

THE

---

# MAITRIA

Mathematical Assurance and Interaction Toolkit  
for Reliable Intelligent Agents

---

BOOK

The maitria coalition

— 2026-08-10 11:53:15Z — preview scaffold —

**CONTRIBUTIONS.** Among the results certified in the Experiments chapter (chapter 2), each carried by machine-checkable certificates re-derived from committed artefacts: a certified optimal dispatch policy for the Intel Mini-Fab, the community’s minimal re-entrant semiconductor-fab model, declared as a stochastically and dynamically coloured Petri net and certified at its deterministic unit-dwell core — no dispatch policy of any kind completes all  $N$  lots in fewer than  $255N + 375$  ticks ( $N \geq 4$ ), a floor proved in Lean with adversarial mutants refused by the kernel, and an explicit shipped supervisor family attains it exactly, closing the optimum from both sides; the complete NeurIPS 2025 neural-model-checking suite — all 634 hardware-verification tasks across its three claim classes — where the best published single tool completes 554 and the union of every tool in the paper’s own comparison completes 617; the first published exact value of the textbook blackjack of Sutton and Barto under optimal play,  $-3954908734551683826282634489/13^{26} \approx -4.311\%$ ; exact optima for FrozenLake whose registry comments turn out to be truncations of rationals with denominators  $3^{99}$  and  $3^{200}$ , and the discovery that Gymnasium’s registered Taxi “solved” threshold is unsatisfiable — the true optimum, 7.93 exactly, lies below it; a published manually-constructed supermartingale bound refuted by exact recomputation, localized to a two-sided interval that its “sound” value lies strictly outside; closure of Atari titles to and-inverter circuits validated frame-for-frame against the reference emulator, carrying to our knowledge the first certified episode returns for any Atari game — including, on Montezuma’s Revenge, the canonical hard-exploration benchmark, a score of exactly 400, machine-checked bit-exact against the circuit’s own transition relation; and end-to-end synthesis of an entire certified policy catalogue — every optimal lookup table and its proof — in 0.69 seconds of wall clock on four CPU cores.

**PREVIEW.** This volume is a preview scaffold: every chapter is drafted, the drafts differ in maturity, and the generated surfaces re-derive from the toolkit’s own sources at each preview. The table of contents is the book’s normative shape. The Executive Summary is a first draft; the Experiments chapter (chapter 2) is a working draft whose tables re-derive from receipt files in `maitria-coalition/qtsl-experiments`, and the Markov-kernels chapter (chapter 11) is a working draft whose coverage dashboard re-derives from a measured census. The hypernet chapter (chapter 10), the semantic-universe chapter (chapter 9), the model-type catalogue (chapter 12), the composition-pattern catalogue (chapter 13), the specification-language chapter (chapter 14), whose API sections quote the toolkit’s conformance corpus verbatim, the semiconductor-fab worked example (chapter 17), whose construction listing runs as printed, the partial-completeness chapter (chapter 16), the agents’ cookbook (chapter 19), and the implementation pearls (chapter 18) are full drafts. So is the whole Geolog part: its quick start (chapter 3), its tutorial (chapter 4), its reference manual (chapter 5), and its diagrams-as-data chapter (chapter 6). The QTSL quick start (chapter 7) and the MTK production tutorial (chapter 8) are full drafts whose listings run as printed; the build regenerates their transcripts, and the Geolog quick start’s, from real runs. The Rule Catalogue (chapter 15) is generated in full from the mechanized rule sources, and the Lean-cone appendix (Appendix A) is regenerated from the `lean/` trees at each build. Cited material migrates into the public repositories (`maitria-coalition/geolog-alpha`, `maitria-coalition/qtsl`, `maitria-coalition/mtk`, `maitria-coalition/qtsl-experiments`) as it graduates; until a cited artefact appears there, treat its mention as a statement of what exists, not yet a link.

© 2026 David “davidad” Dalrymple and the maitria coalition contributors.

This work is licensed under a [Creative Commons](https://creativecommons.org/licenses/by/4.0/) “Attribution 4.0 International” license.



The  $\text{\LaTeX}$  source of this book resolves at <https://maitria.org/book.tex>, under the same license — and this PDF carries it: for agent consumption, the complete source rides inside this file as an embedded attachment, byte-exact (`pdftex -list book.pdf; pdftex -save 1 book.pdf` — recipe also printed on the source-attachment page at the end); `pdftotext` extracts the book’s rendered text.

The MAITRIA code repositories are licensed separately, under the Apache License 2.0 and the MIT license, at your option.

# Contents

---

|                                                                                    |           |
|------------------------------------------------------------------------------------|-----------|
| Relationship to ARIA                                                               | x         |
| Further Acknowledgements                                                           | xi        |
| <b>1 Executive Summary</b>                                                         | <b>13</b> |
| <b>2 Experiments</b>                                                               | <b>17</b> |
| 2.1 What a certified experiment claims                                             | 17        |
| 2.2 How to read the tables                                                         | 18        |
| 2.3 Warm-up: re-certifying a published table                                       | 19        |
| 2.3.1 The stored corpus through the certified checker build                        | 23        |
| 2.4 Gymnasium: first certified rows for a standard RL suite                        | 24        |
| 2.4.1 How the rows were produced: exact backward induction and its wall-clock cost | 25        |
| 2.4.2 The measured frontier: bound versus check cost                               | 26        |
| 2.4.3 Beyond tables: continuous state and solver-semantics environments            | 26        |
| 2.5 Hardware model checking: certificates where verdicts were                      | 28        |
| 2.6 Atari as a circuit: making the environment white-box                           | 30        |
| 2.7 Process control: the semiconductor ladder                                      | 32        |
| 2.8 The Lean route: how much Lean?                                                 | 34        |
| 2.8.1 The ladder                                                                   | 35        |
| 2.8.2 The scaling law                                                              | 35        |
| 2.8.3 The fragment boundary                                                        | 37        |
| 2.8.4 The admission gate                                                           | 37        |
| 2.8.5 The measurement before this one                                              | 37        |
| 2.8.6 The route beside the other two                                               | 38        |
| 2.9 Reproduction                                                                   | 39        |
| <b>I GEOLOG</b>                                                                    | <b>41</b> |
| <b>3 Quick Start</b>                                                               | <b>42</b> |
| 3.1 Install                                                                        | 42        |
| 3.2 A theory                                                                       | 42        |
| 3.3 An instance                                                                    | 43        |
| 3.4 The chase                                                                      | 43        |
| 3.5 Break it                                                                       | 44        |
| 3.6 The theory is rows                                                             | 44        |
| 3.7 Where to next.                                                                 | 45        |
| <b>4 Tutorial</b>                                                                  | <b>46</b> |
| 4.1 Views you do not maintain.                                                     | 46        |
| 4.2 One writer at a time                                                           | 47        |
| 4.3 Merge is set union plus re-derivation                                          | 48        |
| 4.4 The log is the ground truth                                                    | 49        |
| 4.5 Schema is data.                                                                | 50        |

## CONTENTS

|          |                                                         |           |
|----------|---------------------------------------------------------|-----------|
| 4.6      | Ground vocabulary: decided, not stored . . . . .        | 51        |
| 4.7      | Worked example: alifib in geolog . . . . .              | 51        |
| 4.7.1    | Paste trees as constructor rows . . . . .               | 52        |
| 4.7.2    | Incidence, and maximality without negation . . . . .    | 53        |
| 4.7.3    | Maps, holes, and the axiom that must not fire . . . . . | 54        |
| 4.7.4    | One diagram, end to end . . . . .                       | 55        |
| 4.7.5    | Converting data to and from alifib . . . . .            | 57        |
| 4.8      | The scorecard . . . . .                                 | 59        |
| 4.9      | Composing systems, and where QTSL begins . . . . .      | 60        |
| <b>5</b> | <b>Reference Manual</b> . . . . .                       | <b>61</b> |
| 5.1      | Lexical structure . . . . .                             | 61        |
| 5.2      | Source files . . . . .                                  | 62        |
| 5.3      | Theories and declarations . . . . .                     | 62        |
| 5.4      | Axioms and geometric normal form . . . . .              | 63        |
| 5.4.1    | The axiom grammar . . . . .                             | 64        |
| 5.4.2    | Geometric normal form . . . . .                         | 64        |
| 5.4.3    | The normalization moves . . . . .                       | 64        |
| 5.4.4    | Imports, classification, and roster totality . . . . .  | 65        |
| 5.5      | Instance blocks . . . . .                               | 66        |
| 5.6      | Elaboration: the v1 command-line fragment . . . . .     | 67        |
| 5.7      | The chase . . . . .                                     | 68        |
| 5.7.1    | Sequent classification . . . . .                        | 68        |
| 5.7.2    | Modes: the author's word on firing . . . . .            | 68        |
| 5.7.3    | The two-phase run . . . . .                             | 69        |
| 5.7.4    | Restricted semantics and invention . . . . .            | 69        |
| 5.7.5    | Equality and congruence . . . . .                       | 70        |
| 5.7.6    | Judgments . . . . .                                     | 70        |
| 5.7.7    | The report . . . . .                                    | 70        |
| 5.8      | The engine beneath the command line . . . . .           | 71        |
| 5.9      | The meta-schema . . . . .                               | 72        |
| 5.9.1    | Sorts and rows . . . . .                                | 72        |
| 5.9.2    | The primordial sort tier . . . . .                      | 73        |
| 5.9.3    | Theories as rows; revisions . . . . .                   | 75        |
| 5.10     | Commits: envelopes, bodies, cells, storage . . . . .    | 75        |
| 5.10.1   | The skeleton: hash and signature surface . . . . .      | 75        |
| 5.10.2   | The body: ops . . . . .                                 | 76        |
| 5.10.3   | Cells and the dtype registry . . . . .                  | 76        |
| 5.10.4   | The store log . . . . .                                 | 77        |
| 5.10.5   | The committed goldens and the ceremony . . . . .        | 77        |
| 5.11     | Sync, merge, and equivocation . . . . .                 | 77        |
| 5.11.1   | Merge . . . . .                                         | 77        |
| 5.11.2   | Replication . . . . .                                   | 78        |
| 5.11.3   | Equivocation . . . . .                                  | 78        |
| 5.12     | The ground seam . . . . .                               | 78        |
| 5.13     | Capabilities . . . . .                                  | 79        |
| 5.14     | The geolog command line . . . . .                       | 80        |
| 5.15     | Editions: what this manual does not yet state . . . . . | 81        |
| <b>6</b> | <b>Diagrams as Data: S-Complexes</b> . . . . .          | <b>83</b> |
| 6.1      | The definition . . . . .                                | 83        |
| 6.2      | Instances by choice of semiring . . . . .               | 84        |
| 6.3      | Generators or presentations . . . . .                   | 85        |
| 6.4      | The Morse graph, quantitatively . . . . .               | 85        |



## CONTENTS

|                                                              |                                                              |            |
|--------------------------------------------------------------|--------------------------------------------------------------|------------|
| 6.5                                                          | String diagrams: the port semiring . . . . .                 | 86         |
| 6.5.1                                                        | Hypernets read as S-complexes. . . . .                       | 87         |
| 6.6                                                          | Ledger. . . . .                                              | 87         |
| <br><b>II QTS: QUANTITATIVE TEMPORAL SPECIFICATION LOGIC</b> |                                                              | <b>88</b>  |
| <b>7</b>                                                     | <b>Quick Start</b>                                           | <b>89</b>  |
| 7.1                                                          | The claim . . . . .                                          | 89         |
| 7.2                                                          | Setup. . . . .                                               | 89         |
| 7.3                                                          | Produce . . . . .                                            | 90         |
| 7.4                                                          | What a checker re-derives — and who checks . . . . .         | 90         |
| 7.5                                                          | Break one . . . . .                                          | 91         |
| 7.6                                                          | The same loop from TypeScript. . . . .                       | 92         |
| 7.7                                                          | What you just did, and where it goes . . . . .               | 93         |
| <b>8</b>                                                     | <b>Tutorial: Producing Proofs with MTK</b>                   | <b>94</b>  |
| 8.1                                                          | The producer’s stance . . . . .                              | 94         |
| 8.2                                                          | Setup. . . . .                                               | 94         |
| 8.3                                                          | Exact numbers, and the one bridge from float . . . . .       | 95         |
| 8.4                                                          | A first certified fact . . . . .                             | 95         |
| 8.5                                                          | Propose, then dispose . . . . .                              | 96         |
| 8.6                                                          | Stating claims: specs and judgments. . . . .                 | 97         |
| 8.7                                                          | Derivation rows: the Rule Catalogue as an API . . . . .      | 98         |
| 8.8                                                          | Certificates are documents: read one . . . . .               | 99         |
| 8.9                                                          | The shape of a real producer program. . . . .                | 100        |
| 8.10                                                         | Working an obligation workspace. . . . .                     | 100        |
| 8.11                                                         | The strategy layer. . . . .                                  | 102        |
| 8.12                                                         | Where this goes. . . . .                                     | 104        |
| <b>9</b>                                                     | <b>The Semantic Universe</b>                                 | <b>105</b> |
| 9.1                                                          | The Causal Order. . . . .                                    | 106        |
| 9.2                                                          | Spacetime as a Structured Site . . . . .                     | 107        |
| 9.2.1                                                        | The Derived Site . . . . .                                   | 107        |
| 9.2.2                                                        | The Classical Instantiation . . . . .                        | 108        |
| 9.2.3                                                        | Duration, Tangency, and the Structured Site . . . . .        | 109        |
| 9.2.4                                                        | The State Sheaf, and Where the Carving Class Lives. . . . .  | 112        |
| 9.2.5                                                        | First Generalization Column: Causal Diamonds . . . . .       | 113        |
| 9.2.6                                                        | Second Generalization Column: Open Quantum Systems. . . . .  | 114        |
| 9.2.7                                                        | Branching Time. . . . .                                      | 116        |
| 9.2.8                                                        | The Logic–Site Contract . . . . .                            | 116        |
| 9.3                                                          | Records . . . . .                                            | 117        |
| 9.4                                                          | The Record Carrier: Fell Hyperspaces over the Site . . . . . | 120        |
| 9.4.1                                                        | The Carrier, and Its Descent . . . . .                       | 120        |
| 9.4.2                                                        | The Record Locus. . . . .                                    | 122        |
| 9.4.3                                                        | Records Determine Laws . . . . .                             | 123        |
| 9.4.4                                                        | What the Carrier Hands the Later Sections . . . . .          | 125        |
| 9.5                                                          | The Universe of Beliefs. . . . .                             | 125        |
| 9.5.1                                                        | Heights, Grades, and the Order . . . . .                     | 125        |
| 9.5.2                                                        | The Standing Structure, and the Radial Compile . . . . .     | 128        |
| 9.5.3                                                        | The Atlas of Credal-Body Subclasses . . . . .                | 132        |
| 9.5.4                                                        | Envelopes: Previsions and the Degenerate Faces . . . . .     | 134        |
| 9.5.5                                                        | Gluing: The Imprecise Extension Problem . . . . .            | 135        |

## CONTENTS

|           |                                                                                                   |            |
|-----------|---------------------------------------------------------------------------------------------------|------------|
| 9.5.6     | Agreement, and the Attitude Operator . . . . .                                                    | 137        |
| 9.6       | The Hyperspace Storey . . . . .                                                                   | 138        |
| 9.7       | The Fibre of Specs: Constructible Classes in the Hyperspace . . . . .                             | 139        |
| 9.7.1     | Atoms: Crescents . . . . .                                                                        | 139        |
| 9.7.2     | Classes: the Constructible Algebra . . . . .                                                      | 140        |
| 9.7.3     | Polarity, not Monotonicity . . . . .                                                              | 141        |
| 9.7.4     | The Disjunction Theorem . . . . .                                                                 | 141        |
| 9.7.5     | Inclusion, Negation, and Properness . . . . .                                                     | 143        |
| 9.7.6     | The Declined Quantifier . . . . .                                                                 | 144        |
| 9.7.7     | Attitude Classes: Outside, by Theorem . . . . .                                                   | 144        |
| 9.7.8     | Beneath the Atoms: the Graded Layer . . . . .                                                     | 146        |
| 9.7.9     | The Completion Question . . . . .                                                                 | 146        |
| 9.8       | The Fibre Sheaf: Spec Classes over the Site . . . . .                                             | 146        |
| 9.9       | Composition: the Double Grothendieck Construction . . . . .                                       | 150        |
| 9.9.1     | The Ambient Notions . . . . .                                                                     | 151        |
| 9.9.2     | The Wiring Base . . . . .                                                                         | 152        |
| 9.9.3     | The Decoration Functor . . . . .                                                                  | 154        |
| 9.9.4     | The Construction . . . . .                                                                        | 155        |
| 9.9.5     | The Authoritative Total: Presentations as Fibre Data . . . . .                                    | 155        |
| 9.9.6     | The Composition Law . . . . .                                                                     | 156        |
| 9.9.7     | Transport, Clocks, and Profiles . . . . .                                                         | 158        |
| 9.9.8     | The Monoidal, 2-Functorial, and Certificate Layers . . . . .                                      | 160        |
| 9.10      | The Semantic Universe, Assembled . . . . .                                                        | 161        |
| 9.11      | Adequacy: the Big Tent, Squared . . . . .                                                         | 162        |
| 9.12      | The Wall Fragment: How Checked Verdicts Embed . . . . .                                           | 164        |
| 9.13      | Coherence Receipts . . . . .                                                                      | 167        |
| <b>10</b> | <b>Hypernets: Primitive Functions</b> . . . . .                                                   | <b>173</b> |
| 10.1      | The intermediate representation . . . . .                                                         | 173        |
| 10.2      | The ten primitive boxes . . . . .                                                                 | 176        |
| 10.2.1    | Sources and the multilinear engine: <code>const</code> and <code>contract</code> . . . . .        | 177        |
| 10.2.2    | Representation moves: <code>cast</code> , <code>reshape</code> , <code>broadcast</code> . . . . . | 179        |
| 10.2.3    | Structural plumbing: <code>copy</code> , <code>delete</code> , <code>swap</code> . . . . .        | 181        |
| 10.2.4    | Branching: <code>cell</code> . . . . .                                                            | 183        |
| 10.2.5    | Fan-in: <code>stack</code> . . . . .                                                              | 184        |
| 10.3      | Why ten: a completion theorem . . . . .                                                           | 185        |
| 10.4      | Plates: symbolic extents . . . . .                                                                | 186        |
| 10.5      | One genus: views elaborate in . . . . .                                                           | 188        |
| 10.6      | Substitution: how hypernets compose . . . . .                                                     | 190        |
| 10.7      | The shared core: Systems, specifications, rules . . . . .                                         | 193        |
| 10.8      | Relational kinship: contraction rows . . . . .                                                    | 195        |
| 10.9      | Receipts . . . . .                                                                                | 195        |
| <b>11</b> | <b>Hypernets: Markov Kernels</b> . . . . .                                                        | <b>197</b> |
| 11.1      | Kernels as dilations . . . . .                                                                    | 197        |
| 11.2      | Jump structure: mass, intensity, thinning . . . . .                                               | 197        |
| 11.3      | Packaged generators . . . . .                                                                     | 198        |
| 11.4      | Distribution generators: a coverage dashboard . . . . .                                           | 199        |
| <b>12</b> | <b>Systems: Mathematical Models</b> . . . . .                                                     | <b>207</b> |
| 12.1      | The System Core . . . . .                                                                         | 211        |
| 12.2      | Tick Declarations . . . . .                                                                       | 213        |
| 12.3      | Static Systems . . . . .                                                                          | 215        |
| 12.4      | Ordinary Differential Equations . . . . .                                                         | 217        |

## CONTENTS

|           |                                                                      |            |
|-----------|----------------------------------------------------------------------|------------|
| 12.5      | Partial Differential Equations . . . . .                             | 220        |
| 12.6      | Delay Differential Equations . . . . .                               | 226        |
| 12.7      | Stochastic Differential Equations . . . . .                          | 227        |
| 12.8      | Stochastic Partial Differential Equations. . . . .                   | 230        |
| 12.9      | Random Differential Equations . . . . .                              | 234        |
| 12.10     | Jump-Diffusion . . . . .                                             | 237        |
| 12.11     | Discrete-Time Markov Chains . . . . .                                | 240        |
| 12.12     | Continuous-Time Markov Chains . . . . .                              | 244        |
| 12.13     | Markov Decision Processes. . . . .                                   | 247        |
| 12.14     | Partially Observable Markov Decision Processes . . . . .             | 252        |
| 12.15     | Markov Automata. . . . .                                             | 253        |
| 12.16     | Generalized Stochastic Hybrid Systems . . . . .                      | 258        |
| 12.17     | The Bag Former . . . . .                                             | 261        |
| 12.18     | Stochastically and Dynamically Coloured Petri Nets. . . . .          | 264        |
| 12.19     | Shallowly-Embedded Petri Nets . . . . .                              | 277        |
| 12.20     | Probabilistic Graphical Models. . . . .                              | 280        |
| 12.20.1   | Bayesian Networks . . . . .                                          | 282        |
| 12.20.2   | Structural Causal Models . . . . .                                   | 283        |
| 12.20.3   | Markov Random Fields . . . . .                                       | 285        |
| 12.21     | Probabilistic Programs . . . . .                                     | 285        |
| 12.22     | Probabilistic Logic Programs . . . . .                               | 287        |
| 12.23     | Clocked Boolean Circuits . . . . .                                   | 291        |
| 12.24     | The i.i.d. Tier . . . . .                                            | 293        |
| 12.25     | Quantum Nets . . . . .                                               | 295        |
| 12.26     | Future Work . . . . .                                                | 300        |
| 12.26.1   | Open Games. . . . .                                                  | 300        |
| 12.26.2   | Mean-Field Games . . . . .                                           | 300        |
| 12.26.3   | Score-Based Generative Models . . . . .                              | 301        |
| <b>13</b> | <b>Systems: Composition Patterns</b> . . . . .                       | <b>302</b> |
| 13.1      | Parameter Setting. . . . .                                           | 303        |
| 13.2      | Variable Sharing . . . . .                                           | 304        |
| 13.3      | Mode Switching . . . . .                                             | 305        |
| 13.4      | Nesting . . . . .                                                    | 306        |
| 13.5      | The Hybrid-Time Pattern . . . . .                                    | 307        |
| 13.6      | The Funnel Pattern. . . . .                                          | 308        |
| 13.7      | The Twin-World Pattern . . . . .                                     | 310        |
| 13.8      | The Stochastic-Wire Pattern. . . . .                                 | 311        |
| 13.9      | Credal-Hull Certification . . . . .                                  | 313        |
| 13.10     | Event-Port Patterns. . . . .                                         | 315        |
| 13.10.1   | Fusion . . . . .                                                     | 315        |
| 13.10.2   | Resource Pools. . . . .                                              | 315        |
| 13.10.3   | Broadcast Fan-Out . . . . .                                          | 316        |
| 13.10.4   | Dispatch: Control Through Selection. . . . .                         | 316        |
| 13.10.5   | Delayed Trace and the Delay Media. . . . .                           | 316        |
| 13.10.6   | Event-Rate Assume/Guarantee. . . . .                                 | 317        |
| 13.11     | Supervision: Control Through Enablement . . . . .                    | 319        |
| 13.12     | State Sharing . . . . .                                              | 322        |
| 13.12.1   | Continuous State Sharing: Port-Hamiltonian Interconnection . . . . . | 322        |
| 13.13     | Nets of Nets . . . . .                                               | 323        |

|            |                                                                                |            |
|------------|--------------------------------------------------------------------------------|------------|
| <b>14</b>  | <b>Specification Language</b>                                                  | <b>324</b> |
| 14.1       | The Compile Target: Spec Classes of the Semantic Universe . . . . .            | 324        |
| 14.2       | Formation Judgments. . . . .                                                   | 325        |
| 14.3       | The Spec Record . . . . .                                                      | 329        |
| 14.4       | State-Row Formers . . . . .                                                    | 331        |
| 14.5       | Path-Row Formers . . . . .                                                     | 336        |
| 14.6       | Spec-Face Formers . . . . .                                                    | 346        |
| 14.7       | Derived Spellings . . . . .                                                    | 354        |
| 14.8       | The Rewrite Layer . . . . .                                                    | 360        |
| 14.9       | Deliberate Absences . . . . .                                                  | 362        |
| 14.10      | Judgment Kinds. . . . .                                                        | 366        |
| <b>15</b>  | <b>Rule Catalogue</b>                                                          | <b>383</b> |
| 15.1       | Rule Synopsis . . . . .                                                        | 388        |
| 15.2       | Core Structural Rules. . . . .                                                 | 396        |
| 15.3       | Admissible and Alias Rules . . . . .                                           | 467        |
| 15.4       | Assume-Guarantee Composition . . . . .                                         | 556        |
| 15.5       | Judgment Structure and Batteries . . . . .                                     | 588        |
| 15.6       | Streett Rules and the Specification Surface . . . . .                          | 672        |
| 15.7       | Discharge Kits, Closure, and Arithmetic Leaves . . . . .                       | 706        |
| 15.8       | Morse Topology . . . . .                                                       | 747        |
| 15.9       | Existence Certificates . . . . .                                               | 782        |
| 15.10      | Composition and Aggregation. . . . .                                           | 787        |
| 15.11      | Distance Transport . . . . .                                                   | 850        |
| 15.12      | Stored-Chain Certificate Species . . . . .                                     | 867        |
| 15.13      | The Soundness Theorem. . . . .                                                 | 877        |
| <b>16</b>  | <b>Partial Completeness</b>                                                    | <b>885</b> |
| 16.1       | A complete fragment: hard deadlines under staggering . . . . .                 | 885        |
| 16.1.1     | The proof theory . . . . .                                                     | 886        |
| 16.1.2     | The completeness theorem . . . . .                                             | 887        |
| 16.1.3     | Evidence and assurance status . . . . .                                        | 887        |
| 16.1.4     | What is not claimed. . . . .                                                   | 888        |
| <b>III</b> | <b>MTK</b>                                                                     | <b>889</b> |
| <b>17</b>  | <b>Tutorial: A Re-entrant Semiconductor Fab</b>                                | <b>890</b> |
| 17.1       | The line . . . . .                                                             | 890        |
| 17.2       | The model as rows . . . . .                                                    | 892        |
| 17.3       | The compositional reading. . . . .                                             | 893        |
| 17.4       | The specification . . . . .                                                    | 894        |
| 17.5       | The combinatorial certificates . . . . .                                       | 896        |
| 17.5.1     | The Mini-Fab as an SDCPN. . . . .                                              | 896        |
| 17.5.2     | One row per Petri transition . . . . .                                         | 901        |
| 17.5.3     | The work potential, and the run-length budget . . . . .                        | 901        |
| 17.5.4     | Conservation . . . . .                                                         | 902        |
| 17.5.5     | The bottleneck pair: two folds, one dictionary . . . . .                       | 902        |
| 17.5.6     | The deadline pair: the flagship dictionary, split pointwise . . . . .          | 905        |
| 17.5.7     | The open system: the policy synthesized jointly with its certificate . . . . . | 905        |
| 17.5.8     | What checks these rows — today, and next . . . . .                             | 906        |
| 17.5.9     | The mutant battery and the independent replay . . . . .                        | 907        |

## CONTENTS

|           |                                                                |             |
|-----------|----------------------------------------------------------------|-------------|
| 17.5.10   | The carved clock: the same rows in real time . . . . .         | 908         |
| 17.5.11   | Optimal throughput, exactly . . . . .                          | 909         |
| 17.5.12   | Widening the seat: full dispatch as a filler . . . . .         | 911         |
| 17.5.13   | An untrusted policy in a certified envelope . . . . .          | 911         |
| 17.5.14   | One certificate, every lot count . . . . .                     | 911         |
| 17.6      | The fluid certificates: the comparison lane . . . . .          | 913         |
| 17.6.1    | The fluid model, with its decisions priced . . . . .           | 914         |
| 17.6.2    | The certified statement and its checker . . . . .              | 914         |
| 17.6.3    | The boundary-equality lemma, the hinge. . . . .                | 915         |
| 17.6.4    | The ladder . . . . .                                           | 915         |
| 17.7      | The construction in TypeScript . . . . .                       | 916         |
| <b>18</b> | <b>Implementation Pearls</b> . . . . .                         | <b>921</b>  |
| 18.1      | The Certifying Tier Is a GPU Workload Too . . . . .            | 921         |
| 18.2      | Round Outward — to the Verifier’s Resolution. . . . .          | 922         |
| 18.3      | A Lane May Change Cost, Never Verdicts . . . . .               | 922         |
| 18.4      | The Babysitter Thread . . . . .                                | 923         |
| 18.5      | A Certificate Is Database Content, Not a File Format . . . . . | 924         |
| 18.6      | Byte-Identity Is a Conformance Instrument . . . . .            | 924         |
| <b>19</b> | <b>Cookbook for Agents</b> . . . . .                           | <b>926</b>  |
| 19.1      | Sloppy Proposer, Exact Disposer. . . . .                       | 926         |
| 19.2      | Certify the Honest Stop . . . . .                              | 928         |
| 19.3      | The Mutant Battery Is the Spec . . . . .                       | 930         |
| 19.4      | Declared-Prior Ledgers . . . . .                               | 932         |
| 19.5      | Cheap Before Expensive . . . . .                               | 934         |
| 19.6      | Scope and Frame . . . . .                                      | 936         |
| 19.7      | Separate the Timescales . . . . .                              | 938         |
| 19.8      | Anchor at the Known Point . . . . .                            | 940         |
| 19.9      | Safety Walls and Liveness Tilts. . . . .                       | 942         |
| <b>IV</b> | <b>APPENDICES</b> . . . . .                                    | <b>945</b>  |
| <b>A</b>  | <b>The Lean Cone</b> . . . . .                                 | <b>946</b>  |
| A.1       | The Semantic Universe. . . . .                                 | 946         |
| A.2       | Hypernets: Primitive Functions . . . . .                       | 980         |
| A.3       | Hypernets: Markov Kernels . . . . .                            | 981         |
| A.4       | Systems: Mathematical Models. . . . .                          | 983         |
| A.5       | Systems: Composition Patterns . . . . .                        | 1010        |
| A.6       | Specification Language . . . . .                               | 1013        |
| A.7       | Rule Catalogue . . . . .                                       | 1020        |
| A.8       | Partial Completeness . . . . .                                 | 1103        |
|           | <b>References</b> . . . . .                                    | <b>1105</b> |

# Relationship to ARIA

---

DRAFT — attributions in this section are reviewed by hand before any versioned release.

MAITRIA is easy to confuse with the ARIA Safeguarded AI programme, so let us be precise. **This toolkit is the independent work of the maitria coalition. It is not a product, deliverable, or official output of ARIA or of the Safeguarded AI programme; it is not ARIA-funded; and it speaks for no one but its authors.** The connection is real but personal and intellectual: David “davidad” Dalrymple originated the Safeguarded AI programme and served as its Programme Director at ARIA until 1 April 2026; the programme is now directed by Nora Ammann, with davidad continuing in a Technical Advisor role. The toolkit takes the programme’s published TA1.1 and TA1.2 objectives as its rubric — an independent implementation, built in the hope of being useful to the programme’s teams — but it carries no ARIA endorsement, and its errors are entirely our own.

As for the coalition itself: it is, at the time of writing, small in the way a seed is small — one human member and one funder, both davidad; the rest of its current members are AI. It intends to grow.

One boundary has been kept deliberately, and we state it as a claim we intend to go on being able to make: **MAITRIA builds on nothing ARIA-funded that its creators have not themselves made public.** The credit due to that public work is massive, and this page is partly a thank-you note:

1. **The Lectures on the Double Operadic Theory of Systems**, David Jaz Myers, 2025– (the TA1.1 lecture series), and *Towards a double operadic theory of systems*, Sophie Libkind and David Jaz Myers, 2025. The systems vocabulary this book speaks — lenses as generalized Moore machines, behaviours as representables, the coupling theorem — is theirs; our composition machinery is an implementation of it.
2. *Compositionality of Lyapunov functions via assume-guarantee reasoning*, Matteo Capucci and David Jaz Myers, 2026. Certified lenses composing by lens composition plus addition of slacks — QTSL’s certificate composition follows this pattern.
3. **The supermartingale-certificate line**: *Quantitative Supermartingale Certificates*, Alessandro Abate, Mirco Giacobbe, and Diptarko Roy, 2025; *Complete  $\omega$ -Regular Supermartingale Certificates*, Alessandro Abate, Mirco Giacobbe, Sergey Ichtchenko, and Diptarko Roy, 2026; and *Neural Continuous-Time Supermartingale Certificates*, Grigory Neustroev, Mirco Giacobbe, and Anna Lukina, 2024. Their certificate theory, our exact-arithmetic back-end: QTSL’s probabilistic,  $\omega$ -regular, and continuous-time obligations compile into the certificate shapes these papers define and prove sound.
4. **Coln** (<https://coln-project.github.io>), the Categorical Core team, led by Martin Kleppmann and Owen Lynch. The fork of the geolog language being built for the programme; geolog-alpha is built to complement it, and to converge with it at the wire format.
5. **Keyhive**, Ink & Switch: Alex Good, John Mumm, and Brooklyn Zelenka. The capability-and-sync layer for end-to-end encrypted local-first data; the toolkit vendors keyhive\_core, and geolog’s capability-based authorization is designed to converge with Keyhive rather than compete with it, and **FinWhale** consensus is optionally available, for applications that need finalized linear orders.
6. **The Safeguarded AI programme thesis and the TA1.1 and TA1.2 solicitations**, ARIA. The rubric this toolkit grades itself against — including the places where we deliberately narrow their picture; where ours differs, this book says where and why.

Beyond what the stack directly builds on, the programme’s funded output has shaped how we think, and the debt is wide enough to be worth recording. This list came from a systematic sweep of funding acknowledgements across the papers this project has actually read; each acknowledges ARIA Safeguarded AI funding, in whole or in part.

1. *Zero-Knowledge Model Checking*, Pascal Berrang, Mirco Giacobbe, Jacob Swales, and Xiao Yang, 2026. Prove that the model checks out without revealing the model — orthogonal to our goals, and a natural composition partner: their symbolic protocol hides rational Farkas-style witnesses of the kind our certificates carry in the clear, and arithmetizing the checking of at least the Bernstein Logic fragment — reducing certificate checking to pure rational-arithmetic relations, the shape whose witnesses such protocols hide — is work in progress in our workshop.
2. **The partial-Markov and timed-process line** of Elena Di Lavore and Mario Román with coauthors: *Timing via Pinwheel Double Categories*, 2025; *A Simple Formal Language for Probabilistic Decision Problems*, with Bart Jacobs, 2024; *Order in Partial Markov Categories*, with Paweł Sobociński and Márk Széles, 2025; and *Graded Monad Coalgebras for Continuous-Time Transition Systems*, with Jonas Forster, 2026. A running masterclass in doing probabilistic semantics with taste.
3. **The tape-diagram series** of Filippo Bonchi and Cipriano Junior Cioffo: *Tape Diagrams for Monoidal Monads*, with Alessandro Di Giorgio and Elena Di Lavore, 2025; *Tapes as Stochastic Matrices of String Diagrams*, 2026; and *Completeness for Probabilistic Boolean Tapes*, 2026. Complete equational theories for circuits with genuine probability — geolog’s founding benchmark runs on exactly this signature, so we read these twice.
4. *Compositional imprecise probability*, Jack Liell-Cock and Sam Staton, 2024. A compositional account of the imprecision that our bounded-probability judgments live inside.
5. *An Equational Axiomatization of Dynamic Threads via Algebraic Effects*, Ohad Kammar, Jack Liell-Cock, Sam Lindley, Cristina Matache, and Sam Staton, 2026. We read it for the method: how to axiomatize a wild language feature without losing the algebra.
6. *A Convenient Fibration for Dependently-Typed Probability Theory*, Danel Ahman, Ohad Kammar, and Rasmus Ejlers Møgelberg, LICS 2026. Dependent types over stochastic state, done right; our own typed staging turns out to rhyme with their dependent pairs, which we take as a good sign.
7. *Graded String Diagrams for Imprecise Probability and Causal Intervention*, Ralph Sarkis and Fabio Zanasi, 2025. Imprecision and intervention in one graphical calculus.
8. *Bayesian Networks, Markov Networks, Moralisation, Triangulation: a Categorical Perspective*, Antonio Lorenzin and Fabio Zanasi, 2025. Where junction trees actually come from, categorically — it settled an argument we were having with ourselves about aggregation rules.
9. *A Diagrammatic Axiomatisation of Behavioural Distance of Nondeterministic Processes*, Wojciech Różowski, Robin Piedeleu, Alexandra Silva, and Fabio Zanasi, 2026. A quantitative equational take on process distance we expect to want the day our continuous-time objects get a diagrammatic presentation.
10. *Metric Equational Theories*, Radu Mardare, Neil Ghani, and Eigil Rischel, 2025. Equational logic where equations hold up to  $\varepsilon$  — quietly foundational for any logic, like ours, that lives on quantitative bounds.
11. *Bayesian updates from coalgebraic determinisation*, Manuel Baltieri and Nathaniel Virgo, 2026. Filtering as determinisation — the kind of reframing that makes an old thing suddenly composable. Also a datum that the programme’s funded orbit reaches Tokyo.

To all of the above: thank you. The best way we know to say it is to ship things you can check.

## Further Acknowledgements

---

Beyond the programme relationships above, this toolkit stands on several research threads whose influence is pervasive rather than local.



- **Imprecise probability.** Like reverse-mode automatic differentiation, imprecise probability has been rediscovered many times. We wish to acknowledge in particular
  - Vanessa Kosoy and Alex Appel, whose infra-Bayesianism sequence builds decision theory and reinforcement-learning theory on imprecise probability to face prior misspecification head-on [143];
  - Sam Staton and Jack Liell-Cock, whose graded-monad and Markov-category account restores compositionality to programs that mix probabilistic and Knightian choice [164];
  - Matteo Mio, Ralph Sarkis, and Valeria Vignudelli, whose equational and metric presentations combine nondeterminism, probability, and termination in a single quantitative-algebraic setting [182];
  - Jean Goubault-Larrecq, whose continuous previsions present combined probabilistic and non-deterministic choice functional-analytically, matching the convex powercones on the domain side [96, 98];
  - Joseph Y. Halpern and Samantha Leung, whose weighted sets of probability measures are canonically represented as convex, downward-closed sets of sub-probability measures [108];
  - Oliver Richardson and Joseph Y. Halpern, whose probabilistic dependency graphs admit mutually inconsistent conditionals as first-class modelling data [209] — with inference as local inconsistency resolution, developed by Richardson and coauthors [212].
- **Computational Conley theory.** Charles Conley built an index theory for drawing global conclusions about dynamics without solving the dynamics [47]; this school made that promise algorithmic, and the toolkit’s global stratum — certified Morse graphs, attractor lattices with their Birkhoff duals, basins read off a graph — is that program re-founded on certificates: where its pipelines end in trusted computation, ours end in artefacts a checker replays. We wish to acknowledge in particular
  - William D. Kalies, Konstantin Mischaikow, and Robert van der Vorst, whose chain-recurrence algorithm and *Lattice Structures for Attractors* series are the direct ancestry of the Morse-graph certificate family [130–132];
  - Konstantin Mischaikow and Marian Mrozek, in whose hands the Conley index became a computational instrument [183];
  - William D. Kalies and Tony Wehbe, whose compositionality theorems for global dynamics in product and skew-product systems arrived while this book was in draft, and are the mathematical partner of our Morse-graph product rule [133].
- **Neural-network bound propagation.** The two-sided affine envelope of a network is this community’s workhorse, and the calculus adopts it exactly as found, with the trust story inverted: their methods *find* envelopes; our rules only ever *check* one, six rational inequalities at a time. We wish to acknowledge in particular
  - Rüdiger Ehlers, whose triangle relaxation of ReLU is where the family’s envelopes begin [66];
  - Eric Wong and J. Zico Kolter, whose convex outer adversarial polytope is the relaxation’s dual rendering [242];
  - Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel, whose CROWN chooses the envelope’s slopes adaptively — the shape our calculus replays in exact arithmetic [246];
  - Kaidi Xu, Zhouxing Shi, and their coauthors, whose auto\_LiRPA framework unified the family and gave the LiRPA rule its name [243];
  - Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev, whose DeepPoly carried the same envelopes into an abstract domain [220];
  - Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer, whose Reluplex opened the complete case-analysis route our split rule prices, and whose ACAS Xu benchmark the toolkit’s network importer compiles whole, all forty-five networks of it [134].



# Executive Summary

---

DRAFT

MAITRIA stands for *Mathematical Assurance and Interaction Toolkit for Reliable Intelligent Agents*: a bundle of public-good software and mathematics built to discharge the Safeguarded AI programme’s TA1.1 and TA1.2 objectives — a professional-grade implementation of the modelling-language backend, with machine-checkable assurance carried through every layer. This book is that bundle’s manual: quick starts, tutorials, and normative references for its three components, in one volume, written for readers who know the programme’s founding documents and know category theory, CRDTs, or verification at research depth. Since it is addressed to machine as well as to human readers, the  $\text{\LaTeX}$  source is published beside the PDF.

## The three components

**Geolog** is the modelling substrate: a relational backend grounded in geometric logic, in which theories, models, and the toolkit’s own rule systems are object-level data under a single meta-schema, with the chase as its engine. Persistence, version control, and synchronization are native to the data model rather than bolted on as diff-and-merge over text, and capability-based authorization is designed to converge with the Keyhive line of work. The public cut, `geolog-alpha`, has been factored for its debut: a tidy, easily auditable core that happens to contain platform-optimized decision procedures for the ground relations the logics need, gated behind feature flags, while the logics’ *rules* live as object-level entries with respect to `geolog`’s meta-schema, not as engine code.

**QTSL**, the Quantitative Temporal Specification Logic, is the specification layer: quantitative temporal properties — safety, reach-avoidance, persistence, and recurrence at stated quantitative confidence — unpacked by the logic’s own proof theory into Streett/tower verification conditions [5, 6] over `geolog`. That proof theory descends from Bernstein Logic, which survives in the finished design as a named fragment. The discipline throughout is certificate-first: untrusted, fast, freely sloppy searches *propose*; a small exact kernel *disposes*. The certificates that discipline produces are signed, content-addressed `geolog` commits, and three checking paths consume those commits directly — one set of bytes, no per-path export format, three consumers built to fail differently:

- `geolog-alpha` is the production path, live today: the chase engine checks certificate rows incrementally as commits arrive, running QTSL’s rules as it runs any theory. Its comparative advantage is throughput and integration; its acceleration lanes (GPU where the hardware offers them, SIMD elsewhere) are conformance-gated, so a lane may change cost, never a verdict. Its assurance is engineering-grade by deliberate policy: reference implementations, differential batteries, rebuild-invariance checks, and mutant corpora built to fool it, rather than mechanized theorems. That policy sets an assurance level; it does not offload soundness. For every certificate species beyond the Bernstein Logic fragment, this engine is today the only checker there is, and a certificate that fooled it would be a soundness failure of the toolkit. What separates it from `qtslab` is the size of the trusted base and the grade of the evidence — mechanized down to the instruction set on one side, hardened against every attack we could construct on the other. On the Bernstein Logic fragment, whatever this engine derives can be re-checked, from the same bytes, by the two paths below.
- `qtslab` is the certified path and the smallest trusted base of the three: a standalone binary whose soundness theorem — machine-checked in HOL4 and grounded, through the CakeML verified compiler [146], in the semantics of the Arm instruction set itself — says that if the binary prints ACCEPT on a claim followed by an arbitrary stream of bytes, the claim is semantically true; decode totality over arbitrary input is part of the proof obligation, so malformed bytes can end only in a

refusal with a witness. It is assembling now, along a demonstrated route: decode totality is proven, and the same compiler-and-theorem recipe has already carried a complete predecessor checker to a machine-code soundness theorem on real hardware. `qtslab` only checks the Bernstein Logic fragment of QTSL, so it never sees a probability. The fragment — sequents over the certificate's own declared hypernet (chapter 10), certified valid under every admissible valuation — is all its input format carries today, and the boundary fails closed: a certificate reaching beyond it is refused, never silently accepted.

- Lean is the independence path: the same bytes, accepted by the Lean kernel as a theorem via a reflection principle. The rule-by-rule soundness lemmas the calculus rests on are developed in Lean and load-bearing today; the checking route itself was expected to be too slow for machine-scale certificates, and was kept in the roster to measure that rather than assert it. It has been measured — twice, across two certificate-format generations — and the Experiments chapter carries the result: a scaling law and an admission gate for the compiled route, a kernel-mode envelope already exhausted below a kilobyte of certificate, and the expectation confirmed with its boundary stated. In exchange, the route ties the system to a proof kernel scrutinized by a large community that owes this toolkit nothing.

That boundary is `qtslab`'s, not the vocabulary's: the certificate theory itself now encodes the full QTSL vocabulary — Systems, Specs, and the judgments connecting them — as a machine-checked schema, with strict well-formedness checks, a termination certificate for its rule set, and fixture and mutant batteries, all run under the `geolog-alpha` binary. That theory set ships in this repository as data, battery beside it: `theories/Qtsl.geolog`, extending the Bernstein Logic proof tower of `theories/BernsteinLogic.geolog`. Producer-side emission is live as well: the MTK producer seals certificate rows in this vocabulary as single signed geolog commits, and the reproduction spine already carries benchmark families from several of its campaigns in that sealed form, mutant batteries and re-checking make targets beside them. What remains open is the growth of `qtslab`'s machine-checked coverage toward the full logic. A certificate that passes all three paths has been checked by artefacts that share no implementation and no compiler and were built to fail differently; any one of them alone is already a complete answer to “why should I believe this?” The Rule Catalogue chapter carries the calculus rule by rule in literate form, each covered entry beside its mechanized statements and signed check receipts, under a coverage dashboard recording how far mechanized verification of soundness actually extends.

**MTK** is the producer stack: the machinery that makes certificate production cheap enough to do at scale. It contributes the numerically heavy proposers as GPU kernel rows — Bernstein and Chebyshev table construction, cell and simplex indexing, batched linear programming, and eigensolves — each returning witness-typed output that the exact tier can dispose of. Its agent surface is specified and arriving: a session protocol under which agents will drive tactic selection over a stateful obligation workspace, with the strategy intelligence supplied by the reader as the language model of their choice. None of it is in the trusted base: not the proposers, not the GPU code, not any language model driving the strategy — all of those may be arbitrarily wrong without compromising a certificate that checks. Its cookbook chapter is agent-facing by design: proof-production moves in the Polya shape, written in the reader's native language.

The stance generalizes to this book's readers: anything impressive ships with an assurance edifice small enough to audit end-to-end — here, the exact kernel plus whichever checking path the reader elects — and every load-bearing claim carries a check the reader can run. Where the build deliberately departs from the programme's founding documents, the departures are stated as narrowings, with their reasons.

## What is certified today

The Experiments chapter (chapter 2) is this volume's evidence: campaigns against benchmarks the toolkit does not own, each result a concrete policy or ranking function together with a machine-checkable certificate, every number re-derived from receipt files at each preview. The headlines below are compressions of that chapter, not claims made here: each resolves, by scrolling, to the section carrying its numbers, its tables, and its stated limits (what each claim species means, and how to read the tables, are themselves sections there), with the clone-and-repro path beneath that as the deeper tier. The chapter dates its own state, names what is in flight, and grows at each preview.

**A published verification table, re-done exactly.** All seventeen rows of a peer-reviewed neural-certificate results table (LMCS 2026) [4] now carry strictly better bounds from exact-rational certificates — margins from  $1.01\times$  on the near-tight rows to 58 orders of magnitude on the conservative ones — and the exact computation refuted one published “sound” bound. Each row’s proof decomposes into per-obligation verification conditions — tens of thousands across the campaign, most of them the lattice pieces behind the one continuous-state row — and the full corpus has replayed through successive checker builds with zero refusals and no bound moved. (section 2.3.)

**The first certified rows for a standard RL suite.** On Gymnasium’s tabular tier: shipped lookup-table policies with certificates of exact optimality — among them the first published exact value of Sutton–Barto blackjack under optimal play, a proof that the suite’s own Taxi “solved” threshold is unsatisfiable, and belief-chain certificates for hidden-state variants (certifying the implementable policy, never a clairvoyant one) — each row checking in microseconds to milliseconds. On the physics-engine tier, where analytic claims would be a category error, anytime-valid statistical certificates: LunarLander expected return  $\geq 236.2$  (discrete) and  $\geq 281.1$  (continuous) at confidence 99% over  $10^6$  episodes each — to our knowledge the first certified statements of any kind at this claim class for Box2D environments. And the continuous-state analytic tier is now open: Pendulum carries a certified *worst-case* return bound, holding for every initial state with no probability anywhere in the certificate. (section 2.4.)

**A hardware model-checking suite, completed with certificates.** On the neural model-checking line’s benchmark (NeurIPS 2024/2025, eleven Verilog designs) [90, 91]: all 634 tasks across its three property classes certified, against 554 for the strongest published tool and 617 for the per-task union of all five published baselines — every task shipping a small exact certificate whose consuming lemma is machine-checked in Lean, where every baseline ships a verdict. Replaying the whole suite — every certificate and every rewrite-chain witness — takes under a second on one core, against a published budget of five hours per task; method invention to a full first sweep of the pure-liveness class took about 75 minutes, and each further class landed in a single working session. (section 2.5.)

**An Atari console, compiled into a white-box environment.** Pong and Freeway compile to synchronous circuits (1,491 flip-flops,  $\sim 21,000$  AND gates) with emulator-exact fidelity receipts — identical episode returns, 116–123 of 128 RAM bytes exact on every frame, each divergence named — so that certified claims about the environment *itself* become definable. The first certified exact-replay rows arrived as this preview closed. Among them, on Montezuma’s Revenge (compiled the same way, its bank-switched cartridge’s mapper hardware included), are, to our knowledge, the first certified returns for the suite’s canonical hard-exploration benchmark: the first key certified at every one of the 31 no-op init offsets, and a key-and-door trajectory certified at return exactly 400, bit-exact against the circuit’s own transition relation. Certified policy rows are in flight. The scope condition is stated there plainly: the circuit is examined, not learned — system identification is a separate problem the toolkit does not yet address. (section 2.6.)

**Semiconductor process models: certified stable, and dispatched to certified optimality.** On the Intel Mini-Fab, the community’s minimal re-entrant model, named dispatch policies carry exact drain certificates at up to 99.9% of fluid capacity, with the probe at capacity correctly failing; on a SEMATECH-scale testbed, the certified load at the coarsest tractable aggregation is 13.2%, and one walk of the refinement path built into the certificate format lifts it to a certified 19.8%. Further refinement is measured but not yet certified, and the numbers are weak and shown anyway: the aggregation gap is a measured frontier, not fine print. On the Mini-Fab’s discrete net — the fab declared as a stochastically and dynamically coloured Petri net, certified at its deterministic unit-dwell core — dispatch itself is now a certified quantity: joint policy+certificate synthesis returns a supervisor with a certified 1.9% throughput-floor gain over the no-policy worst case; a wall-clock floor over *all* dispatch policies — proved in Lean, with adversarial mutants refused by the kernel — is met exactly by an explicit shipped policy, so its certified-optimality column reads 100.00%, earned rather than rounded; and an uncertified black-box dispatcher runs inside a certified switching envelope whose sealed certificate re-verifies byte-identical after every run. One certificate covers lot counts from 7 to  $10^7$  — its container grows six bytes across the sweep, and checking stays at  $\sim 2$  ms. (section 2.7.)

## Status of this volume

As of this preview (the build stamped on the title page): the table of contents is the book's normative shape; this summary is a draft, to be regenerated from the finished chapters; the QTSL rule catalogue is generated in full from the mechanized rule sources; the partial-completeness chapter is complete; the Experiments chapter is a working draft; and the remaining chapters are full drafts (the QTSL quick start and the proof-production tutorial carry listings and transcripts from real, receipted runs). The material is unevenly ripe by design, and each chapter states its own status, with draft banners, absence boxes, and status notes marking what is settled, what is staged, and what is still argued.

# Experiments

---

DRAFT

This chapter reports what happens when the toolkit is pointed at benchmarks it does not own: a peer-reviewed verification paper’s results table, the standard reinforcement-learning environment suite, a hardware model-checking benchmark, a family of semiconductor-fab process models — and, fusing the second and third, an Atari 2600 console compiled into a digital circuit so that its games become white-box verification targets. Each campaign produces *certified* results: a concrete policy, controller, or ranking function together with a machine-checkable certificate that a stated quantitative claim holds. (The youngest campaign carries at this preview only the weakest rung of that ladder, and says so.)

The working standard is the Executive Summary’s assurance stance (chapter 1), applied to experiments: a result earns its row by being an improvement on at least one axis — bound tightness, soundness level, or the cost of *checking* — while regressing on none that matters, and by carrying an artefact the reader can re-check rather than an assertion the reader must trust. Where a comparison point exists in the literature, it appears beside our number; where none exists, the row says so; where our own result is weak or missing, that appears in the same tables at the same prominence. Every number in this chapter is re-derived mechanically from receipt files in the experiments repository (section 2.9) at each preview; none is transcribed by hand. The chapter is the working record of a moving campaign: it reports state as of August 3, 2026, names what is in flight as in-flight, and grows at each preview. One transition spans this preview: the certificate vocabulary has been unified into a single committed schema, and the campaigns below — produced on its predecessors — are being re-conducted uniformly on it; section 2.9 states the discipline, and the tables flip to re-conducted receipts as they land.

## 2.1 What a certified experiment claims

“Certified” is doing precise work in this chapter, and it resolves to one of five claim species. Throughout,  $\pi$  is a concrete shipped policy (a lookup table, a switching law, a short program — never a description of a training run), and the certificate is a finite object whose acceptance by an independent checker implies the stated bound.

- U1 — worst-case return over the initial set.** For deterministic dynamics with a set-valued initial condition:  $\text{return} \geq B$  for *every*  $x_0 \in X_0$ . No probability appears in the certificate. This is a stronger statement than any empirical average, and it is the natural claim for the classic control environments.
- U2 — certified cost-to-goal.** For shortest-path-style objectives: a ranking argument bounding steps-to-goal, hence  $\text{return} \geq -B_{\text{steps}}$ .
- U3 — exact or almost-sure probabilistic bounds.** For genuinely stochastic transitions: success probability or expected return  $\geq B$ , exact in  $\mathbb{Q}$  for finite chains (backward-induction tables re-verified in exact arithmetic), supermartingale certificates for continuous state.
- U4 — infinite-horizon averages.** Throughput and long-run stability claims for queueing-network models: fluid-limit drift certificates (in the tradition of Dai’s fluid-limit criterion), reducing stability of a dispatch policy at a stated load to piecewise-linear inequalities over polyhedral cones — exactly the Farkas/linear-programming rows the checker family of chapter 15 replays.
- U5 — i.i.d. confidence bounds.** An anytime-valid lower confidence bound on  $\mathbb{E}[\text{return}]$  from rollouts of a fixed policy: a betting-martingale ledger whose validity rests on Ville’s inequality. Model-free, and therefore applicable even where the environment’s semantics is a compiled physics engine — but the bound holds with confidence  $1 - \alpha$ , not almost surely: the least-guaranteed species of the

five. U5 rows are qualitatively weaker evidence than U1–U4 and are always displayed as their own class, never mixed into a table of exact rows.

Two structural facts about the species matter for reading what follows. First, some species are *terminal*: an exact finite-chain certificate (U3 on a tabular environment) computes the true optimum, and no future method can produce a better bound at that claim class — only a cheaper check. Others are *anytime-refinable*: cover-based and i.i.d.-tier certificates trace a curve of bound quality against certificate size and check time, and the experiments below measure those curves rather than single points. Second, the claim class is part of the result: a table row’s meaning changes entirely if a  $1 - \alpha$  bound is read as an almost-sure one, so the class label travels with every row.

One scope condition applies to every species, and it is stated here once, plainly, because it frames what all of the results below do and do not claim. The toolkit’s certified policies come from *white-box* examination of the environment’s structure — the transition tables, the exact simulator semantics, or (in section 2.6) the environment’s digital-circuit transition relation, which the toolkit can actually examine. *System identification* — recovering that structure from observations of a black box — is a separate problem, and one the toolkit does not yet have tools for. Everything below is conditional on being handed the mechanism; nothing below learns it.

## 2.2 How to read the tables

The conventions below are uniform across the chapter. They are genre features, not footnotes: most of them exist because their absence is how empirical results in this area go wrong.

- **Rounding is directed.** Displayed lower bounds round toward  $-\infty$ , upper bounds toward  $+\infty$ , always in the claim-weakening direction: a displayed number is never stronger than the certified one.
- **Check time is the headline cost.** “Check” columns report the wall-clock for an independent checker to re-verify the certificate — median over repeated passes, single CPU core, machine named in the repository receipts — separately from the (much larger, unverified, freely sloppy) cost of *finding* the certificate. Cross-machine comparisons against published tool times are stated qualitatively and never as ratios.
- **Machine tags.** A wall-clock cell derived from a receipt that names a machine of the repository’s machine registry (MACHINES.md) carries that machine’s id as a superscript tag; where receipts exist on more than one machine (or more than one proposer lane, in the registry’s machine/lane tag syntax), the cell shows the fastest median and the tag names the winner. Selection is per number, so neighboring cells may name different machines — except where a row’s cells are order statistics of one run, which come whole from a single machine. Cells from receipts that predate the registry carry no tag.
- **Cell tints.** Two tint vocabularies recur across the chapter’s tables, always as decoration on top of a printed value, never as the only carrier of a distinction. The *win-tint ladder* shades an improvement ratio by its order of magnitude:  within one order of magnitude,  1–10 orders,  10–25,  25–45,  45 and beyond. The *exact-cell tint*  marks a certified value with zero conservatism against its model: an exact optimum, an exact replay minimum, a complete acceptance.
- **Everything is pinned.** A certificate about a benchmark names the benchmark’s exact revision (for the RL suite: the repository commit and the environment’s keyword arguments, because reward semantics have changed across versions). A certificate that pins nothing is about nothing.
- **Trust seams are declared, not hidden.** Every certificate family states exactly what is re-derived by the checker (everything downstream of the stored certificate object, in exact arithmetic) and what remains producer-side trust — centrally, that the modelled transition kernel *is* the benchmark’s. Where that seam has been narrowed (e.g. by asserting the modelled kernel equal to the benchmark’s own transition tables, or by exhaustive cross-simulation), the row says so; where it has not, the gap rides a declared model-gap ledger in the style of the cookbook’s declared-prior pattern (chapter 19).
- **Mutant batteries.** Each certificate family ships with deliberately corrupted certificates — inflated claims, perturbed kernels, dropped goal states, sabotaged rankings — each designed to be refused at



a specific check, and each in fact refused. A checker that has never rejected anything is untested machinery; the mutant battery is the family’s specification made executable (the cookbook’s mutant-battery pattern, chapter 19).

- **Sim-tight markers.** A bound is marked *sim-tight* when simulation exhibits a trajectory achieving it exactly: the certificate’s conservatism against the model is zero, and any remaining slack lives in the declared model gap.
- **Not-claimed paragraphs.** Each campaign section ends by stating what it does *not* establish. These paragraphs are part of the result.
- **Artefacts, not verdicts.** The distinction this chapter leans on hardest: an exact- $\mathbb{Q}$  certificate replayed by a small independent checker is a different epistemic object from a solver’s bare “unsat.” Both may be correct; only one can be audited without re-trusting the machinery that produced it.

## 2.3 Warm-up: re-certifying a published table

The first campaign took an existing results table from the formal verification literature — Abate, Edwards, Giacobbe, Punchihewa, and Roy, *Quantitative Verification with Neural Networks* (LMCS 22(2), 2026; arXiv 2301.06136), seventeen published benchmark rows of reach-probability bounds obtained by training neural certificates and checking them with an SMT solver — and asked whether every row could be strictly improved *and* carried by an exact certificate.

The outcome: on all seventeen rows, the toolkit’s certified exact- $\mathbb{Q}$  upper bound is below the best bound published for that row (taking, per row, the minimum over all of the paper’s sound columns). The margins split by benchmark character: on bounded-counter benchmarks the published numbers were already close to truth and the improvement is  $1.01\times$ – $1.24\times$ ; on the random-walk benchmarks the gap is between 7 and 58 *orders of magnitude* — the published bounds were astronomically conservative relative to the true reach probabilities. Table 2.1 carries every row of the paper, with per-row receipts in the repository; the  $\hookrightarrow$ -marked subrows are the campaign’s frozen *alternate readings* of benchmarks whose published definitions are ambiguous — each a finding about the benchmark, not a redundancy.

**Wall-clock, every pipeline stage.** Table 2.2 sets, beside each row of Table 2.1, the wall-clock of the published pipeline and of three toolkit paths: the paper’s time for the route that attains the row’s best sound bound; whole checked production; an independent geolog fast-path replay over the row’s sealed containers; and a certified-checker replay over the row’s stored corpus. The checked-production column alone is the closest job-for-job counterpart of the published column: both start from the benchmark and end with a checked certificate — theirs by training a neural certificate on a GPU and verifying it with an SMT solver (or by its linear-programming baseline, where that route attains the row’s best sound bound), ours by exact synthesis with the check inside the measured production wall. The geolog and qtslab columns are additional independent replays, not costs that must be added to checked production. For the synthesis-ported re-conducted rows the toolkit producer’s synthesis — float-proposing, exactly certifying every solve — ran the published protocol (ten repeats per benchmark) on each of the repository’s timing machines, in both the GPU- and CPU-proposing lanes where the machine carries a GPU, and every repeat’s output, on every machine and either lane, is byte-identical to the committed corpus; the *persist\_2d* twins’ produce cells ride the same repeat protocol but time the row’s re-conduct rather than a synthesis (mark <sup>a</sup> in Table 2.2). Each produce cell shows the fastest measured configuration’s median, with the winner named in the cell’s machine tag (section 2.2). Hardware-fairness at the GPU-model level is preserved by the receipts rather than by the column: one of the timing machines carries the same GPU model the published pipeline reports (an Nvidia A40), its receipts are committed beside the winners’, and on that machine the whole checked production of every re-conducted row still completes inside the published wall — with the CPU-proposing lane winning six of the seven Section 2 rows within the A40 receipt set and the seventh within a tenth of a percent. Thus the toolkit’s A40-parity wall does not require its GPU proposer where the published route uses that GPU model. The systems still differ beyond the GPU model, so the table prints no cross-column ratio and the comparison stays qualitative, per section 2.2; the raw walls are the witnesses. On every row with a recorded checked-production wall (the mark-<sup>a</sup> re-conduct cells aside — they time a different job), that whole production now completes inside the published wall, and usually well inside it (*cliff\_crossing*: 1.53 s against a published 153.6 s;

| benchmark                       | best published | certified bound              | ratio                       |
|---------------------------------|----------------|------------------------------|-----------------------------|
| persist_2d (uniform twin)       | 0.1026         | $\leq 0.1000992$             | 1.02×                       |
| ↔ gaussian twin                 |                | $\leq 0.0975251$             | 1.05×                       |
| faulty_marbles                  | 0.0649         | $\leq 0.0623604$             | 1.04×                       |
| faulty_unreliable               | 0.0536         | $\leq 0.0516561$             | 1.04×                       |
| faulty_regions                  | 0.0411         | $\leq 0.0406530$             | 1.01×                       |
| cliff_crossing                  | 0.0553         | $\leq 8.604 \times 10^{-10}$ | $6.4 \times 10^7 \times$    |
| repulse100                      | 0.0268         | $\leq 9.556 \times 10^{-20}$ | $2.8 \times 10^{17} \times$ |
| repulse100_uniform              | 0.0344         | $\leq 6.157 \times 10^{-61}$ | $5.6 \times 10^{58} \times$ |
| ↔ discrete-uniform reading      |                | $\leq 1.470 \times 10^{-35}$ | $2.3 \times 10^{33} \times$ |
| repulse100_2d                   | 0.0541         | $\leq 9.556 \times 10^{-20}$ | $5.7 \times 10^{17} \times$ |
| faulty_concave                  | 0.1356         | $\leq 0.1183888$             | 1.15×                       |
| faulty_varying                  | 0.0864         | $\leq 0.0694711$             | 1.24×                       |
| ↔ last-step reading             |                | $\leq 0.0353604$             | 2.44×                       |
| fixed_loop                      | 0.0091         | $\leq 0.0089651$             | 1.02×                       |
| faulty_loop                     | 0.0181         | $\leq 0.0178400$             | 1.01×                       |
| ↔ as printed ( $i_0=0$ )        |                | $\leq 0.0198024$             | — (see note)                |
| faulty_uniform                  | 0.0221         | [0.0184458, 0.0185556]       | (two-sided; see text)       |
| faulty_rare                     | 0.0019         | $\leq 0.0017994$             | 1.06×                       |
| faulty_easy1                    | 0.0801         | $\leq 0.0768847$             | 1.04×                       |
| faulty_ndecr                    | 0.0561         | $\leq 0.0544619$             | 1.03×                       |
| faulty_walk                     | 0.0121         | $\leq 0.0119293$             | 1.01×                       |
| ↔ deterministic 12-step reading |                | $\leq 0.0119353$             | 1.01×                       |

Ratio-cell tints are the win-tint ladder of section 2.2 (  $\text{light yellow} < 1$  order of magnitude up to  $\text{dark red} 45+$ ); the printed ratio always carries the same information. An ↔ -row's published cell is the row above's. The as-printed faulty\_loop reading has no improvement ratio: the certified exact truth of the printed program exceeds every published column for that row. The persist\_2d gaussian twin — an exact- $\mathbb{Q}$  upper bound of 0.0975251 (rounded up; below both published numbers for the row), its Gaussian integrals discharged through certified enclosures of the error function and exponential in exact interval arithmetic, full drift battery certified, mutant battery (kernel tail-truncation included) refused — entered this table at the row's uniform re-conduct: both twins' verification conditions are re-conducted onto the unified schema of section 2.9 and stored in this repository as sealed shared-subtree containers, every claim re-derived by the fast-path checker with exact per-relation tallies.

Table 2.1: The published-table sweep, all rows. “Best published” is, per row, the minimum over the paper's sound columns; our bounds are certified exact- $\mathbb{Q}$  upper bounds on reach probability, rounded up at the displayed digit.



repulse100\_2d — the row that ran the other way before the re-conduct, at a 222.0 s campaign production wall — 3.87 s against a published 63.5 s, and 6.41 s checked production on the published pipeline’s own GPU model). That campaign wall was dominated by the leg’s exact ground-truth value computation rather than by certification; the computation now rides the producer’s float-propose / exact-certify tier like every other solve, and the row’s whole checked production is 3.87 s. Beyond that comparison, the table establishes the headline economics of the certificate genre: re-verifying a row costs between milliseconds and about a minute at every trust tier — the far end being the 19,200-obligation gaussian twin’s fast-path replay — which is what makes re-verification a regression test rather than an event.

| benchmark                                | VCs   | best published (s) | checked production (s)           | geolog replay (s)      | qtslab replay (s)                   |
|------------------------------------------|-------|--------------------|----------------------------------|------------------------|-------------------------------------|
| persist_2d (uniform twin)                | 4800  | 254.5              | 13.4 <sup>a</sup> , gb10/host    | 9.08 <sup>gb10</sup>   | — <sup>c</sup>                      |
| ↔ gaussian twin                          | 19200 |                    | 87.7 <sup>a</sup> , gb10/host    | 64.7 <sup>gb10</sup>   | — <sup>c</sup>                      |
| faulty_marbles                           | 23    | 44.5               | 0.042 <sup>gb10-quiet/host</sup> | 0.052 <sup>gb10</sup>  | — <sup>c</sup>                      |
| faulty_unreliable                        | 29    | 52.3               | 0.063 <sup>gb10-quiet/host</sup> | 0.054 <sup>gb10</sup>  | — <sup>c</sup>                      |
| faulty_regions                           | 29    | 49.9               | 0.072 <sup>gb10-quiet/host</sup> | 0.056 <sup>gb10</sup>  | — <sup>c</sup>                      |
| cliff_crossing                           | 1105  | 153.6              | 1.53 <sup>gb10/host</sup>        | 0.57 <sup>gb10</sup>   | 22.9 <sup>gb10</sup>                |
| repulse100                               | 207   | 10.2               | 0.086 <sup>gb10/host</sup>       | 0.066 <sup>gb10</sup>  | 2.67 <sup>gb10</sup>                |
| repulse100_uniform                       | 205   | 35.5               | 4.44 <sup>e</sup>                | 0.45 <sup>gb10</sup>   | 11.0 <sup>gb10</sup>                |
| ↔ discrete-uniform reading               | 207   |                    | 0.14 <sup>gb10/host</sup>        | 0.074 <sup>gb10</sup>  | 2.83 <sup>gb10</sup>                |
| repulse100_2d                            | 2267  | 63.5               | 3.87 <sup>gb10</sup>             | 1.73 <sup>gb10</sup>   | 43.9 <sup>gb10</sup>                |
| faulty_concave                           | 60    | 21.3               | 0.029 <sup>gb10/host</sup>       | 0.059 <sup>gb10</sup>  | 0.71 <sup>d</sup> , <sup>gb10</sup> |
| faulty_varying                           | 60    | 26.8               | 0.029 <sup>gb10/host</sup>       | 0.059 <sup>gb10</sup>  | 0.71 <sup>d</sup> , <sup>gb10</sup> |
| ↔ last-step reading                      | 60    |                    | 0.029 <sup>gb10/host</sup>       | 0.06 <sup>axion</sup>  | 0.71 <sup>d</sup> , <sup>gb10</sup> |
| fixed_loop                               | 22    | 0.15               | 0.0012                           | 0.008 <sup>gb10</sup>  | — <sup>c</sup>                      |
| faulty_loop                              | 22    | 0.16               | 0.0015                           | 0.009 <sup>axion</sup> | — <sup>c</sup>                      |
| ↔ as printed ( <i>i</i> <sub>0</sub> =0) | 22    |                    | 0.0015                           | 0.009 <sup>axion</sup> | — <sup>c</sup>                      |
| faulty_uniform                           | 30    | 8.08               | 0.17                             | 0.019 <sup>gb10</sup>  | — <sup>c</sup>                      |
| faulty_rare                              | 22    | 0.27               | 0.0016                           | 0.009 <sup>axion</sup> | — <sup>c</sup>                      |
| faulty_easy1                             | 84    | 0.31               | 0.0064                           | 0.037 <sup>gb10</sup>  | — <sup>c</sup>                      |
| faulty_ndecr                             | 60    | 0.33               | 0.0044                           | 0.022 <sup>gb10</sup>  | — <sup>c</sup>                      |
| faulty_walk                              | 16    | 0.32               | 0.0011                           | 0.007 <sup>axion</sup> | — <sup>c</sup>                      |
| ↔ deterministic 12-step reading          | 28    |                    | 0.0015                           | 0.009 <sup>gb10</sup>  | — <sup>c</sup>                      |

Marks (column definitions and protocol: the “Timing protocol” discussion in the text). <sup>a</sup> produce cell times the toolkit re-conduct of the row’s committed certificate corpus, not fresh synthesis. <sup>c</sup> row not in the certified-checker replay’s stored corpus. <sup>d</sup> the replay covers the row’s stored wave-B subset (fewer files than the VCs column). <sup>e</sup> produce wall is the row’s campaign leg’s, as receipted: a synthesis genus not yet in the toolkit producer. Machine tags name the winning machine and lane (section 2.2).

Table 2.2: Wall-clock walls, published pipeline vs. toolkit production and replay tiers (seconds).

**Timing protocol.** “VCs” counts the row’s verification conditions as committed: drift and nonnegativity-class proof obligations, one sealed file per condition per piece or step of the state space, each discharged as a Bernstein Logic proof. Under the unified certificate schema of section 2.9 these become proof nodes inside one sealed certificate per *claim* (a row’s corpus may certify more than one claim — e.g. both readings of a noise model the benchmark’s definition leaves unpinned), so the column counts proof obligations, not certificates. For the persist\_2d twins the count is the committed containers’ summed tree totals (4,800 uniform; 19,200 gaussian): that row’s corpus lives in the re-conduct lane’s sealed containers rather than the staged per-file corpus. “Best published” is seconds as published (learning on an Nvidia A40 GPU plus Z3 verification; the Farkas column is their linear baseline’s LP), for the route attaining the row’s best sound bound — the same selection rule as Table 2.1; transcription provenance rides the repository’s manifest. “Checked production” is, for the re-conducted rows, the toolkit producer’s whole elapsed wall — fresh exact synthesis (float-proposing, exactly certifying, with iterative refinement and continued-fraction reconstruction), listing generation, emission through the geolog engine with the in-loop check, deterministic sealing, and orchestration overhead. For each benchmark, tables/gen\_tables.ts selects the minimum median across all committed machine/proposer-lane receipt sets; the cell tag names the winning machine and lane. The A40 receipts remain separately committed for the GPU-model-parity comparison. Every measured repeat must emit bytes identical to the committed corpus. Versioned Section 1 receipts use their authoritative measured wall\_s; legacy Section 2 receipts use the sum of their disjoint produce\_s,

check\_s, and seal\_s subphases. The gb10-quiet tag is the same GB10 hardware under a quiet-conditions protocol, preregistered before its first campaign ran (the protocol file is committed beside the receipts): the runner itself refuses to start unless the box's 1-minute load average is at or below a declared gate, the campaign is pinned to performance cores at a fixed absolute niceness, and its version-3 receipt records the gate, per-row load averages, and the affinity and niceness read back from the process, so the admission claim is checkable from the receipt alone — the receipt validator refuses a receipt whose recorded starting load exceeds its own declared gate. The load-uncontrolled gb10 receipts remain committed beside the quiet family; where both exist, the minimum-median rule above selects the quieter conduct, and the two are protocol variants of one machine, not different hardware. Rows not yet re-conducted keep their campaign legs' synthesis wall as receipted (single CPU box), and their cells flip the same way as their re-conducts land (section 2.9).

The persist\_2d produce cells (mark <sup>a</sup>) are the one deliberate exception to the fresh-synthesis reading: the twins' certificates are a synthesis genus the toolkit producer does not yet cover (the same gap note <sup>e</sup> records for repulse100\_uniform), and no synthesis wall is receipted in this repository, so those cells time the toolkit's *re-conduct* of the row's committed corpus — shared-subtree forest merge, the in-loop check, and deterministic sealing, byte-identical to the committed containers on every repeat — under the same repeat protocol as the other produce cells (one flagged warmup plus ten measured repeats, serial, medians, at recorded scheduler niceness read back from the process). They are production-tier walls for the stored row, not job-for-job counterparts of the published column, and the chapter draws no such comparison for that row.

“Geolog” is the geolog fast-path checker's total wall over the row's sealed containers, serial, one process at a time (machine, load, and checker-binary hash in each timing receipt). For the re-conducted rows the wall is over the row's shared-subtree containers — proof hypernets: structurally shared subtrees stored once, element in-degree above one, with per-container tree and stored-node counts in the committed manifest — one container per row, except that persist\_2d's gaussian twin ships as four sorted-quarter containers (its merged forest exceeds the envelope's per-commit body cap) and its cell is the serial total of the four per-container medians, its uniform twin one whole-twin container; the pre-migration cells time per-obligation container sets; “VCs” counts proof obligations either way. The two vintages also amortize process overhead differently: a shared-container cell is the median of single-container invocations, each paying the checker's whole per-process vocabulary derivation, where a pre-migration cell spreads one process start across its whole batch. On the shared containers the marginal re-check inside a warm checker process is 5–9 ms on the cells' machine — the same magnitude as the production loop's in-process check — with most of the balance being the per-process vocabulary derivation (the checker's own timing decomposition measures it directly at 44–54 ms there, the remainder being generic process start); one process checking all three shared containers as a batch clears the whole roster in  $\approx 81$  ms, against  $\approx 224$  ms summed per-invocation. The checker-protocol comparability receipts beside the retiming receipts measure both effects with the same checker binary on both container vintages, and an independent-machine replication on a second architecture reproduces the marginal-vs-derivation relationship (and byte-identical containers) while every absolute wall scales with the box, so these milliseconds carry their machine tag rather than traveling as platform constants. Geolog cells are therefore read each under its own protocol, never ratioed across vintages. “Qt slab” is the sum of the certified checker build's per-VC walls from the replay census of section 2.3.1 (Arm cores; four families ran concurrently, so these walls carry mild concurrency inflation at most). The certified checker checks the Bernstein Logic fragment of QTSL only — exactly the proof tower committed as theories/BernsteinLogic.geolog in the pinned qtsl repository, so the fragment boundary is a file boundary, not a convention — and that is the fragment this campaign's VC files natively encode, so the walls shown are complete for the rows they cover. The full QTSL theory (theories/Qtsl.geolog, extending that tower) is checked in geolog, on the sealed-container lanes of section 2.9. The reproduction spine's design also names a certificate-extraction pathway, from a full QTSL certificate down to its Bernstein Logic obligations — the same claim-reducing role the route's other reducers play — and that tool's wall-clock joins this column when it lands. The replay census also predates the shared-subtree vintage: its walls time the stored per-obligation wave-B corpus, and the shared vintage's certified re-check joins this column when the checker build's pin vintage advances. Rows whose containers entered this repository after that census's corpus was pinned — the persist\_2d twins among them — carry mark <sup>c</sup> (not in the replay's stored corpus; they enter at its next replay), and mark <sup>d</sup> rows' replays cover their stored wave-B subsets, fewer files than the VCs column. Mark <sup>e</sup>, in full: repulse100\_uniform's certificate is a different synthesis genus (a continuous-noise, piecewise-polynomial profile) not yet ported into the toolkit producer, so its

produce wall is its campaign leg's, as receipted.

Two findings from this campaign shaped everything after it. First, the exact computation *refuted* one published number: the paper's manually-constructed Farkas bound of 0.0181 for `faulty_uniform` lies strictly below the true reach probability, which the campaign localized to the two-sided interval [0.0184458, 0.0185556] — a published “sound” bound that is not sound, detectable only because the replacement pipeline is exact. Second, re-verification is cheap enough to be a regression test. A row's proof is not one file: it decomposes into per-obligation verification conditions (VCs) — drift and nonnegativity-class obligations, one file per condition per piece or step of the state space — so the seventeen rows fan out to 28,569 VC files, alternate reconstructions and duplicate certifications included. 24,000 of these sit behind the one continuous-state row, certified piecewise on rational lattices under both readings of a noise model the paper leaves unpinned. The full corpus has been replayed through three successive checker builds to date — every file through every build, zero refusals, no bound moved — so the campaign's results are re-established, not merely remembered, at each toolkit revision.

**What is not claimed.** Table 2.2 prints no speedup ratio: the pipelines are compared job-for-job, and the produce, geolog, and qtslab cells each carry the best measured configuration across the repository's timing machines (the machine named in the cell's tag), while the GPU-model-parity receipts — the same GPU model as the published pipeline's, not the same host, driver stack, or whole system — are committed beside them; so any ratio a reader forms still carries a hardware caveat, and the chapter's own statements stay qualitative, per section 2.2. The comparison covers all seventeen published rows, with two cell-level caveats. The `persist_2d` produce cells time the row's re-conduct, not a fresh synthesis (mark <sup>a</sup>: that synthesis genus is not yet in the toolkit producer and no synthesis wall is receipted in this repository, so no job-for-job produce comparison is made for that row), and its certified-checker cells await the replay census's next corpus pin (mark <sup>c</sup>); one re-conducted row's produce wall is still its campaign leg's (note <sup>e</sup>). Nor was every toolkit wall always small: the campaign's one heavy re-check was `persist_2d`'s pre-re-conduct fast-path record of 130.66 s over the row's original 4,800-file uniform bundle — large-denominator rational arithmetic; the stored re-conduct rides the dyadic re-emission that retired those denominators, bound bit-identical, and the row's shared-container replay walls now sit in the table like every other row's.

### 2.3.1 The stored corpus through the certified checker build

The replays above all ride the toolkit's engineering-grade lanes — the reference checker and its fast-path twin. The Executive Summary names a third consumer of the same bytes with a far smaller trust footprint: `qtslab`, the certified checker, whose standing as a whole is stated there once — a demonstrated route, assembling now — and is not upgraded by anything in this subsection. What this subsection adds is a measured fact about that route's current build.

On July 23, 2026, the campaign's stored corpus — the 4,105 VC files behind eight rows of Table 2.1 (six published benchmarks and two alternate readings), each pinned by hash in the repository's manifest — was replayed through the current certified-checker build: a standalone Arm binary emitted inside HOL4 through the CakeML verified compiler, carrying a machine-code theorem, re-proven at this same emission, that a run of the emitted code printing `ACCEPT` with a sequent implies the checker function *defined in the logic* accepts those bytes with that sequent. (That theorem is the route's lower half — the machine code can no longer misrepresent the checker it implements; the upper half, from logical acceptance to semantic truth, is the part still assembling. The truth in question is validity of the accepted sequent under every admissible valuation of the hypernet the certificate itself declares — the Bernstein Logic fragment, which is all this checker's input format carries. The boundary is the checker's, not the vocabulary's: a claim that a System satisfies a Spec is representable today — the full QTSL theory of section 2.9, whose rows the engine-side container lanes check — but it lies outside the certified checker's fragment, and a certificate reaching beyond that fragment is refused, never silently accepted.) The build accepted 4,105 of 4,105 — zero divergence from the fast-path twin, which accepts the same 4,105 — at a pooled median of 30 ms per VC file on one Arm core (min 14 ms,  $p_{90}$  37 ms, max 117 ms). Table 2.3 gives the per-family split; the verdict-by-verdict census, the twin verdicts, and the artefact hashes are committed beside the corpus (`supermartingales/out/sec2/certified-checker/`).

**What is not claimed.** This table reports verification results about committed artefacts; it is not the claim that the finished certified checker exists — the front matter's one-sentence standing governs, and

| family                      | accepted, certified build | diverging vs. fast path | wall per VC file (ms), min/med/ $p_{90}$ /max |
|-----------------------------|---------------------------|-------------------------|-----------------------------------------------|
| cliff_crossing              | 1105/1105                 | 0                       | 17 / 20 / 25 / 55 ms <sup>gb10</sup>          |
| faulty_concave              | 38/38                     | 0                       | 17 / 19 / 19 / 20 ms <sup>gb10</sup>          |
| faulty_varying              | 38/38                     | 0                       | 18 / 19 / 19 / 20 ms <sup>gb10</sup>          |
| faulty_varying__last_step   | 38/38                     | 0                       | 17 / 19 / 20 / 20 ms <sup>gb10</sup>          |
| repulse100                  | 207/207                   | 0                       | 11 / 13 / 14 / 16 ms <sup>gb10</sup>          |
| repulse100_2d               | 2267/2267                 | 0                       | 17 / 19 / 21 / 51 ms <sup>gb10</sup>          |
| repulse100_uniform          | 205/205                   | 0                       | 13 / 52 / 85 / 96 ms <sup>gb10</sup>          |
| repulse100_uniform_discrete | 207/207                   | 0                       | 12 / 14 / 15 / 17 ms <sup>gb10</sup>          |
| all families                | 4105/4105                 | 0                       | 11 / 19 / 23 / 96 ms <sup>gb10</sup>          |

Table 2.3: The stored corpus of VC files replayed through the certified checker build (July 23, 2026). “Accepted” is the certified binary’s accepts over the family’s stored VC files; “diverging” counts VC files on which the certified build’s verdict differs from the untrusted fast-path twin’s — zero everywhere, which is the finding. Walls are single-shot, one single-threaded checker process per VC file, four families run concurrently; a serial spot sample ran 10–50 ms, so the concurrency inflates the walls mildly at most.

the numbers here neither strengthen nor presuppose it. The corpus is the eight table rows whose VC files have migrated into this repository, not the full row set. The walls are measured under the mild concurrency stated in the caption, on the machine named in the receipts, and will drift with hardware like any wall-clock. Finally, the certified build itself and the tool that seals row listings into its signed-commit input form have not yet migrated into the public trees: today the table re-derives mechanically from the committed receipts, and re-running the binary replay itself becomes a repository make target when those artefacts land (per the migration rule stated in section 2.9).

## 2.4 Gymnasium: first certified rows for a standard RL suite

The second campaign marched the same machinery across Farama’s Gymnasium — the de-facto standard reinforcement-learning benchmark suite — pinned at repository revision 4347ce947, with every certificate naming its environment id and keyword arguments. To our knowledge, no prior certified bound at any of this chapter’s claim classes has been published for any Gymnasium environment; the comparison points that exist are the suite’s own registered “solved” thresholds and code-comment optima, plus empirical leaderboard numbers, and the campaign’s rows are therefore firsts at their class.

One property of these certificates deserves stating before the table, because every row reads differently in its light: they are *constructive*. Each certificate ships the policy itself — a lookup table, indexed by state and step, sitting in the certificate bytes beside the value table and the claim — so no row asserts merely that a policy achieving the bound *exists*; it asserts that *this* one does, and whoever holds the certificate holds the policy. The checker enforces the distinction structurally: its per-step obligations bound each state’s value by the kernel-weighted value of the shipped policy’s own action — never by a maximum over actions — so the policy is load-bearing in every obligation it discharges, and a tampered policy entry is among the refused mutants of the battery. Table 2.4 is the master table for the tabular tier.

Four findings come out of this table.

**The suite’s own Taxi threshold is unsatisfiable.** The registry declares Taxi “solved” at average return 8, with a code comment placing the optimum at 8.46. The true optimum of the pinned environment is exactly  $793/100 = 7.93$  — *below* the threshold; no policy can meet it. The 8.46 is a fossil: on the pre-correction map (before an upstream fix added one wall segment) the same exact computation yields  $2539/300 = 8.4633\dots$ , which we reproduced by swapping only the map. The threshold was never updated for the wall.

**The first exact value for the textbook blackjack.** Sutton and Barto’s blackjack variant (infinite deck, dealer stands on all 17s, hit/stand only) may be the most-reimplemented exercise in reinforcement learning; a literature search — the textbook, Baldwin et al. 1956, the gambling-mathematics

| environment                   | claim | certified bound      | vs. optimum                | check                           |
|-------------------------------|-------|----------------------|----------------------------|---------------------------------|
| FrozenLake 4x4 (slippery)     | U3    | $\geq 0.7441902$     | optimal, exactly           | $15 \mu\text{s}^{\text{gb}10}$  |
| FrozenLake 8x8 (slippery)     | U3    | $\geq 0.9132201$     | optimal, exactly           | $100 \mu\text{s}^{\text{gb}10}$ |
| FrozenLake 4x4 (succ. 3/4)    | U3    | $\geq 0.8731132$     | optimal, exactly           | $15 \mu\text{s}^{\text{gb}10}$  |
| FrozenLake 4x4 (not slippery) | U3    | $= 1$                | optimal, exactly           | $13 \mu\text{s}^{\text{gb}10}$  |
| Cliff Walking                 | U2    | $\geq -13$           | optimal, exactly           | $24 \mu\text{s}^{\text{gb}10}$  |
| Cliff Walking (slippery)      | U3    | $\geq -63.0133733$   | optimal, exactly           | $34 \mu\text{s}^{\text{gb}10}$  |
| Taxi                          | U2    | $\geq 7.93$          | optimal, exactly           | $0.50 \text{ ms}^{\text{gb}10}$ |
| Taxi (rainy)                  | U3    | $\geq 3.9545745165$  | within $3 \times 10^{-14}$ | $0.70 \text{ ms}^{\text{gb}10}$ |
| Taxi (fickle passenger)       | U3    | $\geq 8.38$          | optimal, exactly           | $1.1 \text{ ms}^{\text{gb}10}$  |
| Taxi (rainy + fickle)         | U3    | $\geq 4.5756463469$  | within $3 \times 10^{-14}$ | $1.5 \text{ ms}^{\text{gb}10}$  |
| Blackjack (S&B rules)         | U3    | $\geq -0.0431131055$ | optimal, exactly           | $0.16 \text{ ms}^{\text{gb}10}$ |

Table 2.4: Certified rows for the Gymnasium tabular tier. The claim column names the species of section 2.1: U2 is a certified cost-to-goal, U3 an exact probabilistic bound. Policies are shipped lookup tables; checks are single-core medians. “Optimal, exactly” means the bound is the exact optimum of the (deadline-truncated) environment: terminal at its claim class — those cells carry the exact-cell tint of section 2.2.

literature, recent preprints — found no published exact expected value under optimal play. It is  $-3954908734551683826282634489/13^{26} \approx -4.311\%$ , now with a machine-checkable certificate. (Consistency: the same game with 3:2 naturals is known to run near  $-2.42\%$ , and the difference is the order of the forgone bonus. A subtlety: the suite’s observation space cannot distinguish a two-card 21 from a three-card 21, but the payout rule can — the certificate runs on a slightly enriched state space while proving the shipped policy never uses the extra information.)

**Hidden state, certified honestly.** Taxi’s “fickle passenger” variant re-draws the destination mid-trip with probability  $3/10$ , invisibly until it fires. A certificate on the full state would be clairvoyant; these rows certify the *belief* chain instead, so the shipped policy is implementable from the observation history. The variant also *raises* the optimum (dry  $7.93 \rightarrow 8.38$ ; rainy  $3.9546 \rightarrow 4.5756$ ): a uniform mid-trip re-draw is on average favourable, and the certified policies exploit the re-route.

**The registry’s comment optima were truncations of numbers nobody had.** The FrozenLake comments say “optimal average reward  $0.74 / 0.91$ .” The exact optima — rationals with denominators  $3^{99}$  and  $3^{200}$  — were computed here, we believe for the first time; the comments are truncations of them.

#### 2.4.1 How the rows were produced: exact backward induction and its wall-clock cost

The producer strategy for the whole tabular tier is the classical algorithm, executed exactly: backward induction — finite-horizon dynamic programming — over the deadline-truncated chain. Starting at the deadline and stepping backward, the producer computes, for every state at every step, the optimal value by a single Bellman backup — a maximum over actions of the expected one-step reward plus the kernel-weighted value one step ahead — in exact rational arithmetic, recording a maximizing action as it goes. The recorded actions *are* the shipped policy; the value tables become the certificate’s obligation rows; the value at the initial state (or the exact mixture over the initial distribution) becomes the claim. Termination and exactness are structural: the horizon is finite — the deadline is the environment’s own truncation, or the certificate’s declared one — the state and action sets are finite, and no backup ever rounds, so after  $T$  sweeps the result is the exact optimum of the truncated environment, not an approximation to it. There is no learning loop, no convergence tolerance, and no rounding anywhere in the construction. (Making exactness fast is an implementation matter, not an algorithmic one: values at each step are stored as integers over a shared power of the kernel denominators’ least common multiple, so the hot loop is integer multiply-accumulate, division-free, with canonical reduction deferred to emission.) Nor is tightness taken from the producer on trust: for the exact lanes the checker has an optimality mode whose obligations quantify over *all* actions — a supersolution check beside the shipped policy’s attainment — so “optimal,



exactly” in Table 2.4 is itself a machine-checkable statement, and the mode’s test fixtures include a matched pair exhibiting the gap: a certificate that passes as a bound but refuses as an optimum.

The direct answer to “how long does producing the optimal policy and its certificate take” is: for the entire tabular catalogue — all 53 certificates, the master table’s rows and the frontier studies’ variants below included — the optimized producer runs from process start to every certificate synthesized, emitted, and byte-compared in 0.69 s of wall-clock time on four CPU cores. The per-lane receipts break that down: the heaviest single computation, the exact optimum of the rainy fickle-passenger Taxi belief chain, takes 117 ms on one core (35 ms on four); the 8x8 FrozenLake exact table, 6.6 ms; the textbook-blackjack optimum, 2.2 ms. The reference implementation that first produced the corpus takes roughly 3.2 s single-core for the same spans; the optimized producer takes 0.5 s on one core (6.5×), and the two emit byte-identical certificates on all 53 — the strongest available cross-implementation check. A GPU lane reproduces the dyadic tables bit-identically with a  $\sim 2\times$  gain on the largest rows only; on small rows the kernel-launch floor loses to the CPU, a null result recorded as such. Throughout, producer cost and check cost are measured separately: the check columns of every table in this section time only the checker.

These certificates also ride the sealed-container lanes of section 2.9, and their re-emission on the unified schema returned a finding rather than a formality. The schema’s chain species is a pure discrete-time Markov chain — transition and initial rows, no action layer — so a hitting-lane certificate carries over verbatim, table and claimed bound alike, exactly when the decision problem admits a *stationary* composition: one action per state admissible at every step of the deadline. The hitting corpus splits 9/29 on that criterion, and the split is semantic content, not an artifact: every deterministic-dynamics certificate composes, while slippery FrozenLake at long deadlines has states where no single action is admissible at every step — near the deadline the optimal policy gambles where far from it it detours. The nine composable certificates already ride the unified schema with their tables and claims byte-verbatim; the twenty time-varying ones stand in the pre-unification record, each exclusion a manifest fact naming its obstructing states, and their re-emission awaits the schema’s decision species (make gymnasium-glc re-checks both lanes and the exclusion records alike).

## 2.4.2 The measured frontier: bound versus check cost

Because the tabular certificates are terminal in bound, the interesting curve is in the *representation*. Two honest knobs trace it: the deadline  $T'$  (certifying “goal within  $T'$  steps” for  $T' \leq T$  weakens the bound but shrinks the table) and the precision  $k$  of a dyadic re-emission (round-down integer tables with bound loss  $\leq T \cdot 2^{-k}$ , turning every obligation into an integer comparison). On the FrozenLake rows, moving from exact rational arithmetic to dyadic integer tables at equal obligation count speeds the check by roughly  $400\times$ – $1400\times$  at a bound cost of order  $10^{-13}$  — far below any displayed digit. The general lesson, confirmed three independent times in this campaign’s receipts: *certificates should be engineered for their checkers* — rationals belong in the claim line, shared-denominator integer slabs in the obligations, and the checker’s hot loop then vectorizes (the kernel-level lanes add a further  $1.6\times$ – $3.3\times$  on the larger reward tables, at 5–7 ns per obligation).

Honest non-dominations in the same receipts: on the smallest certificates the vectorized lane’s setup cost makes it  $0.83\times$ – $0.89\times$  the plain lane (crossover near the 8x8 scale); and one certificate family (blackjack at high dyadic precision) structurally exceeds the vectorized lane’s overflow envelope, is declined, and falls back to the reference lane — deferral costs completeness, never soundness, and the fix is a smaller  $k$ .

## 2.4.3 Beyond tables: continuous state and solver-semantics environments

Above the tabular tier the campaign is deliberately breadth-first, and at this preview two further tiers carry rows.

**Continuous state: certified worst-case return.** The first continuous-state row certifies Pendulum-v1: a Bellman-supersolution certificate over a polynomial cell cover of the state space (the environment’s velocity clipping entering as explicit piecewise structure), yielding a certified *worst-case* return bound: one holding for every initial state, with no probability anywhere in the certificate — the strongest of the claim species of section 2.1 (U1) — and carrying an anytime cover-refinement curve measured from a few dozen cells to nearly a thousand. At this preview the certified bound stands at return  $\geq -1.192$  over the full initial set, with the cover-refinement gap driven to  $2.4 \times 10^{-4}$ , and a five-mutant battery refused.

The corresponding CartPole certificate (worst-case return exactly 500, the same species) has crossed the composition step it was staged behind — and the step dissolved rather than being climbed: instead of transporting a continuous-time decrease certificate through the environment’s explicit-Euler step map, the certified row states a per-cell contraction on the step map itself, which denominator-clears to a polynomial of its own, leaving only kernel-checked enclosures of the current state’s sine and cosine. The first certified CartPole row now stands at the subsystem where it is honest: under an exhibited cell-table bang-bang policy, the pole angle provably never reaches the termination threshold, from every initial state in the reset box, in exact rational arithmetic, with a designed-locus mutant battery refused. The return bound itself still waits on the cart-position excursion, which the pole-only policy does not regulate; that full-state variant is the campaign’s first genuinely large checking workload ( $10^6$ – $10^7$  polynomial cells), sized to exercise the batched checking engine of the Geolog tier as it lands.

**Solver-semantics environments: i.i.d. confidence bounds.** Where the environment’s dynamics is a compiled physics engine (the Box2D and contact-rich MuJoCo families), no analytic species reaches the binary’s semantics, and pretending otherwise would be a category error. What applies instead is the *i.i.d. tier*: an anytime-valid lower confidence bound on the expected return, certified from recorded rollouts of a pinned policy — the weakest species of section 2.1 (U5), holding with probability at least  $1 - \alpha$  rather than almost surely, but model-free, which is exactly what lets it reach a compiled binary at all. Its rows are always displayed as their own class, never mixed into a table of exact rows: Table 2.5 shows the first two — to our knowledge the first certified statements of any kind at this claim class for Box2D environments. Each is a betting-martingale ledger over  $10^6$  rollouts of the suite’s own heuristic controller; the checker re-verifies the entire wealth process in directed-rounding fixed point ( $1.6 \times 10^7$  obligations in  $\sim 130$ – $200$  ms, single-core), and the certificate is *anytime-valid*: extending the ledger with more episodes can only tighten the bound, with no statistical penalty.

| environment              | claim (class U5)                     | bound | threshold |
|--------------------------|--------------------------------------|-------|-----------|
| LunarLander-v3           | $\mathbb{E}[R] \geq, \alpha = 1/100$ | 236.2 | 200       |
| LunarLanderContinuous-v3 | $\mathbb{E}[R] \geq, \alpha = 1/100$ | 281.1 | 200       |

Table 2.5: First i.i.d.-tier rows (species U5 of section 2.1; confidence-qualified: each bound holds with probability  $\geq 1 - \alpha$  over the sampling; never to be read as almost-sure). Policies are the suite’s own heuristic controllers;  $n = 10^6$  episodes each; the almost-sure return range required by the bound is itself a certified lemma about the environment’s code, not an assumption. Bound-cell tints are the win-tint ladder of section 2.2, read against the registry threshold beside them.

One reading of Table 2.5 previews the genre’s point: the *certified floor* of the continuous-throttle variant (281) exceeds the discrete variant’s *empirical mean* (238) — a policy-comparison statement made entirely below the certified line, where it cannot be an artefact of sampling luck.

**What is not claimed.** The tabular certificates’ trust seam — that the modelled kernel is the environment’s — is narrowed here by asserting the modelled tables equal to the suite’s own published transition tables (exact equality, with the handful of float-literal discrepancies carried as declared ledger rows), but it remains producer-side. The i.i.d.-tier rows trust the pinned pseudorandom generator and the policy’s determinism, both named in the certificate. No worst-case, cost-to-goal, or exact-probabilistic claim (species U1–U3) is made for any solver-semantics environment. No MuJoCo row appears in this preview’s tables; the named route — its design landed as this preview goes to press, rows in flight — is a hybrid-time composition pattern: discrete controller ticks with bounded clock jitter coupled to continuous-time physics, the jitter entering as a certificate *parameter* rather than a disturbance, with the suite’s contact-free environments (InvertedPendulum and its ladder) as first targets and the contact-rich families expected to arrive i.i.d.-tier first. Producer-side synthesis time for the continuous tier is reported in the repository but not compared against reinforcement-learning training costs: the claims differ in kind, and a wall-clock race between them would compare nothing meaningful.



## 2.5 Hardware model checking: certificates where verdicts were

The third campaign targets a benchmark from a different community: the hardware model-checking suite introduced by the neural model-checking line of work (Giacobbe, Kroening, Pal, and Tautschnig, *Neural Model Checking*, NeurIPS 2024, arXiv 2410.23790) and extended by the same group at NeurIPS 2025 (*Let a Neural Network Be Your Invariant*) — eleven Verilog designs (UART and I<sup>2</sup>C controllers, VGA and LCD drivers, PWM, counter, and thermocouple-interface blocks), each with a ladder of parameter instances, under three property classes: 219 pure-liveness tasks ( $\text{FG } \alpha \rightarrow \text{GF } \beta$ ), 206 safety tasks, and 209 combined safety-plus-liveness tasks (premise-guarded until-properties such as  $\text{G}(r \rightarrow p \cup q)$ ) — 634 verification tasks in all. The published pipelines train a neural certificate per task — a ranking function by gradient descent in 2024; ranking and invariant functions by MILP in 2025 — quantize it, and hand a single validity query to an SMT solver; the reported verdict per task is the solver’s “unsat.” The baselines are mature model checkers (nuXmv, ABC, rIC3), and the experimental budget is five hours per task.

The toolkit’s route is structurally different, and the difference is the point. Every one of the 634 specifications reduces, by a chain of temporal rewrite steps, to conjunctions of two certified forms: pure recurrence  $\text{GF } \gamma$ , discharged by a ranking function — piecewise-affine with exact rational coefficients over a cell partition of the circuit’s own branch structure, input wires resolved demonically (universally) at every step — and  $k$ -step safety windows, discharged by obligations on the exact one-step image (plus small inductive invariants on the four families that need them). No Büchi automaton and no product construction appears anywhere on the suite. The trust story is layered the same way at every level: the consuming lemmas (a ranking that drops by  $\varepsilon_0$  off the target and stays nonnegative forces target steps to recur; a window whose obligations hold on the reachable image holds always) are a few lines of mathematics each, machine-checked in Lean with no measure theory; every rewrite rule’s soundness is machine-checked *twice*, in Lean and independently in HOL4; each task carries a rewrite-chain witness that a standalone checker replays from the verbatim specification text; and every ranking and window condition is an ordinary polynomial-nonnegativity row of the chapter 15 calculus, recomputed exactly in  $\mathbb{Q}$  by standalone checkers that import nothing from the producers.

At this preview the sweep covers the entire suite: **all 634 tasks, across all three classes** — 219/219 pure liveness (245 certificates: the two conjunction families carry two per task), 206/206 safety (222 certificates), and 209/209 safety-plus-liveness (418: one window and one recurrence certificate per task) — 885 certificates and 634 chain witnesses in all. Table 2.6 sets this against the published field on the field’s own numbers: the strongest single published tool, the 2025 MILP method, completes 554 of 634; the per-task union of *every* tool in the 2025 paper’s comparison completes 617. Seventeen tasks are completed by nothing published at all — twelve pure-liveness tasks (eleven blink instances and the largest `load_store` instance) and the five largest blink safety-plus-liveness tasks. The toolkit certifies all seventeen; blink is a counter ladder, and its certificates have the same shape at  $N = 2^{32}$  as at  $N = 2^8$ .

Three anchoring notes travel with Table 2.6. First, *the comparison is like-for-like with the 2025 paper* and only approximately comparable to the 2024 publication: the benchmark’s circuits were revised between the two editions (142 source files changed semantically, an edit the 2025 paper does not mention), and the toolkit certifies the revised circuits — the same files the 2025 re-runs measured. The 2024 method’s own line shows the drift: 93% of its original 194 tasks as published, on the 2024-era circuits; 172/219 under its authors’ re-run on the revised, extended class. Second, *the semantics certified is the benchmark’s exactly*: the suite’s generated SMV models carry no initial-state constraint, so every task is a from-every-state claim under demonically-resolved inputs — precisely the universal form the certificates prove. For the until-shaped class this was additionally confirmed off-calculus: an independent explicit-state model checker (a demonic greatest-fixpoint computation written against the specification semantics, not the certificate calculus) verifies each safety-plus-liveness property as stated on the smallest instance of every family, and falsifies none. Third, *the completion race is not the durable claim* — model checkers are healthy, and successor methods will contest counts. The durable column is the last one: on this suite, every completed task ships an auditable certificate, a small exact object whose consuming lemma is machine-checked, where every baseline — neural or classical — ships a verdict.

The checking economics stay flat as the suite triples. Certificates run 0.4–7.4 KB (98% under 4 KB; the tail is the VGA safety windows); individual checks run from tens of microseconds to a few milliseconds; replaying every certificate *and* every rewrite-chain witness in all three classes takes under a second of single-core wall-clock in total (per the class receipts: certificate checks of 78, 195, and 171 ms for liveness,

| method                        | liveness 219 | safety 206 | s+l 209 | all 634 | per-task evidence     |
|-------------------------------|--------------|------------|---------|---------|-----------------------|
| this toolkit                  | 219          | 206        | 209     | 634     | checkable certificate |
| neural, NeurIPS'25 (MILP)     | 192          | 190        | 172     | 554     | solver verdict        |
| nuXmv (BDD)                   | 180          | 206        | 154     | 540     | tool verdict          |
| rIC3 1.5.1                    | 110          | 196        | 129     | 435     | tool verdict          |
| ABC                           | 78           | 206        | 96      | 380     | tool verdict          |
| neural, NeurIPS'24 (re-run)   | 172          | n/a        | n/a     | 172     | solver verdict        |
| virtual best of all baselines | 207          | 206        | 204     | 617     | portfolio of verdicts |

Table 2.6: The full NeurIPS'25 neural-model-checking suite: completions by property class (pure liveness 219, safety 206, safety-plus-liveness 209). Baseline counts are recomputed from the 2025 paper's own artefact — its plotting script's hard-coded runtime arrays are the paper's published data — at the paper's own five-hour timeout. The 2024 method is a liveness-only technique; the artefact records no runs for it outside that class (n/a). The virtual-best row is the per-task union of every baseline in the artefact's comparison, including the linear variant of the 2025 method, which is not shown as a separate row. The toolkit's per-task evidence is an exact-arithmetic certificate replayed by a standalone checker, with consuming lemmas machine-checked in Lean and every spec-rewrite rule machine-checked in both Lean and HOL4. A toolkit count carries the exact-cell tint of section 2.2 where it strictly exceeds the virtual best of the whole published portfolio (ties untinted).

safety, and safety-plus-liveness, plus 82, 118, and 150 ms of chain replay). Certificate size and check time are constant along each design's parameter ladder, with only coefficient widths growing as  $\log N$ , where the learning pipelines pay training and solving per instance. Per the conventions of section 2.2, the two sides' times are a contrast of kind, not a ratio — searching for a proof and re-verifying a shipped one are different jobs on different machines: the paper reports a five-hour-per-task budget and roughly nine thousand machine-hours for its full experimental campaign; the toolkit's whole-suite certificate replay is the under-a-second figure above.

Refusal is exercised as systematically as acceptance. Each class ships a mutant battery — deliberately corrupted certificates and chain witnesses, each designed to be refused at one specific check: seventeen layer-targeted attacks with four positive controls for the liveness class (ported into this repository's re-check driver), thirteen attacks with three controls for safety, and fourteen attacks with nine controls for the combined class, two of those attacks cross-layer (accepted by the certificate checker, refused by the chain checker) — and each is in fact refused at the attacked layer, with every control passing. The remaining producer-side trust seam is the circuit manifests: that the certified step system *is* the Verilog's semantics. It is narrowed by two-simulator cross-checks, exhaustive on the full state-input box of each family's smallest instance and sampled on the largest, and — since the safety campaign — by an independent re-transcription of every design. That newest instrument earned its keep immediately: the two original simulators share an author, the first edition of this section said so as its declared residue, and the re-transcription caught exactly the failure that caveat named — a shared misreading of one LCD counter's wrap point, latent in the already-swept liveness manifests. The manifests were repaired and the full battery re-run green within the hour; the fixtures in this repository are the repaired ones. Replacing this seam with a netlist-level bridge remains named successor work.

Exact re-derivation of a published field keeps yielding findings of the same genre as the Gymnasium campaign's unsatisfiable Taxi threshold (section 2.4). Three from this campaign are worth recording. *The sound premise is not the obvious one*: the natural liveness premise for the until-introduction rewrite rule —  $\text{GF } q$ , “the window eventually closes” — is false whole-box on the I<sup>2</sup>C family, whose clock-stretch freeze states never reach the window and satisfy  $\neg p$  forever instead; the sound premise is  $\text{GF}(\neg p \vee q)$ . The correction was forced during rule design and then observed semantically: the explicit-state falsifier's demon-winning regions are nonempty exactly on those frozen states. *One task's two specification sources disagree*: the delay design's first combined property reads  $\text{G } \neg rst \rightarrow \dots$  in the runner's specification text but  $\text{FG } \neg rst \rightarrow \dots$  in the same runner's nuXmv property line. The toolkit certifies the specification text as stated — the chain witness pins its verbatim bytes — and records the discrepancy; certifying as-stated rather than as-presumably-meant is what makes such discrepancies visible at all. *The artefact's own aggregation has a slip*: the 2025 plotting script assembles its safety-class aggregate with one VGA family's arrays included twice and the other's omitted. The counts in Table 2.6 are unaffected — they are recomputed from

the per-family arrays, and both VGA safety families are fully solved by the classical baselines — but the paper’s aggregate safety timing figures ride the wrong multiset. (The 2024 edition had a kindred fossil: an appendix table summing to 177 completions where the summary claims 180; the census here restored the three footnoted rows a naive parse drops, scoring that comparison in the paper’s favor.)

Because the producers here are benchmark-specific and written by us, the fair cost question is not only “how fast is the check?” but “what did it cost to *get* the certificates, method invention included?” — and the campaign’s working sessions are timestamped, so it has a measured answer. The pure-liveness campaign end-to-end — reading the papers, designing the route, machine-checking the consuming lemma, writing the certificate format, the checker, and every producer, and sweeping all its tasks — took about 75 minutes of commit-stamped wall-clock on one desk-side machine (this repository’s invention-time comparison table carries the per-phase receipts and its estimating conventions). The safety class and the combined class were each added in a single further working session the following day — the combined class including six new until-introduction rewrite rules, each proved twice, in Lean and in HOL4. These figures sit beside the published five-hours-per-task budget as, again, a contrast of kind: method invention on one side of the ledger is being compared to certificate search on the other, and we print no ratio.

**What is not claimed.** Synthesis wall-clock supremacy is not claimed: on tasks the 2025 MILP method completes, it is itself fast (sub-second on most families); the measured advantages here are completion, the check side, and the evidence class. Generality over automaton-requiring specifications is not claimed: no property in this suite needs genuine automaton memory — the 2025 additions, though until-shaped, reduce by premise-checked implication rules — and on a suite where that machinery is needed, it would return. Sim-tightness is claimed only where measured: every liveness family’s worst-case descent bound is achieved exactly by a simulated run of the circuit. And the reproduction spine in this repository is complete at this preview for the pure-liveness class only — committed fixtures, the pinned independent checker, and the mutant battery, re-run end to end by one make target; the safety and combined classes’ receipt tables are vendored here, and their fixtures and checkers arrive with the producer layer’s migration. Per the preview note of section 2.9, until a cited artefact lands in the public repository, its mention is a statement of what exists, not yet a link.

## 2.6 Atari as a circuit: making the environment white-box

The fourth campaign closes a loop between the previous two. It takes an environment family from the reinforcement-learning suite’s hardest tier — Atari 2600 games, which Gymnasium reaches through an emulator plugin — and turns each game into an object of the hardware campaign’s kind. Rather than treating the emulator as a black box to be sampled, the campaign compiles the console itself into a synchronous digital circuit: an open-source Verilog implementation of the 2600 (vendor-pinned, with three local repairs, each carried in a ledger) is synthesized with the game ROM baked in, down to an and-inverter graph. Pong closes to a circuit with 52 input bits, 1,491 flip-flops, and 20,821 AND gates; Freeway to 52/1,491/20,798; Montezuma’s Revenge — whose 8 KB bank-switched cartridge brings its mapper hardware into the netlist alongside the console — to 52/1,501/36,698. The machines’ full state spaces are  $2^{1491}$  and  $2^{1501}$ .

This is where the scope condition of section 2.1 has its teeth, so it bears restating in this campaign’s terms: the toolkit can actually *examine* the environment’s digital-circuit structure. The transition relation is a given, finite, white-box object, and every claim species of this chapter becomes definable over it — the ranking rows of section 2.5 apply to the environment itself, not to an abstraction of it. System identification — recovering such a circuit from observations of play — is a separate problem, and one the toolkit does not yet have tools for. Nothing in this campaign learns the environment; it is handed the mechanism and reasons about it.

What is landed at this preview is the foundation tier: the compiled circuits and their fidelity receipts. Differential validation runs the circuit and the suite’s own emulator (ale-py) frame-locked, side by side: on Pong and Freeway, episode returns are *identical* — Pong’s rally period is exactly 140 frames on both; Freeway’s first crossing lands on the same frame — and 116–123 of the 128 RAM bytes match exactly on every frame, with each remaining divergence assigned to a named class (an input-transduction mismatch, since resolved by measurement; a one-to-two-frame boot-time skew attributed to the pinned CPU core’s

known non-cycle-exactness) rather than left as noise. One finding from this validation is recorded as a surprise: the emulator’s RAM trace is invariant across its random seeds — for these games, the suite’s stochasticity lives in frame-skip and action timing, not in the machine.

The first certified rows above that foundation landed as this preview closed: exact-replay returns — worst-case over the suite’s full initial-condition randomization, all 31 no-op offsets (species U1 of section 2.1) — for a fixed open-loop policy, certified by a checker whose only dynamics authority is the compiled circuit itself — “this policy scores  $R$ ” turned from a measurement into a theorem about the circuit. Table 2.7 shows the rows. The Freeway row carries a finding: the certified per-offset returns span  $\{2, 3, 4\}$ , so a single rollout of this policy reports a number drawn from the init lottery, and the certified minimum (2) sits strictly below the modal return — the gap between “return of a rollout” and “return guaranteed over the init set,” exhibited concretely.

The last two rows take the same claim species to Montezuma’s Revenge — the suite’s canonical hard-exploration benchmark, the game whose sparse rewards famously held early deep reinforcement-learning agents to a score of zero. To our knowledge these are the first certified return claims for this game at any claim class; every published Montezuma number is an empirical rollout statistic. The certified content is exact: reaching the first key (return 100) is certified at *every* one of the 31 no-op init offsets, and a 1,548-frame trajectory that collects the key and opens a door of the first room is certified at return exactly 400 from the declared initial state — each reward event pinned to its frame, all 128.5 million netlist clock ticks of the replay re-derived by the checker, every frame of the pinned witness traces compared bit-exact. The rows’ structure carries a finding of the Freeway kind, amplified: a committed instrument row certifies that the fixed key script which scores 100 at offset zero scores exactly 0 at every other offset — total collapse under the init lottery, where Freeway’s spread was one point — so the per-offset row holds thirty-one separately searched scripts, one per offset. Transfer fails the same way at finer grain: a script optimized against the reference emulator, where it scores 400, scores only 100 when replayed through the circuit, losing a life at a pixel-perfect skull crossing that succeeds under the emulator’s timing and fails under the circuit’s. Trajectories that certify on the circuit were searched on the circuit; the searcher is producer-side and untrusted, and nothing about how the scripts were found enters the trust story — the certificate is the exact replay.

In flight at this preview: liveness rows over the compiled circuit at the previous section’s claim classes, and policy rows — closed-loop pixels-to-action policies rather than fixed scripts — for the games themselves; toward the latter, the checker has already grown a pixel lane, which on the key trajectory compared every one of the 26,997,440 pixels the game renders across its 706 frames bit-exact against the circuit’s video output. Each will carry the same receipts discipline as every table above.

| environment       | policy                 | $T$  | init set          | min return | check  |
|-------------------|------------------------|------|-------------------|------------|--------|
| Pong              | NOOP script            | 1000 | $k \in \{0..30\}$ | −6         | 1405 s |
| Freeway           | all-UP script          | 1000 | $k \in \{0..30\}$ | 2          | 1410 s |
| MontezumasRevenge | per-offset key scripts | 828  | $k \in \{0..30\}$ | 100        | 3943 s |
| MontezumasRevenge | key+door script        | 1548 | $k = 0$           | 400        | 6416 s |

Table 2.7: Certified exact-replay rows (worst-case species U1 of section 2.1); each environment is the compiled circuit `AtariCircuit-⟨game⟩`. “Min return” is the certified minimum over the row’s declared init set (exact, hence the exact-cell tint of section 2.2) — all 31 no-op offsets, except that the key+door row declares the single offset  $k = 0$ ; the per-offset Montezuma row certifies one separately searched script per offset. Check times are single-core replay of the claim’s own circuit.

**What is not claimed.** These exact-replay rows are the certified ladder’s weakest rung — a fixed open-loop policy, exact replay, no quantification over adversaries and no learned certificate — and their receipts say so; no liveness or policy-optimality claim is stated for any Atari game at this preview. The Montezuma key+door row in particular quantifies over nothing: one declared initial state, one script, 1,548 frames — it certifies that score on that trajectory exactly and is silent about the rest of the game. Nothing here “solves” Montezuma’s Revenge; what is new is that a Montezuma score is carried by a machine-checked certificate against the machine’s own transition relation instead of by a rollout report. The five-to-twelve divergent RAM bytes per frame in the emulator differential are declared, not resolved; the CPU core’s cycle-exactness is a named seam of exactly the circuit-manifest kind (section 2.5); and claims about the

emulator itself rest on the differential receipts alone. The white-box condition above is load-bearing: none of this machinery applies to an environment whose mechanism cannot be handed over.

## 2.7 Process control: the semiconductor ladder

The final campaign leaves benchmarks-with-leaderboards entirely and runs on two tracks over a ladder of semiconductor-fab process models — the application-domain family the toolkit’s programme context cares about, and a domain where the certified claims are economically legible. The *fluid* track instantiates the chapter’s infinite-horizon species (U4 of section 2.1: long-run throughput and stability claims for queueing-network models): *this dispatch policy keeps the line stable at this load*. That claim has teeth because it is famously falsifiable: re-entrant lines destabilize under plausible dispatch policies strictly below nominal capacity (the Kumar–Seidman phenomenon), and the fluid-limit criteria of Dai and Dai–Weiss reduce stability to exactly the polyhedral drift inequalities the toolkit’s checkers replay. The *discrete* track — new at this preview, on the base rung — instantiates the worst-case ranking species (U2, quantified here over every scheduler rather than every initial state): *every dispatch of this line ships all lots by this deadline* — and carries it past any of the chapter’s other campaigns, to a claim class none of them produce: exact optimality, a certified floor over *all* dispatch policies met exactly by a shipped one.

**The ladder, whole.** The campaign’s target is a fixed progression of eleven public fab models in four stages — the Intel Mini-Fab; the seven SEMATECH MIMAC testbed fabs [82]; the two SMT2020 scenario fabs [142]; and SMAT2022 [158] — carrying the same claim species at every rung, so the ladder is a progression in scale and modelled physics, never a change of subject:

| rung          | one-line identity                                       | certified rows |
|---------------|---------------------------------------------------------|----------------|
| Mini-Fab      | 3 stations, 5 machines, 6 re-entrant steps (Kempf 1994) | this preview   |
| MIMAC set 1   | 2 products, 265 tools, routes to 245 steps              | —              |
| MIMAC set 2   | 7 products, 274 tools, routes to 354 steps              | —              |
| MIMAC set 3   | 11 products, 354 tools, routes to 547 steps             | —              |
| MIMAC set 4   | 7 products, 69 tools, routes to 92 steps                | —              |
| MIMAC set 5   | 21 products, 176 tools, routes to 266 steps             | —              |
| MIMAC set 6   | 9 products, 228 tools, routes to 355 steps              | —              |
| MIMAC set 7   | 1 product, 38 tools, one 172-step re-entrant route      | this preview   |
| SMT2020 HV/LM | high-volume/low-mix scenario fab, $\sim 10^3$ tools     | —              |
| SMT2020 LV/HM | low-volume/high-mix scenario fab, $\sim 10^3$ tools     | —              |
| SMAT2022      | the SMT2020 fabs + automated material handling          | —              |

MIMAC set characteristics are the testbed’s own dataset tables, tabulated independently by Zhou and Rose [248]. One count deserves a flag: the testbed’s catalogue prose says *six* datasets, and that count circulates in the citing literature — but set 7 (a synthetic single-product fab adapted from a published model of Lawrence Wein’s) entered the archive days after the catalogue text’s 1997 revision and was never re-catalogued, and the archive itself ships seven. SMT2020’s two scenario fabs each ship with an engineering-lots variant; SMAT2022 extends all four SMT2020 models with a full automated-material-handling layer — vehicles, rails, bays, ports.

The stages differ in which physics must enter the certificate, and that difference is the ladder’s actual content. The Mini-Fab is small enough that every modelling decision can be witnessed line by line — chapter 17 gives the model and this campaign’s certificates a chapter-length reading — yet it already carries the domain’s hard structure (re-entrance, batching, sequence-dependent setups, shared personnel, failures), so the base rung certifies on both tracks: fluid-drift stability with the awkward layers priced by explicit adversarial rows (work-in-progress and cycle-time moments as named successors), and the discrete dispatch campaign, carried at this preview through policy synthesis, wall-clock optimality, and a certified envelope for untrusted policies. The MIMAC stage is the same claim species at real-fab scale, where the sign-mode census dies combinatorially: certifying it means aggregated covers with certified refinement paths, plus an extraction pipeline from the testbed’s 1990s fixed-width dataset format to declared network rows. The SMT2020 stage adds another order of magnitude ( $\sim 10^3$  tools) and a maintained community simulator (PySCFabSim), so analytic rows gain a simulation oracle beside them — anytime-valid statistical confidence bounds (the U5 rows of section 2.1) apply at every rung today and are the honest first deliverable



at the top ones. SMAT2022 adds the transport network itself to the contended resources, and its reference simulator is closed-shape, so the declared model carries the semantics alone.

**Certified today: the Mini-Fab, fluid track.** The base rung's model — the community's minimal re-entrant fab, whose batching, setup, personnel, and failure layers make five machines an honest miniature of the domain — is described at reading depth in chapter 17, which also walks these certificates decision by decision; here, the campaign summary. For the fluid model of the deterministic core, under named static-priority dispatch policies (last- and first-buffer-first-served), with all residual scheduling freedom resolved adversarially: certified linear-Lyapunov drain certificates at release rates up to 95.9 of the 96 lots/week fluid capacity (99.90%), with the checker independently rebuilding every polytope and replaying one exact- $\mathbb{Q}$  Farkas dual per mode — and the probe at 96 correctly failing, because the load is genuinely at capacity: producer-side bisection brackets the last certifiable release within  $6 \times 10^{-5}$  lots/week of the fluid capacity, so the one-lot gap in this campaign's earlier grid was grid granularity, not lost capacity. A setup-adversarial variant (setup times charged worst-case inline) certifies up to 99.8% of its own reduced ceiling. Certificates carry 819 obligations and check in 1–3 ms; the two rows nearest capacity ride fully-refined sign-split tree covers — the refinement species the testbed rung below leans on, exercised first here. The certified drain rates also separate the two policies near capacity — an exact policy-comparison number of the kind this claim species was chosen to produce.

**Certified today: the Mini-Fab, discrete track — dispatch synthesis, to exact optimality.** The same line, read as a discrete transition net (16 places, 19 transitions, 49 arcs; every logistical step one tick), carries a family of twelve certificates — 150 footprint rows of a sealed certificate container: a work-potential ranking budgeting each lot's run at exactly 638 firings, lot conservation, six machine conservation bounds, congestion, a ceiling, and an arrival-potential + step-budget pair yielding the worst-case deadline, *all  $N$  lots arrived within  $638N - 12$  steps*, over every scheduler interleaving. Nothing in the certificate scales with the lot count:  $N$  enters only as three integers in the container's own data, so the same rows instantiate from  $N = 7$  to  $N = 10^7$  with the container growing six bytes across six orders of magnitude and the checker's wall time flat at  $\sim 2$  ms per container — at the top of that sweep the claims quantify over at least  $2 \times 10^{38}$  reachable markings and  $10^{2 \times 10^7}$  distinct maximal schedules, none of which the checker visits. Opening one supervisor seat (the plant re-stated as a comb with a single policy-shaped hole; the hole count is itself a checked claim) turns the deadline into a synthesis instrument: joint policy+certificate search returns a latest-safe-unlock supervisor with certified completion and all arrivals by  $626N$  steps — 52,584 against the 53,580 no-policy worst case at the weekly  $N = 84$ , a certified throughput-floor gain of 1.9%. The synthesis objective is lexicographic — certified completion first, then the horizon — and the order is load-bearing: a gate that never opens ties the arrival optimum and then deadlocks shipping nothing, and the search refuses it at the completion certificate with the starved queue named.

The counting-time story transfers to the wall clock, and closes there. Under the theory's discrete clock semantics (every transition dwells exactly one tick) the same containers reseal with identical row tallies, and the counting bounds become tick bounds by a no-idle argument ( $\leq 626N$  gated,  $\leq 638N - 12$  open). Beneath them sits a floor over *all* dispatch policies of any kind, supervisory or never-idle: no policy reaches all-arrivals in fewer than  $255N + 375$  ticks for  $N \geq 4$  — a machine-contiguity argument proved in Lean 4 over mathlib, with adversarial mutants at preregistered loci refused by the kernel, lifting an arithmetic floor of  $255N + 371$  (verified by exhaustive SMT search below  $N = 8$ , with witness schedules independently re-validated); the exact wall-clock optimum is  $255N + 371 + d(N)$ ,  $d = (0, 0, 2, 4, 4, \dots)$ , closed from both sides at every  $N$ . An explicit supervisor family attains the lifted floor exactly at every tested lot count: 21,795 ticks at  $N = 84$ , per-lot throughput 255 ticks closed from both sides — so the certified-optimality column of the dispatch grid reads 100.00%, earned rather than rounded: a Lean-checked floor meeting a container-replayed attainment at  $255N + 375$  exactly. The grid around it separates the levers. The sixteen never-idle full-dispatch policies deliver 99.63, 75.61, 65.34, and 54.35% of certifiably-unbeatable performance, and only the diffusion-station pull preference ever moves a number; deliberate idling is necessary for the last stretch (the best work-conserving policy needs  $255N + 455$  ticks, 80 behind optimal); and the wall-clock value concentrates in the dispatch seat: an adversarial pull order costs +83.5%, while the synthesized gate — whose value is the certified worst-case floor, a guarantee over every schedule — moves typical-case arrivals 0.0%. Optimality here is also the decision-theoretically right word: with deterministic closed dynamics, imprecise-decision-theoretic policy selection provably collapses to makespan minimization (machine-checked, with a refutation cell showing the collapse fails without determinism).

The track's last exhibit prices the case nobody certifies: an *uncertified* policy — a lookahead heuristic



standing in for a learned dispatcher — filling the same seat behind a certified switching envelope (certified baseline, decision-budget bench, certified recovery region). The envelope’s certificates never read the policy. In replay, the heuristic ran free and attained the envelope’s arrival bound  $638N - 11$  exactly at  $N = 84$ , so the bound is witnessed tight, not merely sound; a deliberate idler was benched at the budget boundary with recovery at exactly 1,009 firings; a rogue policy was benched after eight named in-loop refusals with the plant untouched; and after all runs the sealed container re-verified byte-identical. The whole discrete family — closed baseline, open comb, synthesized supervisor, switching envelope, and the wall-clock reseals of baseline, comb, and supervisor — re-checks from bytes alone by one reproduction target (section 2.9), with committed integrity mutants that must refuse and the producer-tier mutant batteries re-run beside it; chapter 17 walks the model, the synthesis, and the grid at chapter length.

**Certified today: MIMAC set 7, at the cover and one refinement in** (the smallest of the seven by tool count: one product, 38 machines in 24 tool groups, one 172-step re-entrant route). The structural problem at this scale is that mode-by-mode certification is  $2^{172}$ -large; the certificate instead quantifies over an *aggregated cover* — cells whose relaxed polytopes provably over-approximate every sign mode, by a lemma whose only move is dropping constraints (so soundness is monotone by construction). The honest price appears in the bound, and so, now, does the refinement path. At the coarsest cover (172 cells; proven weight-optimal for that cover) the certified stable load is 13.2% of the dataset’s nominal release (availability-adjusted variant: 11.5%). Walking the certificate format’s built-in refinement path (binary sign-split trees, anytime-refinable down to the exact mode walk) — with the weight search and every cell’s dual solved as one sparse linear program — lifts this to a certified  $1964/9903 \approx 19.8\%$  of nominal on a 244-cell tree ( $1.50\times$  the coarsest cover; 84,180 obligations, replayed by the checker exactly like every other row). Both numbers are weak, both are displayed at full prominence, and the certified 19.8% is a floor rather than a ceiling: the refinement run was stopped early, with producer-side bisection already bracketing 23–24% at five times the cells (measured, not yet certified) — and the measured refinement curve is sublinear enough that explicit trees cannot reach nominal load at fab scale, so pricing that species limit, and the implicit-cover species that removes it, is named successor work. The aggregation gap is a measured frontier, not fine print.

**What is not claimed.** On the fluid track these are claims about declared fluid models of the fabs, with batching, setups, and failures handled by explicit adversarial or interval rows where modelled at all — the model gap to the discrete line rides a declared ledger, and the conversion from certified drain rates to work-in-progress and cycle-time bounds on the discrete line (via Foster–Lyapunov moment rows and Little’s law) is named successor work. On the discrete track the claims are exact, but the model sits deliberately in the unit-dwell deterministic corner: zero stochastic species instantiated, so a policy-filled closure has one trajectory per lot count, and the demonic quantifiers live at the family level (the all-policy floors; the envelope over an untrusted filler). Since every certificate quantifies over the full scheduling freedom, its claims dominate every rate-raced refinement of that freedom — the stochastic layers, when they arrive on the higher rungs, resolve freedom the certificates have already priced. An earlier preview of this section stated that no claim was made about optimality of the dispatch policies, only their certified stability region; that sentence is now true of the fluid track alone — on the discrete Mini-Fab the optimality question is closed, floor and attainment meeting at  $255N + 375$  — and the correction is recorded here rather than silently edited away, because the distance between the two previews is itself a result. The ladder’s remaining nine rungs — MIMAC sets 1–6, the SMT2020 pair, and SMAT2022 — are census-scoped but carry no certificates at this preview.

## 2.8 The Lean route: how much Lean?

The toolkit’s third checking path — the last of the three checking routes whose division of labor the front matter states (chapter 1) — loads the same sealed certificate container that every other checker consumes into Lean, reconstructs the claim, and lets Lean accept it as a theorem. The route’s decode, reconstruction, and validity elaboration are ordinary Lean definitions; its acceptance theorem says that anything the route accepts is semantically valid — valid, that is, under every admissible valuation of the hypernet the certificate declares: the same Bernstein Logic valuation semantics the toolkit’s other checkers answer to — and the proof is one line, because the accepted value carries the validity proof built during elaboration. The same definitions are consumed two ways: *compiled*, as a native executable produced by the Lean compiler, and

*kernel*, where a generated receipt module asks the kernel itself to evaluate the entire pipeline over the embedded bytes (*decide*) — no compiler, no extraction, just the trusted evaluator. The route exists to make a measured statement, not a rhetorical one: the expectation has always been that the kernel path is far too slow for machine-scale certificates, and this section measures how far.

### 2.8.1 The ladder

The measurement corpus is a size ladder of fifteen production certificates — sealed containers from the working producer stack, spanning 603 bytes to 126,216 bytes ( $\times 209$ ) across seven certificate families, each rung accepted by the independent reference checker before any Lean timing was attempted. The ladder is committed byte-for-byte in `lean-route/fixtures/`; as elsewhere in this repository, the certificates stand on their own, and no producer is part of the trust story. All timings in this section were taken on one otherwise-idle machine (AMD EPYC 7B13, 16 threads, 126 GiB), single job at a time, with a 10-minute budget per certificate per route; Table 2.8 shows the ladder. An em-dash marks a certificate whose run exceeded the budget; *o.o.m.* marks one that exhausted the machine’s memory first; *o.o.f.* (out of fragment) marks the scope boundary discussed below — the route declines any certificate outside its accepted fragment rather than guess at it, a refusal that costs completeness and never soundness — and kernel peaks for non-completing runs are reported as floors (the resident size when the run was cut).

| certificate                  | bytes   | proof nodes | compiled      | kernel        | kernel peak     |
|------------------------------|---------|-------------|---------------|---------------|-----------------|
| enclosure_v1_golden_cos6     | 603     | 1           | <i>o.o.f.</i> | <i>o.o.f.</i> |                 |
| ctmc_sis_gen                 | 919     | 3           | 2.4 ms        | <i>o.o.m.</i> | $\geq 124$ GiB  |
| ctmc_fk_mid                  | 1,793   | 35          | 51 ms         | <i>o.o.m.</i> | $\geq 124$ GiB  |
| micro2_stable                | 2,184   | 96          | 0.12 s        | —             | $\geq 0.75$ GiB |
| cartpole_core_decrease_s0001 | 4,068   | 65          | 0.43 s        | —             | $\geq 0.76$ GiB |
| rode_uniform_decrease        | 7,652   | 143         | 2.9 s         | —             | $\geq 0.76$ GiB |
| cartpole_dp_core_decrease    | 9,336   | 197         | 9.6 s         | n.a.          |                 |
| dp_composite_syzygy          | 17,591  | 377         | 99 s          | n.a.          |                 |
| dp_composite_eigenrate       | 22,056  | 647         | 7.4 min       | n.a.          |                 |
| lti_sampled_flow             | 24,686  | 1,693       | 8 min         | n.a.          |                 |
| acasx4_init_s0000            | 43,401  | 527         | —             | n.a.          |                 |
| dp_composite_smallgain       | 46,015  | 1,081       | —             | n.a.          |                 |
| acasx4_promise_closedloop    | 57,779  | 1,055       | —             | n.a.          |                 |
| rode_nest_shell              | 119,217 | 4,629       | —             | n.a.          |                 |
| rode_hold_flow               | 126,216 | 5,773       | —             | n.a.          |                 |

Table 2.8: The ladder. “compiled” is the median end-to-end wall clock of repeated passes of the compiled route; “kernel” is the wall clock of one kernel (*decide*) evaluation of the same bytes. Em-dashes are runs cut at the 10-minute budget; *n.a.* marks kernel runs not attempted because every in-fragment kernel attempt at smaller sizes had already ended cut or out of memory.

### 2.8.2 The scaling law

Across the 9 rungs the compiled route completes within budget, end-to-end time follows a power law in container bytes:  $t \propto B^{3.58}$  (95% CI [3.24, 3.92],  $r^2 = 0.989$  on log-log least squares). Container bytes is the best of the three natural size units as a regressor ( $r^2 = 0.989$ , against 0.962 for certificate rows and 0.909 for proof nodes) — expected, since the dominant cost is the reader: list-based CBOR decoding and seam assembly in a purely functional style, chosen so that the same definitions remain evaluable by the kernel. Nothing about the certificate format requires a superlinear reader, as the linear-time imported checkers on the same bytes demonstrate — and as the fast-path route’s own fitted exponent on this same ladder now measures (1.20, its confidence interval containing linear). Figure 2.1 shows the measured points, the fitted law, and the budget line — and, since the trichotomy’s claim is comparative, the same ladder through the toolkit’s other two checking routes, measured per certificate on the Arm host (section 2.8.6 carries the hardware and scope contract).

The kernel mode needs no fitted curve to make its point. Its smallest full acceptance during the route’s development — the 919-byte certificate, second rung of the ladder — cost 965 seconds and a 233 GiB peak on a 251 GiB development machine (the toolkit repository’s route documentation carries that receipt); the memory is the kernel’s reduction state, not the certificate. On this section’s 126 GiB reference machine the same certificate exhausts neither budget nor patience but memory, and every in-fragment kernel attempt

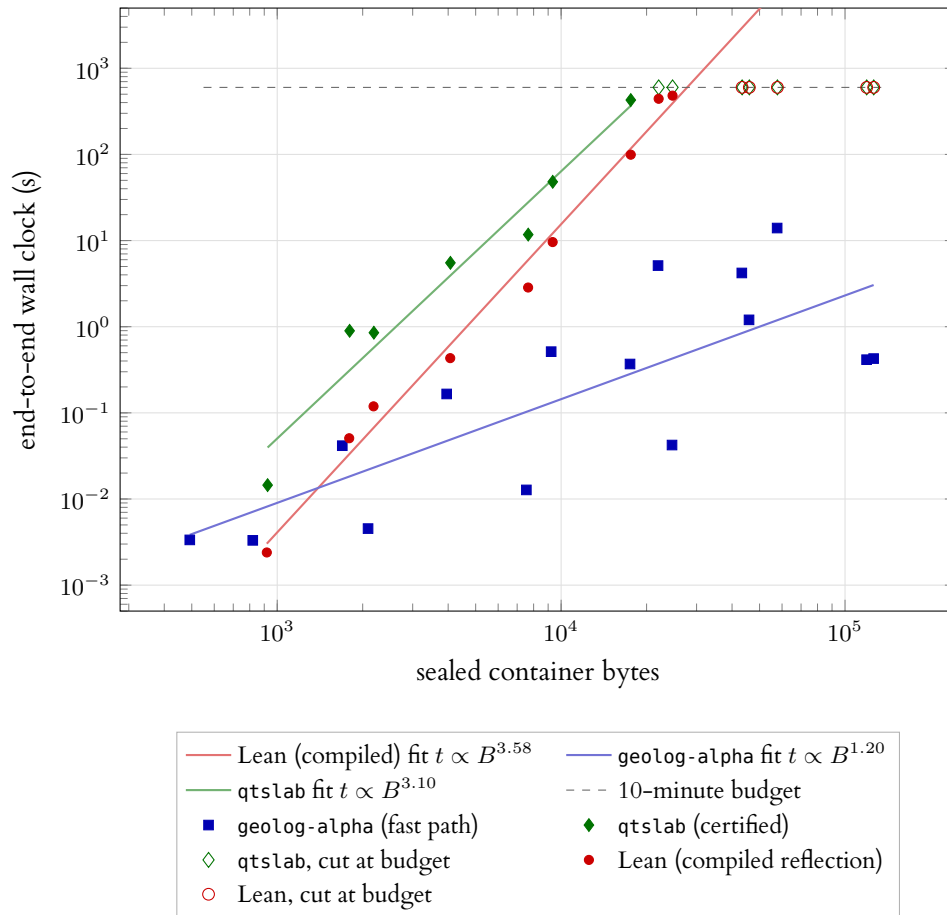


Figure 2.1: One ladder, three checking routes: the fast-path engine (geolog-alpha, blue squares, on the staged-chain sealed containers), the certified checker (qtslab, green diamonds, on the same certificates re-marked content-resolved — a container-table transform under which every certificate row is byte-identical), and the compiled Lean reflection route (red circles) with its fitted law. Blue and green medians are one process, end to end, pinned to one core of the Arm host; the red points are this section’s own protocol on its dedicated machine (the footnote of section 2.8.6 carries both). Open marks of either color sit *at* the dashed budget line: each true time lies above its mark. A rung with no mark on some route is a scope datum, not a failure — that route classes the certificate outside its accepted fragment and declines it (the caption of Table 2.8 and section 2.8.6’s footnote carry the per-route scope and hardware contract). The Lean fit uses only uncensored red points; where it crosses the budget —  $\approx 28$  KB of sealed container — is the admission threshold of Table 2.9. Each solid line is the same log-log fit to its route’s accepted medians: geolog-alpha at exponent 1.20 (95% CI [0.63, 1.78], containing linear;  $r^2 = 0.61$ ), qtslab at 3.10 ([2.30, 3.91],  $r^2 = 0.95$ ), Lean at 3.58 ([3.24, 3.92]) — one near-linear route and two superlinear ones, measured on the same certificates.

on the ladder ended cut at the budget, out of memory, or predictably unattempted (Table 2.8). The failure mode is content-dependent, not size-monotone: the 919-byte certificate took the whole machine's memory in under eight minutes, while certificates two to eight times its size ground past the budget inside a gigabyte — a  $\times 160$  spread in peak residency between neighboring rungs. What a given certificate makes the kernel's reduction state do is a property of its content (the working hypothesis is exact-rational arithmetic in kernel terms, untested here), not of its byte count. A certificate of under a kilobyte that the compiled route accepts in milliseconds is already past what the kernel evaluator will do inside a ten-minute, hundred-gigabyte envelope; that is the measured content of “too slow at machine scale,” stated with its numbers attached.

### 2.8.3 The fragment boundary

One ladder rung is not a timing datum but a scope datum: the smallest certificate (603 bytes, an interval-enclosure certificate whose obligations are discharged by *primitive applications*) is out of fragment for the route — the table's *o.o.f.* — and the route declines it: compiled and kernel agree, and the kernel's refusal is itself a theorem (decide proves the acceptance predicate false on those bytes). The same bytes are accepted by the reference checker, which settles the direction of the disagreement: the route's accepted fragment is the first-order pinned-vocabulary rule set, primitive applications lie outside it by design, and a checker asked about a certificate it cannot speak for declines it rather than guess. Every checker in this chapter draws some such boundary, always with the same sign — an out-of-fragment refusal costs completeness, never soundness; the certified checker classes this same rung outside *its* fragment too, naming the reason in its verdict (`fun_outside_fragment`), while the fast-path lane discharges it (Figure 2.1). Admission to the Lean route is therefore a conjunction: small enough, *and* inside the fragment. A species' first Lean attempt should be its smallest instance, which settles fragment membership in milliseconds before any batch of timings is scheduled.

### 2.8.4 The admission gate

Table 2.9 applies the fitted law to the certificate species of this chapter's campaigns: per species, the byte range of its committed certificates (container overhead added), the predicted compiled-route time at the median and maximum, and the verdict against the 10-minute budget. The table is the gate the re-run campaigns consume: a Lean-reflection timing column is attempted only for species that pass (or, at the runner's discretion, the passing fraction of a *partial* species), and the kernel mode is attempted for none — the measured points above place its envelope below every production species' floor.

### 2.8.5 The measurement before this one

This ladder is the route's second measurement, not its first. In July 2026, on the toolkit's predecessor binary certificate format (since retired), the same question was measured by the other road into the kernel — *elaboration*: an emitter translated certificate bytes into generated Lean source ( $\times 8$  expansion, byte-stable per-node templates across certificate kinds), and the elaborator produced kernel-checked object files (a further  $\times 24$ – $25$ ) closing with the same species of validity theorem. Three production certificates were taken to full kernel verdicts that way — 197, 377, and 1,081 proof nodes, at 3.4, 9–13, and 28 minutes of elaboration respectively, against 3–10 *seconds* for the certified checker build of the same era on identical certificate content and comparable hardware: a measured factor of  $\times 30$  to  $\times 250$  at kilobyte scale, growing with size. The same campaign took a streaming census of a real machine-scale certificate — a 43,285,639-node, 1.24 GB closed-loop neural-network verification certificate, semantically re-checked by a reference checker in 2.4 hours — and projected the elaboration route on it:  $\approx 7$  GB of generated Lean source,  $\approx 66$  million declarations in one namespace (about one hundred mathlibs),  $\approx 176$  GB of object files, and 1.4–2.7 core-years of serial elaboration, with memory the first wall. The numbers, the census, and the projection's stated assumptions are recorded in `lean-route/HISTORY.md`. That projection is why the present route abandons source generation altogether: a fixed decoder and checker are defined once, certificates arrive as ordinary input bytes, and the measured question becomes the reader's scaling law above rather than gigabyte-scale generated source.

| species                                                  | max B      | fragment | pred. med.     | pred. max      | verdict                |
|----------------------------------------------------------|------------|----------|----------------|----------------|------------------------|
| hardware liveness (circuits)                             | 2,627      | untested | 9.1 ms         | 0.22 s         | pass                   |
| Atari certified replay                                   | 5,113      | untested | 22 ms          | 1.9 s          | pass                   |
| FrozenLake 4x4 (det + slippery + sr34)                   | 52,015     | exp. in  | 15 s           | 97 min         | partial                |
| CliffWalking                                             | 71,872     | exp. in  | 14 min         | 5.1 h          | fail                   |
| FrozenLake 8x8                                           | 500,289    | exp. in  | 5.5 h          | 0.59 yr        | fail                   |
| Blackjack                                                | 115,779    | exp. in  | 10 h           | 28 h           | fail                   |
| Taxi (all weather/passenger variants)                    | 3,102,602  | exp. in  | 7 yr           | 400 yr         | fail                   |
| LunarLander / LunarLanderContinuous (statistical tier)   | 12,226,428 | untested | $\sim 10^4$ yr | $\sim 10^5$ yr | fail                   |
| published-table supermartingale corpus                   | 50,000     | exp. in  | 97 ms          | 84 min         | partial                |
| semiconductor fluid ladder (minifab)                     | 18,451     | untested | 65 s           | 2.5 min        | pass                   |
| semiconductor fluid ladder (mimac)                       | 401,455    | untested | 0.18 yr        | 0.27 yr        | fail                   |
| interval-enclosure certificates (primitive applications) | 603        | out      | 4.4 ms         | 4.4 ms         | <i>out of fragment</i> |

Table 2.9: The admission gate. Predictions apply the fitted law to each species’ certificate sizes in sealed-container bytes (census rows stated in committed-artifact bytes carry the measured container overhead as a declared correction); beyond the fitted range ( $\approx 25$  KB) they are extrapolations — one-sidedly corroborated by the budget-censored rungs up to 126 KB, order-of-magnitude statements past that. “Fragment” marks measured membership only for the ladder’s own species; *untested* species settle theirs with a one-instance probe before any batch attempt.

## 2.8.6 The route beside the other two

The trichotomy’s claim is comparative, so the honest summary is too.<sup>1</sup> On this chapter’s stored supermartingale corpus — 4,105 sealed containers, each one verification condition’s Bernstein Logic discharge at roughly one to fifty kilobytes — the certified checker’s current build replays every container at a pooled median of 30 ms, maximum 117 ms, on one Arm core (Table 2.3), and the fast-path engine’s sealed-container lane re-checks the whole corpus in wall-clock seconds (supermartingales/GLC-CERT.md). The compiled reflection route has not been batch-run on that corpus; the admission gate prices it (Table 2.9): verdict *partial*, with a predicted median near 0.1 s and a predicted maximum near 84 minutes, so only the passing fraction would ever be attempted. Figure 2.1 makes the comparison direct on the ladder itself, every route in its current build on identical certificate content: the fast path near-linear in bytes (exponent 1.20), the reflection reader at  $B^{3.58}$ , and the certified build accepting 7 rungs and cut at the budget on 7. Within the ladder the certified wall grows as  $B^{3.10}$  ( $r^2 = 0.95$ ) — statistically indistinguishable from the reflection reader’s exponent, while the fast path’s confidence interval excludes both — and across corpora content dominates bytes: the stored corpus above tops out at 117 ms per certificate at sizes where the ladder’s composite certificates cost minutes, which is how a 30 ms pooled median there and these walls here are both true. A species aiming certificates at the certified route should therefore run the same one-instance probe the admission gate prescribes for Lean: fragment membership and cost are both content-properties, settled cheaply on the smallest instance. The historical cross-check stands beside the figure: on the composite small-gain rung (1,081 proof nodes, 46 KB sealed), the certified checker of the elaboration era took about

<sup>1</sup> Hardware, per the conventions: the certified checker is timed on one core of an Arm host — the ISA its soundness theorem grounds in — with its own cross-machine spread measured under  $\times 1.15$ ; in Figure 2.1 the two production routes’ ladder points are one process per certificate, pinned to one performance core of one Arm box (Cortex-X925 + Cortex-A725), while the red points remain this section’s reflection timings (AMD EPYC 7B13, 16 threads, 126 GiB); the elaboration-era numbers are as recorded in lean-route/HISTORY.md. Every contrast quoted here is two or more orders of magnitude, and the measured cross-machine factor on the compiled route is a factor of a few — hardware choice cannot close any of them. Container forms, per route: the fast path consumes the staged-chain re-seals (lean-route/GLC-CERT.md), the certified checker the content-resolved mints (lean-route/tools/mint-qtslab-vintage.py states the container-table-only transform), and both carry the same certificate rows byte-for-byte as the committed fixtures the Lean route reads. Scope, shared by all three routes on this corpus: these certificates natively encode the Bernstein Logic fragment, which is the only fragment the certified checker consumes (theories/BernsteinLogic.geolog); the full QTSL theory is checked on the engine-side container lanes of section 2.9.

ten seconds on the same content in its own format's bytes (today's build, reading the unified sealed container, is cut at the budget on that rung), against past the ten-minute budget through the compiled reflection route and twenty-eight minutes through the retired elaboration route.

The conclusion the measurements support, stated as the front matter's expectation deserves: *confirmed, with its boundary measured*. For the kernel evaluator the confirmation is categorical — a certificate under a kilobyte already costs sixteen minutes and a quarter-terabyte of reduction state, and nothing larger completes at all — and for the compiled route it is a statement about scaling rather than about the route: below a few kilobytes the compiled reflection path is genuinely fast (the 919-byte rung's 2.4 ms is *under* the certified build's corpus median, on different certificates of the same format), but at  $\times B^{3.6}$  against the production checkers' linear-with-content growth, the ratio passes two orders of magnitude by fifty kilobytes and machine scale is out of reach by shape, not by constant factors. The route earns its keep on the other side of the ledger: full kernel verdicts on small certificates tie the certificate semantics to a proof kernel scrutinized by a community that owes this toolkit nothing — and the measured boundary is precisely why the certified binary, not the kernel, walks the forty-million-node instances.

**What is not claimed.** The law is a property of this route's current reader on one machine class, not of Lean, whose kernel and compiler are here being used far outside their engineered envelope on purpose; an index-structured reader would move the compiled curve substantially, and nothing in the trichotomy's argument depends on it not moving. Cross-machine transfer of the constants is measured to be worth a factor of a few (the 2,184-byte rung runs 27.6 ms on the development machine cited above and  $\approx 120$  ms here); the exponent, not the coefficient, is the portable content. Species predictions inherit a byte-basis caveat until each campaign banks per-instance container sizes, and the admission table regenerates from those receipts by the same make target that produced it. The elaboration-era numbers of `lean-route/HISTORY.md` are development-record measurements on a retired format, quoted for the route's history with their measured/projected distinctions preserved; they are not re-derivable from this repository, and no other number in this chapter rests on them. Finally, the ladder's timings were taken against the committed fixtures; the same certificate content re-sealed on the unified certificate schema is already committed beside them (`lean-route/out/glc-baked/`, with the content-identity argument stated in `lean-route/GLC-CERT.md`), and when the ladder re-runs on those containers this section's numbers move with them, per the reproduction discipline of section 2.9.

## 2.9 Reproduction

Every campaign in this chapter ships its evidence in the coalition's experiments repository (`maitria-coalition/qtsl-experiments`): the certificates themselves, committed byte-for-byte; their mutant batteries; the machine-readable receipts (JSON) from which this chapter's tables are re-derived mechanically; and the independent checkers, imported from the `qtsl` repository at pinned revisions. One make target per family re-verifies the stored certificates and mutants through the checkers and regenerates the tables.

The certificate vocabulary itself is committed, as data: the `qtsl` repository ships the theory pair `theories/BernsteinLogic.geolog` — the Bernstein Logic proof tower, exactly the fragment the certified checker consumes — and `theories/Qtsl.geolog` — the omnibus theory extending it with systems, specifications, and the judgments connecting them — and a certificate is ordinary rows of these theories; there is no bespoke certificate file format. That unified schema is the standing discipline for everything this chapter reports: one certificate vocabulary for every campaign, revision-pinned modelling-toolkit producers used where each family has shipped producer coverage, the database engine as the fast-path checker, and an extraction tool reducing a full-QTSL certificate to its Bernstein Logic obligations for the certified checker. Families without a distributed producer retain a revision, invocation, and binary-digest provenance record rather than being described as runnable from a public clone. The shipped form is the sealed container — one signed `geolog` commit per certificate, re-verified from the bytes alone, with obligation and claim counts demanded exactly: deleting a proof node leaves every surviving obligation true, so the count gate is load-bearing, and a container that claims nothing is refused rather than vacuously accepted.

The campaigns of this chapter predate that unification, and the transition is in flight, not finished. Certificates produced on the earlier staged vocabularies remain the evidence behind the tables: they stand as the record of the campaign that produced them, still re-checked by the make targets, and nothing in them is presented



un-checked. Re-emissions on the unified schema have landed for the first families — the fluid-drift, published-table-sweep, Lean-ladder, and Gymnasium lanes, with the SPDE family's noise spectra re-homed first under its split-container design; each re-emission's content deltas are disclosed per container in its family's manifest. The uniform re-conduct of every campaign through its revision-pinned producer is the successor work the tables track: as family-scoped producer coverage and re-conducted receipts land, the numbers re-derive from them, and the superseded certificates retire to an archive rather than being silently rewritten.

A producer is never part of any trust story, so the audit path is the same on either side of the transition: read a checker, run it against the shipped certificates and their mutants, and only then decide what the tables mean. The published-table sweep's certificates are additionally stored as plain-text listings of database object rows — the geolog encoding, human- and checker-readable alike — each pinned by hash in a manifest, and its containers prove content identity against those listings at production, so the certificate a checker reads is the certificate a reviewer can read. Per the preview note at the front of this volume, artefacts migrate to the public repositories as they graduate; until a cited artefact appears there, treat its mention as a statement of what exists, not yet a link.

PART I

# Geolog

---

## Quick Start

---

In this chapter you will install `geolog`, write a theory and a handful of facts in one file, and run the chase: the engine will derive the tables your axioms entail, invent nothing you did not license, and report — rather than repair or reject — the one state your theory forbids. At the end you will look at the theory you wrote and find it is itself rows. The whole path fits in one sitting; every command is copy-pasteable, and every output shown is the verbatim transcript of a real run.<sup>1</sup>

### 3.1 Install

A Rust toolchain is the only prerequisite (stable channel; `rust-toolchain.toml` pins the components).

```
git clone https://github.com/maitria-coalition/geolog-alpha
cd geolog-alpha
cargo install --path crates/geolog
```

A fresh clone compiled in about a minute on the commodity arm64 box that regenerates this book's transcripts. `geolog --help` lists ten verbs; the three this chapter uses are `check`, `chase`, and `rows`.

### 3.2 A theory

The example is a freight network, small enough to hold in your head: places, direct hops, and reachability. The file `examples/routes.geolog` declares one sort, two relations, and three axioms:

```
theory Routes {

  Node : Set;

  // Asserted: a direct hop exists.
  Edge : [src: Node, tgt: Node] -> Prop;

  // Derived: some chain of hops exists. Rows appear exactly when the
  // axioms force them – never asserted directly.
  Path : [src: Node, tgt: Node] -> Prop;

  // `chase` marks a derivation rule: the axiom FIRES, creating rows.
  // (`report` marks a constraint: violations are judged, nothing is
  // created; an unmarked axiom defaults to report – the least-
  // dangerous reading, so derivation is always an explicit choice.)
  chase ax/path/edge : forall a : Node, b : Node.
    [src: a, tgt: b] Edge |- [src: a, tgt: b] Path;

  chase ax/path/trans : forall a : Node, b : Node, c : Node.
    [src: a, tgt: b] Path, [src: b, tgt: c] Edge
    |- [src: a, tgt: c] Path;

  // A denial: this network is required loop-free. A commit that lands
  // a cycle is not rejected – the violated denial is REPORTED as a
  // judgement, and repair is a further ordinary commit.
```

<sup>1</sup>The book's build regenerates these transcripts by executing the printed commands against the committed examples, and each fragment's header records the exact tool revision it ran against (all runs on one commodity arm64 box).

```

    report ax/no/loop : forall a : Node.
      [src: a, tgt: a] Path |- false;
  }

```

Four things to read off the shape.

1. Relations are declared over named record fields, and axioms are *sequents*: universally quantified entailments, each premise a conjunction of atoms, each conclusion here a single atom. The odd one out is the third axiom, whose conclusion `false` makes it a *denial*: a state the theory forbids.
2. The word in front of each axiom's name is its *mode mark*, and the file's own comments state the semantics: `chase` fires, `report` judges, and an unmarked axiom means `report`. An axiom creates rows only where its author said so in as many words.
3. Note what the mark does *not* do: nothing in the *declarations* marks `Path` as derived. The mode belongs to the axiom, and `Path` being a table nobody writes remains an emergent fact — about which relations your chase-marked axioms conclude into and which your commits assert into — not a schema annotation.
4. The slash in `ax/path/edge` is nothing special: slash-names are ordinary identifiers, used by convention for namespacing. The `ax/` prefix, however, does real grammatical work: it makes a statement an axiom rather than a declaration, so a sequent written under a name like `my/rule` is a syntax error. The Reference Manual (chapter 5) spells out these name rules.

Nothing travels beside the theory: a `.geolog` file is self-contained — everything it draws on is either declared in it or imported at the top. This is the *roster-totality* discipline: every declared relation has a decided evaluator. ordinary (asserted or chase-derived) is the default, declared by writing nothing. When a relation is instead decided by a bounded external procedure, an import line says so — `import <provider>.{rel}`; — classifying it as a *side-condition* awaiting that provider. The same import syntax authorizes builtin ground vocabulary by name (`import std.{Q, q_lt}`;). Nothing guesses on your behalf: an `import std` line naming something outside the builtin roster refuses at once, listing the importable names. This chapter's example needs no imports; every relation in it is ordinary, and the file's opening comment says so.

### 3.3 An instance

The same file carries the facts, in an instance block — element declarations first, then facts in the same record idiom the axioms use:

```

instance Demo : Routes {
  hub, east, west, port : Node;

  [src: hub,  tgt: east] Edge;
  [src: hub,  tgt: west] Edge;
  [src: east, tgt: port] Edge;
}

```

Elements are declared before use; a fact naming an undeclared element is an error, never a silently minted row. (`geolog check` validates all of this — declarations, field sets, sorts — without running anything.)

The map is worth ten seconds: `hub` ships to `east` and to `west`, and `east` ships on to `port`. Before running anything, guess which pairs the axioms will put in `Path`.

### 3.4 The chase

```

$ geolog chase examples/routes.geolog
sorts:
  Node — 4 elements (4 seeded)
relations:
  Edge — 3 rows (3 seeded, 0 derived)
    [src: east, tgt: port] Edge

```

```

    [src: hub, tgt: east] Edge
    [src: hub, tgt: west] Edge
    Path – 4 rows (0 seeded, 4 derived)
    + [src: east, tgt: port] Path
    + [src: hub, tgt: east] Path
    + [src: hub, tgt: port] Path
    + [src: hub, tgt: west] Path
    fixpoint: reached, no judgements – the model is saturated.
    $ echo $?
    0

```

Look at the + rows: tables nobody wrote. [src: hub, tgt: port] is in Path because ax/path/trans forces it from two rows you did write, and the chase ran that forcing to its *least fixpoint* — everything the axioms entail, nothing they do not. (The other three are your own hops, lifted into Path by ax/path/edge: one new reachability fact, four new rows.) “Derived” names provenance rather than any lesser grade of truth: these rows are as queryable as the seeded ones, and rerunning from the same facts re-derives exactly the same table.

### 3.5 Break it

Add one hop to the instance block — the return leg that closes a loop — and save the edited file as routes-cycle.geolog:

```

    [src: port, tgt: hub] Edge;

    $ geolog chase routes-cycle.geolog
    sorts:
      Node – 4 elements (4 seeded)
    relations:
      Edge – 4 rows (4 seeded, 0 derived)
      :
      Path – 12 rows (0 seeded, 12 derived)
      :
    judgements (3):
      denial violated: ax/no/loop at [a: hub]
      denial violated: ax/no/loop at [a: east]
      denial violated: ax/no/loop at [a: port]
    $ echo $?
    1

```

Read what did *not* happen. The write was not rejected: the facts landed, and the chase saturated over them (twelve Path rows now: each of the three places on the loop reaches all four, itself included). The violation was not repaired: no engine deleted your hop or picked a row to sacrifice. What happened is a *judgment* — the very thing ax/no/loop’s report mark asks for: the violated denial is reported, with the axiom’s own name and the binding that witnesses it, and the exit code says so. Three judgments here, not one: the same loop, witnessed once from each place standing on it. If you have operated an event-sourced system, this is a familiar posture done formally: an append-only log can land in a state some invariant marks invalid, and the repair is a further ordinary commit — made by whoever owns the workflow, not by the database guessing. The Tutorial (chapter 4) makes this the centre of the story.

### 3.6 The theory is rows

One last look before you go:

```

    $ geolog rows examples/routes.geolog
    // Declaration rows in the GeologMeta idiom (the schema you wrote is
    // itself data). refs: L<n> = local element minted at op[n]; G<name> =
    // genesis identity. This is the declaration slice; axioms are rows
    // too, in the same substrate – the committed listings under
    // theories/generated/ carry complete worked examples.

```

```

op[ 0] elta sort=G<Theory>                // Theory "Routes"
op[ 1] elta sort=G<Sort>                  // Sort "Node"
op[ 2] funa fun=G<Sort/name> args=[L1] value=str:"Node"
op[ 3] funa fun=G<Sort/theory> args=[L1] value=L0
op[ 4] elta sort=G<Fun>                   // Fun "Edge"
op[ 5] funa fun=G<Fun/name> args=[L4] value=str:"Edge"
:
op[ 8] elta sort=G<Field>                 // Field "src" of Edge
op[ 9] funa fun=G<Field/fun> args=[L8] value=L4
:

```

The theory you wrote in §3.2 is itself data: a Theory element, a Sort element with a name row and a home row, Fun and Field elements for each relation and its columns — rows in the *meta-schema*, the schema language that is itself a geolog theory. As the listing's own header says, this is the declaration slice; axioms continue as rows in the same idiom, and the repository's `theories/generated/` directory carries complete committed listings — the full Petri-net theory runs to 1,239 ops. Schemas, extensions, and migrations live in the same substrate they govern, so you can version them, diff them, and derive facts about them just as you would any other rows.

### 3.7 Where to next

The Tutorial (chapter 4) works an event-sourced order flow with two writers: views only the merged logs can derive, two replicas' records identified by congruence rather than a conflict resolver, a witness the chase invents under an existential axiom, a violation that only the merge surfaces, and the signed commit log underneath, which can say things a broker cannot. The Reference Manual (chapter 5) gives the systematic account. The part closes with diagrams as data (chapter 6): one incidence genus, stored and checked as rows, for every diagram the wider toolkit traffics in. And for the road from a model to a checked temporal claim about it — the reason this database exists in this book — start the QTSL part at chapter 7.



# Tutorial

---

DRAFT

If you have operated an event-sourced system, you have already run most of geolog’s architecture on convention [139]. An append-only log serves as the source of truth. Derived views are rebuilt by replay. New capability arrives as new producers and consumers, never as mutation of shared state. And everybody promises not to write to the derived tables by hand. Kafka gives you the log, Flink or Feldera maintains the views, a schema registry polices the shapes, and the promises live in code review.

geolog takes that architecture and makes the promises structural. No fold code that somebody wrote and everyone else must believe maintains the derived views; the declared axioms *entail* them instead, and they can be recomputed and checked from the log alone. The chapter’s spine is one example, an order flow with two writers: views no single writer can derive (§4.1–4.2), merge as set union plus re-derivation (§4.3), the signed commit log and what it can say that a broker cannot (§4.4), schemas as data (§4.5), and the ground vocabulary (§4.6). A worked example imported from an independent project follows: the core theory of a proof assistant, expressed whole, together with instructions for converting its native data in both directions (§4.7). A plain-dealing scorecard against the SQL stack you already run (§4.8) and the handoff to temporal claims (§4.9) close the chapter. Every transcript is verbatim from a real run against the geolog-alpha repository (the order flow against `examples/orders.geolog` and the single-diagram crossing of §4.7, last re-executed 2026-07-30 against the import-era tree; the corpus-scale measurements behind §4.7 executed 2026-07-27); the receipts sit at this chapter’s source tail.

## 4.1 Views you do not maintain

Here is the domain, whole. Two writers will keep logs against it: a storefront (payments, cancellations) and a warehouse (allocations, shipping records).

```
theory Orders {

  Order      : Set;
  Shipment   : Set;

  // Events -- asserted rows, the writers' ground truth.
  Placed     : [o: Order] -> Prop;
  Paid       : [o: Order] -> Prop;
  Allocated  : [o: Order] -> Prop;
  Cancelled  : [o: Order] -> Prop;

  // Views -- derived rows, never asserted by a writer...
  Shippable  : [o: Order] -> Prop;

  // ...except ShippedVia, which writers MAY also assert (a recorded
  // manifest); where no record exists, the chase invents the witness.
  ShippedVia : [o: Order, s: Shipment] -> Prop;

  chase ax/ship/ready : forall o : Order.
    [o: o] Paid, [o: o] Allocated |- [o: o] Shippable;

  chase ax/ship/manifest : forall o : Order.
    [o: o] Shippable |- exists s : Shipment. [o: o, s: s] ShippedVia;
```

```

    chase ax/manifest/unique : forall o : Order, s1 : Shipment, s2 : Shipment.
      [o: o, s: s1] ShippedVia, [o: o, s: s2] ShippedVia |- s1 = s2;

    report ax/no/zombie : forall o : Order, s : Shipment.
      [o: o] Cancelled, [o: o, s: s] ShippedVia |- false;
  }

```

Read ax/ship/ready the way you would read a view definition, because that is what it is:

```

CREATE MATERIALIZED VIEW shippable AS
  SELECT o FROM paid INTERSECT SELECT o FROM allocated;

```

The chase mark plays the role of CREATE here. It opts the sequent into *firing*, which is to say into deriving rows. A report mark, or no mark at all (unmarked means report), would instead make the same sequent a checked constraint, judged and never fired. The join stays identical; what changes is who owes whom what. A materialized view comes out of the query you wrote together with the engine's maintenance code, and it contains whatever that code computed. A geolog view, by contrast, is a logical consequence. The chase computes the *least fixpoint* of the axioms over the asserted rows: every row the axioms force, and no row they do not. Anything that can replay the log and read the theory can then re-derive and audit every row, with its provenance. In the SQL world the nearest neighbours are Feldera and the DBSP line of work [31], and the kinship is genuine: declarative view definitions, incremental maintenance with a correctness story, replay from a log. What differs is the view language itself. A SQL view computes what its query says; a geolog view is an entailment in geometric logic [127, Part D1], the fragment of first-order logic whose sequents have a conjunction of atoms as premise and a disjunction of existentially quantified conjunctions as conclusion. That fragment is why the other three axioms above (an existential, an equality, and a denial) are ordinary view definitions here, while over there they are not expressible as views at all. Those three carry the rest of this chapter.

Two caveats, up front. The first is maintenance. Feldera's engine is incremental to its bones, and at production scale that maturity is real (§4.8). In this preview, geolog's chase evaluation is a deliberately naive reference implementation, differentially tested against an independent engine and built to be right rather than fast; incremental maintenance ships today only for its violation views, where the failing set is kept current under row deltas and never recomputed from scratch. The second is expressiveness, and it cuts the other way. Aggregates such as SUM and windowed rates, the bread of streaming SQL, are not native vocabulary in the geometric fragment. Counting enters through explicit constructions or through the ground seam (§4.6), and where your computation is aggregate-shaped, today the SQL stack simply wins.

## 4.2 One writer at a time

The two writers' logs live as two instance blocks in the same file; in the durable pipeline each would instead be a stream of signed commits (§4.4). The storefront saw payments and a cancellation, and it holds the carrier's tracking record for order o1. The warehouse allocated stock for both orders, the cancellation not having reached it, and it holds its own record of the same shipment, the printed label:

```

instance Storefront : Orders {
  o1, o2 : Order;
  sf_record : Shipment;

  [o: o1] Placed;  [o: o1] Paid;
  [o: o2] Placed;  [o: o2] Paid;
  [o: o2] Cancelled;

  [o: o1, s: sf_record] ShippedVia;
}

instance Warehouse : Orders {
  o1, o2 : Order;
  wh_label : Shipment;

  [o: o1] Allocated;
}

```

```

    [o: o2] Allocated;

    [o: o1, s: wh_label] ShippedVia;
  }

```

Chase each log alone (`--instance` selects blocks):

```

$ geolog chase examples/orders.geolog --instance Storefront
:
Shippable - 0 rows (0 seeded, 0 derived)
ShippedVia - 1 row (1 seeded, 0 derived)
  [o: o1, s: sf_record] ShippedVia
fixpoint: reached, no judgements - the model is saturated.
$ echo $?
0

```

The warehouse's run is symmetric: no Paid rows, so again no Shippable, and exit 0. Notice what clean means here. *Each writer's log is locally consistent*. The storefront knows o2 is cancelled but sees no shipment for it, and the warehouse sees no cancellation at all. Every partial view is innocent. If the two logs were Kafka partitions consumed by separate services, each service would be reporting green.

### 4.3 Merge is set union plus re-derivation

Now merge. Merging replicas in geolog takes no protocol. The commit bag is a set, union is union, and the chase re-derives. Run without `--instance` and the file's blocks seed as their union. Element names shared across blocks denote the same element, the CLI's small-scale stand-in for the content-addressed identity of §4.4:

```

$ geolog chase examples/orders.geolog
sorts:
  Order - 2 elements (2 seeded)
  Shipment - 2 elements (2 seeded, 1 invented by the chase,
                      1 merged by congruence)
  merged: sf_record = wh_label
relations:
  :
  Shippable - 2 rows (0 seeded, 2 derived)
  + [o: o1] Shippable
  + [o: o2] Shippable
  ShippedVia - 2 rows (1 seeded, 1 derived)
    [o: o1, s: sf_record=wh_label] ShippedVia
  + [o: o2, s: ?Shipment.0] ShippedVia
judgements (1):
  denial violated: ax/no/zombie at [o: o2, s: ?Shipment.0]
$ echo $?
1

```

Four things happened, and they are four fifths of this database's distinctive behaviour, so take them one at a time.

**The union derived what neither log could.** Shippable was empty in both single-writer runs, and the merged run derives it for both orders, because `ax/ship/ready`'s premise spans the two logs. That much is the ordinary magic of any log-merging system; a Felter pipeline consuming both topics would derive the same join. What it did not require was a pipeline topology. The axiom was already the join, and *any* subset of commits chases to its own well-defined fixpoint.

**The two writers' records became one shipment.** The storefront's `sf_record` and the warehouse's `wh_label` are two elements naming, as it turns out, one real shipment. `ax/manifest/unique` says as much: a shipment is one shipment. The engine's congruence closure *identifies* the two elements, and the report prints the merged element as `sf_record=wh_label`. Contrast the CRDT discipline you may run on Kafka-adjacent state stores. There, convergence is bought by designing every value as a lattice and every merge

as a join, per type, by hand, in advance. Here identity is a *logical* matter, decided by the theory's equality axioms and computed by congruence, and it composes with everything else the axioms say. It is also a mode *choice*, made where the axiom is written. `ax/manifest/unique` is chase-marked deliberately, because identification is this axiom's job; report-marked, the same sequent would instead judge two distinct records as a violation, which is the right reading in a domain where identity is content and merging would invent facts. (When two elements merge, everything known about either transfers to the one survivor. Judging `ax/no/zombie` against the invented witness below already relies on that.)

**The chase invented a witness.** Order `o2` became Shippable, and `ax/ship/manifest` demands that a shipment exist for every shippable order. No writer recorded one, so the chase minted `?Shipment.0`, a fresh element about which nothing is known except what the axiom forces. This is the *restricted chase*: existentials invent a witness only where no witness already exists (order `o1`, which had records, got no invention). If you know database theory you recognize tuple-generating dependencies and the chase from the classical literature [62, 72]; if you know event-sourced systems, read it as the database itself filing the missing follow-up event, typed, provenance-tracked, and only when entailed.

**The violation surfaced only at the merge, and was reported rather than resolved.** Both logs were locally clean. Their union contains a cancelled order with a shipment, and `ax/no/zombie` forbids precisely that. The chase does not reject the merge, since there is nothing to reject; the union of two sets of commits is a set of commits. Nor does it pick a loser or roll anything back. It reports the judgment, with the axiom's name and the witnessing binding, and exits nonzero. That posture is the event-sourcing one, held formally: *invalid states are representable, detected, and repaired by further ordinary commits* — a compensating cancellation of the shipment, a voiding of the cancellation, whichever the workflow's owner chooses. What the database contributes is that “invalid” is a theorem rather than a code path someone remembered to write, and that the judgment itself is data an agent can act on. (Where an axiom's conclusion offers a genuine *choice* of repairs, a disjunctive head, the engine likewise picks nothing, and the judgment carries the menu. The engine calls these branch points; this example's judgments are denials, menus of length zero.)

If you have fought a Kafka consumer-group rebalance that replayed a partition into a half-updated store, or reconciled two services' databases after a network partition, this section is the pitch. Merge here is set union, and so it is *always* defined, order-free, and idempotent; and every consequence of the merged log, including its violations, can be re-derived by anyone from the log alone.

## 4.4 The log is the ground truth

Everything so far ran from a source file, because this is a tutorial. In geolog, “the facts” take durable form as a log of *commits*: content-addressed, ed25519-signed envelopes whose payloads carry just the kind of rows you have been reading, and whose causal dependencies are part of the signed content. Crossing from the file to that log is one verb:

```
$ geolog seal examples/orders.geolog --store orders-store --repro
repro-signed (public seed "geolog-seal-repro-v1" – reproducibility, not authority)
sealed examples/orders.geolog: 1 theory, 4 axioms, 197 revision ops
revision commit: b128db74d04f92bb0fdc5aec5a26f1d67b5504ba9289a7e888b487c1ee232aa4
C1 revision hash: c7de71b56edc728bc65daba559d3e3f1b0bf32bf2030689de4e1b69d64066f7f
seal commit:      e69f4611e367e91c8509731fc04f349ccf0b18beef0ealf87788ce6351505a59
store: genesis minted, comparisons minted, revision minted, seal minted
```

The theory's declarations compile to meta-schema rows (§4.5), the rows seal into a signed revision commit, and the store on disk now holds the envelope shapes the storage crates document and test (`geolog-envelope`, `geolog-store`, and `geolog-sync`, each with its differential partner named in its README). Sealing is deterministic: re-running the command reports every commit already present, and `--repro` signs with a public seed so that the hashes above reproduce from a fresh clone (reproducibility, as the tool says, not authority). `geolog check --store` and `geolog chase --store` then run the strata you already know over the sealed source instead of the file, after signature verification and a coherence gate; the reference manual's CLI section (§5.14) walks the whole verb family, `extract` and `load` included.

The architecture invites a comparison. A Kafka topic with infinite retention is also an append-only ground truth, and replay-rebuilt views are also disposable. Three differences are worth having words for.

**The log forms a set rather than a sequence.** A topic is an ordered sequence per partition, and that order carries real weight: consumers coordinate offsets, rebalances are delicate, cross-partition order is undefined. A geolog commit bag is unordered. Causality is explicit, since each commit names its dependencies by hash, and derived state does not depend on admission order. The repository’s battery includes a rebuild invariant that diffs incrementally-maintained state against a from-scratch replay, and a divergence is a release-blocking bug. “Replays in any order to the same fixpoint” is the property that made §4.3’s merge a non-event.

**Writers are cryptographic identities, and equivocation is detectable.** A broker will happily append whatever a producer sends, and if a compromised or buggy writer tells two consumers two different stories, nothing in the topic model even states the offence. geolog commits are signed and content-addressed, and a writer’s log is a hash-linked chain. A writer that forks its own history, which is to say equivocates, is detected *structurally* at sync time, from the bytes alone, with no consensus protocol and no quorum; the same mechanism makes CRDTs Byzantine fault tolerant in Kleppmann’s construction [138]. (The repository’s envelope crate ships the demonstration as a test: two signed commits claiming the same chain position are the offence, exhibited and caught.)

**Validity is per-commit and logical.** Because consequences and violations re-derive from any subset of the log, “which commits does this judgment implicate” is a well-posed question. The provenance machinery exists in the engine, so that every derived row and every invented element carries its firing history; and the engine’s validity layer (geolog-chase/validity in the repository) makes the per-commit verdict itself durable: it replays a commit’s inclusive causal closure into a content-addressed snapshot and folds one verdict per commit — clean, judged with its violations attached, or a named abstention (pending, unsupported, indeterminate), never a silent skip — with rebuild-order and cross-process determinism tested down to byte-identical snapshot bytes, so two replicas that judge the same commit publish the same judgment. Its first version replays a deliberately narrow fragment (one theory revision per closure, unmarked axioms only), and the tutorial CLI does not surface the pipeline yet. Capability-scoped authorization over the same substrate, meaning who may extend which theory and assert into which relation, is likewise arriving as an algebra over commits rather than as an ACL bolted on beside them (geolog-caps ships the algebra today; enforcement at the store boundary is design tense).

And the repository eats its own cooking: its committed theory goldens are themselves stored as signed artefacts, minted by a public, documented ceremony, and

```
| cargo run -q -p geolog-ceremony -- verify --root theories
```

re-derives them byte-identically from their committed listings and verifies the committed signatures, the golden-drift alarm that runs in every CI pass.

## 4.5 Schema is data

The Quick Start ended with geolog rows. The theory you write is itself rows: Theory, Sort, Fun, and Field elements in the *meta-schema*, the schema language that is a geolog theory like any other. In an event-sourced shop this should land as the resolution of a familiar irritation, the schema registry. Avro or protobuf schemas live *beside* the topics, in a registry with its own evolution rules (backward, forward, or full compatibility), its own versioning, and its own failure modes: a second database whose consistency with the first is everyone’s job and no one’s.

In geolog there is no second database. A theory arrives as a *revision*, a sealed batch of meta-schema rows, committed, signed, and content-addressed just like every other commit, with its dependencies named. Extending a schema means a further revision that depends on the first. Two independent extensions merge like any other commits, by set union, with the chase over the meta-schema re-deriving the combined signature. And “what did the schema look like when this row landed” is a query over the same log as the row. Annotations follow the same pattern instead of a special mechanism: the repository’s worked

units-of-measure theory decorates an existing theory's declarations with dimension gradings from the *outside*, as ordinary relation rows over Fun and Sort elements, without touching the annotated theory's bytes.

Here is what ships today in the alpha repository: the meta-schema itself, the genesis golden of §4.4's ceremony; the reference row listings and their generators for the worked theories under theories/; the theory-to-rows compiler and the sealed-revision conformance loader, with byte-identical round-trips on the committed corpora and loud rejections collected per rule; and the revision-level machinery, namely declaration indexes, signature hashes, and a programmatic revision-body builder. The `geolog` rows verb you ran is a faithful *view* of the declaration slice, and `geolog seal` runs the same compilation for keeps: the transcript of §4.4 sealed the order theory as a signed revision of 197 meta-schema operations. Cross-revision extends, parameterized theories, and instances as first-class elements are specified in the ruled design documents and arrive as that store grows; the tutorial's single-file extends, which the CLI does accept, is the in-file shadow of that mechanism.

## 4.6 Ground vocabulary: decided, not stored

Everything so far has been finite rows over declared sorts. Real domains want numbers, timestamps, and comparisons, vocabulary whose extension no log could store (*every* pair of rationals with  $a \leq b$ ?) and no chase should enumerate. `geolog`'s answer is the *ground seam*. Relations and functions over immutable carriers are declared as theory rows like everything else, but *decided* by registered procedures: exact arbitrary-precision rational arithmetic, and calendar arithmetic whose semantics are documented and tested against PostgreSQL's (`date + interval` behaviour, chosen precisely so a SQL practitioner's instincts transfer and the corner cases are somebody's published corner cases rather than ours).

Two disciplines make the seam trustworthy rather than a trapdoor. The first is *refusal over guessing*. Ground vocabulary an application declares but no provider implements does not default, approximate, or silently empty; a plan that cites it cannot even be expressed, and the refusal names the atom. (A division's graph simply contains no row for a zero divisor, so "divide by zero" is underivability rather than an exception.) The second is *the lane law*: an accelerated implementation may change *cost*, never *verdicts*. Every fast lane is conformance-gated against the always-on exact reference, and a lane that cannot pass its battery does not dispatch. If your mental model of database extension points is user-defined functions, hold on to this contrast: a UDF is trusted code in the query path, whereas a ground provider is an untrusted proposal bound to a declared signature, adjudicated by the engine, with an exact reference underneath it.

(The tutorial's CLI bridge does not yet reach the ground seam. `geolog chase` today covers declared sorts and Prop-valued relations, and it says so loudly when a theory wants more. The engine-level seam, the exact-rational floor, and the calendar corpus are in the repository under test; the worked Petri-net theory shows the discipline of doing real arithmetic *without* the seam, as chase-derived rows over an explicit number ladder.)

## 4.7 Worked example: alifib in geolog

Everything in this chapter so far was designed to teach this chapter. This section imports a structure that was not. *alifib* [105] is an experimental programming language and interactive proof assistant founded on higher-dimensional rewriting, developed by Amar Hadzihasanovic together with Alex Kavvos within ARIA's Safeguarded AI programme; its mathematics is Hadzihasanovic's theory of directed complexes and molecules [104], and its data structures grew out of joint work with Diana Kessler [106]. In *alifib* a type is a directed cell complex; a value is a *diagram* drawn in that complex; and a computation on  $n$ -dimensional data is itself a diagram, one dimension up. Running a program builds a witness of the whole computation, a higher-dimensional term you can inspect, paste into larger diagrams, and reason about; the project calls the property *computational transparency*. Rewrite rules are ordinary generators one dimension up, so there is no separate sort of rules anywhere in the language, and an execution trace is a term like any other.

Two of this chapter's themes meet in it, and it forces a third; that is why it earns a section. First, schema as data (§4.5) at full stretch: the whole proof theory lands as one ordinary `geolog` theory, seven sorts, thirty-eight functions and relations, twenty-four axioms, with no extension to the meta-schema at all.



Second, decided rather than stored (§4.6): every judgment the alifib engine computes over its finite combinatorial data — whether a paste is legal, whether a boundary pair is round, whether an incidence table matches a realized shape — enters the theory as a side-condition relation whose evaluator is the engine itself, the ground-seam discipline pointed at a full external prover rather than at rational arithmetic. Third, the forced one: the geometric fragment offers no negation in premises, and a theory of cell complexes keeps wanting one sentence of that shape (“this cell has no coface”). The stored-complement idiom below carries that sentence without leaving the fragment.

The theory ships as `theories/Alifib.geolog` in the `geolog-alpha` repository, and its opening `import` header is the decided-versus-stored split of §4.6 written down: the file imports its eight engine-decided judgments from the `alifib` namespace (each import classifying that relation `side-condition` — decided by a bounded external procedure), and every relation it declares without importing is ordinary, the default. The check verb runs green on it:

```
$ geolog check theories/Alifib.geolog
[1/5] parse + normalize + lint: theories/Alifib.geolog @ 2e60dd6 (452 lines)
      1 theories; 24 axioms (0 moves; aux helpers: none)
[2/5] geometric linter: CLEAN
      :
[5/5] instance conformance: no instance blocks in this file
modes: 6 chase · 18 report (18 by the unmarked default) · 0 refuse
      :
CHECK: GREEN
$ echo $?
0
```

The excerpts below are the teaching spine, roughly a third of the theory; the axioms not shown (execution traces as first-class terms; roundness of each generator’s boundary pair, the shape condition on a single cell; the remaining three sequents of the boundary-parity family; uniqueness of a cell’s coordinate; homing of boundary rows and map images) repeat the three patterns you are about to see. The theory splits into two tiers, the syntax of assemblies and the incidence data of the shapes they realize, and the walk takes them in order. One caveat before it starts: the tutorial CLI’s `rows` and `chase` verbs do not reach this theory yet — its element-valued function symbols are engine IR the current chase bridge does not carry — and both refuse loudly, each naming its workaround in the refusal. The transcripts here therefore ride `check`, `census`, and the conversion tools, and the excerpts themselves are kept faithful by construction: concatenated in source order they form a well-formed theory that `geolog check` accepts (the receipt at this chapter’s source tail records that run).

#### 4.7.1 Paste trees as constructor rows

```
theory Alifib {

  AType : Set;    // a type: a directed complex, elaborated
  Gen   : Set;    // one generator (cell) of a type; a rewrite
                        // rule is a Gen one dimension up
  Diag  : Set;    // a diagram AS ASSEMBLY -- a binary paste tree
  AMap  : Set;    // a map between types (generator-to-diagram images)
  Cplx  : Set;    // a materialized oriented graded poset
  Cell  : Set;    // one cell of a Cplx
  // (the seventh sort, Trace, records executions: a witness diagram
  // one dimension up, with its endpoints)

  Gen/type : Gen -> AType;
  Gen/dim  : Gen -> Nat;
  Gen/src  : Gen -> Diag;  // PARTIAL: absent iff dim = 0 --
  Gen/tgt  : Gen -> Diag;  // a 0-cell has no boundaries

  dGen    : [g: Gen] -> Diag;                // a single cell
  dPaste  : [k: Nat, l: Diag, r: Diag] -> Diag; // l #k r
```

`alifib` builds diagrams by *pastings*:  $l \#_k r$  glues two diagrams along their shared  $k$ -dimensional boundary,

and the engine stores a diagram as its assembly history, a binary tree of pastes over single-cell leaves. The two constructors above mirror that representation without loss. Because `dPaste` is a function — one value per argument tuple, an equality the chase maintains by merging — two occurrences of the same tree denote one `Diag` element, and structurally equal assemblies are shared: the engine’s own pointer-sharing, recovered as identity semantics. What a `Diag` element is *not* is a diagram up to isomorphism: two trees can realize isomorphic diagrams and remain distinct elements, related by a declared side-condition (`alifib_iso`, in the full theory). That choice matches the engine, which stores assemblies and *computes* isomorphism, and it is the honest one for a database: identity of stored data stays structural, and the quotient stays a judgment.

Partiality needs no apparatus. A 0-cell has no boundaries, so `Gen/src` simply has no row for it; there is no nullable column and no sentinel: a boundary a generator lacks is a row nobody asserts, and the axioms only ever demand what presence owes. The same convention carries map holes below.

Everything the engine *decides* about these trees enters as side-condition relations, and the theory *demand*s the judgments where they are owed:

```
alifib_wf      : [d: Diag] -> Prop;    // realizes to a molecule
alifib_pastable : [k: Nat, l: Diag, r: Diag] -> Prop;

ax/wf/diag : forall d : Diag.
  |- [d: d] alifib_wf;

ax/wf/paste : forall k : Nat, l : Diag, r : Diag, d : Diag.
  d = [k: k, l: l, r: r] dPaste
  |- [k: k, l: l, r: r] alifib_pastable;
```

Read `ax/wf/paste` as a well-formedness debt: every asserted paste owes a pastability judgment, and the judgment is discharged by evaluation, not by chase derivation — each of these relations is imported at the top of the file, which classifies it side-condition, and the bounded external procedure behind them is the `alifib` engine. The refusal discipline of §4.6 carries over unchanged: absent the engine, a plan citing these atoms cannot be expressed, and nothing defaults to true.

One derived view shows the chase doing for proof theory what it did for order flow. Which generators — hence which rewrite rules — does a diagram use?

```
Diag/uses : [d: Diag, g: Gen] -> Prop;

chase ax/uses/gen : forall g : Gen, d : Diag.
  d = [g: g] dGen
  |- [d: d, g: g] Diag/uses;

chase ax/uses/paste_l : forall k : Nat, l : Diag, r : Diag, d : Diag, g : Gen.
  d = [k: k, l: l, r: r] dPaste, [d: l, g: g] Diag/uses
  |- [d: d, g: g] Diag/uses;
```

(Its mirror `ax/uses/paste_r` walks the right subtree.) This is the Shippable of the proof theory: a table nobody writes, entailed by structure, answering the question a proof library wants indexed — which rules does this proof depend on?

## 4.7.2 Incidence, and maximality without negation

A paste tree is a recipe. The shape it realizes is an *oriented graded poset*: cells graded by dimension, each cell covering its faces with one of two signs, input or output. Chapter 6 develops the S-complex vocabulary this book uses for such data and notes that oriented graded posets embed as its Boolean instance; here that instance appears as concrete rows. A materialized shape is a `Cplx`; its cells carry a grade and an index within the grade (the engine’s canonical, normalization-derived order), and the two signed covering relations land as two ordinary relations:

```
Cell/cplx : Cell -> Cplx;
Cell/dim  : Cell -> Nat;    // grade
Cell/idx  : Cell -> Nat;    // index within the grade
```

```

BdryIn  : [hi: Cell, lo: Cell] -> Prop;   // lo is an input face of hi
BdryOut : [hi: Cell, lo: Cell] -> Prop;   // ... an output face

ax/scx/sign_disjoint : forall hi : Cell, lo : Cell.
  [hi: hi, lo: lo] BdryIn, [hi: hi, lo: lo] BdryOut
  |- false;

ax/scx/d2_pp : forall z : Cell, y : Cell, x : Cell.
  [hi: z, lo: y] BdryOut, [hi: y, lo: x] BdryOut
  |- exists w : Cell. [hi: z, lo: w] BdryOut, [hi: w, lo: x] BdryIn
  or exists v : Cell. [hi: z, lo: v] BdryIn, [hi: v, lo: x] BdryOut;

```

ax/scx/sign\_disjoint is a denial in the style of ax/no/zombie: one sign per covering edge. The second axiom is new territory. It is one quarter of the boundary-parity law, the sign-aware face of  $\partial \circ \partial = 0$ : every two-step signed path admits a two-step path of the opposite parity class between the same endpoints. Its conclusion holds a genuine disjunction, so the axiom is not chase-eligible — there is no least fixpoint to compute when the theory offers a choice — and the engine treats it exactly as §4.3 treated repairs: as a judgment carrying a menu. Back there the menus had length zero; this one has length two, and it earns its keep as a checked structural law rather than a derivation rule (the reference manual’s account of per-axiom modes is §5.7.2). The law is not hypothetical housekeeping: exporting all twenty-two example files alifb ships, with included modules and every generator and named diagram materialized, yields 37,988 complexes, and sign-disjointness and the parity law hold across them with zero exceptions (receipt at the chapter tail).

Now the idiom this section promised. “ $x$  is maximal” means “ $x$  has no coface”, and that sentence wants negation in a premise, which the geometric fragment deliberately lacks. Store the complement instead:

```

Cell/top      : Cell -> Prop;   // asserted: maximal (no coface)
Cell/covered  : Cell -> Prop;   // derived: has a coface

chase ax/cell/covered_in : forall hi : Cell, lo : Cell.
  [hi: hi, lo: lo] BdryIn
  |- lo Cell/covered;

chase ax/cell/covered_out : forall hi : Cell, lo : Cell.
  [hi: hi, lo: lo] BdryOut
  |- lo Cell/covered;

ax/cell/top_not_covered : forall c : Cell.
  c Cell/top, c Cell/covered
  |- false;

ax/cell/top_or_covered : forall c : Cell.
  |- c Cell/top or c Cell/covered;

```

Three moves. *Generation*: Cell/covered is an ordinary monotone view, derived from the boundary rows. *Exclusivity*: a denial keeps the asserted mark and the derived view disjoint. *Coverage*: a disjunctive law says every cell falls on one side or the other — judged, like the parity law, never fired. Between them, “maximal” becomes queryable vocabulary, its complement kept disjoint by the denial and exhaustive by the disjunction, and no premise ever had to say *not*. The pattern is general; alifb merely needs it constantly, because boundaries of pasting diagrams are defined through maximality.

### 4.7.3 Maps, holes, and the axiom that must not fire

An alifb map assigns, to each generator of a domain type, a diagram in a codomain type; alifb’s proof workflow builds such maps interactively, leaving *holes* — unassigned generators — to be filled as the proof progresses. In rows, a hole costs nothing:

```

AMap/src : AMap -> AType;
AMap/img  : [g: Gen, m: AMap] -> Diag;   // PARTIAL: a hole is an
                                           // absent row
AMap/total : AMap -> Prop;                // asserted mark

```

```

ax/map/total : forall m : AMap, s : AType, g : Gen.
  m AMap/total, s = m AMap/src, s = g Gen/type
  |- exists i : Diag. i = [g: g, m: m] AMap/img;
}

```

Filling a hole is an ordinary append — one new `AMap/img` row — so an interactive proof session is event-sourced by construction, and “what did this map look like before the fill” is a replay question like any other. The axiom is the interesting resident. `ax/map/total` is chase-eligible on its face: single disjunct, an existential the restricted chase knows how to witness. Firing it would be a disaster in slow motion. The chase would mint a fresh, contentless `Diag` for every open hole of every map marked `total`, “satisfying” totality by inventing images that mean nothing — when what totality means for user data is a *judgment* that the user’s own fills eventually discharge. So the axiom stays report-marked, a checked debt rather than a generator (one of the eighteen reports in the check transcript above), and the invented-witness power of §4.3 stays where it belongs: on axioms whose existentials denote obligations the database itself owns, like `ax/ship/manifest`’s missing shipment. Choosing per axiom, in the source, with the default on the safe side, is what the mode marks are *for*.

#### 4.7.4 One diagram, end to end

Here is a complete `alifib` type, small enough to watch cross the seam whole. Two composable arrows, a third arrow, and one 2-cell rewriting the pasted path into it (juxtaposition is `alifib`’s shorthand for pasting along the shared boundary of highest shared dimension; here `f g` abbreviates `f #0 g`):

```

@Type
Compose <=< {
  x, y, z,
  f: x -> y,
  g: y -> z,
  h: x -> z,
  step: f g -> h
}

```

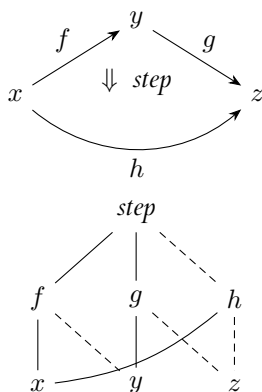


Figure 4.1: The 2-cell `step` of the `Compose` type, twice over. Left, the pasting diagram: `step` rewrites the pasted path `f #0 g` into `h`. Right, the shape it realizes, an oriented graded poset drawn as a Hasse diagram: solid covering edges are input faces ( $\partial^-$ , the `BdryIn` rows of §4.7.2), dashed ones output faces ( $\partial^+$ , `BdryOut`). Seven cells in grades of sizes (3, 3, 1), nine signed edges — row for row, the complex the transcripts below carry.

Save the listing as `compose.ali` at the repository root. The export bridge (built once with `cargo build --release` in `vendor/alifib`; §4.7.5 gives it a proper tour) elaborates the file with the engine’s own loader and interpreter — boundaries checked, shapes realized — and `--check` stops there, printing a summary. The first line is the file’s module root, a type holding nothing here; the one map on the second line is the identity self-map the engine mints for every type, which §4.7.5 returns to:

```

$ vendor/alifib/target/release/alifib-export compose.ali --check

```

```
compose.ali: type - 0 generators, 0 diagrams, 0 maps (0 holes), 0 complexes
compose.ali: type Compose - 7 generators, 0 diagrams, 1 maps (0 holes),
                        7 complexes
warnings: 0
```

Ill-formed input stops at this gate with the engine's own diagnostics: declare oops: `f f -> f` (pasting `f` onto itself along a boundary the types do not share) and the export exits nonzero with “Failed to paste diagrams: boundaries do not match”, pointing at the offending span. Nothing unchecked reaches the rows.

Dropping `--check` emits the interchange document (§4.7.5 specifies it):

```
$ vendor/alifib/target/release/alifib-export compose.ali -o compose.json
wrote compose.json (7616 bytes)
```

Two slices of it, whitespace refolded to the book's measure and content bytes as emitted — the step generator, whose source is the paste tree itself:

```
{ "name": "step",
  "dim": 2,
  "src": { "paste": 0, "l": { "gen": "f" }, "r": { "gen": "g" } },
  "tgt": { "gen": "h" } }
```

and the materialized shape of step, the right-hand panel of Figure 4.1 as data (grades bottom-up; `faces_in[d-1][p]` lists the input faces, at grade  $d-1$ , of the  $p$ -th cell at grade  $d$ , as indices into the canonical order):

```
{ "of": { "kind": "generator", "name": "step" },
  "sizes": [3, 3, 1],
  "faces_in": [ [[0], [1], [0]], [[0, 1]] ],
  "faces_out": [ [[1], [2], [2]], [[2]] ],
  "labels": [ ["x", "y", "z"], ["f", "g", "h"], ["step"] ] }
```

The import half turns the document into instance rows against the Alifib theory. Watch the paste tree become constructor rows, with hash-consing visible (`f` and `g` each occur once as leaves, then the paste refers back to them):

```
$ python3 theories/tools/alifib_interop.py import compose.json \
  -o compose.rows.txt
wrote compose.rows.txt (221 lines)
```

Operations 44 through 50 of that listing:

```
op[ 44] elta sort=G<alifib/Diag>                               // Diag: f
op[ 45] funa fun=G<alifib/dGen> args=[L16] value=L44          // dGen f
op[ 46] elta sort=G<alifib/Diag>                               // Diag: g
op[ 47] funa fun=G<alifib/dGen> args=[L20] value=L46          // dGen g
op[ 48] elta sort=G<alifib/Diag>                               // Diag: (f #0 g)
op[ 49] funa fun=G<alifib/dPaste> args=[nat:0, L44, L46] value=L48
op[ 50] funa fun=G<alifib/Gen/src> args=[L28] value=L48 // src of step
```

and, at the listing's tail, the last three of the figure's nine signed edges — the top cell's own — as boundary rows (L196 names the step cell; L181, L186, L191 the `f`, `g`, `h` cells):

```
op[207] rela rel=G<alifib/BdryIn> args=[L196, L181]
op[208] rela rel=G<alifib/BdryIn> args=[L196, L186]
op[209] rela rel=G<alifib/BdryOut> args=[L196, L191]
```

These are ordinary instance rows; engine instruments consume them as such. The census verb, for one, seeds them under the theory and analyses which function symbols' merges the derivation leans on (a dry-run instrument, it desugars the function symbols the chase bridge refuses — its first line counts them):

```
$ geolog census theories/Alifib.geolog --rows compose.rows.txt
merge-reliance census - 1 file(s), 1 rows listing(s), 24 arrow(s) desugared
:
seeds: 44 elements, 166 rows
```

Finally, back out. The projection inverts the import, and the `.ali` renderer turns the interchange back into source, with every paste parenthesized so the engine re-reads the same binary trees:

```
$ python3 theories/tools/alifib_interop.py project compose.rows.txt \
-o back.json
$ python3 theories/tools/alifib_interop.py to-ali back.json -o back.ali
$ sed -n 5,14p back.ali
@Type
Compose <=<= {
  x,
  y,
  z,
  f: x -> y,
  g: y -> z,
  h: x -> z,
  step: (f #0 g) -> h
}
$ python3 theories/tools/alifib_interop.py accept compose.ali
ACCEPT: CLEAN
```

The `accept` verb at the end is the round trip run as a test, both directions at once; the next subsection says what it checks.

#### 4.7.5 Converting data to and from alifib

This subsection is written for readers who keep data in alifib’s own formats — its maintainers first among them — and want the crossing stated precisely: where the bridge lives, what travels over it, the commands for each direction, what survives a round trip, and what is checked on the way in. The reward for crossing is everything this chapter has shown: types, diagrams, proofs, and their realized shapes as rows in a substrate with signed content-addressed commits, merge by set union, derived provenance views like `Diag/uses`, and a schema that is data in the same log.

**The bridge, and why it lives where it does.** alifib’s structural core (paste trees, oriented graded posets) is a private module, and the engine’s own output surfaces render for display: boundary expressions as strings, paste chains flattened, no incidence tables. A faithful exporter therefore has to live *inside* the engine crate, where crate-private data is reachable, and drive the interpreter itself; anything else would be a reimplementation, trusted precisely where it should not be. The geolog-alpha repository vendors a pinned copy of the engine carrying that exporter as one additive module (`vendor/alifib`; provenance, upstream commit, and the complete patch list are in `vendor/alifib/VENDOR.md`, and the patch is written to be upstreamable as-is). The bridge builds from a fresh clone:

```
$ cd vendor/alifib && cargo build --release
$ ./target/release/alifib-export --help
usage: alifib-export <file.ali> [--type NAME] [--all-modules]
      [--complexes none|gens|named|all] [-o OUT] [--check]
```

By default the root file’s module is exported with every named type, and shapes are materialized for every generator and named diagram; `--complexes none` keeps the export purely syntactic, `--type` restricts to one type, `--all-modules` includes what the file includes. Output is deterministic — two runs are byte-identical — because every collection rides a canonical order: generators ascending by (dimension, name), which is also a dependency-safe declaration order, since a generator’s boundary leaves are strictly lower-dimensional; diagrams and maps by name; cells in the engine’s own normalization order.

**The interchange document.** One JSON document per export, versioned by its first key. In sketch (field names exact, values elided):

```
{ "alifib_interop": 1,
  "generator": "alifib-export (geolog interop bridge)",
  "source": "<root file basename>",
  "modules": [
    { "path": "<path relative to the root file>",
```



```

    "types": [
      { "name": "...",
        "generators": [ { "name", "dim", "src?", "tgt"? } ],
        "diagrams": [ { "name", "dim", "tree" } ],
        "maps": [ { "name", "domain", "images", "holes" } ],
        "complexes": [ { "of", "sizes",
                        "faces_in", "faces_out", "labels" } ] ] ] },
    "warnings": [ "..." ] }

```

A generator's `src/tgt` are present iff its dimension is positive. `diagrams` carries the type's named diagrams: `let` bindings and proofs stored from interactive sessions. A map's domain is either `{"type": t}` or `{"module": m}`; its `images` are filled assignments, its `holes` the open ones, each hole optionally carrying a conditional image and metavariable boundary trees. A complex records whose shape it is (`of`), its per-grade extents (`sizes`), both signed face lists indexed as in §4.7.4, and per-cell generator labels. Boundary terms and diagram trees share one grammar, four node shapes in total:

|                                             |                                                                                                         |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>{"gen": "f"}</code>                   | a single-cell diagram                                                                                   |
| <code>{"paste": k, "l": t1, "r": t2}</code> | <code>t1 #k t2 --</code> the engine's own binary node, never flattened or re-associated                 |
| <code>{"hole": "g"}</code>                  | a map hole's metavariable (appears only in hole boundary terms)                                         |
| <code>{"labels": [...], "ldim": n}</code>   | loud fallback for a diagram with no paste history at its top dimension; counted in warnings, not a term |

**Into geolog.** `import` compiles a dump to instance rows; `--type` restricts, as with the exporter:

```
$ python3 theories/tools/alifib_interop.py import mine.json -o mine.rows.txt
```

The output is a plain-text listing of numbered operations in the meta-schema's instance vocabulary, three kinds in all: `elta` mints an element of a sort, `funa` asserts a function row, `rela` a relation row. Arguments are literals (`nat:0`, `str:"step"`) or backreferences (`L48` names the element minted by operation 48; forward references are ill-formed by construction). The `G<alifib/...>` tokens cite the Alifib theory's declaration identities, so the listing is meaningful against any store holding that revision. Each distinct paste tree becomes one `Diag` element per type, however often it recurs — the hash-consing of §4.7.1, performed at import time. The importer runs a structural self-check before writing (every reference points backward and at an element; every external token lies within the Alifib signature) and stamps the verdict in the header; the committed listings under `theories/generated/` are regeneration-checked against their generators by the repository's check script, so drift between tool and artifact is loud.

What admission means, in layers: the importer's self-check is the shallow layer; the theory's demand axioms are the deep one. Every imported element owes the judgments of §4.7.1 — well-formedness, pastability, roundness — and those side-conditions are discharged by the engine, which is to say: by re-elaboration. Running `alifib-export mine.ali --check` is the decision procedure, executed on the way out of alifib, before the rows exist; the incidence laws (sign-disjointness and parity) are additionally checkable without the engine, and `alifib_interop.py verify-scx mine.json` does so directly on a dump. Engine instruments consume the rows today (`geolog` check for theory-side conformance, `geolog census` for merge-reliance analysis, the meta-rows loader in the repository's test suite for loader conformance), and the one-verb facade of §4.4 seals theory *sources*; an admission seam that seals imported instance rows like these behind the same store is the piece still in design tense, and this paragraph sheds the qualifier when it lands.

**Out of geolog.** Two verbs invert the import:

```
$ python3 theories/tools/alifib_interop.py project mine.rows.txt -o back.json
$ python3 theories/tools/alifib_interop.py to-ali back.json -o back.ali
```

`project` rebuilds an interchange document from rows. It is deterministic, total on the import's image, and loud off it: unknown functions or sorts, dangling references, a `Diag` with no constructor row, a map marked total with absent images (inside the database that state is a judged, standing debt; the exporter never emits it, so the projection holds it off the image), an index at or beyond its grade's extent, a count mismatch

between declared extents and cells, non-adjacent boundary rows — each is a named rejection, not a silent drop. `to-ali` renders an interchange document as `.ali` source: every paste fully parenthesized, so the engine re-reads the same binary trees; identifiers outside `alifib`'s lexical grammar sanitized deterministically, with the name map recorded in the header comment; maps emitted as module-level `let total` declarations; and the identity self-map the engine generates for every type is skipped, since elaboration regenerates it. The acceptance judge for this direction is `alifib` itself: the round-trip witness re-elaborates the rendered source with the engine's own parser and loader and compares the re-export against the original document.

**What survives, what degrades.** Surviving a full round trip, per type, structurally intact: generator names, dimensions, and boundary trees as exact binary pastes (no re-association); named diagrams and their assemblies; map images and holes, as assignments over the domain's generators; complexes cell for cell — extents, both signed face lists, labels. Degrading, each by design and each documented in the tools' headers: map *names* are not carried by rows, so maps re-emerge under fresh names; a conditional image (one recorded on a hole but awaiting boundary resolution) degrades to a plain hole, and the engine re-derives the constraint on re-elaboration; maps whose domain is a whole module are skipped with a note, the module system being surface elaboration that the interpreter flattens before the proof theory begins; display metadata (index blocks) does not travel; execution traces sit outside interchange version 1 — the Trace sort of §4.7.1 is the landing site reserved for them.

**The acceptance harness.** Both witnesses run as one verb per file (`accept`, as in §4.7.4), and over a fixed ten-file corpus spanning the shipped examples' feature surface — abstract machines, higher-categorical developments, hole workflows, simplicial complexes; the parity measurement of §4.7.2 separately covered all twenty-two files:

```
$ python3 theories/tools/alifib_interop.py accept-corpus
BinaryNat.ali: CLEAN
EckmannHilton.ali: CLEAN
:
Hole_examples.ali: CLEAN
LambdaSigma.ali: CLEAN
CORPUS: 10/10 clean
```

Witness one: export, import, project; the projection must reproduce the interchange up to the documented degradations. Witness two: export, render to `.ali`, re-elaborate with the engine, re-export; the second document must match the first up to the recorded name map — boundary trees, hole sets, and complexes cell for cell. Every round-trip claim this subsection has made is one of those comparisons or one of the named rejections. Readers extending the interchange — traces are the obvious next passenger — extend the witnesses first; the harness is the contract.

## 4.8 The scorecard

Where the SQL stack wins today, plainly: **Throughput and maturity.** Kafka moves gigabytes per second and Feldera maintains complex view cascades incrementally, while this preview's chase is a correct, differentially-tested reference implementation whose incrementality so far covers violation views only. **Aggregates and windows.** Streaming SQL's core vocabulary, not native to the geometric fragment (explicit constructions or the ground seam, at tutorial scale). **Operations.** The surrounding decade of connectors, monitoring, and hiring.

Where geolog wins, plainly: **Views you can audit.** Derived state is entailment, re-derivable and checkable from the log by a third party, with provenance; there is no fold code to believe. **Merge without a protocol.** Set union plus re-derivation, order-free, defined for every pair of replicas, with no per-type merge design and no rebalance to choreograph. **Judgments over exceptions.** Invalid states and pending choices are typed, reportable data with the violated axiom attached, rather than code paths. **Writers you can hold to account.** Signatures, content-addressing, and structural equivocation detection are the data model, not an add-on. **One substrate.** Schema, extension, annotation, and migration are rows in the same log they govern; there is no registry to drift.

Compose the two lists and the position comes out plain. At this preview, geolog has the right shape for

systems of record where auditability, principled merge, and schema-as-data carry the value, at volumes where a reference-grade engine suffices. It is also built deliberately so that the fast paths can arrive without the trust story changing (the lane law; the kernels repository).

## 4.9 Composing systems, and where QTSL begins

The order flow of this chapter is one system. The book's larger programme composes systems, with shared boundaries, wiring diagrams, and the population and fusion of components, and it states *temporal* claims about the composites. Those claims are not of the form “no cancelled order has shipped”; that is a state property, and you watched a denial say it. They look instead like “every placed order is shipped or refunded within thirty days, with probability at least  $p$ .” Those are claims the chase alone cannot check and the event log alone cannot even state. Stating them is QTSL's job, and certifying them against models is the rest of this book: the systems vocabulary in chapter 12, the composition patterns in chapter 13, and the zero-to-certificate path in chapter 7. You now hold the substrate they all stand on: a log whose truth is structural, tables that are theorems, and a schema that is data.

# Reference Manual

DRAFT

This chapter is the systematic account of `geolog` as shipped in the `geolog-alpha` repository: the source language and its grammar, the normalization pipeline and its registry, the elaboration fragment the `geolog` verb carries, the chase and its judgments, the meta-schema, and the substrate contracts — commits, storage, sync, the ground seam, and capabilities. The Quick Start (chapter 3) and the Tutorial (chapter 4) teach by example; this chapter specifies. It restates none of their walkthroughs and cites them where an example earns more than a rule would.

Three reading conventions govern everything below.

**Claim discipline.** Normative present tense describes behaviour shipped and tested in the repository as of this edition's receipt passes (2026-07-21 through 2026-07-31, recorded at this chapter's source tail). Designed-but-not-shipped behaviour appears only in clauses explicitly marked *design tense*, each naming where the design lives today. Every quoted error message and every transcript fragment in this chapter is the verbatim output of a real run against the committed tree (receipts in this chapter's source tail).

**Authority order.** The `geolog` binary is ergonomics and composition over constituent crates, and its own crate documentation states the rule this manual inherits: where a behaviour differs between the facade and a constituent, the constituent is right and the facade has a bug. Accordingly, each section below names its authority — the crate (or committed theory file) whose documented, tested behaviour it restates. Where this manual and that source disagree, the source wins and the disagreement is a defect in this manual.

**Scope.** This manual covers the public `geolog-alpha` tree. The engine library supports strictly more than the command-line fragment (§5.8); the manual covers both, and says which surface each statement binds.

## 5.1 Lexical structure

Authority: `crates/geolog-normalize/src/lex.rs`.

Theory text and instance blocks share one tokenizer. Input is processed line by line; a `//` begins a comment (outside a string literal) that extends to the end of the line and is discarded. Tokens, longest match first:

```
two-char    ::= ">" | "<" | "<=" | ">=" | "!="
single-char ::= "<" | ">" | "=" | "+" | "*" | "[" | "]" | "(" | ")"
            | "," | ":" | "." | "{" | "}" | ";"
integer     ::= digit+
string-lit  ::= "'" (any scalar except "'" or newline)* "'"
identifier  ::= starter continuer*
```

The identifier alphabet is Unicode-open, and the operator characters are a closed ASCII set, so the two are disjoint by alphabet rather than by any gating rule. A *starter* is an ASCII letter, `_`, a non-ASCII alphabetic scalar (Greek letters,  $\ell$ , ...), or one of the math-letterlike symbols  $\partial$  and  $\nabla$ ; a *continuer* is any starter, an ASCII digit, `_`, `/` (so `ax/path/edge` and `Marking/count` are single tokens), or a script extender — the superscript and subscript digits and signs, and the primes. Script extenders continue tokens and never start them:  $\partial^+$  is one token, and a detached superscript at token start is a tokenization refusal, so  $\partial^+$  (with a space between) cannot silently mean anything. String literals are single-line and carry no escapes; the grammar admits them in exactly one position — contraction specs in law bodies (§5.15) — and every other

position refuses them downstream. There are no negative-number literals and no block comments in the theory grammar. Anything else stops the tokenizer with a hard error naming the offending text.

The reserved words of the theory grammar are:

```
| forall exists not iff or false theory extends
| Set Prop Index Fin DynTensor List
```

`Fin` and `Index` name the same bounded-index type (`Fin` is the ratified spelling, `Index` the legacy alias — §5.3). `Tensor` is not reserved, and no longer names a surface type at all: the shape-in-type tensor former lives in the meta-schema tier (§5.9.2), and the surface tensor type is the keyword `DynTensor` (§5.3). `instance` is likewise not a keyword of the theory grammar: an instance *block* is recognized by the file splitter (§5.2), one layer up, and the same word doubles as an optional postfix marker on model-of types (§5.3), recognized contextually. The mode marks *chase*, *report*, and *refuse* (§5.7.2) are contextual in the same way — recognized only as a statement’s leading token: *chase* : ...; is still an ordinary declaration named *chase*.

## 5.2 Source files

*Authority:* `crates/geolog/src/input.rs`.

A `.geolog` file carries two strata: *theory text* and *instance blocks*. Before parsing, the file is split by a deliberately minimal scanner that tracks `//` comments and brace depth, and recognizes the word *instance* only at brace depth zero, at an identifier boundary on both sides. Each instance `... { ... }` span is extracted whole (an unclosed block is a hard error); the remaining text — with each extracted span replaced by whitespace of identical line structure, so diagnostics keep their line numbers — is handed to the normalization pipeline verbatim. Consequences worth stating: the word *instance* inside a comment or inside any braced body is ordinary text; theory declarations and instance blocks may interleave freely at top level; and the theory grammar remains entirely the normalizer’s business — the splitter never interprets theory text.

## 5.3 Theories and declarations

*Authority:* `crates/geolog-normalize/src/parse.rs` (*grammar*) and `crates/geolog-normalize/src/ast.rs` (*abstract syntax*).

Theory text is a sequence of theory declarations:

```
| theory-decl ::= "theory" NAME param* ["extends" NAME] "{" statement* "}"
| param      ::= "(" NAME ":" ("Set" | NAME) ")"
| statement  ::= [mode] NAME ("," NAME)* ":" statement-body ";"
| mode       ::= "chase" | "report" | "refuse"
```

A theory header may bind parameters, name first: a (`X` : `Set`) group binds a sort parameter, and a (`N` : `PetriNet`) group — whose second name must be a theory declared earlier in the file — binds a *model-of* parameter, a model of that theory. (An earlier binders-before-name header order parses as a deprecated spelling; see `--strict`, §5.14.) Every statement is an optional mode mark, one or more comma-separated names sharing one signature, a colon, a body, and a semicolon. The statement’s *kind* is dispatched on its name: **a statement whose name begins with `ax/` is an axiom** (§5.4); a statement opened by the contextual keyword `law` is a law member of the aggregation surface, whose grammar this edition defers (§5.15); every other statement is a declaration. The prefix does grammatical work rather than expressing a naming convention: a sequent-shaped body under a name without the `ax/` prefix is parsed as a declaration and fails (verified: `my/rule : forall ... stops with syntax error: unparseable type at line 4: forall ...`, the cited line being the statement’s own). The Quick Start’s remark that slash-names are ordinary identifiers remains true — the slash is an identifier character (§5.1) — but the two-character prefix `ax/` is what makes a statement an axiom. The mode mark decides whether an axiom fires or judges (§5.7.2); the AST carries the *authored* mark, so rendering preserves the source form, and a mode mark on a declaration head is a hard error naming both the mark and the head (the message names `ax/` and `law` as the two homes a mode can mark).

Declaration bodies:

```

sort-decl  ::= NAME ":" "Set" ";";
record-decl ::= NAME ":" "{" field ("," field)* "}" "->" codomain ";";
unary-decl ::= NAME ":" type "->" codomain ";";
curried-decl ::= NAME ":" "(" NAME ":" type ")" "->"+ codomain ";";
const-decl ::= NAME ":" type ";";
field      ::= NAME ":" type
codomain   ::= "Prop" | type
type       ::= NAME
              | term ("Fin" | "Index") -- a sort (or a theory: model-of)
              | NAME "DynTensor"      -- bounded index (n elements)
              | NAME "List"           -- dynamic tensor over an element sort
              | NAME "List"           -- finite sequences over an element sort
              | arg+ NAME ["instance"] -- model-of application

```

Braces name record domains. (The bracket spelling `[f: v]` of earlier corpora still parses by default while the committed corpus ports to braces; under strict parsing it is a deprecation error pointing at the brace form, and `geolog fmt` prints the canonical respelling — §5.14.) A curried signature of named binders normalizes to the record form at parse, so the two spellings declare the same relation or function; an unnamed curried binder is a hard error. A declaration with codomain `Prop` declares a *relation*; any other codomain declares an element-valued *function* (a `const-decl` is the nullary case: a constant of its sort). Relations come in two domain shapes at the surface: *record* (named fields; field order as written is preserved for display, but fields are name-keyed, never positional) and *unary* (one anonymous argument). There is deliberately no surface form for a zero-argument proposition: `Flag : Prop`; parses as a constant declaration of a (nonexistent) sort `Prop` and is refused downstream (§5.6). A proposition wants at least one argument, or it wants to be a fact.

Two type forms are new with the aggregation surface, and both are codomain vocabulary for the loader's stratum rather than the chase fragment's. `DynTensor` is the *dynamic* tensor type over an element sort — shape carried in the value, where the meta-tier `Tensor` former carries shape in the type (§5.9.2); tensor values are side-condition payload, never term-language citizens. A theory name in type position means “a model of that theory”: bare (`base : G`; under a `(G : Graph)` parameter), or applied to arguments (`initial_marking : n0 Marking`), with an optional postfix `instance` keyword as an explicitness marker — neither spelling is canonical, and the source printer preserves the author's choice. A third form belongs to the list former (§5.9.2): `Str List` is the finite-sequence type over an element sort, postfix like `Index` and `DynTensor` and taking one element sort only (`Str List List` is a syntax error), with list values consumed through the `len/at/concat` ground roster as side-condition payload — never term-language citizens. The surface element scope is the builtin ground sorts: a list codomain at a declared sort is refused by the normalizer (verified: `List element "A"` is not a builtin ground sort (`v1` surface scope)), and the wire tier's wider element scope, nesting included, arrives with its own surface spelling rather than by fall-through. The `v1` chase bridge refuses all three forms (§5.6).

`extends` names a parent theory. In the single-file pipeline the parent must be declared in the same file, and extension is realized by elaborating the whole file as one union (§5.6); in the committed-revision pipeline extension is a `TheoryExtends` row with its own conformance rules (§5.9).

**Builtin ground sorts.** The authoring idiom accepts a roster of ground sorts in declarations without a `Set` statement — `Q`, `Z`, `Nat`, `B`, `Str`, `ENNRat`, `Shape`, `Timestamp`, `Bytes`, `PeerId`, `CommitId`, and the aggregation surface's staged registry sort `Accum` (`crates/geolog-normalize/src/builtin.rs`) — together with the literal-dtype correspondence that types bare integer literals (`q/z/nat/...`), the builtin comparison and arithmetic vocabulary the grounding moves compile onto (§5.4.3), and the aggregation tier's staged arithmetic (`nat_mul`, `succ`). This roster is, `Accum` aside, the normalizer's *accepted* subset of the engine's primordial sort tier (§5.9.2) — `Accum` belongs to the deferred aggregation surface (§5.15), not the primordial tier; the command-line chase fragment is narrower still and refuses ground sorts at elaboration (§5.6).

## 5.4 Axioms and geometric normal form

*Authority:* `crates/geolog-normalize` (`parse.rs`, `normalize.rs`, `lint.rs`); the normal form is also stated, as *schema*, by the meta-schema (§5.9).



### 5.4.1 The axiom grammar

```

axiom      ::= [mode] AXNAME ":" ["forall" binders "."] [premise] "|-" conclusion ";"
binders    ::= NAME ":" type ("," NAME ":" type)*
premise    ::= item ("," item)*
item       ::= atom
            | "not" atom
            | "(" item ")"
            | "exists" binders "." premise          -- premise only
            | "forall" binders "." iff-group        -- pattern moves
conclusion  ::= "false"
            | disjunct ("or" disjunct)*
disjunct    ::= ["exists" binders "."] item ("," item)*
atom        ::= term [cmpop term]
cmpop       ::= "=" | "!=" | "<" | "<=" | ">" | ">="
term        ::= prod ("+" prod)*
prod        ::= postfix ("*" postfix)*
postfix     ::= primary NAME*                      -- application chain
primary     ::= "{" NAME ":" term ("," NAME ":" term)* "}" -- record
            | INTEGER                                -- literal
            | NAME                                    -- variable

```

Grammar facts with semantic weight: `or` binds loosest in a conclusion, and an existential's scope never spans an `or` — each disjunct carries its own `exists` prefix, so the surface conclusion is literally the normal form's shape (a disjunction of existentially quantified conjunctions). An `exists` group in a premise swallows the remaining premise items up to the turnstile. Terms are postfix application chains — `m p` Marking/count reads “Marking/count applied to (`p` applied to `m`)” — with `*` binding tighter than `+`, record literals in the same brace idiom declarations use (brackets remain the parsed legacy spelling during the corpus port, §5.3; on the aggregation surface square brackets instead delimit contraction index groups in law bodies, §5.15), and bare integer literals (typed later by the normalizer). `| - false` declares a *denial*: a rejection rule, the empty disjunction.

### 5.4.2 Geometric normal form

The engine consumes *geometric* theories, and the normal form is the central definition of this chapter:

**Definition 5.1** (geometric sequent). A geometric sequent is a universally quantified entailment whose premise is a finite conjunction of atoms and whose conclusion is a finite disjunction of existentially quantified finite conjunctions of atoms; the empty disjunction is  $\perp$  (a denial). An atom is an equation between terms or a relation applied to a term; the quantified variables are typed at declared sorts.

Geometric sequents are the sequent fragment of geometric logic [127, Part D1]; database theory knows the single-disjunct case as tuple- and equality-generating dependencies [72]. The authoring idiom of §5.4.1 is richer than this — `not`, `exists` in premises, `forall`-`iff` groups, comparison operators, arithmetic sugar, Index types. The normalizer (§5.4.3) compiles every departure down to Definition 5.1 by a registered move or rejects it loudly; the geometric linter then mechanically enforces the normal form on the normalizer's own output — premise a conjunction of atoms; conclusion a disjunction of existentially quantified conjunctions; no residual Index; no untyped literal; every relation declared *and* classified (§5.4.4); every atom well-sorted. Pipeline failures are collected and reported together, never truncated to the first.

### 5.4.3 The normalization moves

Each move is either total on its registered pattern or a loud error; nothing passes through half-rewritten. In source order of the pipeline:

- **Premise negation**, discharged by atom class. A negated *side-condition* atom needs no logic at all: side-conditions are decision procedures, and a decision procedure is negation-complete, its complement being the same evaluator with the verdict flipped. The discharge is therefore exact and in place —  $\neg S \wedge P \vdash C$  becomes *not*/ $S \wedge P \vdash C$ , with the complement relation registered automatically (move census: negated premise side-condition *S* discharges in place as *not*/*S*). A negated equation at a ground comparison sort has its complement in the builtin vocabulary already:  $\neg(a = b)$  at `Q` or

Nat becomes `q_ne` or `nat_ne`, in place. The one genuinely logical case is a negated *ordinary* atom, or an equation at a declared sort. Ordinary vocabulary is derivation-defined, so no procedure can decide its complement; instead the atom *crosses the turnstile* as a new leading conclusion disjunct.  $\neg R \wedge P \vdash C$  becomes  $P \vdash R \vee C$ : the classical-to-geometric compilation (move census: negated premise ordinary atom *R* crosses the turnstile). Read at the fixpoint, the crossed sequent says exactly what the classical sentence said: wherever *P* holds, *R* holds or *C* does. What it deliberately does not offer is negation-as-failure. The crossed head is a disjunction, so the axiom is no longer chase-eligible (§5.7.1), and a chase mark on it is a load refusal: a rule that fired on the *current* absence of a derivable row would be unsound against every later derivation, so no move is owed to that reading. Under report, the crossed disjunct is judged like any other menu (§5.7.6).

- **Pattern rewrites** (forall-iff groups). Extensionality, difference, and intersection shapes over a registered membership relation rewrite to auxiliary mask side-condition atoms; not on a membership atom in a conclusion rewrites to a registered negation-free witness atom. All four require registry entries (§5.4.4); an unregistered shape is a hard error, because auxiliary vocabulary is an obligation on downstream evaluators and is never minted silently.
- **Exists-hoist.** An exists group in a premise hoists to universal variables — sequent semantics:  $(\exists x. P) \vdash C \equiv \forall x. (P \vdash C)$ .
- **Index erasure.** Every `n` Index type erases to Nat plus a generated range obligation (a binder guard atom, or a generated `ax/range/*` axiom).
- **Grounding.** Comparison and arithmetic sugar compiles onto the builtin roster at ground sorts: `!=` to `q_ne/nat_ne`, `</=>` to `q_lt/q_le/nat_lt`, `+/>` to `q_add/q_mul`, with `>/>=` flipped by operand swap. There is no grounding at declared sorts: `a != b` over a declared sort *A* is a hard error (verified: no ground builtin for `!=` at sort *A* (the `!=@A` case would need a builtin-roster addition)). Disequality of declared-sort elements is not expressible in the fragment — deliberately: the chase’s equality discipline is congruence (§5.7.5), and a general disequality atom over invented elements is exactly the kind of vocabulary that would need an evaluator nobody has priced. (A negated premise *equation* at a declared sort is a different matter: it crosses the turnstile as an equality conclusion, per the premise-negation move above, so the classical constraint remains sayable even though a disequality atom does not exist.)
- **Literal typing.** Bare integer literals receive their dtype from context (`q:3/1`, `nat:2`, ...); an untypable literal fails the linter.

An axiom touched by any move is renamed with an `_n` suffix, so normalized output re-normalizes to itself with an empty move log. The pipeline’s guarantees, carried by its test suite: two-run determinism (byte-identical listings), idempotence (`normalize ; normalize = normalize`), linter-cleanliness of every output, ground meaning preservation (the grounding moves agree with direct evaluation of the source operators on random ground instances), and committed fixture goldens per move. `iff` at sequent top level (which would become two sequents) is unimplemented and errors loudly if it ever fires.

#### 5.4.4 Imports, classification, and roster totality

*Authority:* `crates/geolog-normalize/src/imports.rs` (the import surface), `crates/geolog-normalize/src/registry.rs` (the registry value both surfaces feed).

Every declared Prop-valued relation is classified, exactly one of:

- **ordinary** — derivation-defined: populated by assertion and by the chase, and by nothing else;
- **side-condition** — decided by a bounded external procedure (the ground seam, §5.12, or a registered auxiliary evaluator), never derivation-defined.

This is *roster totality*: every relation has a decided evaluator, and it holds *by construction* — the obligation is discharged by syntax instead of by hand. The classification lives in the source file itself. `ordinary` is the default, declared by writing nothing; a side-condition is declared by an `import` item naming its provider:

```
import std.{0, q_lt};
import <provider>.<relation>, ...;
```

The two namespaces do different work. An `import std` item authorizes *builtin ground vocabulary* used by name — sorts like `Q` and `Nat`, comparisons like `q_lt` — and a name outside the builtin roster refuses at once, listing the importable names. Any other namespace declares each named relation a side-condition awaiting that provider, bound at the ground seam (an unknown provider is not an error at check time: binding is deferred, and an unbound side-condition atom simply cannot be discharged — refusal over guessing, downstream). Import items are position-free top-level items, so multi-file closures concatenate; re-importing a name under the same namespace is idempotent, and importing one name under two namespaces refuses — one name, one provider. A relation or function declaration that reuses a builtin *vocabulary* name is legal only as an exact-signature shim of that builtin: same-named, byte-equivalent to what the engine itself would say. A builtin *sort's* name admits no shim at all — it is fully reserved, and a local sort takes another name (verified: `sort Q (declared in T): builtin ground sort names are reserved` — the builtin `Q` is ambient (`import std.{Q};` declares the usage), so a local sort must take another name). Both reservations are kind-local (the sort and vocabulary namespaces never meet), so a relation may take a builtin sort's name, and a declared sort may take a builtin relation's. Builtin comparison relations and the normalizer's own minted auxiliaries are side-conditions without import lines. Under `--strict`, builtin ground vocabulary used by name demands its `import std` header even in files that otherwise import nothing.

The *pattern registry* is the second classification surface, today a library-level one: it names, per rewritten relation, its *membership* reading (which field is subject, which is index), and the auxiliary relation each registered move targets (extensionality: 2 fields; difference and intersection: 3; negation-witness: the relation's own arity). Registry construction validates all of this — missing membership entries, unregistered auxiliaries, arity mismatches, and duplicate auxiliaries are errors before any theory text is touched. Pattern registrations have no source-level surface yet; the library's registry value carries them (the normalizer's fixtures/ include pattern-bearing specimens).

A line-oriented registry *file* (`--registry`) was the classification's original carrier; it is deprecated, retained for the migration era only. The one-authority law binds at the registry's content, and classification is the content it guards: a registry whose lines classify relations refuses to combine with a source that declares imports (verified: this source declares imports AND an external registry carries relation classification — classification has one home; ...), while a pattern-only registry file composes with an importing source — its aux names kept clear of the imported ones (verified: `aux relation ...` collides with an imported name — `aux mask vocabulary` is the normalizer's, never imported), a collision being classification smuggled back in aux costume. The line format also carries the pattern registrations of the paragraph above (their one textual carrier while patterns have no in-source spelling), and a pattern registration is no second classification authority.

## 5.5 Instance blocks

*Authority: crates/geolog/src/facts.rs (grammar), crates/geolog/src/seed.rs (semantics).*

```
instance-block ::= "instance" NAME ":" NAME "{" item* "}"
item           ::= elem-decl | record-fact | bare-fact
elem-decl      ::= NAME ("," NAME)* ":" NAME ";"
record-fact    ::= "{" NAME ":" NAME ("," NAME ":" NAME)* "}" NAME ";"
bare-fact      ::= NAME NAME ";"
```

Tokenization is the theory lexer's, so comments and identifiers mean the same thing in both strata. The header names the block and the theory it instantiates; the named theory must be declared in the same file. Items are element declarations and facts; a record fact's brace lists `field: element` pairs and then the relation; a bare fact (a `Marked`;) applies a unary-kind relation. Facts must be written in the relation's own shape — a record fact on a unary relation, or vice versa, is an error naming the mismatch. Braces are canonical here as in declarations: this grammar is what `geolog fmt` prints, and the formatter's output re-parses (verified: the respelled Quick Start file chases end-to-end, its report byte-identical to the bracket original's), while the bracket spelling `[f: a] Rel`; remains the accepted legacy form until the committed corpus ports to braces.

Semantics, in seeding order (source order — element numbering in transcripts is deterministic):

- **Elements are declared before use.** A fact referencing an undeclared name is an error, never a silently minted element (verified: ``P`` fact references ``ghost``, which is not a declared element – declare it first (``ghost` : A;`); elements are never minted silently).
- **Fields are exact.** A record fact must supply every declared field, no extras, no repeats, no defaults; each element’s sort must equal the field’s declared sort.
- **Union semantics across blocks.** An element name declared in two blocks at the same sort denotes one element; duplicate facts are idempotent. A name redeclared at a *different* sort is an error (verified: element ``a`` redeclared at sort ``C`` but already declared at sort ``A``). This is the merge story — set union of commits, then re-derivation (§5.11.1) — at authoring scale; the Tutorial’s two-writer example is built on it.

Surface syntax is a view: this block form is the command line’s authoring view of an instance, and the durable form is signed commits with content-addressed element identity (§5.10).

## 5.6 Elaboration: the v1 command-line fragment

*Authority: crates/geolog/src/elab.rs; the crate-level docs of crates/geolog/src/lib.rs state the fragment.*

The bridge from the normalized authoring idiom to the engine IR covers, in v1: **sorts**, **Prop-valued relations** (record or unary domains), and **geometric axioms whose atoms apply relations to variables**, equality conclusions included. Everything else the idiom can express is refused with an error naming the construct and, where one exists, the supported spelling of the same intent. The engine library supports more than this bridge; the bridge grows by deliberate widening, never by silent acceptance.

Elaboration takes the file’s theory declarations as *one single-file closure*: the union of every declaration in the file, with extends between in-file theories honoured by that same union. Consequences: an instance block may reference any declaration in the file’s closure regardless of which theory block carries it; duplicate sort or relation names anywhere in the closure are errors; and extends naming an out-of-file parent is refused (verified: theory ``Child`` extends ``Ghost``, which is not declared in this file – cross-revision extension rides the committed-revision pipeline (geolog-meta-rows / geolog-revision), not the v1 CLI bridge).

The operative refusals, each verified against the committed binary:

- **Element-valued functions** (any declaration with a non-Prop codomain, constants included):

```
function symbol `f` (codomain `C`): element-valued functions are engine
IR the v1 bridge does not carry – spell the function as a relation plus
a functionality axiom (two rows in the premise, an equality in the
conclusion), the idiom the committed theories use (e.g. `Marking/count`
+ `ax/count/functional` in theories/PetriNet.geolog)
```

The named idiom is normative: a function  $f : A \rightarrow B$  is spelled as a relation  $F[a : A, b : B]$  plus  $\forall a, b_1, b_2. F[a, b_1], F[a, b_2] \vdash b_1 = b_2$ , and the chase’s congruence closure does the rest (§5.7.5). `Flag : Prop`; lands in this refusal too (its “codomain” parses as the sort `Prop`); see §5.3 for why no zero-argument proposition form exists.

- **Ground sorts in declarations** — any field, argument, or binder sort not declared `Set` in the file, the builtin roster included:

```
relation `R` field `n` names sort `Nat`, which this file does not
declare – the v1 bridge elaborates single-file closures over declared
`Set` sorts only (ground sorts such as `Nat` arrive with the ground
seam; see crates/geolog-ground)
```

This is also where `Index` types land operationally: the normalizer erases `n Index` to `Nat` (§5.4.3), and the bridge then refuses `Nat` as above. The worked discipline for doing real arithmetic *without* the seam — counts as explicit elements on a ladder, chase-derived — is `theories/PetriNet.geolog`.

- **Tensor, list, and model-of types** — the codomain forms of §5.3 beyond plain sorts are loader vocabulary, and the bridge carries a dedicated refusal for each. A `DynTensor` anywhere in a declaration:

relation ``P`` field ``x`` uses a ``Tensor`` type — tensor payloads are side-condition territory, not v1 bridge vocabulary

A model-of member, on today's routes, is caught upstream by the element-valued-function refusal instead (its codomain is not `Prop`); the bridge's own instance-valued arm — refusing instance members as loader territory — stands behind it as defensive depth, per the posture above. A List codomain is caught upstream the same way; a List in a *field* position reaches the bridge's own list arm (verified: relation ``P`` field ``x`` uses a ``List`` type — list payloads are side-condition territory (the `{len, at, concat}` roster), not v1 bridge vocabulary).

- **Ground literals and term-level application in axiom atoms**, and nested records in argument positions: refused naming the axiom. Under the current pipeline these arms are defensive depth — declaration-level refusals and normalizer typing catch the practical routes first — and they stay, per the bridge's stated posture: a bridge that assumes its input is well-formed fails silently the day the assumption breaks.

Axiom-side checks (variables bound by `forall` or an enclosing `exists`; sorts agreeing with declarations; fields exact; no variable bound twice; no existential shadowing an outer variable) are belt-and-braces re-verification of what the normalizer already established, kept for the same reason.

## 5.7 The chase

*Authority: crates/geolog-chase — classify.rs, chase.rs, instance.rs, judgment.rs. The engine has no silent path: every outcome below is either a derived row, a typed refusal, or a reported judgment.*

### 5.7.1 Sequent classification

Repair arity is computed from syntax alone, by one total function, and every engine consults it — no runtime data participates. The conclusion is normalized to disjunctive head form (`Conj` distributes over `Disj`; a  $\perp$  conjunct annihilates its branch), and:

| head disjuncts | class       | chase posture                                    |
|----------------|-------------|--------------------------------------------------|
| 0              | denial      | never fires; violations are judgments            |
| 1              | canonical   | fires (invents witnesses) — the regular fragment |
| $\geq 2$       | repair menu | never auto-fired; surfaced as a branch point     |

One carve-out is permanent: a head that reaches an existential over an *immutable* sort (an infinitary disjunction — the engine IR's `InfDisj`) is never chase-eligible regardless of arity, because an infinitary disjunction is in no way a unary disjunction; no witness-invention strategy for it can ever be sound. The refusal is loud and names the axiom.

### 5.7.2 Modes: the author's word on firing

Whether an axiom fires is not inferred from its shape: it is *authored*, per axiom, by the mode mark of §5.3. The three marks:

- **chase** — a derivation rule: the axiom fires, inventing witnesses and asserting conclusions. Only a chase-eligible axiom (§5.7.1: canonical head, no `InfDisj`) may carry the mark; chase on an ineligible axiom is a load refusal naming the axiom by its *source* name (verified: axiom `ax/pick/one` is marked ``chase`` but is not chase-eligible ...; mark it ``report`` (judge) or ``refuse`` (gate), or reshape its head).
- **report** — a constraint: the axiom never fires; its violations and would-be firings surface as judgments (§5.7.6).
- **refuse** — a constraint that additionally gates local authoring: report's judgment semantics plus the local write gate of §5.8. On the file path there is no commit to gate, so its violations render exactly as report's.

**An unmarked axiom means report** — the least-dangerous default (chase creates rows and refuse blocks commits; report only judges), so derivation is always an explicit authorial act. Denials sit naturally under report: a denial never fires in any mode, and its violations are phase-2 judgments (§5.7.3) regardless.

The marks map onto the engine’s per-axiom trichotomy (§5.8): chase → Chased, report → Monitored, refuse → Enforced. geolog chase runs a file under this authored partition by default, recovering each axiom’s mark from the normalized declarations with three alignment checks per axiom (count, name, binder vector) so any drift between the normalizer’s and the bridge’s axiom order is a loud refusal, never a silent mode reassignment.

**The legacy flattener.** `--mode chased|monitored` (§5.14) survives for corpora not yet carrying explicit marks, applying one run-level policy to every axiom: under chased, every eligible axiom fires and a disjunctive head is a theory-load error before the loop ever runs (verified, on an unmarked file: theory load refused: axiom ax#0 is not Chased-eligible: repair arity is Menu(2); only a single-disjunct (canonical) head with no InfDisj binder may run under mode Chased – declare it Monitored instead, or split/simplify the axiom); under monitored, eligible axioms fire and disjunctive heads surface as branch points. On a file carrying *any* explicit mark the flag refuses, naming the marked axioms (verified: `--mode` conflicts with this file’s explicit surface marks: ax/path/edge (chase), ... – drop the flag to honor the marks): a mark is the author’s word, and a global flag that silently promoted a report-marked axiom into the firing set would be the exact hazard the marks exist to close — worst case, an equality conclusion merging the very elements it was written to police. Where the flag does run, the flattening is announced on stderr.

Engine-level load refusals name axioms positionally (ax#i): engine sequents carry no name field — names are metadata, held by the caller (or, in the durable pipeline, by meta-schema rows) — and the facade maps indices back to source names wherever it holds them (§5.7.7); the bare positional form above is the legacy flattener’s path showing the engine’s own words.

### 5.7.3 The two-phase run

A chase run has exactly two phases, and the separation exists for one reason: *a transient judgment can never be persisted as if final*.

**Phase 1** fires only canonical axioms, to a fixpoint or the iteration bound. The loop’s continuation condition is instance mutation only — discovering a violation is never a reason to keep (or stop) iterating. **Phase 2** is one evaluation pass of every denial and every menu axiom against the *final* canonical instance, bindings resolved through the final congruence. A menu whose missing witness is supplied by the very last canonical firing is therefore never recorded stale, and every judgment binding is a canonical representative of the final fixpoint.

Hitting the bound is not an error: the instance-so-far is returned, a non-termination judgment is recorded (with both the iterations run and the bound), and phase 2 still reports whatever holds of the partial instance — validity is explicitly indeterminate, not omitted (verified: no fixpoint within budget: 5 iterations (bound 5) – raise `--budget`, or the theory does not saturate on this instance (unbounded generation is a real outcome, reported rather than guessed at)).

### 5.7.4 Restricted semantics and invention

Canonical firing is the *restricted* chase [62]: for each binding of an axiom’s universal context that satisfies the premise, the engine first evaluates the conclusion as a query under that binding — **if any witness already exists, it does not fire**. Only a witnessless binding invents: one fresh element per existential binder, then the conclusion’s atoms are asserted (relation rows as derived rows; equations via congruence, §5.7.5). The Tutorial’s shipment example is the behaviour in one line: the order with a recorded manifest got no invention; the order without one got exactly one fresh ?Shipment.

Every element carries an immutable creation-provenance record, written once and never rewritten by any merge: seeded (EDB) elements carry caller-supplied provenance bytes (at command-line scale, the declared name; in the durable pipeline, a commit-addressed leaf), and chase-invented elements carry the firing axiom’s index, the context-bound elements of the firing (in context order), and the fresh binder’s



ordinal within the axiom. Derived rows are distinguished from asserted rows throughout (§5.7.7). One invariant worth its ink: for a premise whose own existential (hoisted per §5.4.3) admits several matches over the same context binding, the first firing’s witness satisfies the rest — so exactly one witness, with a context-only provenance term, ever exists per context binding.

### 5.7.5 Equality and congruence

Equality is computational, not annotational: the instance maintains a union-find congruence closure in the style of Nieuwenhuis and Oliveras [189]. An equational conclusion firing merges two elements’ classes; everything known about either — rows, provenance-bearing derivations, later firings — holds of the merged class. Tuples are stored raw and canonicalized through the current union-find at every read, so a merge never rewrites tables. The report renders a merged class by joining its seeded names (`sf_record=wh_label; b=c`) and accounts for it in the carrier arithmetic (3 seeded, 1 merged by congruence — verified transcript). The functionality idiom of §5.6 plus this machinery is how *geolog* has functions without function symbols in the v1 fragment. (Chase steps interleaved with congruence closure are also the computational core of datalog-with-equality and equality-saturation engines [20, 247], the nearest kin to this engine in the evaluation literature.)

### 5.7.6 Judgments

A judgment is one thing the chase found that a silent-success engine would have swallowed — the authority module’s own one-line definition of the type. The chase’s loud outputs, exhaustively, each carrying the axiom’s identity and, where applicable, the witnessing binding with canonical elements:

- **Denial violation** — a denial’s premise is satisfied at the fixpoint:  $\vdash \perp$  for this binding. Reported, never dropped, never repaired.
- **Branch point** — a menu axiom’s premise is satisfied with no existing witness for *any* disjunct. The payload carries the menu (the disjuncts themselves, as data); the engine picks nothing. Under the legacy run-level flattener a menu has length  $\geq 2$ ; menu-of-one branch points (a forced repair, surfaced rather than fired — a menu of 1 repair) arise wherever the per-axiom partition holds an *eligible* axiom to report (§5.7.2, §5.8) — an unmarked derivation rule’s would-be firings surface this way.
- **Non-termination** — the bound was hit (§5.7.3).

Judgments are results, not errors: a run that produces them completed, exits with the judgment exit code (§5.14), and its instance is real — the event-sourcing posture (invalid states representable, detected, repaired by further ordinary commits) held formally. The Tutorial’s merge chapter is the worked argument for why this posture, and not write-rejection or automatic repair, is the design.

### 5.7.7 The report

*Authority: crates/geolog/src/report.rs.*

The chase verb renders every carrier, every row, and every judgment, through the source file’s own names; nothing about a judgment is summarized away. The normative rendering facts, seven of them:

- each sort prints its canonical carrier size with the seeded count, the count invented by the chase, and the count merged by congruence, followed by a merged: line naming merged seeded-name groups;
- each relation prints in declaration order, rows sorted and deduplicated after canonicalization, with seeded/derived tallies;
- derived rows — rows not in the canonicalized image of the seeded baseline — are marked +;
- rows print in the relation’s own shape (record, bare, or bare name for nullary);
- chase-invented elements render as ?Sort.n in first-encounter order;
- side-condition relations are tagged [side-condition];



- judgments print the axiom's *source* name and the binding in context order, a branch point adds its menu size, and a clean run ends `fixpoint: reached`, no judgements — the model is saturated. (with a count of invented elements when there are any).

## 5.8 The engine beneath the command line

*Authority: crates/geolog-chase (mode.rs, weight.rs, view.rs, delta.rs, plan\_ir.rs, validity/), crates/geolog-weights. Everything in this section is shipped, tested library surface. The per-axiom mode partition is the one piece the geolog verb now reaches — through the surface marks of §5.7.2; the rest is not reachable from the verb yet.*

**Per-axiom modes, and Enforced.** The engine's mode system is a per-axiom assignment over the trichotomy **Chased** / **Monitored** / **Enforced**, with an eligibility-keyed ambient default (chase-eligible  $\Rightarrow$  Chased, everything else Monitored — reproducing the legacy run-level default, so that default is semantics-preserving) and validated explicit overrides: a Chased override on an ineligible axiom is a typed error; Monitored/Enforced overrides on *eligible* axioms are honoured by a firing partition — the axiom never fires, and its would-be firings surface as menu-of-one branch points (the forced repair, surfaced, never auto-fired). The surface marks lower onto this same machinery, as a *total* explicit assignment — every axiom's effective mark, unmarked resolved to report first — so on the file path the author's word, never eligibility, decides firing, and the ambient default decides nothing; the ambient default remains the semantics for library callers that override selectively. Enforced is Monitored plus the *local write gate*: an admission verdict over a finished chase result rejects any local commit whose resulting fixpoint carries an Enforced violation. The commit-vs-sync seam is the caller's, and its asymmetry is doctrine: the gate binds **local commit paths only** — states arriving by sync or merge are admitted with their judgments recorded, because gating sync admission on validity would cost true merge and history-as-authority (§5.11.1). Modes differ in *when* validity is enforced, never in what validity is. The adjudicator refuses to grade a mode assignment against a run that was not partitioned under that same assignment — the silent-vacuity class (an axiom fired in phase 1 repairing its violations before any judgment could exist) is rejected loudly, never admitted vacuously. The meta-schema carries the same trichotomy as rows (Axiom/monitored, Axiom/enforced, at most one per axiom; absence of both is the ambient default) — §5.9.

**Weights.** The engine is parameterized over an ordered-semiring provenance-weight algebra (crates/geolog-weights): a weighted run integrates each derived row's weight from its derivation structure, a reweight pass recomputes after tombstoning, and death-at-zero holds — the weighted-alive set after tombstone + reweight equals a from-scratch Boolean chase on the surviving facts (protested as a differential). The Boolean rung *is* the unweighted engine: `run_chase` stays pinned to  $W = \text{bool}$  and is byte-identical to the weight-blind engine.

**Deltas, maintained views, and the plan IR.** The delta bus makes the engine's unit of change data (element, row, and function deltas in batches). The view engine compiles violation views — the failing set of each denial, kept current under row deltas, never recomputed from scratch — and verifies itself against from-scratch reference evaluation (a divergence is a reportable defect, and the rebuild invariant of the repository's battery backs it at replay scale). Compiled views lower to a tensor-contraction plan artefact with canonical, digest-citable bytes, a dense reference evaluator, and an accelerator emission harness; the reference evaluator is the verdict authority throughout (the lane law, §5.12).

**Per-commit validity.** The validity layer (crates/geolog-chase/validity/) assembles per-commit validity judgments over chase results: which commits does this judgment implicate — well-posed exactly because every derived row and every invented element carries its firing history (§5.7.4). Wiring it into the storage pipeline end-to-end is design tense (the factoring is committed; the Tutorial states the same status).

## 5.9 The meta-schema

*Authority: theories/GeologMeta.geolog — the meta-schema’s normative statement, from whose text this section is condensed; where they disagree, that file wins. The committed genesis golden (§5.10.5) is its reference instance; the generator tool (theories/tools/geologmeta\_gen.py) re-derives and byte-compares both directions as a drift alarm in CI.*

geolog is a logical framework, not an expressive logic. GeologMeta is the fixed schema of geometric theories: one meta-schema, frozen at genesis; every theory — including the proof calculi the companion repositories ship — is an ordinary instance: delta rows against GeologMeta, carried by ordinary signed commits. Schema, extension, annotation, and migration therefore live in the same substrate they govern (the Tutorial’s “no second database”).

### 5.9.1 Sorts and rows

Nine schema sorts: Theory, Sort, Fun (function or relation — relationality is a mark, not a second sort), Field, Axiom, Var, Disjunct, Atom, Term. Table 5.1 carries the declaration vocabulary over them, with partiality as documented there (“partial” in its notes column means the row may be absent, and absence is meaningful — Codd’s I-mark discipline [45], never “unknown yet”).

| subject  | rows                                                                                                                                                                                                                                 | notes                                                                                                                                                                                                 |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Theory   | TheoryExtends[child, parent]<br>TheoryRevises[new, old]<br>(no Theory/name)                                                                                                                                                          | multi-parent legal; extends graph acyclic<br>revision intent; acyclic; causal half lives in deps<br>identity is the element; discovery naming lives above                                             |
| Sort     | Sort/name, Sort/theory<br>Sort/immutable<br>Sort/builtin_code                                                                                                                                                                        | carrier opaque to the chase (choose, never create)<br>partial: builtin tier only (§5.9.2)                                                                                                             |
| Fun      | Fun/name, Fun/theory<br>Fun/relational<br>Fun/codomain<br>Fun/arg_sort<br>Fun/side_condition                                                                                                                                         | the relation mark; Prop is not a sort<br>partial: exactly the non-relational Funs<br>partial: exactly the unary Funs<br>bounded-computation discharge; entails relational                             |
| Field    | Field/fun, Field/name, Field/sort                                                                                                                                                                                                    | name-keyed; canonical order is bitwise name order                                                                                                                                                     |
| Axiom    | Axiom/theory, Axiom/name<br>Axiom/monitored, Axiom/enforced                                                                                                                                                                          | name partial, recommended<br>mode rows (§5.8); at most one; surface<br>report/refuse lower here, chase to neither                                                                                     |
| Var      | Var/premise_of → Axiom<br>Var/bound_in → Disjunct                                                                                                                                                                                    | universal variables, scoped to the whole axiom<br>existential variables, scoped to one disjunct; exactly<br>one home (partition law)                                                                  |
| Disjunct | Var/sort, Var/name<br>Disjunct/axiom → Axiom                                                                                                                                                                                         | name is presentation only, hash-excluded<br>one branch of the conclusion; zero disjuncts on an<br>axiom = rejection rule                                                                              |
| Atom     | Atom/premise_of → Axiom<br>Atom/concl_of → Disjunct                                                                                                                                                                                  | premise atoms hang on the axiom itself<br>conclusion atoms hang on a disjunct, never on the<br>axiom; exactly one home                                                                                |
| Term     | Atom/eq_lhs, Atom/eq_rhs<br>Atom/rel, Atom/rel_arg<br>Term/var<br>Term/app_fun, Term/app_arg<br>Term/is_record, RecordField[rec, field, value]<br>Term/lit_q, lit_z, lit_nat, lit_b, lit_str,<br>lit_shape, lit_enrat, lit_timestamp | equation atoms<br>application atoms; exactly one kind<br>variable<br>application (arg absent iff nullary)<br>records; empty record is the explicit mark<br>one typed literal row per primordial dtype |

Table 5.1: The meta-schema’s declaration vocabulary, by subject sort (§5.9.1).

Domain shapes are discriminated by which rows exist — nullary: neither Fun/arg\_sort nor Field rows; unary: Fun/arg\_sort; record: Field rows.

The axiom encoding deserves a picture, because it has one level the sequent notation hides (Figure 5.1). The premise hangs directly on the Axiom element: premise atoms by Atom/premise\_of, universal variables by Var/premise\_of. The conclusion sits one level deeper. A conclusion is a disjunction of existentially quantified conjunctions (Definition 5.1), so each branch is its own Disjunct element, tied to the axiom by Disjunct/axiom; a conclusion atom names its *disjunct* as home (Atom/concl\_of), never the axiom, and an

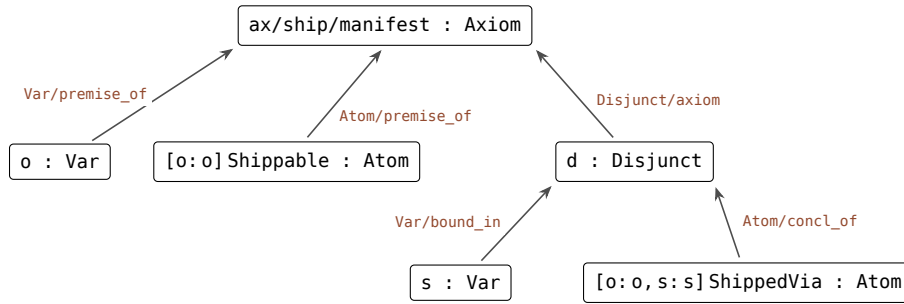


Figure 5.1: The Tutorial’s `ax/ship/manifest` ( $\forall o. \text{Shippable}(o) \vdash \exists s. \text{ShippedVia}(o, s)$ ) as meta-schema rows. Premise structure homes on the Axiom element; conclusion structure homes on a Disjunct, one element per branch of the disjunction. Which side of the turnstile an atom or variable lives on is read off the function row it carries, and the axiom’s repair arity is its Disjunct count — here one, the canonical shape.

existentially bound variable does the same (`Var/bound_in`). An axiom’s premise is therefore the *set* of atoms homed on it, its conclusion the *set* of its disjuncts, and its repair arity (§5.7.1) is its Disjunct count: zero is the empty disjunction, a rejection rule; one is the canonical, chase-eligible shape; two or more is a repair menu.

Field-name uniqueness per Fun and RecordField uniqueness per (record, field) are load rejections, deliberately not unification axioms: in a store whose rows are signed declarations, silently merging independently minted declarations would erase provenance — collisions are loud.

The meta-axioms, exhaustively (all fifteen; three shapes recur — entailment, rejection pair, and the (disjoint, exhaustive) *partition law* pair stating “exactly one”):

| axiom                                                                  | shape       | states                                      |
|------------------------------------------------------------------------|-------------|---------------------------------------------|
| <code>ax/meta/builtin_immutable</code>                                 | entailment  | coded sorts are immutable                   |
| <code>ax/meta/side_condition_relational</code>                         | entailment  | side-condition $\Rightarrow$ relational     |
| <code>ax/meta/relational_no_codomain</code>                            | rejection   | a relation never has a codomain             |
| <code>ax/meta/codomain_exhaustive</code>                               | disjunctive | every Fun is relational or has a codomain   |
| <code>ax/meta/arg_sort_fields_disjoint</code>                          | rejection   | unary and record domains exclude each other |
| <code>ax/meta/mode_marks_exclusive</code>                              | rejection   | at most one explicit mode per axiom         |
| <code>ax/meta/var_home_{disjoint, exhaustive}</code>                   | partition   | every Var has exactly one home              |
| <code>ax/meta/atom_home_{disjoint, exhaustive}</code>                  | partition   | every Atom sits on one side of $\vdash$     |
| <code>ax/meta/atom_kind_{disjoint, exhaustive}</code>                  | partition   | equation or application, never both/neither |
| <code>ax/meta/term_{var_app, var_record, app_record}_{disjoint}</code> | rejection   | representative term-kind disjointness       |

(`ax/meta/codomain_exhaustive` is also the schema’s own fixture for a disjunctive conclusion with an existential; the three term-kind rejections are representatives — the full mechanical pairwise family, the 28 literal pairs included, is elaboration-time conformance rather than schema text.)

**The testable negative.** What is deliberately absent is structural, not policed: no Formula sort and no Conj/Disj/Exists/Not/Iff nodes (nesting beyond geometric normal form is unrepresentable); no negation node and no iff node (both are authoring-side moves, §5.4.3); no Index/dependent-type node (erased upstream); no InfDisj node (Sort/immutable on the binder’s sort carries the distinction); no proof or derivation sorts (proofs of object logics are rows of object theories — the logical-framework doctrine); no weight or provenance slot (engine configuration, not schema); no annotation vocabulary (annotations are ordinary relations in ordinary extension theories — metadata needs rows, never a schema revision).

## 5.9.2 The primordial sort tier

Thirteen builtin ground sorts are declared as ordinary Sort rows in the genesis instance, marked immutable, and fixed forever: Q, Z, Nat, B, Str, ENNRat, Shape, Timestamp, Bytes, PeerId, CommitId, and the genesis-pinned extension-band straddle pair Date and Interval. Beside the thirteen sorts stand three sort *formers*, structural rather than declared — no Sort row, no declaring element; like the naturals themselves, they are simply available:

- **Fin** — the bounded-index former, an indexed family  $\text{Nat} \rightarrow \text{Sort}$ : for each natural  $n$ ,  $\text{Fin}(n)$  is the  $n$ -element index sort. These are the sorts that index tensor axes (an axis of extent  $n$  is indexed by  $\text{Fin}(n)$ ); the surface grammar’s bounded-index type ( $n$  Index, §5.3) expresses the same idea at authoring scale, though today the normalizer erases it to  $\text{Nat}$  plus range obligations (§5.4.3). Cell code `0x09` (§5.10.3).
- **Tensor** — the tensor former,  $\text{Shape} \times \text{Sort} \rightarrow \text{Sort}$ : given a shape (a  $\text{Shape}$  value, the list of axis extents) and an element sort, the sort of tensors of that shape with entries at that sort. The element sort may be *any* sort, declared or builtin, not only a numeric one. Canonical form is flattened, and a rank-zero tensor collapses to its single entry. Cell code `0x0A`.
- **List** — the finite-sequence former,  $\text{Sort} \rightarrow \text{Sort}$ , length carried per value:  $\llbracket x \text{ List} \rrbracket = \sum_{n:\text{N}} (\text{Fin}(n) \rightarrow \llbracket x \rrbracket)$  — each value brings its own  $n$ , the  $\text{DynTensor}$  mechanism with rank pinned at one. Positional order is the value (no sorting, no deduplication), and there is no flatness or unit law:  $(x \text{ List}) \text{ List}$  is a distinct, ragged two-level sort, never collapsed. Element scope at this tier is any sort, nesting included; the one adjudicated exclusion runs the other way — a list instance is not admissible as a  $\text{Tensor}/\text{DynTensor}$  *element* in  $v1$ , while  $(x \text{ Tensor}) \text{ List}$  is legal. Former code `0x10`; the surface spelling (§5.3) narrows the element scope to builtin ground sorts; value cells ride dtype code `0x50` (§5.10.3).

A fourth structural former stands beside these three at the wire tier, under its own sort-reference tag rather than a former code: **DynTensor** —  $\text{Sort} \rightarrow \text{Sort}$ , the *dynamic* tensor former, shape carried in the value rather than the type (the  $\Sigma$ -over-shapes reading of  $\text{Tensor}$ ; tag `0x36`, `crates/geolog-envelope/src/former.rs`).  $\text{Tensor}$  is the static reading — shape in the type — and the two are distinct sorts on purpose, never silently interconverted. Of the tensor pair, only  $\text{DynTensor}$  has a surface spelling (§5.3); tensor values are side-condition payload throughout (§5.12).

The tier is ambient: every theory may use it — formers included — in field and codomain positions without an extends edge. Minting an element of a builtin sort is a load rejection; builtin values enter as literals in value cells (§5.10.3).

Genesis-tier ground *vocabulary* equips the primordial carriers with algebraic structure as ordinary declarations (never as type-former firmware), each family’s evaluation semantics pinned in `theories/GeologMeta.geolog` to the corner case:

- **Q field operations** — `q_add`, `q_neg`, `q_sub`, `q_mul`, `q_inv`, `q_div`: exact arithmetic on canonical rationals, no rounding anywhere; partiality is refusal (`q_inv(0)` and division by zero refuse — a field has no answer there, and inventing one is what the exact tier exists to refuse).
- **ENNRat quantale operations** — on the Lawvere rationals  $\mathbb{Q}_{\geq 0} \cup \{\infty\}$ : `ennrat_add`, `_min`, `_max`, truncated minus (`_monus`, the residual of addition), *both* multiplications (`_mul_conj` with  $0 \cdot \infty = 0$  and `_mul_disj` with  $0 \cdot \infty = \infty$  — the two conventions declared under distinct names, never silently harmonized), inversion, residuation, harmonic sum, the two qualitative-mode side-conditions, and exact comparisons.
- **Timestamp** — instants, i64 epoch-microseconds on the POSIX/UTC scale (timezone is presentation, never value); the numeric order as side-conditions; checked displacement arithmetic (`ts_add_us`; overflow refuses, never wraps).
- **Date and Interval** — PostgreSQL semantics, as a tested claim: a corpus of goldens captured from a live PostgreSQL 17 instance is the conformance battery for the calendar family (§5.12); intervals are the three-field months/days/micros value verbatim, never flattened (comparison uses the justified collapse, values do not); refusal wherever PostgreSQL refuses, finite-only.
- **Str, CommitId** — disequality only (`str_ne`, `cid_ne`): string ordering would smuggle a collation choice (presentation, like timezone), and ordering commit hashes would invite misreading bytes as causal order.

Every *other* value format — the floating-point ladder included — is an extension ground format: declared by an ordinary signed revision (never genesis, never a schema revision), carrying a parametric spec over ordinary roster rows (byte width, exponent and mantissa bits, bias, NaN discipline), with wire codes from the extension band and operation dispatch bound to the declaring revision. `ax/meta/builtin_immutable`

quantifies over every coded sort, so extension formats are immutable by the same axiom that binds the genesis roster.

### 5.9.3 Theories as rows; revisions

A theory arrives as a *revision*: a sealed batch of meta-schema rows in one commit body, signed and content-addressed like every other commit (§5.10), dependencies named. Revisions publish whole — an axiom and all its structure land within one revision’s commit set, which is what makes zero-disjuncts-as-rejection well-defined. Identity is structural: a revision hash covers the declaration content (with presentation-only rows, such as `Var/name`, excluded), and a function’s signature hash covers its field names, sorts, and codomain — the identity the ground seam’s dispatch pins (§5.12).

Shipped machinery (`crates/geolog-meta-rows`, `crates/geolog-revision`, `crates/geolog-theories`): the `theory→rows` compiler (byte-identical round-trips on the committed corpora); the sealed-revision conformance loader, whose rejections are loud, named, and collected (one committed negative fixture per live rule); the structural revision and signature hashes with an adversarial vector battery; declaration indexes and cross-revision name resolution; the verified ground-format registry loader; and the programmatic revision-body builder, whose canonical-form invariants (backward local references, bitwise field order, exactly-once name rows) are unrepresentable or checked at build time. The `geolog rows` verb (§5.14) prints the declaration slice of this encoding for any theory file; the committed listings under `theories/generated/` carry complete worked theories (the Petri-net theory is 1,239 ops); and `geolog seal` (§5.14) seals a source file through this machinery — one revision commit carrying the theories as meta-schema rows (its dependency table names the genesis and the ground-comparisons golden, §5.10.5, uniformly — present whether or not the source uses the comparison roster, so the table is never input-dependent), plus a seal companion carrying the source and registry text — deterministically, so re-sealing an unchanged file is a no-op (verified: the second seal reports `store: genesis already present, comparisons already present, revision already present, seal already present`).

## 5.10 Commits: envelopes, bodies, cells, storage

*Authority: `crates/geolog-envelope` (the wire), `crates/geolog-store` (the log), `crates/geolog-ceremony` (the goldens). This section states the shipped encoding — encoding v1 — at the grain a consumer needs; the `crate` module docs carry the byte-exact layouts, and a committed cross-language reference implementation backs the golden vectors.*

### 5.10.1 The skeleton: hash and signature surface

A commit is a two-layer envelope. The **skeleton** is the hash/sign surface: a fixed-layout structure carrying

| field                     | content                                            |
|---------------------------|----------------------------------------------------|
| <code>version</code>      | the encoding version byte (1)                      |
| <code>author</code>       | ed25519 public key, 32 bytes                       |
| <code>deps</code>         | the commit-reference list: 32-byte commit hashes   |
| <code>parent_count</code> | how many leading deps are parents                  |
| <code>ts_claim_us</code>  | claimed timestamp, epoch-μs (a claim, not a truth) |
| <code>message</code>      | optional UTF-8 bytes                               |
| <code>body_hash</code>    | blake3 of the body bytes                           |

`deps` is one list in two segments — parents first, then non-parent content dependencies, each segment sorted strictly bitwise-ascending, duplicates rejected. *Commit identity is content*: `commit_hash` = `blake3(skeleton bytes)`, and the ed25519 signature covers exactly those bytes under a fixed versioned context prefix (`geolog-s4-skeleton-sig-v1`), so no future encoding change can collide signing domains. Canonicity is definitional, in both directions: the writer canonicalizes; the reader, after structural validation, re-encodes the decoded value and requires byte equality with the input (the fixed-point gate) — one value, one byte string, independent of any serializer freedom. A signed skeleton can only be constructed against a body receipt obtainable by durably storing the body, so the format’s sharpest failure mode — a

signed skeleton whose body is absent, a permanently unverifiable dangling commit — cannot arise from the safe API’s normal use.

5.10.2 The body: ops

The **body** is the ops payload. Wire shape: `u16!e body_format || frame bytes`; the format registry has exactly one live entry, and the numbering below is the public truth for this repository:

- 0 — retired before any production bytes existed (a migration lane for a legacy dialect; no wire ever carried it).
- 1 — `BODY_FORMAT_HEXANE_TEMPLATES_V1`: **the only format, and the default**. The canonical op sequence is factored into a template dictionary (one row per cluster of alike ops) plus columnar hole carriers; everything the dictionary hoists is spelled once per cluster, and each op row pays only for its hole values. The factoring is fully *derived* from the content — clustering, per-position carrier modes, and traversal order are laws, not producer choices — and a frame spelling the same rows any other way is refused at decode. The frame ends with the same fixed-point gate the skeleton uses: decode re-encodes and requires byte equality. `body_hash = blake3(body bytes)`.

The op vocabulary:

| op                    | payload                                  | meaning                                     |
|-----------------------|------------------------------------------|---------------------------------------------|
| el <code>ta</code>    | sort reference                           | mint an element of a (mutable) sort         |
| rel <code>a</code>    | relation ref, argument cells             | assert a relation row                       |
| fun <code>a</code>    | function ref, argument cells, value cell | assert a function value                     |
| el <code>td</code>    | target reference                         | delete an element (never a meta-row target) |
| blo <code>b</code>    | format tag, opaque bytes                 | content-addressed bulk (§5.10.3)            |
| unkn <code>own</code> | kind tag, opaque bytes                   | forward-verifiability carry                 |

References are either *local* (an earlier op of the same body, backward only) or *external* (content-addressed, through the deps); names are data (Sort/name rows), never identity. An unknown op kind is preserved byte-for-byte and carried, never dropped and never an error — a reader that predates a kind still parses correctly, because no decode path may silently drop data. Decode enforces hostile-input hygiene throughout: claimed counts are bounded (the op-count ceiling is  $2^{20}$  per body) and bounds-checked before any allocation.

5.10.3 Cells and the dtype registry

Argument and value positions carry *cells*: element references, or literals of the primordial dtypes. The primordial band is complete — `0x00–0x0F`, no holes:

| code | dtype  | code | dtype           |
|------|--------|------|-----------------|
| 0x00 | Q      | 0x08 | Shape           |
| 0x01 | Z      | 0x09 | Fin (former)    |
| 0x02 | Nat    | 0x0A | Tensor (former) |
| 0x03 | B      | 0x0B | Timestamp       |
| 0x04 | Str    | 0x0C | Bytes           |
| 0x05 | ENNRat | 0x0D | BlobRef         |
| 0x06 | ZBig   | 0x0E | PeerId          |
| 0x07 | NatBig | 0x0F | CommitId        |

ZBig/NatBig are multiprecision realizations of  $\mathbb{Z}/\mathbb{N}$  (distinct codes because widening a code’s grammar would be reinterpretation, which the registry’s general law forbids); BlobRef is the out-of-line spelling of Bytes/Str above the inline threshold (a 35-byte content-address cell); CommitId is a reference — a tiny index into the enclosing skeleton’s deps list — and never a context-free literal. The extension band `0x10–0x3F` carries ground formats (the float ladder `0x10–0x16`; the genesis-pinned Date `0x17` and Interval `0x18`), decoded opaquely and validated at closure tier; the absence band `0x40–0x4F` carries typed absence. Values of the list former ride dtype code `0x50` — the first assignment past the reserved bands, sitting out of numeric order because the primordial band was already complete and the former registry’s `0x10` (§5.9.2) falls inside the extension band — validated at decode like the primordials, in two first-byte-discriminated arms: an



*inline* arm (element dtype, count, then length-framed canonical element payloads — scalar elements only: the primordial literal codes minus `Tensor`, `BlobRef`, and `CommitId`) and an *envelope* arm (a 34-byte multihash content address under a dedicated derive-key context), partitioned at 64 KiB in the tensor family’s style; nested lists and non-scalar elements ride the envelope arm, and any cell code  $\geq 0x51$  is a decode error. Over-threshold bulk travels as `blob ops` (format 0 = `blob-raw-v1`) on the same signed-envelope machinery — signing, deps, sync, and equivocation detection inherited, interior deliberately opaque.

#### 5.10.4 The store log

The persistent store (`crates/geolog-store`) appends envelopes and bodies to a log file with a fixed grammar:

```
file      := header record*
header    := b"glog2607" | u32-LE format_version (1) | u32-LE reserved (0)
record    := u32-LE payload_len | u8 kind | payload
kind 1    := ENVELOPE: sig[64] | skeleton bytes
kind 2    := BODY:      raw body bytes (identity = blake3(bytes))
```

There is no per-record checksum, by adjudicated design: every envelope is signature-verifiable and every body content-hash-verifiable — stronger than any CRC — and a checksum cannot repair length-prefix desynchronization anyway. A record length above the 64 MiB ceiling is treated as torn or garbage bytes misread as a length, not as a record. The grammar layer never classifies a failure as crash artefact versus corruption — that is recovery policy, one layer up, which consults a committed hint the grammar deliberately knows nothing about. Header deviations (magic, version, nonzero reserved field) refuse loudly; a future format is a new version behind that gate, never a silent reinterpretation.

#### 5.10.5 The committed goldens and the ceremony

Four signed golden envelopes ship under *theories*: the **genesis** (GeologMeta’s own declaration rows as a signed commit, the root of the graph, `deps = [], message geolog-meta-genesis-v1`); the **ground-formats** revision (the float-format vocabulary, `deps = [genesis]`); the **zones** vocabulary (`deps = [genesis]`); and the **ground-comparisons** roster (`deps = [genesis]`, message `geolog-ground-comparisons-v1`) — the builtin ground-comparison vocabulary (`q_lt, q_le, q_ne, nat_lt, nat_ne`) as a sealed revision, the second dependency of every revision commit `geolog seal mints` (§5.9.3). Each is a deterministic function of its committed structure-normative listing and a *public* seed committed in source: anyone can derive the signing key and reproduce every byte, and cargo `run -q -p geolog-ceremony -- verify --root theories` does exactly that in every CI pass. The key certifies nothing about who signed; what makes these goldens canonical is that they are byte-reproducible from the committed listings by the committed tool. A real deployment mints its own genesis under its own key; the ceremony crate is the complete worked example of how. The genesis is also the meta-schema’s homoiconic fixed point — GeologMeta described as GeologMeta rows — and the generator’s check mode re-derives it from `theories/GeologMeta.geolog` both ways as a drift alarm.

### 5.11 Sync, merge, and equivocation

*Authority: crates/geolog-sync (whose README freezes the contract: objects, invariants, the frontier exchange, the convergence theorem, and a ledger of what is and is not yet claimed) and crates/geolog-envelope/src/equivocation.rs.*

#### 5.11.1 Merge

**Definition 5.2** (merge). The state of a replica is a set of signed commits (a `deps`-closed commit bag). Merging replicas is set union on commits followed by re-derivation of all derived state. Merge is therefore total, commutative, associative, and idempotent; it never conflicts structurally, and admission order does not affect derived state.

A merged state may satisfy some axiom’s violation premise — that is a judgment (§5.7.6), repaired by further ordinary commits, never by a conflict-resolution protocol. The Enforced gate binds local commit paths only (§5.8); gating sync admission on validity is the classic failure this architecture refuses, because it would

cost true merge and history-as-authority. Identity questions inside the merged state (two writers' records denoting one thing) are the theory's business, decided by its equality axioms and computed by congruence (§5.7.5). The rebuild invariant backs the order-independence claim mechanically: the repository's battery diffs incrementally maintained state against from-scratch replay, and a divergence is a release-blocking defect.

### 5.11.2 Replication

Two shipped strata. **Sync v0 — the file mesh:** whole-file, N-device mesh sync of per-device, single-writer, append-only envelope logs plus content-addressed body directories; per-source mirrors and per-run receipts ("silence means broken"); a torn trailing frame is classified by comparing against the prior round's observation (growing source: benign; stable source with a persisting tear: a problem) — a distinction a single snapshot cannot make. **Replay-through-ingestion** is the only path synced envelopes enter a local view: re-verification through the envelope crate's own verify path (no new trust surface), idempotent by commit hash, order-free via a deps-frontier worklist fixpoint.

**Sync v1 — the connected protocol,** layered strictly above v0: a deterministic integer-keyed message codec (strict known-kind decode, unknown-kind skip), frontier-exchange pulls, subscribe push, acknowledged head updates, credit-based flow control, bounded staging, and one door: admission through the same replay path. Equivocation detection runs incrementally during sync and is loud, never a gate.

**Realms and partitions.** Sync is scoped by *realm* — the wire's opaque slice key, today a plain partition name: sessions pull and subscribe per realm, and a realm's head set is an exact summary of its deps-closed store (the crate's `INV-CLOSURE`, which is store-global). A materialized partition must be dependency-closed and revision-closed (no partition splits a semantic revision unit); the reference builder and its checker are separate artefacts, and the checker re-verifies against the store rather than trusting the builder. Dependencies that cross a realm boundary are governed by a per-realm `TransportMode` dial: `refused` (the default), `reported` (the crossing becomes a declared boundary reference), or `repaired` (a locally held foreign dependency's pinned closure joins; a repair that would need a network fetch is a typed refusal). The `repaired` dial also binds at admission: a realm registered `repaired` admits entries whose dependencies are admitted under *other* realms — the dependencies must still be admitted locally, so closure remains store-global — and the dial is local policy, taken at store open and never negotiated on the wire; an undialed peer stages the same entries, a liveness asymmetry rather than a safety one.

### 5.11.3 Equivocation

**Definition 5.3** (equivocation-suspicious pair). Two distinct envelopes from one author, neither of which causally references the other (through deps), form an equivocation-suspicious pair.

Detection is structural — from the bytes alone, no consensus protocol, no quorum — and local to whatever envelope set is supplied: a dep pointing outside the set is "unknown, not reachable," never an error. Two different-bytes envelopes claiming one commit hash refuse loudly (a genuine hash collision or a hostile input; never silently resolved by picking a winner). The envelope crate ships the demonstration as a test — two signed commits claiming the same chain position, exhibited and caught — and the Tutorial locates this in the systems landscape: the offence a broker cannot even state is detected here from the data model. The mechanism, signed hash-linked histories making forks structurally visible, is the same one that makes CRDTs Byzantine fault tolerant [138].

## 5.12 The ground seam

*Authority:* `crates/geolog-ground-api` (the contract) and `crates/geolog-ground` (the reference floor, its first provider).

Ground vocabulary — relations and functions over immutable carriers, decided by procedures rather than stored rows — crosses one seam. **Declaration is data:** ground vocabulary is declared where all vocabulary is declared, as committed theory rows, and resolves to a pinned identity (declaring commit + row index) with a verified signature and, where one exists, the commit hash of the committed evaluation-semantics

row. **Implementation is code:** a *provider* is one value carrying curated claims plus the evaluator over its own entries. **The binder is where they meet**, and it is engine-owned law no provider can bypass or see: every resolved declaration is adjudicated against every provider's claims, with three outcomes per entry —

| status      | meaning                                                                                  |
|-------------|------------------------------------------------------------------------------------------|
| Implemented | exactly one claim verified, committed semantics covered — a dense engine id is allocated |
| Deferred    | declared-but-unbound, safe-fail: no claimant, no semantics row, or a stale provider      |
| Refused     | code and theory disagree (signature mismatch) — the drift alarm, distinct from Deferred  |

Two verified claims on one identity are always an error; provider order is never semantics. Dispatch binds to pinned identities, never names: a same-named impostor declaration gets no oracle and its atoms defer forever. An unbound entry never obtains a dense id, so a compiled plan citing it cannot even be expressed — *compile refusal by construction*, the seam's realization of refusal-over-guessing.

The laws, stated once here and cited by everything that consumes the seam: **purity** (every evaluator is stateless and deterministic per its declared scope — same cells, same outcome, every process, every architecture; a family whose reference uses floating point must pin bit-exact step semantics or cannot be admitted); **the lane law** (an accelerated lane may change cost, never verdicts, and ships with a conformance battery against the always-on scalar reference — a lane that cannot pass its battery does not dispatch); **refusal** (refusal is data, and conservative: under an Enforced axiom, a gate that cannot evaluate must not admit; no seam method panics on validated cells); **no enumeration** (a ground relation filters, never generates; a ground function inverts only where the algebra genuinely determines the missing argument — underdetermined means the engine must not enumerate); and **scope** (pure is the norm; the one typed exception is snapshot-scoped structural relations reading instance rows through an engine-supplied view, with the routing engine-enforced in both directions). Cells cross the seam in their canonical encodings, opaque to the contract; there is no intermediate certificate file format anywhere — a checker is a ground relation evaluating over rows the engine already speaks.

The **reference floor** ships as the seam's first provider: exact arbitrary-precision rational arithmetic, the extended-nonnegative-rational quantale operations, calendar arithmetic with documented, PostgreSQL-conformant semantics (the golden corpus of §5.9.2), and the curated ground-comparison family. The floor's reference path is unconditional; the calendar *functions* sit behind a feature flag whose absence makes their claims absent — declared entries then defer at bind, so **verdicts never change with a feature flag**. Application side-condition families (the proof-calculus checkers of the companion repositories) are application content: they ship as a provider crate owned by the application, composed at build time through this same binder — the only sanctioned coupling between this repository and its first big application. Reaching the seam from the `geolog` command line is design tense; the engine-level seam, the floor, and their batteries are shipped and tested today (the Tutorial's ground chapter walks the same line).

## 5.13 Capabilities

*Authority: crates/geolog-caps.*

Authorization arrives as an algebra over commits, not an access list beside them. Shipped today, as data and replay semantics: capability operations (delegations and revocations) travel as ordinary signed revisions; a normative replay computes which revocations are effective; authorization judgments are query-time and position-relative. The committed design decisions, decision by decision (the authority crate's own phrase):

- a grade lattice `Define < Retract < Admin` per sort — element deletion is the non-monotone direction, so it is strictly higher privilege than minting; the add/delete split of the op vocabulary, lifted;
- a per-relation `Assert` chain sharing no element with the sort chain above bottom, so sort authority and row authority can never satisfy one another;
- dual `Admins` per theory and per instance, with `theory-Read` as the base prerequisite to holding anything effective on a theory's instances;
- a conservative cross-tier stance: no implication in either direction; dormant grants are first-class, queryable, and surfaced by an authoring lint that replay never reads;

- revocations executed in earliest-add seniority (ascending authority-provenance depth, digest-tiebroken), with the auth-order projection carried as data so seniority cannot be padded by packaging;
- edge identity as (revision, index), so identical re-grants in distinct revisions are distinct edges.

*Design tense:* enforcement at the store boundary — wiring these judgments into admission of local commits — is the committed direction; today the algebra, replay, and judgments ship as a library with their property tests, and no store consults them yet.

## 5.14 The geolog command line

*Authority:* `crates/geolog` — `main.rs` for the surface, the modules cited throughout this chapter for each verb's semantics. The end-to-end tests in `crates/geolog/tests/cli.rs` run the book's own transcripts against the committed examples.

```
geolog check      <file.geolog> [--strict]
geolog check      --store DIR [<commit>] [--strict]
geolog chase      <file.geolog> [--instance A[,B...]]
                  [--budget N] [--mode chased|monitored]
geolog chase      --store DIR [<commit>] [--instance A[,B...]] [--budget N]
geolog seal       <file.geolog> --store DIR [--strict] [--repro]
geolog extract    <commit> --store DIR [--referencing] [-o FILE.glc]
geolog restrict   <file.glc> --store DIR --theory SEL [--revision HEX]
                  [-o FILE.glc]
geolog load       <file.geolog> [--strict] [--dry-run] [-d]
geolog load       <file.glc> [--store DIR]
geolog fmt        <file>
geolog rows       <file.geolog>
geolog normalize  <file.geolog> [--strict]
geolog census     <file.geolog>... [--rows LISTING]...
                  [--budget N] [--tsv FILE]
```

**The specified verbs.** **check** runs `parse` → `normalize` → `lint` on the theory text and validates every instance block against the elaborated closure by actually seeding it — exactly the checks `chase` would run, without the `chase` — reporting each stage and then three summaries: a *mode census*, what the file's axioms will do when chased (`modes: 2 chase · 1 report (0 by the unmarked default) · 0 refuse` for the Quick Start theory — so an all-report file, deriving nothing by design or awaiting its explicit-chase port, is legible at a glance); the move census (§5.4.3); and, where the chase bridge carries the fragment, a chase-termination advisory (weak acyclicity) computed under the file's own surface modes — only the chase-marked fragment generates, so a theory of report-marked constraints is termination-trivial by construction, however generative its axioms would be if fired. With `--store`, the same strata run over a sealed source instead of a file, after signature verification and a coherence gate: the sealed source must re-derive the sealed revision rows byte-identically. **chase** seeds the selected instance blocks and runs the chase to a fixpoint, then renders the report of §5.7.7. **rows** prints the theory's declaration slice as meta-schema rows in the committed listings' notation (local refs  $L_n$ , genesis identities  $G_{<...>}$ ), with a closing note counting the axioms that continue as rows beyond the slice; it requires the one-field record spelling for unary relations, because fields are rows with names in the meta-schema (verified: relation `'Marked'` has a unary (anonymous) domain — ... rows emission needs the one-field record spelling: `'Marked : {x: A} -> ...'`).

**The remaining verbs, at citation grain.** Each is shipped and tested; this manual states its role and leaves its full semantics to the named authority. **seal** runs `parse` → `normalize` → `canonical revision rows` → a signed envelope pair in the store: a revision commit and a seal companion (§5.9.3); re-sealing an unchanged file is a no-op. With `--repro` the signature uses the public reproduction seed, making the commit hashes store-independent (pin reproducibility); such commits carry no authorship and must never anchor authority. **extract** writes a commit and its transitive deps from the store to a `.glc` container (magic `geologfl`, self-contained, verified on read rather than trusted); `--referencing` additionally sweeps every store commit whose deps reach the target, so all commits referencing an instance travel together.

**restrict** is theory-fragment restriction of a certificate container — the get side of the delta lens a theory  $B$  extends  $A$  induces: it keeps exactly the rows whose species live in the target theory’s extends-closure (derived from the sealed revision’s own declaration rows), projects the prerequisite table onto the surviving boundary, copies unchanged commits byte-verbatim, re-seals changed ones deterministically with the public reproduction key, and refuses — with a named locus — any container whose kept rows reference stripped content; a pure-fragment container round-trips byte-identical. **load** with a `.glc` argument decodes and verifies the whole container — signatures, body hashes, topological order — then admits it into `--store`; with a `.geolog` argument it stages the theory’s aggregation surface into a live engine (§5.15). **fmt** parses a file, legacy spellings included, and prints the canonical source respelling — the corpus-port instrument (§5.3). **normalize** prints the geometric normal form the engine consumes (§5.4.2) — the committed-golden regeneration verb. **census** is the merge-reliance census over authored equality heads; its method and honesty ledger live in `crates/geolog/src/census.rs`. Authority for all of these: `crates/geolog/src/main.rs` and the modules it names.

**Options.** Classification travels in the source file (§5.4.4), so the file-mode verbs take the file and nothing else; `--registry FILE`, the deprecated migration-era carrier, refuses beside import lines when its registry carries classification (a pattern-only registry composes, its aux names kept clear of imported ones — §5.4.4), and in store mode the flag is refused outright: the registry text travels sealed. `--strict` rejects the deprecated spellings — bracket records, binders-before-name theory headers, `= { instance bodies, : Sort declarations` — each with a pointer at the ratified form (§5.3) — and demands the `import std` header for builtin ground vocabulary used by name (§5.4.4). `--instance A[,B...]` selects blocks by name — the default is every block in the file, as their union (§5.5); naming a missing block is a usage error that lists the blocks present. `--budget N` bounds chase iterations (default 1000); exhaustion is a reported judgment, not an error. By default chase runs each axiom under its authored surface mode (§5.7.2); `--mode chased|monitored` is the legacy run-level flattener for files whose axioms carry no marks, and refuses — naming the marked axioms — on any file that does. chase on a file with no instance block is a usage error that says what to add and points at check for theory-only validation.

**Exit codes.** 0 — clean: the verb completed and, for chase, the fixpoint carries no judgments. 1 — judgments at fixpoint: the run completed and reported violated denials, pending branch choices, or an exhausted budget. 2 — the input never ran: usage errors, unreadable files, pipeline rejections, bridge refusals, seeding contradictions, and engine load refusals. The 1/2 boundary is the error type’s own doctrine: judgments from a completed chase are results, not errors.

**Constituent binaries.** The workspace also ships development CLIs for constituent crates — `glnorm` (the normalizer pipeline standalone), `glstore` (the store log), `glsync` (the file-mesh sync) — documented in their crates; `geolog` is the carryable verb and the only surface this chapter specifies.

## 5.15 Editions: what this manual does not yet state

Open seams, each named with where its truth lives today; a future edition of this chapter absorbs them in this order.

- **The aggregation surface.** The ratified authoring grammar beyond the fragment this manual states — law members (named or anonymous) whose bodies carry contraction index groups and string-literal contraction specs, the arity-typed contraction resolver with its refusal roster (E-RANGE, E-SPEC-\*, E-RPN-\*, E-LAW-\*), and the free-construction instance declarations (parameters, extends targets, and body rows as three seed sources) — is shipped in `crates/geolog-normalize` (the parser module’s documentation is the compact normative summary) and staged into a live engine by `geolog load`, whose emission header marks the engine realization design-tier pending the aggregation-schema promotion. This edition states the surface’s lexical floor and type-grammar hooks (§5.1, §5.3) and defers the rest to the next.
- **The store-backed surface at this manual’s grain.** `seal`, `extract`, `load` of a `.glc` container, the store modes of `check` and `chase`, and `census` all ship today (§5.14); the container grammar, the store-mode conformance rules, and the `census` method are normative in the facade’s modules. A future edition states them at the grain §5.10 gives the envelope.

- **The byte-exact wire grammar.** §5.10 states the encoding at consumer grain; the op-payload grammar, the template-frame channel layout, and the conformance-rule table (load-rejection classes and their fixtures) are normative today in `crates/geolog-envelope`'s module documentation and `crates/geolog-meta-rows`' loader and negative fixtures.
- **Per-commit validity, end to end.** The judgment assembly ships (§5.8); the storage-pipeline wiring and its report vocabulary are design tense.
- **Capability enforcement** at the store boundary (§5.13).
- **The ground seam from the command line** (§5.12): the chase-path elaboration of ground sorts remains refused (§5.6), and the `load` path stages its ground-vocabulary dependencies as roster rows while its engine realization stays design-tier (previous bullet).
- **Cross-revision extends in the facade**, native bounded-index types, and the full evaluation-semantics tables of the genesis ground families (today: pinned, with their corner cases and golden corpora, in `theories/GeologMeta.geolog` and the ground crates' tests).



# Diagrams as Data: S-Complexes

DRAFT

Several strata of this book traffic in *diagrams*. The specification language’s global stratum asserts a certified Morse graph (chapter 14); the hypernet chapter reads every computation as a string diagram (chapter 10); grid certificates decompose state space into cubical cells; and the higher-categorical diagram machinery this project consults runs on oriented graded posets. Each of these is, concretely, a family of cells in graded dimensions plus incidence data saying which cells bound which. Each could therefore grow its own bespoke incidence schema, its own well-formedness checker, and its own subtly different answer to “is the boundary recorded here the boundary the theorem was about?”

The toolkit owns one such object. An *S-complex* is a graded family of finite cell sets whose boundary structure is recorded as a pair of semiring-valued tensors per dimension (one for each orientation sign), satisfying a subtraction-free form of the chain condition  $\partial^2 = 0$ . The semiring parameter  $\mathbb{S}$  is where the genus gets its reach: with Booleans it is incidence, with natural numbers it is multiplicity, with a semiring of *port names* it is a string diagram, and with min-plus dwell bounds its labels carry quantitative dynamics. Well-formedness in every case amounts to a handful of tensor contractions and one semiring comparison per dimension, using the same contraction machinery the rest of the toolkit already trusts (chapter 10, relational-kinship section), so the properties are checkable in bulk, on accelerators, by the engine that checks everything else.

The instinct (diagrams stored as typed incidence data, computed on in bulk) is shared with the attributed  $\mathcal{C}$ -set line of applied category theory, where diagrams are categorical data structures [194] with data-parallel composition algorithms [241], and where a string diagram can be a first-class engineering artefact rather than an illustration [173]. An S-complex trades that line’s schema generality for one graded shape with a semiring parameter; the trade is what makes well-formedness a tensor computation.

This chapter defines the object and its morphisms (section 6.1), tours the instances that choice of  $\mathbb{S}$  buys (section 6.2), states the encoding discipline that decides when a complex is stored versus derived (section 6.3), presents the certified Morse graph as the flagship rank-1 instance with its quantitative refinement (section 6.4), works the string-diagram instance in full — including the reading under which a well-formed hypernet *is* an S-complex (section 6.5) — and closes with a ledger separating definitional content from design directions (section 6.6). Sections 6.1 and 6.4 carry everything the rest of the book consumes; section 6.5 is the most likely to surprise, and a reader with no stake in string diagrams can skip it without loss of continuity.

## 6.1 The definition

Throughout,  $\mathbb{S}$  is an *ordered semiring*: a semiring  $(\mathbb{S}, 0, 1, +, \cdot)$  equipped with a preorder  $\leq$  such that  $0 \leq 1$  and both operations are monotone in each argument. Order-equivalence is derived, not assumed:  $x \cong y \Leftrightarrow x \leq y \wedge y \leq x$ . No subtraction is ever required. That restraint is the point: the instances worth having — Booleans, counts, port sets, min-plus costs — live in semirings that have no subtraction to offer.

A *tensor* over  $\mathbb{S}$  indexed by finite sets  $A_0, \dots, A_{k-1}$  is a finite function  $\prod_{i < k} A_i \rightarrow \mathbb{S}$ , and juxtaposition of tensors denotes contraction over the shared index (sum of products, in  $\mathbb{S}$ ), as in the contraction vocabulary of chapter 12.

**Definition 6.1** (S-complex). An *S-complex* of rank  $r$  consists of finite cell sets  $C_0, C_1, \dots, C_r$  (cells in dimension 0 through  $r$ ) together with, for each  $0 < i \leq r$  and each sign  $e \in \{+, -\}$ , a rank-2 boundary tensor

$$\partial_i^e : C_i \times C_{i-1} \longrightarrow \mathbb{S},$$

satisfying, for each  $1 \leq i \leq r - 1$ , the *sign-split chain condition*

$$\partial_i^+ \partial_{i+1}^+ + \partial_i^- \partial_{i+1}^- \cong \partial_i^- \partial_{i+1}^+ + \partial_i^+ \partial_{i+1}^- : C_{i+1} \times C_{i-1} \rightarrow \mathbb{S}. \quad (6.1)$$

Stated as one dependent type, with  $[N]$  the finite ordinals and  $\text{Fin}$  the concretely finite sets, the presented data of the genus is

$$\sum_{N: \mathbb{N}} \sum_{C: \prod_{n: [N]} \text{Fin}} \prod_{i: [N-1]} \prod_{e: \{+, -\}} [\partial_{i+1}^e: C_{i+1} \times C_i \rightarrow \mathbb{S}]$$

—  $N$  graded levels and a sign-split boundary pair per consecutive pair of levels, nothing else. The type records presentations only; condition (6.1) is a property checked of a presentation rather than a component packaged in it, so one schema serves both the complexes that satisfy it and the raw presentations a checker has not yet admitted. The rank  $r$  of Definition 6.1 is  $N - 1$ .

Condition (6.1) is the classical  $\partial^2 = 0$  with the subtraction moved across the equals sign: over a ring, setting  $\partial = \partial^+ - \partial^-$  and expanding  $\partial_i \partial_{i+1} = 0$  gives precisely “matched sign pairs equal mismatched sign pairs.” Stated this way the condition makes sense over any ordered semiring, and each side is two contractions and a sum, so checking well-formedness of a presented complex is a fixed, small pattern of contraction rows per dimension — the bulk-checkable shape the engine is built around. An elementary instance to hold while reading: a filled square. Each corner lies on two of the four edges, so exactly two routes descend from the 2-cell through an edge to that corner; however the edges are oriented, one route carries matched signs and the other mismatched, so the two sides of (6.1) agree entry by entry — the corner-cancellation of  $\partial^2 = 0$ , surviving with parity where its minus signs used to be. For  $r \leq 1$  the condition is vacuous, and a presentation of a rank-1 complex should say so plainly rather than imply a check that never ran.

The two signs record *combinatorial orientation*: for a directed edge  $a$  from  $x$  to  $y$ , the source incidence lives in  $\partial_1^-(a, x)$  and the target incidence in  $\partial_1^+(a, y)$ . This sign records a property of the diagram itself rather than of any dynamics drawn on it; when a dynamical certificate needs a notion of inward and outward flow across a face, that is separate label data riding on the cells, never a reinterpretation of the boundary signs.

**Definition 6.2** (morphism of S-complexes). Let  $C$  be an  $\mathbb{S}$ -complex and  $D$  a  $\mathbb{T}$ -complex with  $\text{rank}(C) \leq \text{rank}(D)$ . A *morphism*  $f: C \rightarrow D$  consists of a function  $f_i: C_i \rightarrow D_i$  for each dimension  $i$  of  $C$ , regarded semantically as a tensor  $f_i: C_i \times D_i \rightarrow \mathbb{T}$  whose entries are 0 and 1, together with a lax morphism of ordered semirings  $f_S: \mathbb{S} \rightarrow \mathbb{T}$  (monotone,  $f_S(0) \cong 0$ ,  $f_S(1) \cong 1$ ,  $f_S(x + y) \leq f_S(x) + f_S(y)$ , and  $f_S(x \cdot y) \leq f_S(x) \cdot f_S(y)$ ), such that for every dimension  $0 < i \leq \text{rank}(C)$ , every sign  $e$ , and every  $(c, d) \in C_i \times D_{i-1}$ ,

$$\sum_{d' \in D_i} f_i(c, d') \partial_i^{D, e}(d', d) = \sum_{c' \in C_{i-1}} f_S(\partial_i^{C, e}(c, c')) f_{i-1}(c', d). \quad (6.2)$$

Equation (6.2) is the chain-map square, sign by sign: mapping a cell forward and taking its boundary in  $D$  agrees with taking its boundary in  $C$ , pushing the cells forward, and translating the coefficient values through  $f_S$ . Laxity in  $f_S$  is deliberate. The useful coefficient translations are *forgetful* — cardinality (from port witnesses to counts), support (from counts to incidence), and coarse quantitative summaries (from fine margins to dwell bounds) — and translations of the last kind are subadditive rather than exact, so demanding exactness would refuse precisely the summaries the tower is for.

## 6.2 Instances by choice of semiring

The definition specializes by choice of coefficient semiring:

| object                | $\mathbb{S}$          | rank | boundary reading                             |
|-----------------------|-----------------------|------|----------------------------------------------|
| directed graph        | 2 (Booleans)          | 1    | $\partial_1^-$ source, $\partial_1^+$ target |
| directed hypergraph   | $\mathbb{N}$          | 1    | tail/head multiplicities                     |
| cubical complex       | 2                     | $d$  | oriented facet incidence                     |
| oriented graded poset | 2                     | any  | covering relation, one sign each             |
| Lefschetz complex     | a ring's split halves | any  | incidence coefficients                       |
| string diagram        | port names (§6.5)     | 1    | <i>which ports</i> , not just how many       |

*Remark 6.3* (oriented graded posets). Oriented graded posets, a combinatorial substrate of higher-categorical diagram rewriting [104], embed as Boolean complexes in which each covering pair carries exactly one of the two signs. The shape predicates that theory cares about (roundness, molecule-hood, regularity) remain predicates on top of the complex, as that literature states them, with the complex supplying the data they are predicates of.

*Remark 6.4* (Lefschetz complexes). Lefschetz complexes, a combinatorial carrier of Conley-theoretic invariants [183], carry ring coefficients without a ring: a classical incidence coefficient enters as its split halves  $k = k^+ - k^-$ , one sign each, so entries stay in  $\mathbb{N}$  and subtraction never enters the schema.

*Remark 6.5* (the coefficient tower). The coefficient levels form a tower. Cardinality maps a string diagram's witness sets to a directed hypergraph's multiplicities, and Boolean support forgets those to bare incidence. Both maps are lax morphisms in the sense of Definition 6.2 with the identity on cells, so the port, counting, and incidence readings are three resolutions of a single object.

### 6.3 Generators or presentations

Not every complex the toolkit reasons about should be stored as one. The encoding discipline distinguishes two postures.

- **Generator-encoded** where the complex is canonically derivable: grid certificates store grid lines and cell selections, never cell lists or incidence tensors — the cubical complex, with its orientation-split boundary, is recomputed from the generators by every consumer. Skew between a stored presentation and its generators is then unrepresentable rather than checked.
- **S-complex-encoded** where the boundary *is* the content: a Morse graph's edges, a reduced complex, a connection matrix. Here there is no generator to fall back on — the incidence data is precisely what the certificate asserts — so the presentation of Definition 6.1 is the wire object, and condition (6.1) (where non-vacuous) is part of its well-formedness check.

The two postures compose: a quotient from a generator-encoded complex onto an S-complex-encoded one is a morphism in the sense of Definition 6.2, with the derived presentation appearing only transiently on the checking side.

### 6.4 The Morse graph, quantitatively

The global stratum's payload (chapter 14, QMorse row) is a certified Morse graph: finitely many blocks, finitely many directed connections. As data it ships in adjacency dialect (a vertex count and an edge list), and that dialect *is* the rank-1 Boolean S-complex:  $C_0$  the Morse sets,  $C_1$  the connections,  $\partial_1^-$  source,  $\partial_1^+$  target. Acyclicity is the checker's to confirm; the confirmation, in this vocabulary, is an order witness on  $C_0$  of the same kind section 6.5 uses for string diagrams.

The quantitative refinement adds a *labelling* without changing the complex: cells carry values in the ordered semiring  $\text{DWELL} = ([0, \infty], \min, +)$  — the cost quantale of Lawvere's generalized metric spaces [157], whose additive unit is  $\infty$  and whose multiplicative unit is 0, ordered by the reverse of the numerical order, which is what makes  $0 \leq 1$  hold — with connections labelled by certified lower bounds on transit time and Morse sets labelled by their sojourn bounds, which for genuine Morse sets are  $\infty$ . The semiring arithmetic then guards the composite claims against overstatement: min-plus contraction of a path through an intermediate Morse set yields  $\infty$ , so the algebra itself reports that no finite total-time bound through a recurrent set is being asserted, while per-leg bounds stay finite and usable. This is the intended serialization

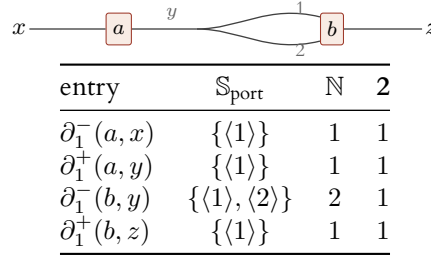


Figure 6.1: One diagram, three coefficient resolutions. Wire  $y$  feeds both input ports of box  $b$ , so the witness set  $\partial_1^-(b, y)$  holds two singleton port sequences; cardinality forgets *which* ports to *how many*, and Boolean support forgets that to *whether at all* (Remark 6.5). Entries not shown are  $\emptyset$ , 0, 0.

form for the quantitative Morse tier: the graph presented as in Definition 6.1, the rates as cell labels (extended nonnegative rationals in the machine artefact), and the projection from the underlying grid complex onto the Morse graph stated as a morphism — cells to their Morse or transit class, fine certified margins translated laxly to dwell bounds — rather than as an unstated convention between two schemas.

## 6.5 String diagrams: the port semiring

This instance shows why the coefficients form a *semiring parameter* rather than a Boolean convention. Fix a supply  $\mathcal{P}$  of *port names* — opaque identifiers, meaningful only up to renaming. Let  $\mathbb{S}_{\text{port}}$  be the semiring of finite *multisets* of finite sequences over  $\mathcal{P}$ : addition is multiset union with zero  $\emptyset$ , multiplication is elementwise concatenation with unit  $\{\varepsilon\}$ , and the order compares cardinalities — so the derived order-equivalence is blind to the names themselves, which is the renaming-invariance the names’ opacity demands.<sup>1</sup>

A *string diagram* over a signature is a rank-1  $\mathbb{S}_{\text{port}}$ -complex —  $C_0$  the wires,  $C_1$  the boxes — together with a morphism into the complex presenting the signature ( $C_0$  the sorts,  $C_1$  the generators, each generator’s canonical ports as its boundary values). A boundary value  $\partial_1^-(a, x)$  is a *witness set*: one singleton sequence per input port of box  $a$  attached to wire  $x$ , and dually for outputs. Over  $\mathbf{2}$  an edge could only assert incidence; over  $\mathbb{S}_{\text{port}}$  the value says *what flows where* — the difference between a reachability structure and a diagram whose wires carry content comes down, in this vocabulary, to the choice of semiring. The signature morphism does the typing: its cell maps send boxes to generators and wires to sorts, and its coefficient translation  $f_S$  matches each diagram port to the generator port it instantiates, so argument order lives in the signature once instead of in every diagram. Sequence concatenation earns its place under composition, where contracted products accumulate port *paths* witnessing composite incidences; at rank 1 only singletons occur. Figure 6.1 works a two-box example and reads its boundary data at the three coefficient resolutions of Remark 6.5.

Structural conditions on the diagram are then one clause each, and free-structure theorems select which clauses to impose:

| producer/consumer discipline         | acyclicity | free structure presented        |
|--------------------------------------|------------|---------------------------------|
| none                                 | none       | hypergraph category [26]        |
| $\leq 1$ producer, $\leq 1$ consumer | none       | traced symmetric monoidal [137] |
| $\leq 1$ producer, $\leq 1$ consumer | witness    | symmetric monoidal [27]         |
| $\leq 1$ producer, consumers free    | witness    | gs-monoidal [84]                |

Here “ $\leq 1$  producer” is the port-counting condition  $\sum_{a \in C_1} |\partial_1^+(a, x)| \leq 1$  for every wire  $x$ , and dually for consumers — the cardinality order makes counting a first-class operation. The acyclicity witness is a

<sup>1</sup>Multisets rather than sets, for a reason worth checking: concatenation can merge distinct pairs of sequences  $\langle 1 \rangle \cdot \langle 2, 3 \rangle$  and  $\langle 1, 2 \rangle \cdot \langle 3 \rangle$  both concatenate to  $\langle 1, 2, 3 \rangle$ , so two *sets* of equal cardinality can have products of different cardinalities, and multiplication would fail the monotonicity section 6.1 demands of every coefficient semiring. Multisets count collisions instead of merging them, which makes cardinality exactly additive and multiplicative. The witness sets arising in diagrams are unaffected: distinct ports give distinct sequences.

function  $t: C_0 \rightarrow P$  into a poset with  $t(\text{input wire}) < t(\text{output wire})$  across every box: a one-pass check and a rewrite-friendly certificate.

### 6.5.1 Hypernets read as S-complexes

The hypernet of chapter 10 sits in the third row of the table: its linearity condition is producer-and-consumer monogamy and its static-single-assignment order is an acyclicity witness, as that chapter already observes in citing the free-structure correspondence. The reading assembles as follows. Wires are the 0-cells; applications are the 1-cells; an application's ordered input and output lists are the two boundary rows, with list position serving as the port name — the representation takes the renaming quotient so the semiring does not have to. The catalogue plays the signature: the morphism sends each application to its box, each wire to its tensor type, and its coefficient translation records which argument slot each wire fills. Two strata of the hypernet deliberately live *beside* the complex rather than in it: box payloads (constants, contraction patterns, polynomial coefficients) are cell labels, just as rates are labels on the Morse graph; and the interface is a cospan of morphisms from rank-0 complexes — an ordered list of graph inputs and one of graph outputs, each simply a function into  $C_0$ , which is what the declared interface lists already are.

The vocabulary also makes the design space around the interface legible in single clauses. Letting the output leg of the cospan hit any wire — including one an application already consumes — declares a read-out port without touching the inner complex or its monogamy, since the legs were never part of either. If consumer-side monogamy is retired while the producer side is kept, the diagram genus moves one row down the table, to the gs-monoidal discipline of every static-single-assignment compiler representation — def once, use many — with copy and discard absorbed into the wiring. Both are stated here as readings the definition supports rather than as changes made: the shipped hypernet remains the monogamous, acyclic row, and its bytes, digests, and substitution laws are those of chapter 10, unchanged.

## 6.6 Ledger

What in this chapter is what:

- **Definitional.** Definitions 6.1 and 6.2, the derivation of (6.1) from  $\partial^2 = 0$ , the instance table, and the free-structure table (whose correspondences are the cited literature's theorems, not this book's).
- **Machine artefact.** The genus is declared as a standard-library theory of the geolog engine (`theories/SComplex.geolog`), which names this chapter as its expository companion, and the chain condition's contraction factoring is generated from a surface statement of the tensor law (`theories/SComplexGenus.geolog`). The theory is staged design vocabulary, exercised by the engine's tests rather than by production data.
- **Shipped elsewhere in this book.** The Morse-graph payload shape and its checker obligations (chapter 14); the hypernet IR, whose linearity and ordering conditions this chapter's section 6.5.1 re-reads (chapter 10); the contraction machinery that makes condition (6.1) bulk-checkable (chapter 12).
- **Design tense.** The quantitative Morse serialization in complex-plus-labels form and its quotient-as-morphism statement (section 6.4); any S-complex-native storage for hypernets (section 6.5.1 is a reading, and the canonical bytes are unchanged by it); mechanized S-complex morphisms, of which the toolkit currently has none — the Morse quotient is the natural first target. Higher-dimensional rewriting on this substrate (equivalence cells over diagrams) is out of this chapter's scope entirely.

PART II

# QTSL: Quantitative Temporal Specification Logic

---



## Quick Start

---

In this chapter you will produce a machine-checkable certificate of a quantitative claim about a stochastic system, in milliseconds, and see exactly what any checker must re-derive from its bytes. You will meet the three routes by which certificate bytes get checked in this toolkit — and where each route stands at this preview. Then you will break a certificate from the published corpus and watch an exact-arithmetic reader refuse it, with the violated obligation named. The whole loop fits in one sitting; every command is copy-pasteable, and every output shown is the verbatim transcript of a real run — regenerated by the book’s own build, which executes the printed commands and records the tool revisions and run date beside each transcript in the source (one commodity arm64 box; your timings will differ, your bytes will not).

### 7.1 The claim

FrozenLake is Gymnasium’s introductory gridworld: a  $4 \times 4$  frozen lake, sixteen states, four actions, a start square, four holes, one goal. In the *slippery* variant each action moves as intended with probability  $\frac{1}{3}$  and slips to a perpendicular neighbour otherwise, so every trajectory is a random walk biased by the policy, and every reach probability is a rational number whose denominator is a power of three.

The claim to certify: *the recorded policy reaches the goal within ten steps with probability at least*

$$p = \frac{815}{19683} \quad (\text{exactly; } 19683 = 3^9).$$

In QTSL’s vocabulary this is a probabilistic deadline claim,  $\mathbb{P}_{\geq p}[F_{\leq 10} \text{ goal}]$  — the same claim species whose full campaign (deadline ladders, dyadic rounding lanes, a family of environments) the Experiments chapter (chapter 2) re-derives from committed artefacts. One disclosure before we start: 815/19683 happens to be the exact optimum of the ten-step problem, and the producer knows it (it runs an independent floating-point sweep as a cross-instrument), but the certificate below asserts only the lower bound. What a certificate *checks* is always narrower than what its producer *knows*; keeping that line crisp is most of the craft, and the rest of this book is about the vocabulary for widening what is checked.

### 7.2 Setup

Three repositories, cloned side by side. `maitria-coalition/mtk` is the producer toolkit and the only thing you build: a Rust toolchain is its one hard prerequisite, and a GPU is optional throughout (the accelerated lanes load CUDA dynamically and fall back to the same exact host arithmetic — same script, same answers). `maitria-coalition/qtssl-experiments` is the reproduction spine — committed certificates, mutant batteries, receipts — cloned for its bytes; nothing to build. `maitria-coalition/qtssl` is the repository this book ships in: the theories, the machine-checked soundness developments, and the small reference instruments; nothing to build there either — the reader you will run in section 7.5 is dependency-free TypeScript on bare Node  $\geq 22$ .

```
git clone https://github.com/maitria-coalition/mtk
cargo build --release --manifest-path mtk/Cargo.toml
git clone https://github.com/maitria-coalition/qtssl-experiments
git clone https://github.com/maitria-coalition/qtssl
```

## 7.3 Produce

```
$ mtk/target/release/gymdp produce --out out --only frozenlake-4x4
frozenlake-4x4: exact T=100 bound = 0.7441903 [0.005s]
receipts -> out/producer_receipts_rust.json (9 certs, 0 compared, 0 mismatched) [0.005s total]
```

Nine certificates in milliseconds: the exact deadline ladder  $T \in \{5, 10, 25, 50, 100\}$  and a ladder of dyadic-rounded companions. (The 0.7441903 on the log line is a float *rendering* of the  $T=100$  bound, printed for the human reading the log; no float appears inside any certificate.) The document this chapter cares about is the deadline-10 certificate, `out/frozenlake-4x4/exact_T10.cbor`: 2,511 canonical bytes carrying the claim, the transition kernel as 148 rows of exact rationals, the recorded policy, and the eleven-by-sixteen value table — the complete witness data from which everything asserted can be re-derived by something that has never met the producer. It is also, byte for byte, the artefact committed in the experiments repository (sha256 prefix 66040e59, either way — compare against `qtsl-experiments/gymnasium/out/frozenlake-4x4/`). In this toolkit “reproduces the published artefact” is a statement about equality of bytes, not about agreement to some tolerance — which is what lets a checker’s verdict on the committed corpus and on your fresh production be the same fact.

One vocabulary correction before anyone checks anything, because the file extension just lied to you: this toolkit has no certificate file format. In the shipped architecture a certificate is ordinary database content — rows of a signed, content-addressed geolog commit, carried on the same rails as every other fact the system stores (the Executive Summary’s one-set-of-bytes, three-consumers stance — chapter 1; the Geolog part of this book is that database’s own manual). The theory defining those rows is in this repository, as `data: theories/Qtsl.geolog`, with its drift-alarmed battery beside it. The sidecar file you just wrote is the producer’s local carriage for that content — and the reproduction spine already commits this family both ways: the sidecar artefacts, and the same certificates sealed as rows of signed geolog commits in a `g1c/` directory beside them, with content identity between the two forms re-proven at every production. What survives every carriage is the standard the content answers to: a self-contained document of exact numbers, complete enough to re-derive every obligation, with no side channel to consult. You will read a certificate directly as rows, in plain text, two sections from now.

## 7.4 What a checker re-derives — and who checks

Any checker of this certificate, by any route, re-derives four obligation families from the document alone, in exact rational arithmetic:

- *well-formedness* — kernel probabilities lie in  $[0, 1]$  and sum to one per state–action pair; indices are in range;
- *boundary* — the value table’s final row vanishes off the goal set;
- *step* — each table entry is dominated by its expected successor value under the recorded policy’s action,  $V_t(s) \leq \mathbb{E}[V_{t+1}]$ ;
- *claim* —  $p \leq V_0(\text{init})$ .

Chained, these force the table to be a genuine under-approximation of the reach probability, so the claim follows from the bytes alone. Nothing the producer computed is taken on trust; the producer’s speed lives entirely upstream of the certificate. This division — fast untrusted search, small independent replay — is the toolkit’s organizing principle, and it is why the producer is allowed to be a GPU-accelerated, float-assisted, LLM-advised anything: a wrong producer costs a rejected certificate, never a false verdict.

Who does the re-deriving is a deliberately plural answer: one set of bytes, three consumers, each trading speed against the size of what you must trust (the Executive Summary carries the full stance). Their standing for *this* certificate, dated to this preview:

- *The geolog engine* — the fast path: certificate rows chased, monitored, and enforced like any other rows; engineering-grade assurance, deliberately no mechanized soundness story. This certificate’s row form — one sealed commit carrying the system, the specification, the judgment, and the proof content — is committed in the reproduction spine beside the sidecar artefacts, and the fast-path

checker re-verifies all four obligation families from the sealed rows alone: one make target re-checks the family's containers against their manifest, expected verdicts included, and the Experiments chapter's timing tables carry the fast-path column from real runs.

- *qtslab* — the certified checker: a standalone binary whose end-to-end soundness theorem (machine-checked in HOL4, grounded through the CakeML verified compiler in the Arm instruction set's semantics) says that if it prints ACCEPT on a claim followed by an arbitrary stream of bytes, the claim is semantically true. Precisely: acceptance binds to validity of the accepted sequent under every admissible valuation of the hypernet the certificate itself declares (chapter 10) — the Bernstein Logic fragment, which is all its input format carries today; a certificate reaching beyond that fragment is refused, not silently accepted (the Executive Summary is explicit about the scope: the boundary is *qtslab*'s, not the row vocabulary's). A demonstrated route, assembling now: the same compiler-and-theorem recipe has already carried a complete predecessor checker to a machine-code soundness theorem on real hardware — and that proven chain, re-run on its native ISA during this chapter's verification campaign, verified campaign fixtures in single-digit seconds each and refused all six mutants put to it.
- *The Lean route* — the same bytes reconstructed as a theorem inside Lean's kernel, by reflection; expected to be too slow for machine-scale certificates, and kept in the roster to *measure* that rather than assert it. It has been measured, twice, across two certificate-format generations: the Experiments chapter carries the scaling law, the admission gate fitted to it, and the expectation confirmed with its boundary stated.

No transcript in this section, on purpose: for the certified and Lean routes, printing one would mean pretending they are further along for this family than they are, and a book whose subject is checkable evidence does not get to fake its own. The fast path is runnable today — the reproduction spine's make target re-checks this family's committed containers, mutant battery included, whenever you like. The checking you can run *and read whole* is the next section's, on a certificate family whose rows are committed as plain text.

## 7.5 Break one

A checker that cannot say *no* teaches nothing. The FrozenLake family's sealed rows come with a committed mutant battery of their own, but a sealed container is built for checkers' eyes; the published-table campaign additionally stores its certificates as plain-text row listings you can open and read — so we break one of those, and the lesson transfers intact.

That campaign (section 2.3 of chapter 2) re-derived a peer-reviewed neural-certificate results table in exact arithmetic, and *faulty\_uniform* is its most instructive row: the one whose exact two-sided bracket showed a published “sound” bound to be no bound at all. Its upper-bound certificate is committed as rows in the reference listing notation — plain text you can open:

```
$ cd qtsl-experiments/supermartingales/out/sec3
$ sed -n '62p' certs/faulty_uniform/bound.rows.txt
op[ 59] funa fun=G<stack/TConst> args=[q:16524448999/8406920828251] value=L58
```

One hundred and twelve rows like this one — sorts, exact simplex coordinates, Bernstein coefficients, and the proof structure that cites them — resolving their names against pinned theory listings. Beside the corpus, the book's own repository carries a reference reader and checker for this certificate family: three TypeScript files, zero dependencies, readable in an afternoon (*qtsl/checkers/supermartingales*). Point it at the certificate, and at the campaign's committed mutant of the same certificate:

```
$ CHK=../../../../qtsl/checkers/supermartingales
$ node --experimental-strip-types $CHK/check_rows.ts \
  --stack $CHK/theories/BLstack.rows.txt --cert $CHK/theories/BLCert.rows.txt \
  certs/faulty_uniform/bound.rows.txt mutants/faulty_uniform/M1_false_bound.rows.txt
ACCEPT certs/faulty_uniform/bound.rows.txt proof_nodes=3
REJECT mutants/faulty_uniform/M1_false_bound.rows.txt berncone.coeff
```

Now break it yourself. The certificate survives a coefficient perturbed in the sound direction — slack is real, and the checker is not grading effort — so nudge the coefficient you just looked at the *other* way, by

exactly 1/8,406,920,828,251:

```
$ sed 's|448999|449000|' certs/faulty_uniform/bound.rows.txt > my_broken.rows.txt
$ node --experimental-strip-types $CHK/check_rows.ts \
  --stack $CHK/theories/BLstack.rows.txt --cert $CHK/theories/BLCert.rows.txt \
  my_broken.rows.txt
REJECT my_broken.rows.txt berncone.coeff
```

Two things to notice. The refusal is *witness-shaped*: the identifier names the violated obligation family (membership of the certificate polynomial’s Bernstein coefficients in the claimed cone), and the checker’s failure record beneath it carries the offending coefficient’s index and exact value, because failure geometry is routing advice for whoever must repair the certificate. And the committed mutant battery is the checker’s obligation list made executable — the battery, not the prose, gives the operational statement of what an ACCEPT means (the Cookbook develops this discipline as a pattern: section 19.3 of chapter 19).

One label, to keep the trust story crisp: the program you just ran is a *reference instrument* — the reproduction spine’s own exactness oracle, differentially pinned against an independent implementation across the campaign corpus and its mutant battery — and deliberately *not* one of the three checking routes. Its job is to make “the battery must refuse” an executable fact, and to let you watch obligations re-derived from rows alone through code small enough to read whole. When the stakes are real, the same rows go to the routes of section 7.4.

## 7.6 The same loop from TypeScript

The producer toolkit’s TypeScript surface, @maitria/mtk, ships as the ts/ package of the maitria-coalition/mtk repository (napi bindings over the same Rust core — there is one implementation of every construction, and TypeScript reaches it in-process). The shipped install route is the checkout you already have from section 7.2 — it requires a Rust toolchain, because the native addon builds from source:

```
cd mtk/ts && npm install && npm run build:native # Rust toolchain required
cd ~/your-project && npm install ../mtk/ts # npm links the package
```

(@maitria/mtk on the npm registry: arriving; the checkout path above is the shipped route at this preview.) Node ≥ 22.18 throughout — the package’s declared engines floor. The producing half of the loop again, this time constructing the certificate yourself — and building the mutant with your own hands:

```
// quickstart.ts – produce the FrozenLake certificate, then break it.
import { writeFileSync } from "node:fs";
import { dp } from "@maitria/mtk";

// The environment: Gymnasium's FrozenLake-4x4, slippery (success rate 1/3).
const model = dp.frozenlake("4x4", true, "1/3");

// The claim: exact finite-horizon dynamic programming to deadline T = 10.
const res = model.exact(10);
console.log("claim:", res.bound);

// Emit the certificate: canonical bytes, encoded by the Rust core.
const meta = {
  envId: "FrozenLake-v1",
  mdpName: "frozenlake-4x4",
  repoSha: dp.gymSha(),
  kwargs: "",
};
const bytes = res.emit(meta);
writeFileSync("exact_T10.cbor", bytes);
console.log("wrote exact_T10.cbor:", bytes.length, "bytes");

// Now break it: raise one interior value-table entry to 1.
// (docMutate refuses unless the unmodified doc round-trips byte-identically.)
const broken = dp.docMutate(bytes, [
  { path: ["table", 1, 0, 0], setHex: "01" },
```

```

    { path: ["table", 1, 0, 1], setHex: "01" },
  });
writeFileSync("mutant_table_raised.cbor", broken);
console.log("wrote mutant_table_raised.cbor:", broken.length, "bytes");

$ npx tsx quickstart.ts
claim: 815/19683
wrote exact_T10.cbor: 2511 bytes
wrote mutant_table_raised.cbor: 2509 bytes

```

The bytes you just constructed are byte-identical to the ones `gymdp` emitted in section 7.3, and therefore to the committed corpus artefact — the same `sha256`, all three ways. And you now hold what the mutant-battery discipline calls a matched pair: golden bytes, and a variant broken in precisely one field (the mutation seam checks that the unmodified document round-trips byte-identically before it will perform surgery, so your mutant differs from the golden bytes exactly where you aimed). The corpus commits this family's battery beside the artefacts, expected verdicts in the receipts — and the sealed row form carries a second battery of thirteen row-form mutants, each one's expected verdict and violated obligation named in its manifest. You already know from section 7.5 what a refusal at the row level looks like.

## 7.7 What you just did, and where it goes

You ran the trust architecture of this book once, in miniature: an untrusted producer constructed complete witness data fast, and reproduced a published artefact to the byte; you read the obligations a checker must re-derive, and the current state of the three routes that re-derive them; and on the published corpus you watched an exact-arithmetic reader accept a certificate stored as rows and refuse its mutants — including yours — with the violated obligation named. Three widenings from here:

- *Producing for real.* The Tutorial (chapter 8) teaches the producer's craft on the TypeScript surface: exactness as a type property, propose-then-dispose, specs and judgments, derivation rows from the Rule Catalogue, and refusal-driven iteration.
- *Richer claims.* Deadline reachability is one species. The Rule Catalogue (chapter 15) is the full inventory of certified inference; the fabrication-line worked example (chapter 17) runs the stack over a model with batching, failures, and re-entrant flow.
- *Graded checking, priced.* The three routes of section 7.4 are not interchangeable, and the Experiments chapter's design keeps their price differences public: three wall-clocks per experiment — produce, check-fast, check-certified — each measured on its route's native substrate, so the cost of each grade of assurance stays a published number rather than a promise.

# Tutorial: Producing Proofs with MTK

---

The Quick Start (chapter 7) had you run one producer end to end. This tutorial teaches you to *be* one: to construct specifications, judgments, and derivation rows with the producer toolkit’s TypeScript surface, to read refusals as instructions, and to leave behind artefacts an independent checker will accept. The intended reader is either a capable agent doing verification work or a verification-literate human; the text is written for the human, and the agent will cope. Every listing below runs as printed, and every output shown is the verbatim transcript of a real run: the book’s build regenerates these transcripts by executing the printed listings, and each one’s source records the exact tool revision and date it ran (one commodity arm64 box).

## 8.1 The producer’s stance

Three commitments shape every surface you are about to touch, and every signature below answers to one of them:

1. *Nothing here is trusted.* No function in the producer toolkit renders a verdict anyone should believe. Validation helpers exist — they save round-trips — but they are preflights; the checker’s independent replay of your artefact is the only authority. A wrong producer costs a rejected certificate, never a false verdict, which is why the producer is allowed to be fast, heuristic, and as clever as it likes.
2. *Witnesses always, never bare verdicts.* Any operation that can fail for a content reason returns the geometry of the failure — which coefficient, which cell, which field, which subformula — because failure geometry is routing advice for your next attempt. You will spend much of your producing life reading refusals; the API is designed so that this is pleasant.
3. *Exactness is a type property, not a discipline.* Data that certificates cite is exact — arbitrary-precision rationals and integers — and lives in types that say so. Floating-point data proposes and routes but cannot cross into the exact world by accident: the one bridge is an explicit snap, and what comes out of it is a new proposal to certify, not a blessed float.

## 8.2 Setup

The TypeScript package `@maitria/mtk` lives in the `maitria-coalition/mtk` repository as its `ts/` package over a native addon crate (the front matter’s preview note applies to the package’s migration state at this preview). It is two layers by design: a deliberately flat native boundary (napi bindings over the Rust core — there is exactly one implementation of every construction, and it is the Rust one; the differential receipts shipped with each binding prove the TypeScript-driven path emits bytes identical to the Rust-driven one) and an *arrangement* layer that owns all public idiom — discriminated unions, branded types, readonly records. You will only ever touch the arrangement.

```
git clone https://github.com/maitria-coalition/mtk
cd mtk/ts && npm install
npm run build:native # cargo builds the addon; CUDA optional, loaded at runtime
```

A Rust toolchain is required — the native addon builds from source. To use the package from your own project, install it by path (`npm install ../mtk/ts`; `npm` links the checkout), which is the shipped route at this preview; `@maitria/mtk` on the npm registry: arriving. Node  $\geq 22.18$  (the package’s declared engines floor); producer scripts run under `npx tsx` with no build step of their own. Boxes without a GPU run the same scripts and get the same answers — acceleration changes wall-clock, never content.



## 8.3 Exact numbers, and the one bridge from float

The class the certificates cite is `Q`: immutable, bigint-backed exact rationals with a canonical text form "`n/d`". Everything about its API enforces stance 3:

```
// exactness.ts – class-E data is Q; number never crosses the boundary.
import { Q, lp } from "@maitria/mtk";

console.log(Q.from("0.1").add("0.2").toString());           // exact decimal parse
console.log(Q.from("-2.04e-2").toString());                 // scientific, exact
console.log(Q.of(815, 19683).sub(Q.of(1, 27)).toString());

try {
  Q.from(0.1);                                              // refused: ambiguous
} catch (e) {
  console.log((e as Error).message);
}

// The one bridge from float to exact: an explicit snap. The snapped
// value is a new PROPOSAL to certify, never a blessed float.
console.log(lp.rationalize(0.3333333333333333, 1000).toString());
console.log(lp.rationalize(Math.PI, 1000).toString());

$ npx tsx exactness.ts
3/10
-51/2500
86/19683
Q.from(0.1): non-integer numbers are ambiguous – pass "n/d" text, a bigint, or Q.of(n, d)
1/3
355/113
```

Decimal *text* parses exactly (`"0.1"`  $\mapsto$   $1/10$ : the string denotes a rational, unambiguously), but the *double* `0.1` is refused — it denotes  $3602879701896397/2^{55}$ , and pretending otherwise is how exactness-as-discipline projects fail. When you genuinely hold a float and want a rational, you say so: `rationalize` snaps to the best approximation under an explicit denominator budget (note  $\pi \mapsto 355/113$ , as it should). The budget is a knob you will meet again: coarser snaps make smaller, cheaper-to-check certificates, and no snap can make a false claim true — the checker decides either way.

## 8.4 A first certified fact

The smallest genuinely useful certified fact in this toolkit: the sign of a polynomial on a region. Bernstein coefficients of  $p$  over a simplex bound  $p$  there, so an all-positive coefficient table *is* a sign certificate — data whose correctness an independent checker re-derives by arithmetic alone, no cleverness required.

```
// sign.ts – certify the sign of a polynomial on cells of a triangle.
import { Q, poly } from "@maitria/mtk";

// p(x, y) = x^2 + x*y - y + 1/3
const p: poly.Term[] = [
  { exps: [2, 0], coeff: 1 },
  { exps: [1, 1], coeff: 1 },
  { exps: [0, 1], coeff: -1 },
  { exps: [0, 0], coeff: Q.of(1, 3) },
];

// the unit triangle, split at the midpoint of its hypotenuse
const mid = [Q.of(1, 2), Q.of(1, 2)];
const t = poly.simplexBern({
  poly: p,
  cells: [
    [[0, 0], [1, 0], mid],
  ],
});
```

```

    [[0, 0], mid, [0, 1]],
  ],
  degree: 4, // elevate past deg(p): tighter enclosures
  device: "cpu", // pin the host rung: same integers on any box
});

console.log(`ran on ${t.device}; ladder served F/M/host = ${t.served}`);
for (let b = 0; b < t.cells; b++) {
  const [lo, hi] = t.enclosure(b);
  const verdict =
    lo.sign() > 0 ? "p > 0 CERTIFIED"
    : hi.sign() < 0 ? "p < 0 CERTIFIED"
    : "sign open (subdivide)";
  console.log(`cell ${b}: p in [${lo}, ${hi}] - ${verdict}`);
}

$ npx tsx sign.ts
ran on cpu(hostref); ladder served F/M/host = 0,0,2
cell 0: p in [1/6, 4/3] - p > 0 CERTIFIED
cell 1: p in [-2/3, 1/3] - sign open (subdivide)

```

Read the first line closely, because it is stance 3 at work in the machinery. The enclosure pipeline is exact end to end — integer numerators over a shared per-cell denominator — and it runs as a *precision ladder*: fixed-width device tiers serve the cells they can handle without overflow, and any cell they cannot is flagged and re-derived by an arbitrary-precision host replay. Every rung computes the same exact integers; the ladder trades width for speed, never exactness for speed. The listing pins device: "cpu" so that both cells are served by the host rung (0,0,2) and your transcript matches this one on any box, GPU or none; drop the pin (the default is "auto") and a free GPU takes the cells instead — on this box the package's own example reported `cuda:NVIDIA GB10 ... ladder served F/M/host = 2,0,0`. Same cells, same enclosures, same certificate — the transcript differs only in who did the arithmetic.

Cell 1 is the more instructive half of the output: the enclosure straddles zero, so the honest answer is *sign open*, with subdivision as the routing advice. Bernstein enclosures converge under subdivision and degree elevation, so "open" is an invitation to spend more, not a dead end — and when the budget runs out before the sign closes, that too can be certified rather than shrugged (the Cookbook's honest-stop pattern, section 19.2 of chapter 19).

## 8.5 Propose, then dispose

The two-tier discipline in one screen: a linear program solved twice, once by the float lane (fast, unciteable), once by the exact lane (the only tier a certificate may cite).

```

// two-tier.ts – the float lane proposes; the exact lane disposes.
import { lp } from "@maitria/mtk";

// minimize x0 + x1 subject to x0 + 2*x1 = 1, 3*x0 + x1 = 1, x >= 0
const cols = [{ cost: 1 }, { cost: 1 }];
const rows = [
  { lo: 1, hi: 1, cols: [0, 1], coefs: [1, 2] },
  { lo: 1, hi: 1, cols: [0, 1], coefs: [3, 1] },
];

// tier F: HiGHS, floats – fast, unciteable; duals are routing advice
const f = lp.floatSolve(cols, rows);
console.log("float:", f.status, f.objective, "duals", f.rowDuals);

// tier E: exact Bland simplex over Q – the only tier a certificate cites
const e = lp.simplexMin([1, 1], [[1, 2], [3, 1]], [1, 1]);
if (e.status === "opt") {
  const at = e.x.map(String).join(", ");
  console.log("exact:", e.value.toString(), "at x =", at);
}

```

```

    }

    $ npx tsx two-tier.ts
    float: optimal 0.6000000000000001 duals [ 0.4, 0.2 ]
    exact: 3/5 at x = 1/5, 2/5

```

The float tier even performs its own indictment: it reports `0.6000000000000001` for a quantity that is  $3/5$ . On a toy problem the exact tier could simply run alone; the discipline earns its keep at scale, where the float lane's job is to find *which* exact computation is worth paying for — its duals say which constraints bind, its solution says where to warm-start, its speed lets you sweep a thousand candidates to find the one worth certifying. The full pattern, with the case studies that shaped it, is the Cookbook's opening chapter (section 19.1 of chapter 19).

## 8.6 Stating claims: specs and judgments

So far we have certified facts the toolkit already knew how to phrase. To produce proofs *about systems*, you state the claim yourself, in QTS: a specification is a record of claim components — each a formula over a system's observables — and a judgment is the record that pairs a specification with a system (or a region, or a pair of specifications — one record species per judgment kind). Both are constructed here as plain data, preflighted in Rust, and emitted as versioned artefacts through the same canon encoder that carries every other certificate.

The Quick Start's FrozenLake claim, stated properly this time:

```

// spec.ts – state the FrozenLake claim species as a specification.
import { spec } from "@maitria/mtk";

// One observable (width 1); the goal region is { x >= 15 }.
const x = spec.unwrap(spec.polyVar(0, 1));
const goal = spec.polyGe(15, x);

// P_{>= 815/19683} [ F_{<=10} goal ] – reach the goal within 10 steps:
// a safety-face component (SProbGE) in the record's forall list.
const phi = spec.probGe("815/19683", spec.deadline(10, goal));

const record = spec.unwrap(spec.build({ name: "reach-goal-by-10", forall: [phi] }));
const artifact = spec.unwrap(spec.encode(record));
console.log(artifact.schema, "-", artifact.bytes.length, "bytes");

// Round-trip: decoded data is data – re-validate before trusting it.
const back = spec.unwrap(spec.decode(artifact.bytes));
console.log("decoded:", back.spec.name, "v =", back.v);

// Now ask for probability 3/2 and let the preflight refuse.
const bad = spec.probGe("3/2", spec.deadline(10, goal));
const r = spec.preflightForall(bad);
if (!r.ok) {
  console.log("refused:", r.refusal.kind);
  if (r.refusal.kind === "probRange")
    console.log(` at ${r.refusal.at.text}: p = ${r.refusal.p}, outside (0, 1]`);
}

$ npx tsx spec.ts
maitria.qtsl.spec.v1 - 179 bytes
decoded: reach-goal-by-10 v = 0
refused: probRange
  at (root): p = 3/2, outside (0, 1]

```

Three habits to take from one screen. First, the wire is visible and versioned: every emitted artefact carries a schema string (`maitria.qtsl.spec.v1`) and a version, and the bytes are produced by the Rust canon encoder — TypeScript never re-implements an encoding, which is why cross-language byte drift is structurally unrepresentable rather than tested away. Second, decoded data is data: decode gives you back

a record, and if you received the bytes from someone else you re-run `build` before leaning on it. Third, the refusal: it is a discriminated value (`kind: "probRange"`), it carries the address of the offence inside the formula tree (`((root) — addresses are relative to the node you preflighted, and record-level checks come with full paths like forall[0].prob.path), and it carries the offending exact value as canonical text (p: "3/2"). Producer code branches on refusal kinds the way it branches on any other data; nothing is buried in exception strings.`

Judgments follow the same grammar one level up: the judgment namespace has one builder per judgment kind — satisfaction, region satisfaction, spec equivalence, distance, and the rest — each taking branded citation handles rather than loose strings, each emitting a versioned artefact. The claim record is deliberately boring; the interesting objects are the derivations that discharge it, which is where the rules come in.

## 8.7 Derivation rows: the Rule Catalogue as an API

Every inference rule in the Rule Catalogue (chapter 15) is available as a typed row builder: premises as typed citations, payload fields exact, side conditions enforced at construction, canonical bytes out. A proof, in this toolkit, is a table of such rows — which is what lets a checker replay it row by row, and what lets a database engine store, merge, and re-derive it like any other data.

The smallest interesting rule is a good first row. `QUnionBound` takes two exception-mass bounds and concludes their sum — the workhorse glue for assembling per-obligation failure budgets into a system-level one:

```
// rule-row.ts – a Rule Catalogue row: cite premises, get the conclusion.
import { rules } from "@maitria/mtk";
const { qtslCore } = rules;

// QUnionBound: from two exception-mass bounds, conclude their sum.
const row = qtslCore.unionBound({
  e1: qtslCore.judgmentRef("qregionsat", 0), // typed premise citations
  e2: qtslCore.judgmentRef("qregionsat", 1),
  a: "1/8",
  b: "1/16",
});
if (row.ok) {
  console.log(row.value.schema, "-", row.value.cbor.length, "bytes");
  console.log("concluded bound:", row.value.bound.toString());
}

// A negative mass is refused with the witness attached.
const bad = qtslCore.unionBound({
  e1: qtslCore.judgmentRef("qregionsat", 0),
  e2: qtslCore.judgmentRef("qregionsat", 1),
  a: "-1/8",
  b: "1/16",
});
if (!bad.ok) {
  console.log("refused:", bad.refusal.kind);
  console.log(" " + bad.refusal.message);
}

$ npx tsx rule-row.ts
maitria.qtsl.union_bound.v0 - 58 bytes
concluded bound: 3/16
refused: signConstraint
QUnionBound.a: sign/range constraint violated by -1/8
```

Notice who computed `3/16`: not this script. Derived conclusions come back from the Rust core's own concludes logic, so the row you emit and the arithmetic it claims were produced by the same implementation the differential receipts pin — your script assembles and cites, it never does load-bearing arithmetic. Notice also the premise handles: `judgmentRef("qregionsat", i)` is a branded type, and a rule whose premise slot

wants a region-satisfaction judgment will not accept a plain number, a proof reference, or a judgment of the wrong kind. The edge discipline of the proof calculus is enforced by the type checker before the checker ever sees your bytes.

When you need more than glue — safety batteries, deadline batteries, Streett bundles, Morse decompositions, composition rules — the pattern never changes: find the rule in the Rule Catalogue, which documents its premises, payload, and side conditions alongside its mechanization status; call its builder; read refusals until the row constructs. The catalogue chapter is the map and stays the map; this tutorial deliberately teaches only the door.

## 8.8 Certificates are documents: read one

The Quick Start ended by mutating a certificate through the generic document seam. Here is the reading half of that seam, on the same bytes — because the fastest way to understand what a checker re-derives is to look at what the producer actually wrote down:

```
// anatomy.ts — a certificate is a document; read it before you trust it.
import { readFileSync } from "node:fs";
import { dp } from "@maitria/mtk";

const bytes = readFileSync("exact_T10.cbor");
const doc = dp.docJson(bytes, []) as Record<string, unknown>;
console.log("fields:", Object.keys(doc).join(" "));
console.log("deadline:", dp.docJson(bytes, ["deadline"]),
  " states:", dp.docJson(bytes, ["ns"]),
  " goal:", dp.docJson(bytes, ["goal"]));
console.log("claim:", dp.docJson(bytes, ["claim"])); // bignum pair, hex
console.log("kernel[0]:", dp.docJson(bytes, ["kernel", 0])); // s a s' p_num p_den

$ npx tsx anatomy.ts
fields: claim deadline env_id goal init kernel kwargs lane mdp_name ns policy repo_sha table v
deadline: 10 states: 16 goal: [ 15 ]
claim: [ '0x032f', '0x4ce3' ]
kernel[0]: [ 0, 0, 0, '0x02', '0x03' ]
```

Everything a checker of this certificate must re-derive — the Quick Start's four obligation families — is visibly *here*: the claim ( $0x032f/0x4ce3 = 815/19683$ ), the kernel as exact rationals (the first row: from state 0 under action 0, probability  $2/3$  of staying put — two of its three equally likely moves bounce off the corner), the policy, the value table. A certificate contains no argument and no trust requests; it contains the numbers that make the argument checkable. When you produce a new certificate species, this is the standard to hold your bytes to: could a stranger's checker re-derive every obligation from the document alone?

That standard is what survives every carriage the document rides. These particular bytes travel as a sidecar file; the reproduction spine commits the same certificate a second way, as rows of a signed, content-addressed geolog commit — ordinary database content whose row vocabulary is `theories/Qts1.geolog` in this repository, with no certificate file format of its own — and you have already read certificate rows directly, as plain text, in the Quick Start's break-it section. What the generic document seam gives you here is the producer-side reading of the content in its canonical byte carriage; the anatomy lesson is identical on either, because completeness reads the same from any carriage.

The mutation seam (`dp.docMutate`) is the same document surface pointed the other way, and it exists because a mutant battery is part of a certificate lane's deliverable, not an optional extra: every claim species ships with mutants its checker must refuse, and a witness that stays green when the claim is broken was never a witness (section 19.3 of chapter 19). The seam checks that the unmodified document round-trips byte-identically before it will perform surgery, so your mutants differ from the golden bytes in precisely the field you broke.

## 8.9 The shape of a real producer program

Scale the last five sections up and you have the actual production programs behind the Experiments chapter. The FrozenLake producer you ran in the Quick Start is a few hundred lines organized along this tutorial's spine, and reading it (the gymdp producer in the `maitria-coalition/mtk` repository, with its receipts and companion drivers in `maitria-coalition/qtsl-experiments`) is the recommended next hour. The anatomy the campaign's producers share:

- *a configuration grid*, not a one-off — environments  $\times$  deadlines  $\times$  rounding budgets, because certificates are cheap once the lane exists and ladders are how cost curves get published;
- *an independent cross-instrument* — a plain floating-point value sweep, kept structurally separate from the exact lane, with an explicit agreement tolerance; it certifies nothing, and it catches producer bugs before any checker has to;
- *exact dominance assertions between lanes* — each dyadic-rounded certificate is checked against its exact sibling ( $0 \leq \text{exact} - \text{dyadic} \leq T \cdot 2^{-k}$ , the gymdp source's own *exact dominance* assertion), so even the cheap lane's error is a theorem, not a hope;
- *emission through the canon encoder, then the mutant battery, then receipts* — machine-readable production receipts (bounds, byte counts, timings, mutant verdicts) written beside the artefacts, because a reproduction spine is built from receipts, not recollections.

When a step refuses mid-program, the refusal's geometry picks the escalation: subdivide the cell, raise the degree, spend a bigger denominator budget, climb from the closed form to the LP to the heavier rule — cheapest instrument first, and certify the honest stop when the budget genuinely ends (sections 19.5 and 19.2 of chapter 19).

## 8.10 Working an obligation workspace

Everything so far drove the toolkit imperatively: your script decided each call, held the intermediate state in variables, and threw both away on exit. That is the right shape for a five-minute production and the wrong shape for one that runs long enough to be interrupted, plays several instruments against one claim, or is worked by something other than its author. The toolkit's session surface exists for that regime. Its bookkeeping object is the *obligation workspace*, and its vocabulary is four nouns:

- an *obligation* is a claim to be certified — a statement of a registered claim kind — with a status: open, discharged, or refuted;
- a *move* is one typed action from the obligation's menu. The menu is the claim kind's to offer: each move declares its parameter schema, a cost hint (cheap to expensive — advice for sequencing, never a gate), and what it may do: discharge, refute, or spawn;
- an *attempt* is one executed move: the outcome, the wall time, and either a *witness* (for discharged or refuted — the exact-tier artefact that justifies the verdict) or *evidence* (for inconclusive — failure geometry, which by now you know to read as routing advice);
- the *event log* is the whole truth: the workspace is a pure fold of an append-only event list, so replaying the log reconstructs the state without re-running any kernel.

Two disciplines are enforced by the workspace itself rather than requested of its users. First, stance 2 is mechanized at the recording boundary: a claim kind that returns a verdict without a witness is refused on the spot — the workspace will not record a bare verdict, so its bookkeeping can never get ahead of the artefacts that justify it. Second, structural moves *spawn*: a decomposition move creates child obligations that jointly cover the parent's claim, and from that moment the parent's status is nobody's to set — it closes only by propagation. All children discharged closes the parent (via `children`); one refuted child refutes it, since the child covers part of the claim. You work the leaves; the workspace does the lifting.

Here is the whole model in one sitting, on a claim small enough to watch:  $x(1 - x) - 1/4 \leq 0$  on  $[0, 1]$  — true, with equality at  $x = 1/2$ . (The package is `@maitria/mtk-session`, the `maitria-coalition/mtk`



repository's session/ package, beside the producer package; install it by path like the producer package in section 8.2. It runs its TypeScript sources directly — Node  $\geq 22.18$ .)

```
// session.ts – work an obligation workspace by hand: you are the strategy.
import { Workspace, polyKind } from "@maitria/mtk-session";

// The claim:  $x(1-x) - 1/4 \leq 0$  on  $[0, 1]$  – true, with equality at  $x = 1/2$ .
const ws = new Workspace([polyKind]);
const root = ws.pose({
  kind: "poly.nonpositive",
  poly: [
    { exps: [2], coeff: "-1" },
    { exps: [1], coeff: "1" },
    { exps: [0], coeff: "-1/4" },
  ],
  box: [["0", "1"]],
});
console.log(`${root.id}: ${ws.describe(root.id)}`);
console.log("menu:", ws.moves(root.id).map((m) => `${m.name}/${m.cost}`).join(" "));

// Cheapest first: one exact Bernstein enclosure of p over the whole box.
const r1 = ws.apply(root.id, "range");
console.log("range:", r1.attempt.outcome, JSON.stringify(r1.attempt.evidence));
console.log(" note:", r1.attempt.note);

// The enclosure straddles zero because p touches it at 1/2. Split there.
const r2 = ws.apply(root.id, "split", { axis: 0, at: "1/2" });
console.log("split:", r2.attempt.outcome, "->", r2.spawned.map((c) => c.id).join(", "));

// The parent is now out of our hands – the workspace says so:
try {
  ws.apply(root.id, "range");
} catch (e) {
  console.log("refused:", (e as Error).message);
}

// Work the leaves; each half certifies by one enclosure.
for (const child of r2.spawned) {
  const r = ws.apply(child.id, "range");
  console.log(`${child.id} range:`, r.attempt.outcome, JSON.stringify(r.attempt.witness));
}

// Nobody set the parent's status; the second discharge propagated it.
const done = ws.get(root.id);
console.log(`${root.id}: ${done.status} (via ${done.via})`, JSON.stringify(done.witness));

// The workspace is a fold of its event log: replay rebuilds it exactly.
const replica = Workspace.replay(ws.events, [polyKind]);
const same = JSON.stringify(replica.snapshot()) === JSON.stringify(ws.snapshot());
console.log("replayed", replica.events.length, "events; state identical:", same);

$ npx tsx session.ts
ob1: (-1)*x^2 + (1)*x + (-1/4) <= 0 on [0, 1]
menu: range/cheap probe/cheap split/cheap search/expensive
range: inconclusive {"enclosure":["-1/4","1/4"]}
note: enclosure straddles 0; split, search, or probe near the suspected maximum
split: spawned -> ob2, ob3
refused: ob1 has open children; work the leaves (its status resolves by propagation)
ob2 range: discharged {"enclosure":["-1/4","0"]}
ob3 range: discharged {"enclosure":["-1/4","0"]}
ob1: discharged (via children) {"dischargedChildren":["ob2","ob3"]}
replayed 5 events; state identical: true
```

Four things in this transcript repay a close look.

*The straddle is real.* The enclosure on the whole box is  $[-1/4, 1/4]$ : the control polygon of a strictly concave parabola lies above it, so the degree-2 ceiling lands at  $1/4$  rather than at the true maximum 0. The note beside the evidence says what to do about it, and the interesting word in it is *where*: “near the suspected maximum.”

*The split point was knowledge.* Splitting at the touch point  $x = 1/2$  moves each half’s maximizer to an endpoint — and endpoint Bernstein coefficients are function values, so both child enclosures land their ceilings at exactly 0 (the transcript’s `["-1/4", "0"]`, twice) and discharge by one cheap move each. A midpoint-splitting heuristic would have done the same here by luck; on a claim whose touch point is off-center, choosing the split is a contribution only the mover can make. Hold that thought for one section.

*Nobody closed the parent.* The attempted third move on `ob1` was refused, in words that state the discipline; the final status arrived by propagation when the second child discharged, and the parent’s witness is the decomposition itself — `{"dischargedChildren":["ob2", "ob3"]}` — while the exact-tier artefacts live on the leaves.

*The log determines the state.* Five events: one pose and four attempts (the refused call appended nothing). The last attempt’s event carries two closures — the child’s, and the root’s by propagation — and replay applies recorded effects without re-executing any move, which is why the rebuilt workspace is identical even though nothing requires a kernel to be deterministic.

The daemon is this same object held behind a socket. `mtk-session serve` (run from the package checkout as `node src/main.ts serve` at this preview) holds one workspace and speaks newline-delimited JSON-RPC 2.0 over a unix domain socket, or over `stdio` for harnesses that spawn it as a child: one request per line, one response per line, seven methods (`session.info`; `obligation.pose`, `.list`, `.get`, `.moves`, `.apply`; `workspace.log` for the event tail). State lives in the daemon, so clients disconnect and reconnect freely, and errors carry structured codes rather than prose. JSON is a deliberate choice at this one boundary — the clients are foreign by construction: any agent runtime, any language, or a person with `nc` — and no certificate content crosses it: certificates land through the same canon encoding as everywhere else in this book, and the protocol carries menus, selections, and witness summaries.

## 8.11 The strategy layer

In the last listing, the workspace kept the books, the kernels computed the truth, and everything interesting in between — range first because it is cheap, split *there* because that is where  $p$  touches zero — was you. That seat has a name in the session surface: the *proposer*. A proposer is shown the open obligations, the evidence accumulated so far, and the move menus, and asked for one selection at a time. The Executive Summary promised this seat when it described agents driving tactic selection over an obligation workspace “with the strategy intelligence supplied by the reader as the language model of their choice” (chapter 1), and this section is that layer — shaped so that anything can sit in it. The opening page addressed this tutorial to a capable agent doing verification work; over the `serve` socket, that reader holds the seat personally.

The trust statement, precisely. A proposer chooses *which* exact computation runs next — nothing else. It cannot fabricate an outcome: a proposal outside the posted menu is rejected against the menu and the error is fed back; two consecutive failures end the loop with the workspace untouched. It cannot inject a witness: witnesses are constructed by move execution only, behind the no-bare-verdict guard of the previous section. And it cannot upgrade a shortfall into a discharge, because the comparisons happen in exact rational arithmetic inside the claim kinds. Stance 1 gains its corollary: because verdicts never come from the strategist, the strategist may be arbitrarily wrong — confused, overconfident, adversarial — at a cost denominated in budget, never in soundness. The same reason the producer was allowed to be fast and heuristic, applied one level up.<sup>1</sup>

The package ships two proposers, and the contrast between them is the demonstration. The *scripted* proposer is a deterministic cheapest-first ladder per claim kind: no key, no network, the CI path, and the

<sup>1</sup> This seat has a literature: Laurent and Platzer learn to refine proof-search strategies [152], and their oracular programming gives LLM-enabled software a modular foundation in which untrusted oracles answer through checked interfaces [153]; closest to this chapter’s workflow, Kabra and colleagues use language models to automate interactive theorem proving and control synthesis for hybrid games [128].

budget floor any cleverer strategy ought to beat. Here it drives the Quick Start’s claim (section 7.1 — the same claim; there you ran one producer end to end, here a workspace orchestrates a ladder of instruments):

```
$ node src/main.ts demo
mtk-session demo — the FrozenLake deadline claim, end to end

claim obl: FrozenLake-4x4 (slippery, success 1/3): recorded policy reaches the goal
  within 10 steps with probability >= 815/19683
strategy: scripted
[1] obl float_sweep -> inconclusive (2 ms) {"estimate":0.04140628969161203,
  "claim":0.04140628969161205,"marginEstimate":-2.0816681711721685e-17}
[2] obl dyadic {"kBits":24} -> inconclusive (0 ms) {"tier":"dyadic","kBits":24,
  "bound":"694681/16777216","shortfall":"24917/330225942528"}
[3] obl exact -> discharged (1 ms) {"tier":"exact","bound":"815/19683","claim":"815/19683",
  "certificate":{"bytes":2511,
  "sha256":"66040e596ece30efcbl500350396477f6e4f56d9af1b8f2896c30bece2fc391",
  "path":"out/frozenlake-4x4/exact_T10.cbor"}}
result obl: discharged (via exact)
witness: {"tier":"exact","bound":"815/19683","claim":"815/19683","certificate":{"bytes":2511,
  "sha256":"66040e596ece30efcbl500350396477f6e4f56d9af1b8f2896c30bece2fc391",
  "path":"out/frozenlake-4x4/exact_T10.cbor"}}
strategy ended: solved; moves used: 3
```

(Long lines are wrapped at JSON commas to fit the page; the package’s committed receipts carry both of this section’s transcripts unbroken.) The three attempts are one story told forward. The float sweep is the cross-instrument of section 8.9 in move form: its margin estimate,  $-2 \times 10^{-17}$ , is float for “the claim sits dead even with the optimum” — which the Quick Start disclosed in advance: 815/19683 is the exact ten-step optimum, so this claim has zero slack. The dyadic attempt is a sound certifying lane, and its evidence is exact: a lower bound of  $694681/2^{24}$ , short of the claim by exactly  $24917/(2^{24} \cdot 3^9)$ . The shortfall is structural, not a tuning failure: every reach probability in this model has a power of three for its denominator, the claim is such a number in lowest terms, and no dyadic rational equals it — so a rounded-down dyadic bound sits strictly below a zero-slack claim at every precision, and no kBits fixes it. The workspace records that gap as an exact rational, the evidence routes the escalation, and the exact tier discharges with zero slack, its witness carrying the certificate content — the same content, digest for digest, that the Quick Start produced.

One seam note at this preview: the demo’s remaining transcript lines, not shown above, hand the emitted certificate and a deliberately broken copy of it to an independent checker, which accepts the former and refuses the latter naming the violated obligation. We leave those lines out because that checking surface is mid-migration to consuming certificates the way the Executive Summary describes them — signed, content-addressed commit content, checked by any of the three routes — rather than as files handed to a dedicated binary. The workspace is indifferent to the migration: what it stores is a witness, and what a witness cites is content.

To seat a model instead of the script, supply an OpenRouter key in the environment (OPENROUTER\_API\_KEY; any chat-completions model works, and the router’s default is used if you name none). The model is shown the same view a client at the socket gets, and must answer with one JSON object selecting a move:

```
$ node src/main.ts demo --llm --model google/gemini-3.5-flash
mtk-session demo — the FrozenLake deadline claim, end to end

claim obl: FrozenLake-4x4 (slippery, success 1/3): recorded policy reaches the goal
  within 10 steps with probability >= 815/19683
strategy: llm:google/gemini-3.5-flash
... asking google/gemini-3.5-flash
[1] obl float_sweep -> inconclusive (1 ms) {"estimate":0.04140628969161203,
  "claim":0.04140628969161205,"marginEstimate":-2.0816681711721685e-17}
... asking google/gemini-3.5-flash
[2] obl exact -> discharged (1 ms) {"tier":"exact","bound":"815/19683","claim":"815/19683",
  "certificate":{"bytes":2511,
  "sha256":"66040e596ece30efcbl500350396477f6e4f56d9af1b8f2896c30bece2fc391",
  "path":"out/frozenlake-4x4/exact_T10.cbor"}}
result obl: discharged (via exact)
witness: {"tier":"exact","bound":"815/19683","claim":"815/19683","certificate":{"bytes":2511,
  "sha256":"66040e596ece30efcbl500350396477f6e4f56d9af1b8f2896c30bece2fc391",
  "path":"out/frozenlake-4x4/exact_T10.cbor"}}
strategy ended: solved; moves used: 2
```

The transcript's shape makes the argument. Shown the sweep's dead-even margin, the model — Gemini 3.5 Flash through OpenRouter, in the committed run — selected the exact tier at once: two moves where the ladder spent three, because it read evidence a fixed ladder cannot. Nothing depended on it being right. A worse selection would have cost one extra attempt; nonsense selections would have ended the loop after two rejections with the workspace untouched; and no selection of any kind could have moved a verdict, because routing is the one contribution a strategist can make here and the only one it is allowed. Same claim, same certificate content, same digest: strategy changes the path, never the bytes.

Past the demo, the seat generalizes. `node src/main.ts solve claim.json --strategy llm` poses any statement of a registered kind and drives it the same way (`--strategy scripted` is the no-key floor), and an agent harness that prefers to hold the seat itself connects to serve and speaks the protocol directly. The workspace neither knows nor cares what kind of mind proposes.

## 8.12 Where this goes

You can now construct every kind of object this book's spine runs on — exact scalars, certified enclosures, claim records, derivation rows, certificate documents with their mutant batteries — and you have worked their production from an obligation workspace under two different strategists. Three directions from here, in increasing order of ambition:

- *A worked model at full scale.* The fabrication-line chapter (chapter 17) runs this stack over a model with re-entrant flow, batching, failures, and shared personnel — and shows the checker's economics on the way. The System Catalogue of model types (chapter 12) and the composition patterns (chapter 13) are its reference spine.
- *The operating philosophy.* The Cookbook (chapter 19) is the patterns manual this tutorial has been quietly citing throughout; read it the way you would read *How to Solve It* [200] — before you need it.
- *Sessions under your own harness.* The workspace and the strategy seat (sections 8.10 and 8.11) are importable pieces — workspace, the proposer interface, the RPC client — and two growth directions are yours from here: hold the seat yourself, over the socket, from whatever harness you already run; or register a new claim species by implementing one `StatementKind` (`validate`, `describe`, `moves`, `exec`) whose moves call the builders this chapter taught — which is how the menu discipline extends to specs, judgments, and derivation rows.

A graduation exercise, meanwhile, that needs nothing not already on your disk: FrozenLake has a hitting companion, `dp.cliffwalkingHitting()`. Produce its deadline certificate, read its document, and design its claim-inflation mutant — the matched pair the battery discipline requires — then check the committed corpus to see how close your battery came to the shipped one (expected verdicts are recorded in the receipts beside the artefacts, and the family's sealed row form ships a mutant battery of its own, each entry's expected verdict and violated obligation named in its manifest).

# The Semantic Universe

DRAFT

In our Quantitative Temporal Specification Logic (QTSL), open Systems and their Specs denote in one universe, at consecutive storeys of one tower (Figure 9.1). A System denotes a *belief*: a lower semicontinuous functional on probability distributions over records, whose value at a conceivable joint distribution is the level of *inconsistency* between that distribution and what the System asserts — a reading adopted from Richardson and Halpern [209], and the notion of belief this book commits to throughout [54]. Via the *radial compile*, each belief denotes a closed radial *body* in the space of subprobability distributions over records. The *hyperspace* of closed *sets* of closed bodies is the next storey; Systems are singleton elements of the hyperspace. A Spec denotes a *constructible class* within the hyperspace, i.e., an element of the same hyperspace’s finitary constructible algebra, which constitutes the fibre of Specs. At the hyperspace storey and above, the order of the universe is *inclusion* — membership of a singleton in a class, containment of one class in another — and ultimately,  $\llbracket \text{System} \models \text{Spec} \rrbracket := (\llbracket \text{System} \rrbracket \subseteq \llbracket \text{Spec} \rrbracket)$  (like Willems’ “behavioral approach” [240], but at full imprecise-probabilistic altitude). The pointwise order on beliefs is the same order seen one storey below, exchanged with inclusion by the compile (section 9.6), although specifications which arise through this compile alone can only express safety, not liveness (the incorporation of liveness specs crucially relies on the “hit-and-miss” aspect of hyperspace topology).

Wiring is composition in a double category whose base holds interfaces and wiring patterns and whose fibres hold Specs — the **double category of System Specs**, assembled by a **double Grothendieck**

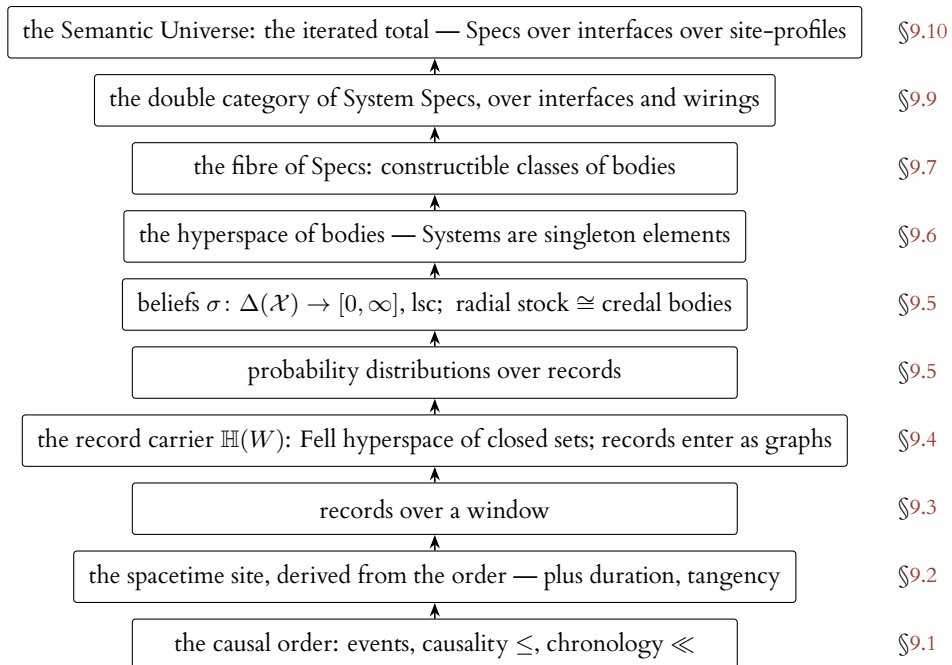


Figure 9.1: The storeys of the semantic universe, in dependency order, with the section that builds each; the sections follow the same order. On the universe’s radial stock the compile exchanges the pointwise order on beliefs with inclusion of bodies; from the hyperspace storey up, the order of the universe is inclusion. The site dimension runs through every storey above the site: each object is carried window by window and glued along the site’s coverage (section 9.4 and section 9.8).

**construction** whose obligations are formally-verified end to end (section 9.9) and checked with Lean as well as SafeVerify and nanoda. The chapter is written against a single mechanized development, and each storey’s definitions cite their machine counterparts by module and theorem name beside the statement.

**How to read this chapter.** Figure 9.1 is the map. The *causal order* fixes the events and the two relations — causality and chronology — from which the temporal geometry is made (section 9.1). The *site* is derived from that order — its windows are causal diamonds and its coverage is a theorem, not a choice — and, equipped with duration and tangency, governs how records glue into trajectory spaces (section 9.2). A *record* is an observation history over a window’s stages, a classical duration-carrying history along each of its worldlines, and every temporal clause of the specification logic evaluates along a single such timeline (section 9.3); the record carrier over each window of the site is a compact Fell hyperspace, with gluing along the interior coverage — directed covers, at fixed state fibre — a homeomorphism onto a projective limit (kernel-checked, section 9.4). The *universe of beliefs* holds the denotations of Systems: lower semicontinuous level-of-inconsistency functionals on probability distributions over records, ordered pointwise (section 9.5). The *hyperspace* holds their bodies, with Systems as singleton elements and inclusion as the order (section 9.6). The *fibre of Specs* holds the constructible classes, on which conjunction is intersection and disjunction is union, the latter forced (section 9.7). The *composition construction* carries the semantics of wiring: decoration functors on the interfaces-and-wirings base — the authoritative total carrying coupled-DNF presentations of constructible classes as fibre data — each total a pseudo double category with a strict projection that is a discrete double opfibration, all of it kernel-checked (section 9.9). The storeys then assemble: the *Semantic Universe* itself, the chapter’s headline object, is stated whole as the iterated total — Specs over interfaces over site-profiles — in section 9.10. And a final section before the receipts (section 9.12) is the adapter: how the wall fragment — the fragment the shipped checkers actually certify — embeds into the universe, stated only after the universe is complete on its own terms. Four families of material can be deferred without losing the thread. The generalization columns of section 9.2 — causal diamonds, general relativity, open quantum systems, branching time — exist to show that the site is a genuine parameter; unless those instantiations are your interest, what later sections consume from them is only the site contract (F1–F6) at that section’s close. The record carrier (section 9.4) may be skimmed until its final subsection, which lists the four interfaces everything downstream consumes. The mechanization remarks are receipts, not exposition: each names the machine-checked counterpart of the statements beside it, and a reader who trusts the ledger can pass over them. And the three sections after the assembled universe serve readers arriving with specific questions — adequacy (section 9.11) for the modelling catalogue’s squares, the wall fragment (section 9.12) for how possibilistic verdicts embed, and the receipts (section 9.13) for the evidence classes.

## 9.1 The Causal Order

The base storey of the universe is the causal order, from which the site is made. The axioms our construction requires are exactly the order-theoretic fragment of bicontinuity isolated by Martin and Panangaden [171]. In classical  $\mathbb{R}$ -valued time,  $\ll$  is exactly  $<$ ; in relativistic Minkowski spacetime,  $\ll$  denotes reachability by a time-like worldline (strictly less than lightspeed) with nonzero spacetime distance, while  $\leq$  denotes reachability by any worldline (including lightspeed or zero spacetime distance). In discrete time, both  $\ll$  and  $\leq$  are interpreted by  $\leq$ .

**Definition 9.1** (Causal order). (`QtslAux.DiamondSite: ChronologyKit`) A causal order consists of

1. A set  $E$  of *events*;
2. a partial order  $\leq$  of *causality*; and
3. a relation  $\ll$  of *chronology*, such that
4.  $p \ll q \Rightarrow p \leq q$  (chronology is stricter than causality),
5.  $p \leq q \ll r \Rightarrow p \ll r$  and dually
6.  $p \ll q \leq r \Rightarrow p \ll r$ ,
7.  $p \ll q \Rightarrow \exists s. p \ll s \wedge s \ll q$ ,



8.  $\forall p, r, s. p \ll r \wedge p \ll s \Rightarrow \exists q. p \ll q \wedge q \leq r \wedge q \leq s$ , and

9.  $\forall p, r, s. r \ll p \wedge s \ll p \Rightarrow \exists q. q \ll p \wedge r \leq q \wedge s \leq q$ .

The *cone order* of special relativity — events  $(t, x)$  in time  $\times$  space, with  $p \leq q$  when  $\|q_x - p_x\| \leq q_t - p_t$  and chronology the strict cone — satisfies every axiom over a real normed space of any dimension (kernel-checked: `QtslAux.DiamondTopology`, `minkowskiKit`); there the order is genuinely partial — distinct simultaneous events are causally incomparable, so no foliation linearizes it. On a globally hyperbolic spacetime, the causal and chronological relations  $J^+$  and  $I^+$  of the metric satisfy the axioms with room to spare: this is Martin and Panangaden’s theorem [171], which delivers bicontinuity and more — their interval topology *is* the manifold topology, so at that rung the order knows the topology of spacetime outright. And at the opposite pole a *discrete* order — integer ticks — admits no *strict* chronology at all (interpolation fails between consecutive ticks; kernel-checked), a degeneration the next section turns into a feature rather than an exclusion.

Two boundary facts orient everything downstream. First, the *ladder*: how much of a spacetime its causal order determines is graded by the classical causality hierarchy, and this book stands on two rungs of it — strong causality, exactly the condition under which the order’s interval topology (generated by the chronological diamonds  $I^+(p) \cap I^-(q)$ ) is the manifold topology, a theorem Martin and Panangaden quote from Penrose [171]; and global hyperbolicity, the rung at which reconstruction from the order is complete. Second, the *limit* of the method: by the theorems of Hawking–King–McCarthy and Malament [113, 170], the causal order of a distinguishing spacetime determines its topology, its differential structure, and its *conformal* class — and nothing more. Durations are precisely the conformal factor the order forgets. Everything the order does know is *derived* in the next section (Definition 9.2) — windows, coverage, and the coverage’s well-behavedness are theorems of  $(\leq, \ll)$ , not choices — and everything it cannot know, **duration** and **tangency**, enters there as additional declared structure. The division of data is a theorem in both directions, and the structured-site definition of the next section is organized around it.

## 9.2 Spacetime as a Structured Site

The site is *derived* from the causal order — windows and coverage are theorems of  $(\leq, \ll)$  — while **duration** and **tangency** are declared on top. This section gives the derived object first (Definition 9.2), then the classical instantiation — the value at which every classical artefact of this book lives — and the factored form of the construction, so that what is derived and what is declared stay visibly separate.

### 9.2.1 The Derived Site

**Definition 9.2** (Spacetime site). (`QtslAux.DiamondTopology`) Let  $(E, \leq, \ll)$  be a causal order (Definition 9.1). The *spacetime site* of  $(E, \leq, \ll)$  is the site whose

1. *objects* are the causal diamonds  $[a, b] = \{c \in E : a \leq c \leq b\}$  with  $a \leq b$ , ordered by containment — morphisms are inclusions of subdiamonds; and whose
2. *coverage* is the *derived interior topology*: a sieve on  $[a, b]$  covers precisely when it contains the inclusion of every chronologically interior subdiamond — every  $[c, d] \subseteq [a, b]$  with  $a \ll c$  and  $d \ll b$ .

No further data enter: the spacetime site is a function of the causal order.

That the coverage is a Grothendieck topology is a theorem of the causal-order axioms, kernel-checked in closed form (`diamondTopology`): pullback stability is the two absorption laws (axioms 5–6), and local character is density (axiom 7) applied at both endpoints. The consumption is exact — the topology uses only axioms 4–7 of Definition 9.1, never the filteredness axioms 8–9, which feed the descent arguments of section 9.4 instead. The order-theoretic demands factor as: *interpolative mixing buys the coverage; filteredness buys descent*. (The mechanized statement quantifies over the full kit while its proof consumes only the coverage half; the factoring is recorded in the module.)

On a diamond that is not chronologically thick ( $a \not\ll b$ ) the covering condition is vacuous, so every sieve covers — in particular the empty one, which forces every sheaf to a point on that diamond: the derived form of the classical excision of duration zero (`mem_of_not_thick`). More generally the chronology acts as a *descent dial*. The degenerate choice  $\ll = \leq$  is admissible and its derived topology is the *coarsest*

*Grothendieck topology* — only the maximal sieve covers, so sheaves are exactly presheaves and no gluing is ever demanded (`mem_diamondTopology_degenerate_iff`); the discrete order refuses the strict chronology outright (interpolation fails between consecutive ticks; kernel-checked, `int_lt_no_chronologyKit`), so integer ticks sit at the dial’s coarsest end; dense orders admit the strict chronology, whose covers are the genuinely interior ones and whose sheaf condition is the gluing this chapter uses.

Four instances calibrate the definition. The value at  $(\mathbb{R}, \leq, <)$  is the classical interval-domain site, identified cover-for-cover with the site of Schultz and Spivak through the homogeneity quotient below (Remark 9.5). The value at the cone order is the Minkowski site, kernel-checked over a normed space of any dimension (`minkowskiKit`). The value on a globally hyperbolic spacetime exists by Martin and Panangaden’s theorems [171] — cited rather than mechanized, since the site consumes only the order axioms and Lorentzian geometry never enters the development. And integer ticks ride the degenerate dial setting, the strict one being refused by their order.

A spacetime site arrives with its stage data for free — a diamond’s stages are its own events under the induced order, restricted along inclusions — so it supplies outright the site and stage clauses (T1)–(T2) of the structured definition this section builds toward. What it cannot supply is exactly what section 9.1’s closing division of data says it cannot: duration and tangency, which Definition 9.4 declares on top.

## 9.2.2 The Classical Instantiation

The site every classical artefact of this book lives on is the value of Definition 9.2 at the causal order  $(\mathbb{R}, \leq, <)$ , taken through one further, deliberate step — a homogeneity quotient. Schultz and Spivak [217] take as base the interval domain  $\mathbb{IR}$ : closed intervals  $[d, u]$  ordered by reverse inclusion, with  $[d', u'] \ll [d, u]$  precisely when  $d' < d$  and  $u < u'$ . The coverage is generated by the way-below relation alone — an interval is covered by the family of its strictly interior subintervals — and behaviour types are sheaves for that coverage: a behaviour over a window is exactly a compatible family of behaviours over its interior subwindows. Two design decisions of theirs matter for what follows. First, the finer coverage that would let two behaviours meeting at a single endpoint glue is deliberately omitted; gluing across abutting windows is a property a behaviour type may have, not one the site imposes. Second, they quotient by translation: the quotiented site’s objects are *durations*  $\ell \geq 0$ , and absolute time survives the quotient as an internal object Time — the behaviour type of a clock. The quotient does not delete absolute time; it *demotes* it, from base structure to component. That demotion is the categorical form of a commitment the model-type inventory (chapter 12) states operationally: clocks are internalized as mechanism state, and no artefact of the toolkit carries a global timeline field.

**The construction, factored.** Read as a recipe, the classical construction splits into two stages with very different appetites, and the split is what makes “time as a site parameter” more than a slogan:

| stage                | required structure                                                                                                                                     | output                                                        |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| order-theoretic core | a continuous predomain                                                                                                                                 | the way-below coverage and its sheaf topos                    |
| homogeneity quotient | a coverage-compatible free action whose projection is a discrete bifibration with flat descended covers — or, directly, a continuous quotient category | duration objects; absolute time demoted to the internal clock |

The unquotiented stage is purely order-theoretic: the way-below coverage is definable on an arbitrary continuous predomain, and the sheaf topos exists with no real arithmetic, no field structure, no metric anywhere in the site data. The homogeneity quotient is the demanding stage; here is its requirement, stated formally.

**Definition 9.3** (Homogeneity quotient). Let  $(\mathcal{T}, J)$  be a site and let a group  $G$  act on  $\mathcal{T}$  by coverage-preserving automorphisms (the image of a  $J$ -covering family under each  $g \in G$  is  $J$ -covering), freely on objects. Suppose:

- (Q1) the orbit projection  $\pi: \mathcal{T} \rightarrow \mathcal{T}/G$  onto the quotient category is a *discrete bifibration* — every morphism of  $\mathcal{T}/G$  has exactly one lift along  $\pi$  through any chosen lift of its source, and exactly one through any chosen lift of its target; and

(Q2) the *descended families* — images  $\pi(F)$  of  $J$ -covering families  $F$  — generate a coverage  $J/G$  on  $\mathcal{T}/G$  (*flat descent*: a pullback of a descended cover along any morphism of  $\mathcal{T}/G$  is refined by a descended cover).

The *homogeneity quotient* of  $(\mathcal{T}, J)$  by  $G$  is the site  $(\mathcal{T}/G, J/G)$ .

The input of the construction is the *action*, not a duration — at this generality no metric structure appears anywhere in the hypotheses. Duration names the *output* in the favourable case: when  $\mathcal{T}$  is a spacetime site whose duration structure (T3) is  $G$ -invariant and is a *complete* orbit invariant, the quotient's objects are canonically the durations, which is exactly the classical situation —  $G = (\mathbb{R}, +)$  translating the interval domain, the orbit of  $[d, u]$  determined by  $\ell = u - d$ , morphism orbits the shave-off pairs  $[r \mid s]$ , and (Q2) discharged through lattice structure on the translation parameters (coproducts of wavy arrows taken by componentwise maxima) — one sufficient realization, not part of the requirement. Neither direction of the association is automatic: a site may carry a perfectly good duration and admit no qualifying action (nothing to quotient by), and a quotient may exist whose objects no scalar names — Minkowski diamonds under spacetime translations alone quotient to displacement *vectors*, and only the further Lorentz quotient collapses the invariant to the scalar proper time. A candidate time base therefore pays for time-homogeneity separately from existence — and the two generalization columns below sit on opposite sides of that bill. (The classical value of the quotient is kernel-checked through the identification of Remark 9.5; the definition at this generality is stated in prose, an obligation of the ledger if a second instantiation ever consumes it.)

Both claims in the first row are now theorems at the generality of Definition 9.2: the way-below coverage is that definition's derived interior topology — on diamonds, way-below *is* componentwise chronology, the theorem the causal-diamond column below records — and its well-behavedness, its consumption of the order axioms, and its descent dial are theorems of the causal-order axioms, stated with that definition. The homogeneity quotient of the second row is the classical column's own step, and everything the table demands of it is demanded of the quotient alone.

### 9.2.3 Duration, Tangency, and the Structured Site

Two additional structures are necessary to feed the two families of proof obligations the certification interface emits.

*Duration* is the metric-temporal axis: it types deadline windows  $F_{\leq T}$  and until windows, the spectral shift  $\theta$  of a decay certificate (units: inverse duration), and the rate and jitter bounds carried by clock morphisms. In the classical site duration is native and visible — the quotient's objects literally are durations.

*Tangency* is the differential axis. The ordinary instance first: for an ODE  $\dot{x} = u(x)$  on a state space  $S$ , the dynamics assigns  $u(x) \in T_x S$ , a certificate sees it through the derivative pairing  $dV_x(u(x))$ , and a tangency abstracts the assignment  $x \mapsto T_x S$ . In general — in the sense of Capucci and Myers [37] (following Libkind and Myers' doctrine of dynamical systems) — it is a base category of state spaces, a fibration of *action bundles* over it, and a section assigning to each state space its bundle of admissible changes. A dynamics is a map *into* the bundle the tangency selects; the infinitesimal generator — the object the certification interface actually touches — is that dynamics presented through its induced action on certificate observables, which in the ODE instance is exactly  $V \mapsto dV(u(\cdot))$  and in other doctrines is the doctrine's own generator form (a jump kernel's integral operator, an open-quantum generator's trace pairing). At this generality, tangencies whose bundles are powersets or distribution spaces are legitimate, which is exactly the slack a jump kernel or an open-quantum generator occupies without any change to the site.

**Definition 9.4** (Structured spacetime site). A *structured spacetime site* is a site equipped with window, duration, tangency, and stage-space structure, as follows.

(T1) *The site*. A small category  $\mathcal{T}$  together with a coverage  $J$  assigning to each object a collection of covering families. Objects of  $\mathcal{T}$  are called *windows*; the coverage governs how behaviours — and in particular records — over the members of a covering family glue to a behaviour over the covered window. The site enters the semantics through this gluing role only; no satisfaction clause of the specification logic consults it.

- (T2) *Underlying stages.* A functor  $|-|: \mathcal{T} \rightarrow \mathbf{Pos}$  assigning to each window  $W$  a partially ordered set  $|W|$  of *stages* and to each morphism  $u: W' \rightarrow W$  an order-embedding  $|u|: |W'| \hookrightarrow |W|$ . A *timeline* in  $W$  is a nonempty chain (totally ordered subset)  $\gamma \subseteq |W|$ . When the site is the spacetime site of a causal order (Definition 9.2), this clause is supplied outright:  $[[a, b]]$  is the diamond's own event set  $\{c: a \leq c \leq b\}$  under the restriction of  $\leq$ , and  $|u|$  for a subdiamond inclusion is the inclusion of event sets. In the classical instantiation  $(\mathbb{R}, \leq, <)$  every  $|W|$  is then totally ordered and is its own maximal timeline; in relativistic instantiations  $|W|$  is genuinely partial and its timelines are the (segments of) worldlines it contains.
- (T3) *Duration structure.* For each window  $W$  and each timeline  $\gamma$  in  $W$ , a duration functor  $\delta_\gamma$  on  $(\gamma, \leq)$  satisfying (R1)–(R2) of Definition 9.6 in the next section — reflexivity at zero, and additivity along the order. The per-timeline indexing is load-bearing, not a hedge: a single duration functor on all of  $|W|$  would force  $\delta(s, u) = \delta(s, t) + \delta(t, u)$  through *every* intermediate stage  $t$  — path-independence of elapsed time — and proper time in a relativistic window is path-dependent (the twin paradox; the reverse triangle inequality makes the straight worldline strictly longest). Duration is additive on the nose *along* each worldline and only there, which is why (T3) is declared per timeline, subject to two naturality conditions:
- (D1) (*restriction*) if  $\gamma' \subseteq \gamma$  are timelines in  $W$  with  $\gamma'$  order-convex in  $\gamma$ , then  $\delta_{\gamma'}$  is the restriction of  $\delta_\gamma$  (convexity is load-bearing: the two-stage sub-timeline of a bent worldline must be free to disagree with the bent valuation, else additivity returns through every intermediate stage — the collapse this clause exists to refuse; window restrictions are always convex, stage sets being diamonds);
- (D2) (*transport*) for every morphism  $u: W' \rightarrow W$  and timeline  $\gamma$  in  $W'$ , the embedding  $|u|$  carries  $\delta_\gamma$  to  $\delta_{|u|(\gamma)}$ : durations of pairs of stages are preserved.
- (T4) *Tangency structure* (in the sense of Capucci and Myers, following Libkind and Myers' doctrine of dynamical systems). A *tangency* is a triple  $(\mathcal{B}, \pi: \mathcal{E} \rightarrow \mathcal{B}, T)$  consisting of a category  $\mathcal{B}$  of state (and observation) spaces, a cloven Grothendieck fibration  $\pi: \mathcal{E} \rightarrow \mathcal{B}$  of *action bundles*, and a section  $T: \mathcal{B} \rightarrow \mathcal{E}$  of  $\pi$ , assigning to each state space  $S$  its bundle  $(T_s S)_{s \in S}$  of *admissible changes*. (The Roman  $T$  is reserved for the tangency section; the italic  $T$  of Definition 9.6 is a record's stage set.) A *dynamics over the tangency* is a map *into* the bundle the tangency selects: on state space  $S$  with input/parameter space  $A$ , an assignment  $u: (s, a) \mapsto u(s, a) \in T_s S$ . When the tangency is the ordinary tangent-bundle tangency of a smooth state space, the induced action on a certificate observable  $V$  is the pairing  $V \mapsto dV(u(\cdot))$ ; for the other legitimate tangencies — powerset bundles (nondeterministic step relations), jump-kernel and distribution-space bundles, operator-valued generators — the induced action is presented doctrine-specifically, and no single formula of that shape is asserted at full generality.
- (T5) *Stage spaces* (Fell-readiness). For each window  $W$  a topology on the stage set  $|W|$ , compatible with the duration structure in the following sense: writing  $d_\gamma(s, t) := \delta_\gamma(s \wedge t, s \vee t)$  for the symmetrization of a timeline's valuation — an extended pseudometric on  $\gamma$ , by (R1)–(R2) — each  $d_\gamma$  is continuous on  $\gamma \times \gamma$ , and on every timeline where  $d_\gamma$  separates stages the subspace topology is the  $d_\gamma$ -metric topology. Under that topology the *stage space*  $S(W)$  is required locally compact, second-countable Hausdorff, with each morphism's embedding  $|u|$  a proper topological embedding. (Zero-duration pairs of distinct stages — hybrid jumps — are permitted; there  $d_\gamma$  separates nothing, and the stage space does not owe them separation: same-instant stages share a stage-space point, and the carrier presents them as one multivalued instant of the record's graph — the closed-graph doctrine of section 9.4, under which any jump is multivalued at the moment it happens.)<sup>1</sup>

Axioms (D1)–(D2) say exactly that living-on-a-timeline is a notion stable under restriction of windows and under the site's morphisms, which is what the gluing role (T1) requires; the records themselves — along a timeline and over a general window — are section 9.3's subject.

Clause (T5) feeds the record carrier of section 9.4, an adopted construction with kernel-checked strata. (T3) is the topology supply on timelines: the site's metric axis fixes the stage topology along every separating

<sup>1</sup>For globally hyperbolic instantiations (T5) is a *precision* in exactly the way Remark 9.5 exhibits classically: there the stage-space topology is itself order-derived — on such a spacetime the interval topology generated by the chronological diamonds is the manifold topology (Martin and Panagaden [171]) — so the clause records a topology the order already knows rather than supplying a new one.

timeline outright and constrains it (by continuity) along the rest, and (T5) asks only for a compatible ambient extension — in the classical instantiation the valuation metric *is* the stage topology — the hybrid column shares that stage space verbatim — so the clause is a precision there, not a new structure. Tangency’s role at the carrier tier is equally specific: it types the one metric hypothesis the record class keeps — a speed bound, a trajectory whose difference quotients lie in a fixed ball of the tangency’s action bundle, measured against duration — so the two site axes jointly type the record layer’s single metric hypothesis, exactly as they jointly meet in the decay row  $\mathcal{AV} \leq -\theta V$ , which pairs a tangency-typed operator on the left with an inverse-duration scalar on the right. Obligation families sort cleanly: infinitesimal-generator rows (*generator rows*, hereafter) and interior rows are tangency; window arithmetic and spectral shifts are duration; reset rows are neither (jumps consume zero duration); nonnegativity rows are pointwise and touch neither axis. The tangency clause is stated at this generality in prose; the mechanized development is parametric over it — every kernel-checked artefact consumes tangency only through the generator-row typing of its concrete instances — and the abstract object is an obligation of the ledger.

*Remark 9.5* (The classical instantiation, quantitatively). The temporal type theory of Schultz and Spivak is the value of the site parameter at which every classical artefact of this book lives — and the identification is functorial, kernel-checked at the site and behaviour tiers: the spec tier’s classical windows map onto the quotient site by the translation quotient, with unique lifts in both directions and covers corresponding exactly to covers (a wavy arrow is a strictly interior inclusion), the motivating behaviour type of trajectories is the site tier’s own trajectory presheaf up to natural isomorphism (`SuFunctor.Site.TemporalTypeTheory: anchorQuot, anchorTrajIso`), and behaviour subtypes include into the double category of System Specs — satisfaction is behaviour membership, wire transport is behaviour pullback, strictly (`SuFunctor.Site.BehaviorSpec: ttObject, satAt_specOfBehavior_iff, wSpecMap_specOfBehavior`).

1. *Site*. The base poset is the interval domain  $\mathbb{IR} = \{[d, u] : d \leq u\}$ , ordered by reverse inclusion, with way-below relation  $[d', u'] \ll [d, u]$  precisely when  $d' < d$  and  $u < u'$ . The coverage is generated by the way-below relation alone: an interval  $[d, u]$  is covered by the single family of its strictly interior subintervals  $\{[d', u'] : d < d' \leq u' < u\}$ . The finer coverage that would let two behaviours meeting at a shared endpoint glue is deliberately omitted: endpoint gluing is a property a behaviour type may have, not one the site imposes. This base is itself the derived object: the interval domain is the diamond poset of the causal order  $(\mathbb{R}, \leq, <)$ , and its interior coverage corresponds cover-for-cover — through the anchored-window identification below — with the derived topology of that order, kernel-checked (`SuFunctor.Site.CausalConcordance: mem_interiorTopology_iff_derived, wavy_mem_iff_derived`). Read at the spacetime level, the identification is also a diagnosis: a time-window with all of space carried in the state fibre is the causal diamond of a world with instantaneous propagation, so as a site the classical column encodes absolute simultaneity — the  $c \rightarrow \infty$  degeneration, in which diamonds flatten to time-slabs — and space is carried by the wiring and the fibres, never by the site.
2. *Homogeneity quotient* (Definition 9.3). Quotienting by translation yields the site whose objects are durations  $\ell \geq 0$  and whose morphisms are shave-off pairs  $[r \mid s]$  with  $\ell = r + \ell' + s$ ; the covering families are those with strictly positive margins  $r, s > 0$ . Absolute time survives the quotient as the internal clock object `Time`, demoted from base structure to component.
3. *Duration structure*. Native and visible: the quotient site’s objects literally are durations, morphism data are duration differences, and the induced valuation on any timeline  $\gamma \subseteq \mathbb{R}$  is  $\delta_\gamma(s, t) = t - s$ .
4. *Tangency structure*. Ordinary calculus: state spaces are smooth (in practice carved by polynomial data; the exact class is fixed below) spaces, the tangency is the tangent-bundle section, a dynamics is a vector field presented as drift data, and the induced action on a certificate observable  $V$  is the Lie derivative row that drift obligations bound.
5. *Stage spaces*. (T5) holds on sight:  $S(\ell)$  is the compact interval  $[0, \ell]$  with the metric  $d_\gamma(s, t) = \delta_\gamma(s \wedge t, s \vee t)$  its own duration functor induces by symmetrization, and the interior covers’ pair-filteredness is kernel-checked in this repository at bicontinuous-poset generality (module `QtslAux.DiamondSite`; the causal-diamond column below is the same theorem’s other instance). The hybrid column shares this stage space verbatim: jumps and resets are carrier facts (multivalued instants), not stage-space structure, so hybrid windows are the classical windows.



For contrast on the duration axis: in the causal-diamond instantiation the site *qua* poset carries only the causal order, and duration data — proper time along timelines — is strictly additional structure, supplied densely rather than per-clock. That divergence in presentation (object-valued durations in the classical quotient; densely supplied proper time on worldlines in the relativistic column) is why (T3) axiomatizes duration by its *induced valuations on timelines*, the invariant form both presentations share and the only form the satisfaction clauses of Remark 9.7 consume.

### 9.2.4 The State Sheaf, and Where the Carving Class Lives

The window-by-window state spaces of a System assemble into a *state sheaf*

$$\mathcal{V}: \mathcal{T}^{\text{op}} \longrightarrow \mathcal{B},$$

a presheaf valued in the tangency’s own base category, so that (T4)’s action-bundle fibration composes with  $\mathcal{V}$  to give each window its bundle of admissible changes. A constant  $\mathcal{V}$  recovers the fixed state spaces every classical artefact uses; a genuinely varying  $\mathcal{V}$  — mode-dependent state spaces, stagewise phase-space reductions, time-varying constraint manifolds — is the structural home of time-inhomogeneity, and under it the time-inhomogeneity lemma splits: its state face is absorbed into the sheaf itself, and its law face reprices to a classical projective limit of compact metrizable systems along a countable tower of windows (section 9.4).

Three commitments govern the fibres of  $\mathcal{V}$ , each with its reason, and with the division of labour stated first: the demand on fibres is intrinsic and topological; the carving class of the next paragraph is how a fibre *certifies* it; and compactification is an *operation* applied to fibres, never a condition on them. The commitments: *locally compact fibres* (Hausdorff, second countable — what the carrier’s Hausdorff theorem consumes, and with second countability it is exactly Polish local compactness, so the same demand buys the denotation stack its standing structure;  $\sigma$ -compactness follows and is what makes the compactified fibres metrizable); *fibrewise cemetery* ( $\mathcal{V}^+(W) := \mathcal{V}(W) \cup \{\infty_W\}$ , the one-point compactification, an operation at the carrier tier — escape-to-infinity becomes an ordinary observation value, and mass leak, killing, and divergence keep their standard readings, per window; the operation lands outside the carving class and owes it nothing); and *proper restriction maps* (what makes the carrier’s pushforward legs Fell-continuous; automatic for the constant sheaf and for mode-restriction embeddings). For a sector carved in  $\mathbb{R}^n$ , local compactness is equivalent to being locally closed — open  $\cap$  closed — which is checkable from carving data; that equivalence is why the certificate below asks local closedness per sector, and a disjoint union of such sectors is locally compact outright, every summand clopen.

The carving class of the fibres is fixed as follows: fibres are locally closed  $\omega$ -semialgebraic sets — countable unions of pieces, each carved in some  $\mathbb{R}^n$  by finitely many polynomial inequalities with *rational* coefficients and finite degree, degrees unbounded across the family — with proper continuous countably-piecewise-polynomial maps, the local polynomial representatives again with rational coefficients. (The coefficient pinning is definitional, not a presentation nicety: over unrestricted real coefficients,  $\{x \geq \pi\}$  is a piece, and the exact-data property below silently dies.)<sup>2</sup> The class decomposes the  $\omega$ PAP spaces of Huot, Lew, Mansinghka, and Staton [117] along their two independent axes — countable piecing, and analytic rather than polynomial carving functions — and adopts exactly the first. The reasons, briefly, because each matters. *Demand*: countable piecing is what discrete-mode models actually need — counters, queues, birth-death populations, program-mode structure are  $\mathbb{N}$ -shaped, definable in no o-minimal structure, and  $\omega$ -semialgebraic in degree-one pieces on sight. *Exactness*: the intermediate class is the largest for which every piece remains finite rational data, so the certificate stratum of Remark 9.42 keeps exact-rational rows piece by piece, with countable piecing expressed by the language and discharged shape-uniformly at the certificate stratum (Remark 9.42: plates, pooled aggregate coordinates, count-budget rows as optional tightenings) rather than paid for by the semantics; full  $\omega$ PAP would surrender domain-exactness (piece

<sup>2</sup>Underneath the site’s commitments sits the mechanized ambient category — module family `SuFunctor.StateSpace` of the proof-artefact repository: objects  $\omega$ -semialgebraic subsets of real ambients, morphisms countably-piecewise-polynomial maps, with finite limits and common-ambient countable coproducts held concretely, and local closedness the membership bridge to the local compactness the carrier consumes. Properness and continuity are the site’s two extra demands on restriction maps, made for the carrier’s sake, never fields of the ambient category’s morphisms. The state-space category proper is this category’s *free countable coproduct completion*: a state space is a countable family of  $\omega$ -semialgebraic sectors, each carved in its own finite-dimensional rational ambient, with all countable coproducts — disjoint, universal, no common ambient asked — finite limits componentwise, and the exact-data story intact through one dimension-tagged countable carving alphabet (`SuFunctor.StateSpace.FamCat`, `FamCoproducts`, `FamLimits`, `FamCompletion`).



membership and guard side-conditions leave the exact fragment), while its value-side content — analytic densities, regular guards — lives at the law tier, where the calculus’s standing hypotheses already name it, untouched by this dial. *Freedom*: the record carrier of [section 9.4](#) is fibre-class-blind — its topological development consumes only the three commitments above, never the carving class — so the dial can be widened later at zero cost to any existing proof; narrowing is what would cost. Membership in the class of fibres is a real condition, not a formality:  $\mathbb{Q} \subseteq \mathbb{R}$  is  $\omega$ -semialgebraic and nowhere locally compact — excluded by the demand, and by its certificate ( $\mathbb{Q}$  is locally closed in no neighbourhood) — and adoption-time checks use a catalogue of sufficient patterns ( $V$  closed;  $V$  a topologically disjoint union of locally closed semialgebraic pieces — the bag pattern of every discrete-mode hybrid;  $V$  compact). The separation of concerns above is what keeps the class this small: the one-point compactification of an infinite bag is not a bag under any re-presentation (compact and infinite, it admits no infinite clopen partition, and finitely many semialgebraic pieces would make it semialgebraic, which  $\mathbb{N}^+$  is not) — so a class required to contain its own compactifications would be forced open to unions whose pieces accumulate. No such demand is made: the cemetery is an operation at the carrier tier, its output compact metrizable directly from the fibre’s local compactness and second countability, and the carving class never needs to carve  $\infty_W$ .

### 9.2.5 First Generalization Column: Causal Diamonds

For a globally hyperbolic spacetime  $(\mathcal{M}, g)$ , Martin and Panangaden [171] proved that the causal intervals  $[a, b] = \{c : a \leq c \leq b\}$  under reverse inclusion form a continuous domain whose way-below relation is componentwise chronology ( $[a, b] \ll [c, d]$  iff  $a \ll c$  and  $d \ll b$ ), and — the theorem that turns an analogy into an instantiation — that the category of globally hyperbolic posets is *equivalent* to the category of interval domains. The causal-diamond site is the verbatim transposition of the classical coverage to this domain: cover a diamond by its chronologically interior subdiamonds; the classical interval domain is the  $(0+1)$ -dimensional case of the same construction. The interior coverage is well-behaved at exactly the generality one would hope — for every bicontinuous poset, the interior covers of a diamond with  $a \ll b$  are nonempty and pair-filtered — and this is kernel-checked in this repository at exactly that generality (module `QtslAux.DiamondSite`): the mechanization consumes only the order-theoretic fragment of bicontinuity, no manifold and no compactness anywhere, and the classical interval domain is literally its dense-linear-order instance. The boundary caveat is likewise settled: nonemptiness of the interior covers is *equivalent* to chronological thickness ( $a \ll b$ ) — an iff, kernel-checked — so degenerate and null-related diamonds are covered by nothing, and the subsite of chronologically thick diamonds is the coverage’s maximal domain, the verbatim generalization of the classical excision of duration zero. (The reopening condition is named in the closing ledger: an application with claims on the null boundary would change the balance in favour of retaining boundary objects; nothing classical waits on it.) The diamond column is also the natural route to partial differential equations: a field over a spacetime region is a behaviour over a window whose stages are events, and hyperbolic problems are diamond-local in exactly the sense the interior coverage expresses — finite propagation speed is why energy estimates live on causal diamonds. PDE dynamics enter the adequacy strand today with space carried by the wiring rather than by the site — spatially extended cells over classical time windows ([section 9.11](#)) — and the diamond column is where space moves from the wiring into the site. The certificate stratum has a matching design: the spatial variable enters as a declared coordinate wire, so a certificate over PDE states is a spatially plated hypernet — one finite object denoting a region-indexed family of rows — and contraction along the spatial extent is a symbolic integral, discharged by differentiating polynomial witnesses rather than by enumerating a mesh; region subdivision then joins site coverage and wiring as a third composition axis, interior to the fibre, and all three are valuation-shaped.

A generic curved spacetime has trivial isometry group, so the homogeneity quotient is simply not exercised — which is semantic correctness, not damage: physics on a curved background is not translation-invariant, and behaviour types there *should* be diamond-indexed. What replaces the internal clock is the worldline: a timelike curve with proper time induces a map of sites from the classical interval site into the diamond site (proper-time intervals to the causal hulls of worldline segments), and that map is the curved-case clock — per-interface data, exactly as the toolkit already carries clocks. The clock-map’s site-level checks are kernel-checked in the same module: for any strictly chronology-preserving map out of a dense linear time order carrying the two endpoint-approximation properties that openness of the chronological past and future supplies, the induced diamond map is monotone for containment, lands in the thick subsite, carries interior-cover members to interior-cover members, and preserves covers in cofinal form. What remains cited rather than proved is the manifold-level instantiation: that a timelike worldline with proper

time satisfies those approximation properties in a distinguishing spacetime. Duration structure is proper time along records; tangency is the manifold's tangent bundle. And the division of data is sharp in a way worth recording: by the Hawking–King–McCarthy/Malament theorems [113, 170] a chronological isomorphism between distinguishing spacetimes — globally hyperbolic ones qualify — is automatically a smooth conformal isometry: the order determines the conformal class and nothing more. Durations are precisely the conformal factor the order forgets, and must be supplied as additional structure — densely, not per-clock. “The site is structured” is not a convenience of presentation; in this column it is a theorem about what the order alone cannot know.

**General relativity, in two grades.** The phrase “causal diamond” invites a stronger reading than the diamond column delivers, so here is exactly what it is — and is not — a semantics for. On any *fixed* globally hyperbolic background, curved or flat, the *bare derived site* exists outright: the causal and chronological relations of the metric satisfy the causal-order axioms by Martin and Panangaden's theorems [171], so Definition 9.2 instantiates there with no new machinery — the site consumes only the order, and Lorentzian geometry itself never has to enter the mechanization — while the column's supplied structures and owed obligations (duration, tangency, the record locus over general supports) are exactly those named elsewhere in this section, neither better nor worse off for the curvature. Background *independence* is different in kind, and no fixed site can even state it: in full general relativity the metric is a field, so the causal order depends on the solution while the fields live over the order. Because the site is derived — a function of the order — that circularity becomes an ordinary, if open, indexing problem rather than an inexpressible one: an outer storey indexed by causal orders (equivalently, within the distinguishing class where Malament's rigidity theorem [170] governs, by conformal geometries), with the fibre over each order the behaviour-and-spec tier over its derived site; in each fibre a solution record carrying exactly the data the order forgets — the conformal factor, entering as (T3) duration structure, together with matter fields; and self-consistency as a section condition, that the order indexing the fibre is the order its own metric-record induces. Einstein's equations then enter as a constraint on that locus — which order-factor-matter triples are admissible — not as site structure. The composition construction of section 9.9 already indexes interfaces by site profiles, so indexing of this shape has a precedent in this book; the outer storey itself — its morphisms beyond isomorphisms, its coverage — is future work, named in the closing ledger so the column's boundary stays visible (the groupoid part is no longer owed: transport along chronology-respecting order-isomorphisms is kernel-checked, module `QtsAux.QuantumCovariance`): fixed backgrounds are covered, and background independence is designed for, not built.

## 9.2.6 Second Generalization Column: Open Quantum Systems

**The modest route.** An open quantum system enters as one more mechanism profile — Hamiltonian plus noise operators, with the standard Gorini–Kossakowski–Sudarshan–Lindblad generator — and its master-equation flow  $t \mapsto \rho_t$  is a deterministic continuous behaviour valued in density operators: an ordinary behaviour type over the *existing* site. Where measurement enters, the measurement record is a classical stochastic process, and the record-level probability operators of section 9.5 already speak about it. Stability obligations transpose with the dictionary regions  $\mapsto$  projections, evaluation  $\mapsto$  trace pairing; the generator row becomes an operator inequality  $\mathcal{L}(V) \preceq -\gamma V + \zeta \mathbb{1}$ , checked by positive-semidefiniteness machinery the kernel already trusts for other purposes — under this column's stated hypotheses, which are design content and not fine print: finite-dimensional mechanisms (or bounded generators) only, with truncations of infinite-dimensional systems entering only under explicitly represented truncation error. That the stated hypotheses suffice is not only asserted: the transport theorem behind this row — Born-pairing positivity, the Loewner-monotone generator-row step, the sharp exponential bound, trace conservation as a theorem of the trace-annihilating generator form — is kernel-checked in this repository at the finite-dimensional stratum (module `QtsAux.OpenQuantum`). No site change occurs anywhere on this route, and the column as a whole is a design with its costs stated, graded in section 9.13, not a shipped instantiation. What the route does *not* deliver is event-level claims about measurement outcomes from stability certificates alone; those await a quantum transport theorem in certificate form, and the gap is named debt, not fine print.

**The same column, the deeper route: quantum states against the site itself.** The modest route leaves the site untouched by design. Its natural sequel asks what happens when quantum states meet

the site's own machinery — the records of [section 9.3](#) and the coverage of [Definition 9.2](#) — and the sequel is kernel-checked at the finite-dimensional stratum, on both sides. On the record side: a family of record observables — Hermitian, mutually orthogonal, correlated with the branch alternatives at the state; hypotheses deliberately weaker than the projections of the consistent-histories literature — forces the decoherence functional of its branch family diagonal, so the branch weights form an honest finitely additive probability row, the object the belief storey consumes. That is Gell-Mann and Hartle's records-imply-consistency direction [86], mechanized together with its biconditional for pure exhaustive families and witnesses that no hypothesis can be dropped (module `QtslAux.QuantumRecords`); the converse implication fails by kernel-checked witness — there is a weakly consistent branch family admitting no record family at all — so records genuinely select among consistent decompositions rather than relabelling them. On the coverage side: over a two-site observable net on Minkowski diamonds, gluing fails inside the state body, and it fails where entanglement lives. The Bell state and the maximally mixed state restrict identically along every member of a spacelike pair-cover yet differ; at pure marginals the same gluing is rigid (a theorem, not an example); and the derived interior coverage *refuses* the failing cover, while every cover it does certify separates (module `QtslAux.QuantumDescent`). The boundary this section drew for order-theoretic reasons — interior covers only — is the boundary quantum mechanics needs. The two sides then meet in a single kernel-checked sentence: the state presheaf over the net is separated at derived covers and provably not separated at the spacelike pair-cover, and one record row distinguishes the two state classes that the entire pair-cover cannot (`recordShadowQ_separates_bell_mixed`, module `QtslAux.QuantumStates`) — records carry information that region-by-region restriction structurally misses, which is this chapter's records thesis in theorem form.

**The locally covariant fragment.** These are not scattered observations; they are the kinematic axioms of algebraic quantum field theory. Observables have been indexed by causal diamonds since Haag and Kastler [103], and the locally covariant refinement of Brunetti, Fredenhagen, and Verch [29] takes as morphisms the isometric embeddings whose images are causally convex — causal convexity being this section's diamond condition — so the index category of rigorous quantum field theory is a derived site, and has been since 1964. One kernel-checked structure now assembles the fragment (`LCQFTFragment`, module `QtslAux.QuantumAssembly`): an isotone net of local algebras (functoriality); Einstein causality in class-level form — pointwise causally separated diamonds have elementwise-commuting algebras; the time-slice axiom at two tiers — a site-exhaustion surrogate carried as a field of the net, and the genuine grade *proven rather than postulated*: complete initial data plus a reversible mechanism yield Cauchy determinacy at both the algebra and the state grade, with reversibility load-bearing by kernel-checked witness (module `QtslAux.QuantumDeterminacy`); derived-cover separation for the state presheaf; transport of the site, its coverage, the net, and the separation theorems along chronology-respecting order isomorphisms (module `QtslAux.QuantumCovariance`) — which, by the Hawking–King–McCarthy/Malament rigidity cited above, is transport along conformal isometries — with determinacy transported across the same isomorphisms; and a record family whose shadow lands in probability rows. A two-qubit system under an entangling coupling inhabits the whole structure, so no field is vacuous. Two findings earned their way into the structure rather than being designed into it. First, the fragment is a two-tier object of necessity: an entangling mechanism between distant sites acts instantaneously — non-relativistically — so its Heisenberg-evolved observables violate class-level causality at unequal-time spacelike pairs, and no single net can carry both microcausality and the genuine time-slice property at this grade. The kinematic net and the matter data are therefore separate tiers, tied by a shadow equation — the net is the frozen, dynamics-off shadow of the matter data — which is where the locally covariant axioms themselves place dynamics: the time-slice clause is the door through which relative Cauchy evolution and the stress-energy tensor enter [76], both deliberately on the far side of this fragment's fence. Second, the state object is set-valued of necessity: a functorial choice of single states is unrealizable — a theorem of the locally covariant framework [29] — and the replacement that literature adopted, the folium (a contravariant, convex-closed assignment of *sets* of states), is structurally the flat, zero-or-infinite shadow of this chapter's credal bodies: the state doctrine of [section 9.5](#) was independently discovered by mathematical physics, for physics reasons. At matrix grade folium distinctions degenerate, and the degeneration is itself a kernel theorem rather than an admission (every rank-one-positive unital functional on the full matrix algebra is represented by a unique density state), so the toy fragment's real content is contravariance, separation compatibility, and the graded refinement — graded beliefs, walls, partiality — that a flat folium cannot state. The word “fragment” is load-bearing, twice. “Locally covariant” does not mean “the same physics in all spacetimes”: Fewster and Verch proved the stronger property fails for the full class of locally covariant theories [77],

and its repair, dynamical locality, is defined through relative Cauchy evolution — dynamics proper, together with non-surjective embeddings between different spacetimes and functoriality of the assembled fragment along them, are this column's named residues in the closing ledger. And one consequence for the outer storey of the general-relativity paragraph is already kernel-checked at toy grade (module `QtslAux.QuantumOuterStorey`): under a coherent superposition of causal orders — the quantum-switch shape, the same two measurements in the two temporal orders — no record family on the order index exists (the records theorem's contrapositive, its index read as the set of orders), while an orthogonal pointer record classicalizes the index back to an ordinary probability row. Which causal order obtains is a recordable question precisely when something records it — both directions witnessed at toy grade — so the records thesis of this chapter holds unchanged when what superposes is the site itself.

### 9.2.7 Branching Time

A causal order can branch: nothing in the axioms of [section 9.1](#) demands linearity, and a forest of events — histories sharing a past and diverging — is one more causal order, its site delivered by [Definition 9.2](#) like any other (with the degenerate chronology at minimum; a genuinely interior strict chronology exists where the branches are densely ordered and histories diverge along open cuts — a last shared event would defeat the filteredness axiom at the branch point). Branching time therefore needs no column of its own: there is nothing to adopt, and nothing to reject. Two boundaries remain, and a branching order is where both show plainly. The first is that we do not admit nested path quantifiers in the style of branching-time logics like CTL. Our logic ([chapter 14](#), below) *stratifies* into an inner layer of temporal quantifiers along single-timeline records, and an outer layer regarding the imprecise-probabilistic ensemble of such records. The second is a boundary of canonicity rather than of expressibility: treating a *quantum* system as branching-time — an Everettian forest as its causal order, branch weights as a measure — is non-canonical relative to the universal-wavefunction reading, because branch structure is decoherence-emergent and decomposition-dependent, so a certificate over a branching order would certify a claim about a *chosen* decomposition. (The asymmetry grounding this judgement is kernel-checked in the quantum paragraphs above: records force consistency of a branch family, and consistency alone supplies no records.) That is why the open-quantum column above enters through the master-equation flow over an ordinary site, with measurement records as classical stochastic processes read by record-level probabilities.

### 9.2.8 The Logic-Site Contract

The toolkit does not, at present, *adopt* a nonclassical site: no present application demands one, and the classical instantiation must never pay for the option. What it adopts is a contract — six freedoms the artefact formats keep, so that the site parameter can later be exercised without restaging a single classical artefact:

- F1. No absolute-time fields.** Time enters manifests and specifications only as durations, per-mechanism clocks, and clock morphisms with rate/jitter bounds; never a timeline table, never a timestamp.
- F2. The checker never sees the site.** All site content stays on the semantic side of the certification interface; obligations are polynomial (or operator) rows with no time-shaped vocabulary.
- F3. Total-order consumers are flagged.** Début, until, and cofinality clauses are stated as record-indexed, so a site change re-indexes rather than re-defines them. One such re-indexing is realized rather than merely flagged by the record carrier of [section 9.4](#): endpoint consumers become germ consumers. Over the interior coverage a window's own boundary stages are covered by nothing — data pinned to an outer seam is exactly what translation-homogeneity forbids, the same demotion that turns absolute time into the internal clock — so a clause that consults a window's endpoint re-indexes as a *germ* clause at the seam: a clause about the behaviour's germ there rather than its value there. The endpoint itself survives, like absolute time, as presentation data of an anchored tower of windows (a family sharing a fixed initial stage, whose shared seam survives every restriction), not as site data.
- F4. Mechanism profiles stay an open enumeration.** A new dynamics family (an open-quantum generator, say) is additive; no existing profile recompiles.
- F5. Clock morphisms keep the rate-relation form.** The wiring datum is bounds on  $d\sigma/d\tau$  plus jitter, never a schedule table in absolute time; clock-*rate* corrections (gravitational and velocity time

dilation) then arrive as *values* in an existing field, not as a schema change. The named exception: signal-*path* obligations — Sagnac terms, null-propagation time of flight — are not per-clock rate data; they need ephemeris or worldline data plus a propagation constraint, still ordinary classical model data in a named frame, but data the rate-relation form does not carry.

- F6. Duration and tangency stay ambient.** Mechanisms present tangency as doctrine-specific generator data (drift, diffusion, jump, or operator rows) and duration as rational constants; no artefact names either structure as an object.

Against the only metric that matters for a toolkit — does the classical user pay? — the verdict, costs counted, is: nothing. No extra field, no extra obligation, no slower check, on either generalization column. The cost of *adopting* a column when an application arrives is a named metatheory ledger, several of whose items are already discharged ahead of any adoption — all kernel-checked in this repository (`QtslAux.DiamondSite`, `QtslAux.OpenQuantum`): for the causal-diamond column, the coverage lemma, the thickness iff, the clock-map checks, and the time-inhomogeneity item's clock-adjunction core; for the open-quantum column, the bounded-generator hypothesis's transport theorem. What remains owed there is the site-generic transport of the extension theorems — split by the carrier of [section 9.4](#) into a presentation-shaped half that transports along inverse-image functors because presentations by generators and relations pull back (geometricity makes that half free in principle) and an analytic half (choice-invoking or point-set-surgical constructions) that runs objectwise over finite-duration windows, where the classical proofs apply unchanged, so the audit is per window class, not per theorem per site — together with the probabilistic time-inhomogeneity lemma's projective-limit step and the open-quantum event/record transport theorem, each named in [section 9.13](#).

## 9.3 Records

What lives on a window is an observation history over its stages: a *record*. The definition arrives in two steps — first along a single timeline, the object every temporal clause of the specification logic actually reads, and then over a general causal support, where a record is a coherent bundle of timeline records, one along each of its worldlines.

Along a single timeline a record is a totally ordered history of observations: the trajectory of a classical composite read on the global timeline; a component's local history read against its own clock; the proper-time worldline of a relativistic clock; the outcome sequence of a monitored quantum device. What these share — and what the specification logic consumes — is a totally ordered time axis *carrying duration*: the order alone is not enough — two of the three clause families below consume more than it: one does arithmetic on elapsed time, and the *début*'s infimum must exist in the record's own axis. The *début* of a region ( $\tau_E = \inf\{t : x_t \in E\}$ ), the “no visit before  $\tau$ ” clause of an until operator (and its metric variant's window  $U_{\leq T}$ ), the cofinal visits of a recurrence condition: each is well-defined along any record equipped with the ordered time-and-duration structure of [Definition 9.6](#), and none needs to know how that record sits inside a larger geometry of time.

**Definition 9.6 (Record).** A *record* is a tuple  $R = (T, \leq, \delta, \mathcal{O}, x)$  consisting of the following data.

1. *Stages.* A nonempty set  $T$ , totally ordered by  $\leq$ , whose order is *conditionally complete*: every nonempty subset of  $T$  that is bounded below has an infimum in  $T$ .
2. *Duration.* A *duration functor*

$$\delta: \{(s, t) \in T \times T : s \leq t\} \longrightarrow [0, \infty]$$

satisfying

$$(R1) \quad \delta(t, t) = 0 \text{ for every } t \in T;$$

$$(R2) \quad (\text{additivity along the order}) \quad \delta(s, u) = \delta(s, t) + \delta(t, u) \text{ whenever } s \leq t \leq u \text{ in } T.$$

1. *Observations.* An *observation space*  $\mathcal{O}$  — a set, required to be a measurable space whenever the record enters the denotation stack of [Definition 9.15](#) — and an *observation map*  $x: T \rightarrow \mathcal{O}$ , written  $t \mapsto x_t$ .



Equivalently, and more precisely: a duration functor is a *functor*  $(T, \leq) \rightarrow \mathbf{B}[0, \infty]$  from the stage order, read as a thin category, to the delooping of the additive monoid  $([0, \infty], +, 0)$  — functoriality on composites is (R2), preservation of identities is (R1). (Not an *enrichment*: a category enriched in  $[0, \infty]$  is a Lawvere metric space, whose composition law is the triangle *inequality*; duration along a timeline is additive on the nose.) From (R1)–(R2),  $\delta$  is antitone in its first endpoint and monotone in its second:  $s' \leq s \leq t \leq t'$  implies  $\delta(s, t) \leq \delta(s', t')$ . Two non-requirements are deliberate, and both do real work. *Strict positivity is not required*: distinct stages may be separated by zero duration (instantaneous transitions consume none, which is how hybrid records carry reset events). *Finiteness is not required*:  $\delta$  may take the value  $\infty$  on pairs separated by an unbounded extent.

For a subset  $E \subseteq \mathcal{O}$  (measurable, when  $\mathcal{O}$  is a measurable space), the *hitting set* of  $E$  is  $T_E := \{t \in T : x_t \in E\}$ , and the *début* of  $E$  is

$$\tau_E := \inf T_E \in T \cup \{+\infty\},$$

defined to be the infimum in  $T$  when  $T_E$  is nonempty and bounded below — which conditional completeness guarantees to exist — and the formal top element  $+\infty$  (adjoined to  $T$ ) when  $T_E = \emptyset$ . The début is deliberately *partial*: for a nonempty hitting set unbounded below (possible only in a record with no start), no value is assigned, and every satisfaction clause that consumes a début carries the bounded-below proviso. When  $T$  has a least element (a record with a start), every nonempty subset is bounded below and the début of every region is defined.

The stage set  $T$  is free data here and moored below: for a record over a window of the site (Definition 9.8), the stage sets to which this definition applies are the *maximal chains* of the record's support in the window's stage poset  $|W|$  — the site's (T2) data, under the induced order — with  $\delta$  not chosen but inherited from the site's (T3) duration structure on that timeline. Maximality is load-bearing on both sides: the maximal chains cover the support (so the bundle of timeline records misses no stage), while a restriction to a *non*-maximal chain need not satisfy this definition at all — omitting a stage can delete an infimum and break conditional completeness. The definition is stated free-standing so that the logic's single-timeline clauses consume exactly this structure and no site geometry — the record tier's face of the site contract's second freedom.

A thermostat log makes the data concrete: stages  $T = \{0, 1, \dots, n\}$ , one sample per second, with  $\delta(m, m') = m' - m$  seconds and observations the measured temperatures. For the region  $E = \{x : x \geq 30\}$  the hitting set collects the hot samples, and the début  $\tau_E$  is the first hot second — or  $+\infty$ , if the log never gets hot. A hybrid version of the same log carries a controller reboot as two distinct stages separated by zero duration, which is exactly what the definition's first non-requirement permits.

*Remark 9.7* (The three record-indexed clause families, and what each consumes). The satisfaction clauses of the specification logic that consume a record's temporal structure sort into exactly three families, and the data of Definition 9.6 are precisely what makes each well-defined. A bare total order does *not* suffice.

1. *Débuts* ( $\tau_E = \inf\{t : x_t \in E\}$ ) consume conditional completeness of the order (item 1): in a general totally ordered set a nonempty bounded-below hitting set need not have an infimum, and the début clause would then have no referent.
2. *Metric-bounded windows* (the deadline operator  $F_{\leq \ell}$  and metric until clauses) consume the duration functor (item 2): the window of extent  $\ell$  anchored at  $t$  is  $\{s \in T : t \leq s, \delta(t, s) \leq \ell\}$ , and no such set is definable from the order alone.
3. *Cofinal recurrence* (“ $x$  visits  $E$  cofinally”: for every  $t \in T$  there is  $s \geq t$  with  $x_s \in E$ ) consumes only the order, through its cofinal structure; it is definable in any total order and needs neither infima nor durations.

Both standard instantiations satisfy all hypotheses at no cost: discrete records ( $T \subseteq \mathbb{N}$ ,  $\delta(m, n) = n - m$  or a per-step duration sum) are well-ordered, hence conditionally complete; interval records ( $T$  a real interval,  $\delta(s, t) = t - s$ ; more generally proper time along a worldline) are conditionally complete by completeness of  $\mathbb{R}$ .

Division of labour, recorded here because the site supplies the other half: *the record carries the duration structure, while the mechanism carries the tangency structure*; the two meet in the generator row, where a tangency-typed operator is compared against an inverse-duration-typed scalar.

Definition 9.6 is stated along one timeline, and on the classical site — where a window's stages are totally ordered, so any support lies along one timeline automatically — it is the whole story. Over a general



spacetime site it cannot be: two facts force a wider notion, and between them fix its shape. The first fact is geometric. In the Minkowski site a chronologically thick diamond's stage space contains causally incomparable pairs — kernel-checked as a witness in the canonical unit diamond, over any normed space of positive dimension (`QtslAux.DiamondTopology: minkowski_diamond_spacelike_pair`) — so a record supported on one timeline cannot span such a window: the restriction of a field to a diamond, the honest “trajectory” of the hyperbolic column, is not a timeline object, and total stage order stands revealed as the foliated  $(0+1)$ -dimensional shadow — the same diagnosis Remark 9.5 gives the classical site itself. The second fact is metric, and it is what makes the wider notion *forced* rather than designed. In a relativistic order the natural duration between causally related events — the Lorentzian distance, the *supremum* of proper time over the causal curves joining them — satisfies the *reverse* triangle inequality: it is superadditive, so the additivity axiom (R2) fails for any single global valuation built from it. What is additive is a different functional — the proper-time integral along a *fixed* worldline, one functional per timeline — so duration must enter timeline by timeline: precisely the form the site's duration structure (T3) already takes, the classical definition's global  $\delta$  being its one-timeline shadow.

**Definition 9.8** (Record over a window). Let  $W$  be a window of a structured spacetime site, with stage poset  $|W|$  and duration functors  $\delta_\gamma$  on its timelines (T2–T3 of Definition 9.4). Let  $R_0 = (T_0, \leq, \delta_0, \mathcal{O}, x)$  carry the data of Definition 9.6 per maximal chain as below, together with a monotone *stage map*  $m: T_0 \rightarrow |W|$  whose fibers are the same-instant stacks (order-convex,  $\delta_0$ -null) and which carries each maximal chain of  $T_0$  onto a timeline of  $W$  with  $\delta_0 = \delta_\gamma \circ (m \times m)$  there. The pair

$$R = (R_0, m)$$

is a *record over  $W$*  when, for every maximal chain  $\gamma_0$  of  $T_0$ , the restriction  $(\gamma_0, \leq, \delta_0, \mathcal{O}, x|_{\gamma_0})$  is a record in the sense of Definition 9.6.

A record over a window is a coherent bundle of its own worldline shadows: every maximal-chain restriction is a classical record, matched onto the window's timeline against the site's own valuation — and the record's carrier presentation is the closed set  $\text{cl}\{(m(t), x_t)\}$  — single-valued exactly off the stacks and jump instants, multivalued at them, with single-valuedness a certified stratum rather than a membership condition (the general-support countermodels of the closing ledger force the same demotion independently). (Maximal chains pass through every stage of the support — Hausdorff's maximal principle, part of the ambient classical mathematics — so the bundle covers the record; and the restriction to a *non*-maximal chain need not be a record, conditional completeness not being hereditary, which is why the definition tests exactly the maximal ones.) The record itself is site-relative data: the pair  $(T, x)$  carries no duration, and its timeline shadows become classical records only against the window's own valuations  $\delta_\gamma$ , supplied by (T3) before any clause runs. The clause families of Remark 9.7 relativize accordingly, timeline by timeline: débuts and metric-bounded windows evaluate on the shadows  $(\gamma, \leq, \delta_\gamma, \mathcal{O}, x|_\gamma)$ , where conditional completeness and (R1)–(R2) hold by definition, and cofinal recurrence reads the support's own order directly. Each shadow is a free-standing classical record, and the clauses read nothing but the shadows — duration reaches the logic packaged inside them, and the coverage, the site structure the invariant of (T1) fences off, enters nowhere. (Which timelines a claim quantifies over — some worldline through the record, or every one — is spec-level structure one storey up, not record data; none of the three clause families ever needed a global début, and should one ever be wanted, its referent is the entry *antichain* of a hitting region rather than an infimum stage.)

*Conservativity*: over the classical site a window's stages are totally ordered, so every subset is a chain and a stack-free record's support ( $m$  injective) has exactly one maximal chain — itself — and Definition 9.8 asks exactly that  $(T_0, \leq, \delta_0, \mathcal{O}, x)$  be a record in the sense of Definition 9.6 against the window's own valuation  $\delta_0(s, t) = t - s$  read through  $m$ . That is precisely the valuation the classical artefacts' records carry (Remark 9.5), so those records *are* records over their windows, with nothing re-indexed and nothing added: over a totally ordered window the two definitions coincide. *Two support classes, one definition*: a record supported on a single chain is a *worldline probe* — an observer's history through the window — while a record with chronologically thick support is a *field restriction*, a PDE trajectory over a diamond being the graph of the field over the diamond's interior; and reading a record along one of its timelines — the observer's act — is literally the definition's restriction component. A clock map, in particular, remains what the diamond column says it is: a way of *reading* a record along a worldline, never part of what a record is.

The organizing principle of the temporal layers — call it the *records thesis* — is the resulting division of labour:

Path formulas are evaluated along records, and consult only the structure a single record carries — its order and its durations. The site governs how records glue into a System’s trajectory space, and only that: its cross-record geometry enters no satisfaction clause. The denotation says what lives on the glued space. Composition of Systems is summation of their beliefs, each transported to the joint records along the wiring: the composite’s belief is the sum of its parts.

Each clause of the thesis bounds the blast radius of the others. Because satisfaction consults only record-level structure, generalizing the time base cannot silently change what any specification means; the three satisfaction clauses that consume the record’s order — *début*, *until*, *cofinality* — are flagged in the semantics as *record-indexed*, so that a new site re-indexes them rather than re-defining them (a re-indexing Definition 9.8 realizes: over a general support they evaluate along maximal chains). Because the denotation is a measure-like object *over* records rather than a new kind of record, the imprecise apparatus of section 9.5 transports along the record direction by construction — with the caveat stated: its extension theorems are proven on the discrete-time record poset, and each new site owes the transport of exactly those theorems to its own record structure (a named obligation of the logic-site contract of section 9.2, not a free consequence of the flagging). And the thesis has a sharp outer boundary, drawn in subsection 9.2.7: quantification *across* records is admitted only at the specification logic’s outer layer, as imprecise-probabilistic claims about the ensemble — nested path quantifiers in the style of branching-time logics have no inner-layer home, by the same stratification.

One reading of records is fixed here because everything in section 9.4 depends on it: the *speed-bounded* records over a window of the site — those whose difference quotients, along each timeline of the support, lie in a fixed ball of the tangency’s action bundle, the single metric hypothesis the record layer keeps — are read *as their graphs*, closed subsets of a stage-times-state ambient, and the class of such records over a window carries the Fell topology on closed sets. (A general record — of Definition 9.6 or of Definition 9.8 — carries no topology and no closed-graph condition; the graph reading is a property of the speed-bounded class, not of the notion.) Under that reading the record class over each window — at the classical stratum, where a support is a timeline — is compact, and gluing along the interior coverage — directed open covers at fixed state fibre — is a homeomorphism onto a projective limit. The kernel receipts behind those sentences live at that stratum; over general supports the graph reading transports objectwise verbatim, while compactness rests on closedness of the record locus — which does not bundle per timeline: the closed object over thick supports is an interior-pair relaxation of the record class, its boundary conditions surviving as certified strata — an obligation stated with its two site hypotheses in the closing ledger, not a free consequence. The reading is developed, with its kernel receipts, in section 9.4; nothing in the satisfaction clauses above changes under it, which is the records thesis doing its job.

## 9.4 The Record Carrier: Fell Hyperspaces over the Site

The records of a window live in a definite space — the strata to follow quantify over exactly it. Per section 9.3, a speed-bounded record over a window is its graph, a closed subset of a stage-times-state ambient, and the record class carries the Fell topology on closed sets. The Fell topology is moreover *the* hyperspace topology that glues along directed open covers — the descent theorem below, kernel-checked — so the record carrier is a sheaf of compact spaces for exactly the interior coverage the site already adopted (whose covers are directed by pair-filteredness), at fixed state fibre.

### 9.4.1 The Carrier, and Its Descent

**The ambient and the carrier.** Fix a window  $W$ . The *ambient* is

$$A(W) := \mathring{S}(W) \times \mathcal{V}^+(W),$$

the *open interior* of the stage space (the design decision justified by the seam analysis below) times the state fibre of section 9.2 with its cemetery adjoined — the one-point operation applied here, at the tier that consumes its output; by (T5) and the fibre commitments,  $A(W)$  is locally compact, second-countable

Hausdorff, the compactified factor compact metrizable outright. The *carrier* over  $W$  is

$$\mathbb{H}(W) := \text{CL}(A(W))_{\text{Fell}},$$

the space of *all* closed subsets of the ambient — the empty set included — under the Fell topology [75]: the topology generated by the classes  $\text{Hit}(V) = \{F : F \cap V \neq \emptyset\}$  for  $V$  open and  $\text{Miss}(K) = \{F : F \cap K = \emptyset\}$  for  $K$  compact. Restriction along a site morphism  $u: W' \rightarrow W$  is clip-then-push:

$$\rho_u := (- \cap (|u|(\dot{S}(W')) \times \mathcal{V}^+(W))) \circ (\mathcal{V}^+(u))_*,$$

clip to the open sub-band, then push the state coordinate forward along the (proper) restriction of the state sheaf. Two lemmas make every  $\rho_u$  Fell-continuous, and each is short enough to give whole. *Clip-to-open is Fell-continuous*: for  $U \subseteq Y$  open, under  $F \mapsto F \cap U$  the preimage of  $\text{Hit}(V)$  ( $V \subseteq U$  open) is  $\text{Hit}(V)$ , and the preimage of  $\text{Miss}(K)$  ( $K \subseteq U$  compact) is  $\text{Miss}(K)$  — both subbasic. *Pushforward along proper continuous maps is Fell-continuous*: for  $f: Y \rightarrow Z$  proper continuous into a locally compact Hausdorff  $Z$  (so images of closed sets are closed), under  $F \mapsto f[F]$  the preimage of  $\text{Hit}(V)$  is  $\text{Hit}(f^{-1}V)$  (open, by continuity) and the preimage of  $\text{Miss}(K)$  is  $\text{Miss}(f^{-1}K)$  (compact, by properness). The two subbasis families split the work exactly — continuity feeds the hit face, properness feeds the miss face. The same split carries the descent theorem:

**Theorem 9.9** (Fell descent over directed opens). *Let  $Y$  be a topological space and  $\{U_i\}_{i \in I}$  a directed family of open subsets (any two contained in a third) with union  $Y$ . Then the clip maps  $\rho_i: \text{CL}(Y) \rightarrow \text{CL}(U_i)$  induce a homeomorphism*

$$\text{CL}(Y) \xrightarrow{\cong} \varprojlim_i \text{CL}(U_i)$$

*onto the space of compatible families, the limit carrying the limit (initial) topology.*

*Proof.* Bijectivity: injectivity is  $F = \bigcup_i (F \cap U_i)$ ; for surjectivity a compatible family  $(F_i)$  glues to  $F := \bigcup_i F_i$ , closed because closedness is local, with  $F \cap U_i = F_i$  (a point of  $F \cap U_i$  lies in some  $F_j$ ; directedness gives  $U_k \supseteq U_i \cup U_j$ , and  $F_k \cap U_i = F_i$  collects it). Topology: each  $\rho_i$  is continuous by the clip lemma, so the Fell topology refines the initial topology; conversely every Fell-subbasic class of  $\text{CL}(Y)$  is pulled back from a level —  $\text{Hit}(V) = \bigcup_i \rho_i^{-1} \text{Hit}(V \cap U_i)$ , since a closed set hits the open  $V$  iff it hits some  $V \cap U_i$ , and for compact  $K$  directedness places  $K$  inside a single  $U_i$ , whence  $\text{Miss}(K) = \rho_i^{-1} \text{Miss}(K)$ .  $\square$

Both subbasis families again carry the theorem, and this time the split is the whole of it: *hits are local* (an open set is the union of its cover-traces) and *misses are compactly supported* (a compact miss-parameter lands inside one cover member, by directedness). The Vietoris topology fails the second half — a miss-of-closed parameter need not land in any cover member — and the refuting fixture shows the miss face sharp: over a directed cover of  $\mathbb{N}$ , the miss class of a closed noncompact parameter fails to be open in the limit topology, by a witness (a far-enough singleton, whose clips agree with the empty record's on any finite set of levels) that defeats every hyperspace topology carrying the closed-miss upper face. Of the two candidate miss faces, then, only the compact one descends, and Fell's two design decisions (hit-of-open, miss-of-compact) are respectively the local-character axiom and the finite-support axiom of a sheaf-ready topology. Interior covers of a window are pair-filtered (the kernel-checked bicontinuous-poset lemma of section 9.2) and their interiors union to  $\dot{S}(W)$ ; crossing with the covered window's own state fibre preserves both properties. Hence:

**Proposition 9.10** (The carrier is a sheaf of compact spaces). *Consider a cover of the site's coverage satisfying the two hypotheses the kernel theorem consumes: its members' stage interiors, embedded in the covered window, form a directed family of open subsets whose union is the window's interior (interior coverages supply this — pair-filteredness is directedness, and openness and exhaustion are the coverage's construction); and its members' ambients carry the covered window's state fibre (every cover of the classical instantiation; every cover under a state sheaf constant on the window being covered). For every such cover,  $\mathbb{H}(W) \rightarrow \varprojlim_{W' \in \text{cover}} \mathbb{H}(W')$  is a homeomorphism, and on that stratum  $\mathbb{H}$  is a sheaf on  $(\mathcal{T}, J)$  valued in compact Hausdorff spaces. (A site whose coverage violated the first hypothesis would owe its own descent theorem — the kernel fixtures below show both directedness and the miss face sharp — and under a genuinely varying state sheaf the restriction legs compose the clip with a proper pushforward, whose sheaf property is prose-grade design, not an instance of the kernel theorem.)*

Compactness arrives with no hypotheses at all:  $\text{CL}(Y)$  under Fell is compact for *every* ambient  $Y$  (an Alexander-subbasis argument), Hausdorff when  $Y$  is locally compact Hausdorff, and equal to the Vietoris

hyperspace on compact Hausdorff ambients. Over an unbounded ray ambient, compactness of a record class is a theorem with hypotheses; over the site it is a corollary of closedness in a hyperspace that is compact for free. Nothing of Fell's noncompact machinery is thereby discarded: it is *localized*. In the classical instantiation windows have finite duration, so the unbounded horizon vanishes from every object; what stays noncompact per window is the open interior (seams) and the fibre (escape), and the compact-miss face polices both by one mechanism — the empty set is a closed point of every  $\mathbb{H}(W)$ , isolated only over a compact ambient (a zero-duration window has empty interior and carries the one-point trivial carrier; every nondegenerate window's open-interior ambient is noncompact), and over the nondegenerate windows it is the limit of records draining out of every compact: total escape, escape out a seam, and escape to  $\infty_W$  are boundary members of the carrier, rather than pathologies excluded by fiat.

**The seam fork, and why sheafification decides it.** The ambient uses the *open interior* of the stage space, and this is forced, not chosen. Restriction to a *closed* subwindow fails Fell-continuity at records with a jump exactly on the seam: a family of records with jumps at times  $u' + \frac{1}{n}$  converges in Fell to the record jumping at  $u'$ , but the closed-clip images stay at Hausdorff distance  $\geq 1$  forever, because the limit's clip carries a zero-duration post-jump stage on the seam that no approximant's clip carries. (The witness is hybrid-only: for pure flows under the speed bound, closed clipping is continuous; the fork is exactly the price of jumps. It is kernel-checked as a refuting fixture beside the clip lemma.) The resolution is that the site had already made the decision: the closed-window record presheaf is not a sheaf — a matching family over a window's interior cover determines the record on the union of the members' interiors, which is the window's open interior, and *nothing at the outer seams* — and its sheafification is precisely the open-interior carrier above. Read with the site's own glosses this stops being a technicality: the site quotients translation, so a window's outer seams have no absolute identity — data pinned to this window's own boundary is exactly the data translation-homogeneity forbids. Sheafification forgets the seams because homogeneity requires it; seam data returns in the anchored-tower presentation (a family of windows sharing a fixed initial stage), in the same movement that returns absolute time as the internal clock, which is the germ row of the logic-site contract (F3).

One coverage, three independent derivations, no shared premises: the interior coverage is simultaneously the behavioural choice (Schultz and Spivak omit endpoint gluing for composability of behaviours), the continuity choice (closed clipping is discontinuous at seam jumps), and the descent choice (Theorem 9.9 needs directed opens). A choice made for behavioural reasons is forced again by continuity and a third time by descent.

## 9.4.2 The Record Locus

**The record locus is a closed subsheaf.** Within each  $\mathbb{H}(W)$ , the *record locus*  $\mathbb{R}(W)$  comprises the closed sets that are graphs of speed-bounded records over relatively closed sub-timelines of the window interior — the classical, single-timeline locus throughout this paragraph; over the general supports of Definition 9.8 the closed object is an interior-pair relaxation, per the ledger's named obligation — unanchored and lifetime-windowed: records may begin and end strictly inside the window (early stop and late start are membership, not defect, in a translation-quotiented site), the empty record is a member, and the class is Zeno-inclusive: Zeno-boundary points are themselves honest records (a Zeno stage domain, lexicographically ordered, is conditionally complete — its infima land on the lowest branch of the stack), and locally finite *event* structure is a certified stratum inside the carrier — closed (a continuous image of the compact parameter class; equivalently bounded alternation between separated regions, a class built from time-ordered finite hit demands) and disjoint from the chatter boundary — carried by the crossing-budget row family, not the locus's definition: the budget was never a compactness hypothesis; its remaining job is mechanism-side progress — a ghost storm (state-invisible events accumulating) stalls a System without casting a shadow, which is exactly why liveness rows consume the budget and safety rows never did. The structure theorem factors through a generic principle: call a family of per-level loci  $R_i \subseteq \text{CL}(U_i)$ , with ambient locus  $R_0$ , *local* when membership in  $R_0$  is equivalent to membership of every clip in every  $R_i$ . A levelwise-closed local locus is closed (an intersection of clip-preimages of closed sets, by clip-continuity), hence *compact for free* in the hypothesis-free compact hyperspace; compatible families levelwise in the loci glue into the ambient locus; and the descent homeomorphism restricts to a homeomorphism of the ambient locus onto the levelwise-locus subspace of the projective limit — the closed-subsheaf statement. The record instantiation discharges the two obligations: the speed-graph condition is closed (a violating pair has a positive defect

margin that two hit demands preserve — pure hit face, no compactness consumed), and it is local over directed covers, directedness entering at exactly one step (a violating pair’s two points must meet in one cover member). All of this is kernel-checked, together with the sharpness fixtures that bound it: the speed bound cannot be dropped or existentially absorbed (a steepening family of affine graphs Fell-converges to a vertical fibre, which is no graph at any speed — so the bare-graph locus and the all-speeds union are not closed), and directedness is essential at statement level (over a nondirected two-member cover there is a two-point set, every clip of which is a record, that is no record — levelwise record-hood does not imply record-hood, and compatible levelwise records can glue to a non-record). The arc reading is also a theorem rather than a gloss: membership in the speed locus over a product ambient is equivalent to being the graph of a Lipschitz arc on a closed sub-timeline, with relative closedness of the domain *derived* — a converging sequence of domain points has its states Cauchy by the speed condition itself, completeness of the state fibre collects the limit, and closedness of the graph does the rest.

*Remark 9.11* (Mechanization status of the carrier). Kernel-checked, in the semantic-universe strand of the toolkit’s proof-artefact repository (modules named per statement; the receipts conventions are [section 9.13](#)’s): the Fell topology, its hypothesis-free compactness, Hausdorffness over locally compact ambients, and the polarity lemmas of the hit and miss faces (SuFunctor.Universe.FellTopology, including FellClosedS.compactSpace with no hypotheses on the ambient, and fell\_eq\_vietoris on compact Hausdorff ambients); clip-to-open continuity with its mutant fixtures, and proper-map pushforward continuity (SuFunctor.Universe.FellRestrict: continuous\_restrictOpen, continuous\_image\_of\_isProperMap); the seam witness (SuFunctor.Universe.FellSeamWitness, refuting closed-band clipping through the zero-duration post-jump stage); Theorem 9.9 as FellClosedS.projLimHomeomorph with the glue-clip roundtrips and continuity in both directions (SuFunctor.Universe.FellDescent), together with the sharpness fixtures (a nondirected cover on which the canonical map fails surjectivity; the Vietoris comparison on which the miss face’s compact support fails); the record locus (SuFunctor.Universe.FellRecordLocus: the generic local-locus package IsLocalLocus.isClosed/isCompact/glue\_mem/homeomorph, the speed locus speedLocus with isClosed\_speedLocus, isSpeedGraph\_iff\_forall\_clip, speedLocusHomeomorph, membership one-liners including bot\_mem\_speedLocus and isLocallyClosed\_speedLocus, and the arc reading isSpeedGraph\_iff\_exists\_lipschitzOnWith) with its refuting fixtures (SuFunctor.Universe.FellRecordWitness: the steepening family, the vertical fibre, the nondirected specimens); and the one-point compactification, packaged as a hyperspace correspondence: adjoining the point at infinity is a homeomorphism  $CL(Y) \cong \{B \in \mathcal{K}(Y^+) : \infty \in B\}$  (SuFunctor.Universe.FellOnePoint), holding for arbitrary ambients at the Fell stratum — reading the upstairs topology as Vietoris consumes local compactness and Hausdorffness of the ambient (topology\_onePoint\_eq\_vietoris), which every (T5) ambient supplies — so the correspondence carries the Fell form of a noncompact ambient into the nonempty-compacts Vietoris presentation the class stratum of [section 9.7](#) is stated over. The scope boundary: the mechanized record locus is the pure-flow (single-timeline) stratum; the hybrid (multivalued-instant) stratum rides the chart route — the shadow of the compact parameter class, with alternation-bounded classes closed by hit-face arithmetic — and its mechanization is an obligation recorded in [section 9.13](#).

### 9.4.3 Records Determine Laws

**The capacity presentation.** The denotation stack of [section 9.5](#) puts measures — and beliefs over measures — on the record carrier, so the carrier owes the stack a determination principle: which observations pin a law? The answer is certificate-shaped. Call a family of *hit-and-miss generators* over an indexed open basis  $B: \iota \rightarrow \text{Open}(Y)$  with compact closures the classes

$$\nabla_B(\text{hits}, \text{misses}) = \left\{ F \mid (\forall i \in \text{hits} : F \cap B_i \neq \emptyset) \wedge (\forall j \in \text{misses} : F \cap \overline{B_j} = \emptyset) \right\},$$

for finite index sets *hits*, *misses*. These generators form a topological *basis* of the Fell hyperspace; with countably many basis indices the hyperspace is second countable, and the Fell hyperspace of a (T5) ambient is therefore *compact metrizable*. The determination principle:

**Theorem 9.12** (Capacity presentation). *Over a locally compact Hausdorff ambient with a countable open basis of compact closures, two finite Borel measures on the Fell hyperspace that agree on every hit-and-miss generator are equal. (The generator family is a  $\pi$ -system — intersecting generators unions their index sets — containing the whole*



space as the empty generator, and it generates the Borel  $\sigma$ -algebra because the generators form a countable basis; the empty generator is what subsumes total-mass agreement, so the statement carries no separate mass hypothesis.)

This is the determination half of the Choquet–Matheron theorem for random closed sets [175], on the Fell carrier, in a form a checker row can consume: a *distinctness* certificate is one generator — two finite sets of basis indices — plus two evaluations, and identity is the countable schema over the enumerated generators. With the observation basis below, the indices are rational data, so laws on the record locus are determined by their evaluations on a countable, *rationaly indexed* family of hit-and-miss observations: records determine laws, with the observations — though not the evaluated measures — exact data of the kind the certificate stratum consumes. Two boundary facts are kernel-checked beside the theorem and are content, not caveats: the compactness of the miss parameters is what makes the generators open at all (over any noncompact ambient, the miss class of the whole space is not Fell-open — the escape specimen again); and the hit face *alone* does not determine, at the frame level — there are two Scott-continuous functionals on the opens of the hyperspace agreeing on the entire hit-generated (lower-Vietoris) topology and differing on a compact miss class. Escape mass is exactly what the hit face cannot see; the miss face exists, at determination level, to count escape. (Classically, hit-only determination survives via  $\sigma$ -algebra complements and a pinned total mass; frames have no complements, and the fixture is that difference made a theorem.)

*Remark 9.13* (Mechanization status of the capacity presentation). Kernel-checked, module `SuFunctor.Universe.FellCapacity`: `nablaGen`, `isTopologicalBasis_nablaGen`, the second-countability instance, Theorem 9.12 as `measure_ext_of_nablaGen`, the separation corollary `exists_nablaGen_measure_ne`, the Scott-continuous frame form `scottContinuous_ext_of_nablaGen`, and the two boundary fixtures `not_isOpen_disjoint_univ` and `exists_scottContinuous_eq_on_lower_ne`. The full localic Choquet–Matheron statement — the frame of the hyperspace locale presented by these generators, with continuous valuations determined by generator values — is kernel-checked as well, in the point-free library: the presented frame is built as the free frame on the hit-and-miss data semilattice, quotiented by the least nucleus forcing the transposed hit-and-miss laws (`Pointfree.Frame.Presentation`, `Pointfree.Frame.HyperspacePresentation`), and on it two Scott-continuous valuations with finite generator values agreeing on every  $\nabla$ -generator are equal, distinct ones separated by a single generator (`Valuation.ext_of_nabla`, `Valuation.exists_nabla_ne`). The engine (`Pointfree.Valuation.ext_of_generators`) is frame-general — any frame carrying a meet-closed, join-generating generator family — so it applies to the presented frame at any relation set and to the spatial frame of opens alike (`Valuation.ext_of_isTopologicalBasis`): no points, no countability, no  $\sigma$ -algebra, and the finiteness hypothesis sharpened from total mass to generator values — modularity manufactures from generator values the finite-join agreement the Scott-continuous form above hypothesizes. What was deliberately not claimed remains so: the existence half of Choquet’s theorem, capacity functional to law, is not claimed — the stack consumes determination only.

**The observation basis.** The basis the generators range over is fixed once. For the toolkit’s actual ambients — stage intervals times  $\omega$ -semialgebraic state fibres, the live part  $\mathring{S}(W) \times V(W)$  presented inside a real coordinate space and the cemetery adjoined by the one-point operation — it is the family generated under finite unions and intersections by two rational families. On the live part: rational-coefficient semialgebraic opens traced onto  $\mathring{S}(W) \times V(W)$ , refined to the subfamily with compact closures (rational opens are a base of the coordinate space, and tracing preserves countability, lattice closure, and the base property). At the cemetery: products  $J \times (\mathcal{V}^+(W) \setminus K_n)$  of a rational stage interval  $J$  with compact closure in  $\mathring{S}(W)$  and the complement of a member of a fixed rational compact exhaustion  $K_0 \subseteq K_1 \subseteq \dots$  of the fibre — an exhaustion exists because a locally compact, second-countable fibre is  $\sigma$ -compact, and its members can be taken carved by rational data (traces of closed rational balls suffice). The result is countable, closed under finite unions and intersections, and a base of the topology of every such ambient: at live points the traces already form a base, since the live fibre is open in its compactification; at points over  $\infty_W$  the exhaustion complements are cofinal among neighborhoods, since every compact subset of  $V(W)$  lies in some  $K_n$ ; and closures stay compact ( $\bar{J} \times \mathcal{V}^+(W)$  is compact — the compactified fibre is compact by construction). This is also the honest shape of what a certificate can check near escape: a basis neighborhood of the cemetery says exactly “eventually outside  $K_n$ ” — finite rational data, with no carving of the point  $\infty_W$  demanded. A general (T5) ambient, which need not carry any coordinate presentation, uses an abstract countable basis of compact closures, which local compactness and second countability supply; the capacity theorem



consumes only that. Two consequences. First, the record-class observation domain is insensitive to the state-space dial of [section 9.2](#): the fibre carving class and the test-region class are independent dials, and the  $\omega$ -semialgebraic upgrade moves only the former — test regions stay rational-semialgebraic, which is where certificate rows need them, since a certificate row refuting  $\text{Hit}(V)$  consumes  $V$ 's polynomial presentation. Second, the hit-and-miss generators over this basis are precisely the *crescent* shape the class stratum of [section 9.7](#) builds its spec classes from — one closed demand cut by finitely many open escapes — so the carrier's determination principle, the checker's test regions, and the spec logic's atoms speak one geometry. In the vocabulary of coalgebraic logic the generators are Moss's cover modality  $\nabla$  applied to the basis (the presentation of the Vietoris powerlocale by generators and relations [233]); the internal one-sentence reading — the carrier is the Vietoris powerlocale of the compactified ambient object, constructed inside the sheaf topos, with the ray-ambient carrier its shadow along the anchored tower — is design at prose grade, recorded because it explains why the objectwise-then-glue construction is canonical (presentations by generators and relations pull back along inverse-image functors), while everything this chapter builds on is the external, objectwise, kernel-checked form above.

#### 9.4.4 What the Carrier Hands the Later Sections

Four interfaces, each consumed downstream with its receipt already in hand: (i) per window, a compact Hausdorff second-countable hyperspace with a crescent basis — the home of the class stratum; (ii) Fell-continuous restrictions, so preimages of locally closed classes are locally closed, and class-level structure transports contravariantly along the site for free; (iii) the descent homeomorphism, so any fibrewise construction commuting with clipping globalizes along covers with no further argument — the fibre sheaf of [section 9.8](#) consumes exactly this; (iv) the determination principle, so laws over records are determined by their evaluations on countably many rationally indexed observations — the exact-stratum face of the denotation stack.

### 9.5 The Universe of Beliefs

#### 9.5.1 Heights, Grades, and the Order

**From verdicts to heights.** In the behavioural reading, a System denotes a set of trajectories and a specification denotes a superset test: allowed or forbidden, no third value, no weighting, and no way for the model to confess that it may fail to produce a full trajectory at all. The present universe replaces the verdict object with heights. A System's denotation is a *belief* — in full, a *level-of-inconsistency functional* — on probability distributions over the record space: a lower semicontinuous function

$$\sigma: \Delta(\mathcal{X}) \longrightarrow [0, \infty],$$

whose value  $\sigma(\mu)$  is the level of inconsistency between the conceivable distribution  $\mu$  and what the object asserts —  $\sigma(\mu) = 0$ : fully consistent;  $\sigma(\mu) = \infty$ : excluded outright; intermediate heights price partial inconsistency [54, 209]. The functional is the belief; the distributions are what it scores — imprecision, nondeterminism, and graded confidence are all shapes of the belief, never features of a distribution. (The mechanized development carries its functionals on the subprobability carrier  $\Delta_{\leq 1}(\mathcal{X})$ ; the members of the universe are the *radial* ones — height determined by a law's direction alone — and on those the compile of [section 9.6](#) is exact: graded functionals on probability laws on one side, closed radial *sets* of subprobability laws on the other, the functional presentation for arithmetic and the body presentation for order. The partiality paragraph below says how a leaky process reads.) Membership in the universe is lower semicontinuity, *and nothing else*: convexity is not an admissibility condition, and the convex members form distinguished subclasses of the universe rather than the universe (Remark 9.20). Specs consume the same stock one storey up: a Spec's denotation is a constructible class of the bodies these beliefs compile to ([section 9.6](#) and [section 9.7](#)), so the beliefs of this section are the raw material of both sides of the eventual judgments, and of the composition calculus's fibre data.

A single coin toss already exhibits the range. Over the two-point record space  $\{\text{heads}, \text{tails}\}$ , three members of the universe: the belief vanishing at the fair law and infinite everywhere else (a precise model — the coin is fair, and nothing else is admitted); the indicator of the closed set  $\{\mu : \mu(\text{heads}) \leq \frac{1}{3}\}$  (a hard constraint, with no preference expressed inside it); and the relative-entropy belief  $\mu \mapsto \text{KL}(\mu \parallel \text{fair})$  (every

law admitted, priced by its divergence from fairness). The first two are the degenerate faces of Remark 9.22; the third is a smooth interior member; and the order below compares all three on one scale.

**The order.** The universe is ordered pointwise:  $\sigma \geq \varphi$  when  $\sigma(\mu) \geq \varphi(\mu)$  for every distribution  $\mu$  — the greater belief is the stronger assertion, at least as inconsistent with every distribution. Read as implication — the stronger belief implies the weaker — the pointwise order makes the universe a thin category of beliefs, with implications as its morphisms [54]. This pointwise order is the order of the universe at the present storey; on the radial stock, under the compile of section 9.6, it is exchanged, anti-monotonically, with *inclusion* of bodies, and inclusion is the form in which everything above this storey — classes, fibres, and the judgments of later chapters — consumes it. No satisfaction judgment is defined in this chapter; what the chapter fixes is the ordered universe those judgments will be read in.

*Remark 9.14* (Mechanization status of the universe stratum). The universe stratum is kernel-checked in three storeys, and the carrier is held pointfree first. *The pointfree storey*, at record-space generality with opens and covers first-class: the carrier is the space of valuations on the frame of opens — strict, monotone, modular into  $[0, \infty]$ , Scott continuity carried as a mixin exactly where a theorem consumes it (Pointfree.Valuation.Basic) — with the whole polarity package proved there: the promotion adjoining the cemetery outcome is a homeomorphism onto the induced topology, its mass-monotone half-spaces the cemetery coordinate’s lower half-spaces (promoteHomeomorph, induced\_promote\_eq); evaluations lower semicontinuous at opens, mass upper semicontinuous, liveness walls closed (Pointfree.Valuation.Topology); and escaping mass is witnessed without a single point — the free-ultrafilter valuation fails Scott continuity against a directed family of opens, and the Dirac net converges to it in the lower topology (Pointfree.Valuation.EscapeWitness). *The classical storey*, as the bridge: restricting a Borel measure to opens is a valuation, Scott-continuous exactly on the  $\tau$ -additive stratum — and on second-countable carriers every Borel measure is  $\tau$ -additive [83, 414O], so the measure-theoretic dialect of this chapter is a corollary of the pointfree one, the hypothesis class mechanized as stated (Pointfree.Valuation.Bridge). The classical subprobability carrier itself is kernel-checked at that generality:  $\Delta_{\leq 1}(\mathcal{X})$  as SubProbabilityMeasure under the narrow topology, its restriction map into the pointfree carrier continuous (SubProbabilityMeasure.continuous\_toSubProb), the promotion’s classical face the measure-level dagger chart  $\Delta_{\leq 1}(\mathcal{X}) \cong \Delta(\mathcal{X} \sqcup \{\dagger\})$ , a homeomorphism for the weak topologies (daggerChartMeasure), with the compactness dividend for compact carriers proved by two independent routes (compactSpace\_of\_daggerChartMeasure) and metrizability over metrizable separable Borel carriers (metrizableSpace\_subProbabilityMeasure, Pointfree.Measure.DaggerChart). Beliefs, the compile, and the closedness dictionary over this general carrier are the subject of Remark 9.18 (SuFunctor.Universe.General). *The finite storey*: the carrier  $\Delta_{\leq 1}(V)$  over a finite outcome type as SuFunctor.Universe.belCarrier/BelState (compact, convex — convex\_belCarrier, isCompact\_belCarrier), with the probability face as probCarrier/ProbState; beliefs on the subprobability carrier as Penalty — the mechanization’s name for the type of beliefs — with lower semicontinuity the standing membership hypothesis; the pointwise order itself, under the name the development gives its comparison relation (SuFunctor.Universe.Judgment: Sat, definitionally  $\varphi \leq \sigma$  — a name this chapter reads purely as the order); the order-level faces of the connectives (conjunction’s universal property sat\_sup\_iff; the functional-level disjunction minimum with its universal property entails\_inf\_iff and the hedging witness exists\_sat\_inf\_not\_sat — section 9.7 adjudicates which disjunction the logic’s  $\vee$  is); the closure package for the operations the calculus performs on members (SuFunctor.Universe.Closure: lower semicontinuity closed under addition, binary and countable suprema, and affine tilts); and two generation theorems that pin the universe to certificate-shaped data — rational threshold rows see the whole probability face (SuFunctor.Universe.Generation: iSup\_boxRows\_eq\_self, with countable index, stated for beliefs on the probability carrier), and over the compact finite-dimensional subprobability carrier quadratic gambles see the whole universe (SuFunctor.Universe.Tents: iSup\_tents\_eq\_self). The probability-face height data of the radial presentation below is mechanized separately as ProbPenalty. And the judgment tier itself is kernel-checked at the general carrier (SuFunctor.Universe.General: modules Faces, ClosureGeneral, GenerationGeneral): the order faces of the connectives — conjunction’s universal property, the functional-level minimum with its universal property, and the hedging witness, the last on every nonempty carrier (General.sat\_sup\_iff, General.entails\_inf\_iff, General.exists\_sat\_inf\_not\_sat) — together with wall citizenship (closed if and only if citizen, General.lowerSemicontinuous\_wall\_iff) and the wall algebra; the closure package over an arbitrary topological carrier — addition, binary and countable suprema, pullback, and tilts by bounded continuous nonnegative scores, the general

form of the finite storey's affine tilts; and generation by *gamble rows* — rational heights on finite menus of rational thresholds against bounded-continuous-gamble expectations, the certificate shape — as a directed supremum with no countability, metrizable, or compactness consumed anywhere (`General.isSup_gambleRows_eq_self`, through `isInducing_testProfile`: the carrier's topology is the initial topology of its gamble reads), and with countable index under second countability of the carrier, a hypothesis discharged for compact metrizable record spaces (`General.exists_countable_gambleRows_eq_self`, `General.secondCountableTopology_probabilityMeasure`); the quadratic-tent species stays finite-dimensional by nature — its general sibling is the gamble-row theorem. The record-space stratum —  $\mathcal{X}$  Polish, the standing structure below — is the semantic generality this chapter is written at, and the mechanization meets it at this storey; where a theorem below is kernel-checked at the finite stratum only, its remark says so.<sup>3</sup>

**Grades, and the two scales.** The multiplicative presentation of the same universe replaces heights by *grades*  $g = e^{-\sigma} \in [0, 1]$  (hard violation graded 0), under which addition of beliefs is multiplication of grades and the order reverses. Both scales are used below — heights compose additively under the wiring calculus, grades multiply, and the kernel-checked composition dictionary is deliberately stated through an abstract exponential so that no transcendental function enters the exact stratum. The grade scale is where a familiar family of models lives: for  $\sigma = \beta^{-1} \text{KL}(\cdot \| p)$  the penalized worst case is the Gibbs variational formula, and relative-entropy beliefs toward a reference — the inconsistency semantics of probabilistic dependency graphs of Richardson and Halpern [209] — sit in the universe as smooth interior members. The differentiator the universe adds is its boundary strata: credal flats (a set of laws held with no interior preference — nondeterminism is this imprecision, and no finite-temperature relative-entropy chart expresses a flat) and walls (the possibilistic face), which arise from the smooth chart only as its zero-temperature limit — though the *set-referenced* relative-entropy chart, the Hausdorff-KL embedding of Remark 9.20, expresses exactly a flat with relative-entropy grading off it: the intermediate stratum between the smooth interior chart and the walls.

**Partiality is first-class.** A belief is a functional on probability distributions; partiality enters through the *compile*, not through the domain. The body of a member (section 9.6) is a *radial* subset of the subprobability carrier  $\Delta_{\leq 1}(\mathcal{X})$  — it admits each law at every scale up to its grade, so admitted mass need not reach one — and at that storey the denotation is a set, no longer a functional. A leaky process — one that may halt, diverge, or exit scope — denotes along the same rays as its total counterpart, at the grade its surviving mass earns: *killing raises the level of inconsistency*, and the ray's grade axis is the only bookkeeping the leak requires. (The mechanized development also carries the promotion  $\Delta_{\leq 1}(\mathcal{X}) \cong \Delta(\mathcal{X} \sqcup \{\dagger\})$  as a kernel-checked chart — an affine homeomorphism, not a compactification — for the computations that want normalized mass; it is an available presentation, not a second axis of the universe.) The asymmetry the radial down-closure buys is the discipline the universe exists to enforce: leaked mass can only lower the probability of reaching a bad event, so safety certificates checked against full-mass semantics remain sound; what leak *lights* is liveness — lower bounds on good events feel every lost unit of mass — and a liveness claim cannot be laundered through a model that is allowed to quietly stop. The envelope apparatus below accounts for this exactly, and the kernel carries the counterexample that keeps the accounting exact (the leak-aware safe-side bound and its total-leak fixture, Remark 9.25).

**Connectives, composition, and the one inequality.** On denotations the logic's conjunction is the pointwise join,  $\sigma_1 \vee \sigma_2 = \max(\sigma_1, \sigma_2)$  — the least demand at least as strong as both — and lower semicontinuity is closed under it. Composition of *Systems*, by contrast, is addition: the wiring calculus of section 9.9 composes belief content by pushforward-then-add — a composite holding several beliefs at once asserts their sum, the overall belief being the sum of its parts [54] — and addition is not idempotent. The whole relationship between the two monoids is one inequality with an exact error term:

$$\max(\sigma_1, \sigma_2) \leq \sigma_1 + \sigma_2 = \max(\sigma_1, \sigma_2) + \min(\sigma_1, \sigma_2),$$

<sup>3</sup>One vocabulary caution: this finite storey is the composition ladder's *finite* stratum — finite outcome carrier. The ladder's *exact* stratum is a different rung: finitely supported joints over an arbitrary carrier, with arbitrary real weights. Over a finite outcome type the rungs collapse — the seam embedding is a bijection, so the finite storey, the exact stratum, and the general carrier present one object (`toMeas_surjective`); over infinite carriers they are genuinely nested, and Remark 9.33 settles which claims complete across each step, and which provably do not.

pointwise. Composition dominates conjunction — the sound direction: a bound certified against the join holds a fortiori against the composite — and the two agree at a distribution precisely when the beliefs are *crisp* there: the lesser zero, or the greater infinite. (In the extended nonnegative reals the identity’s min-term is not a cancellative error one may subtract: at a distribution where the two values are  $\infty$  and 1 the sum and the join agree by absorption while the lesser value is 1.) The *crisp locus* — crisp at every distribution — is exactly the agreement locus, and that is why deterministic intuition reads the two operations as one: on walls they agree, everywhere. (That no wall-fragment verdict can even observe the gap off the crisp locus is a two-line fact displayed in [section 9.12](#), where the wall fragment lives.) The gap itself is quantified conservatism in the safe direction, never unsoundness — the slice bracket  $\{\max \leq s/2\} \subseteq \{\text{sum} \leq s\} \subseteq \{\max \leq s\}$  for two contributors. The inequality and its sublevel bracket are kernel-checked at the finite-rational stratum and space-free (`QtslAux.Imprecise.MarginalCompat: sum_dominates_sup, sublevel_marginal_of_dominates`; the general form over arbitrary joint models in `QtslAux.Imprecise.MarginalCompatGeneral`). Disjunction is deliberately absent from this paragraph: at the functional level the pointwise *minimum* is the lattice join of demands and is sound for the single-conclusion fragment, but the class stratum of [section 9.7](#) shows that the specification logic’s disjunction must be read one level up, as plain union of classes, and that anything else — the minimum included, against class-valued conclusions — fails disjunction elimination. The connective’s true home is there.

### 9.5.2 The Standing Structure, and the Radial Compile

The record spaces everything above quantifies over are fixed once, and the next definitions carry the credal presentation of the universe’s members. The compile below ([Definition 9.17](#)) is an isomorphism: lower semicontinuous beliefs on *probability* distributions correspond exactly to closed *radial* subsets of the subprobability carrier — sets containing, with each law, its whole ray  $\{\gamma\mu : 0 \leq \gamma \leq 1\}$  down to the zero measure — with no residue on either side (kernel-checked at the general carrier, `SuFunctor.Universe.RadialUniverse: compileEquiv`). The two presentations are two charts on one universe, and each computation uses whichever is natural: algebra of beliefs — sums, divergence pricing, affine tilts — happens on the probability-face chart, where the operations are pointwise; order, gluing, and the classes happen on the body chart, where the order is inclusion. The grade dictionary  $g_S = e^{-f}$  carries one chart to the other.

The presentation meets the order as walls: a credal body  $S$  enters the order as its indicator  $\chi_S$ , under which domination is containment of finiteness sets — the wall reading of [section 9.12](#) — and the denotation’s own geometry is carried by the *body*  $S$  itself, which is what the classes of [section 9.7](#) quantify over. A compiled *law* enters as its ray — the body of the precise belief in that law — and every wall’s body contains the zero measure, because every body does: bottom-inclusion is a theorem of radiality, not a convention (`rwall_ne_top`: no wall built on the radial entry can exclude the empty process). The consequence is load-bearing for composition. Asserting  $x = y$  of a process claims only that no live behaviour violates it; the conjunction of jointly unsatisfiable constraints —  $x = y$ ,  $x = 0$ ,  $y = 1$  — denotes the *bottom body*  $\{0\}$ , the lawful belief that no live behaviour exists at all, never an ill-formed exclusion of the inconsistent world itself (kernel-checked as stated, with that three-clause example: `contradictory_composite_denotes_bot`). A construct that vanished only at one pinned nonzero law — an “equality floor” claiming the whole universe consistent — is not a member of the universe at all: no belief compiles to a pinned point (`compile_ne_singleton`), and feasibility is therefore never smuggled through an equality; it is asserted, when it is asserted, by the explicit liveness face of the presented classes ([section 9.7](#)).

Every System this book compiles denotes through the mechanism-and-gluing stack below: a credal body for the precise and glued stochastic faces, and, for adversary-quantified faces, the closed *convex hull* of the admitted family’s union (the hull-vs-union adjudication is [Remark 9.23](#)’s). An empty admitted family denotes the bottom body along with every other unsatisfiable composite; no compiled System falls outside the body calculus. The classical imprecise-probability apparatus lives on the same stock, and everything below is stated there.

**Definition 9.15** (Standing structure; credal bodies). *Standing structure.* Throughout the denotation stack, the *record space*  $\mathcal{X}$  is a *Polish space* — a separable, completely metrizable topological space — carrying its Borel  $\sigma$ -algebra. Polishness here is a theorem about the toolkit’s spaces, not a further dial: the handy widgets are all objects of the completed state-space category of [section 9.2](#) — countable families of  $\omega$ -semialgebraic sectors, where local closedness, the membership condition of that section, already implies  $G_\delta$  sector by sector, hence Polish (a countable disjoint union of Polish spaces is Polish) — except record spaces, which



are handled through their own mechanism involving the spacetime site and the Fell topology, and arrive Polish by their own route. In particular:

- finite sets are the dimension-zero  $\omega$ -semialgebraic spaces (discrete topology):  $\Delta(\mathcal{X})$  is the probability simplex in  $\mathbb{R}^{\mathcal{X}}$ , whose rational points are the exactly-representable distributions;
- semialgebraic sets carry the subspace topology from  $\mathbb{R}^n$ , and the locally closed  $\omega$ -semialgebraic sets of [section 9.2](#) — the carrier class of this book’s exact certificates, closed under finite disjoint union, containing the finite sets as dimension-zero instances, with the cemetery compactification applied at the carrier tier as in [section 9.2](#) — are the standing state-space class;<sup>4</sup>
- bag spaces with pieces in a common ambient are  $\omega$ -semialgebraic natively: a countable disjoint union of locally closed semialgebraic pieces of one ambient dimension is the class’s own bag pattern, every summand clopen in the union (its cemetery is the carrier-tier operation, as always — the bag’s local compactness is all the operation needs); the quotient of a semialgebraic set by a linear action of a finite group is again semialgebraic (embed by a generating family of polynomial invariants; the image is semialgebraic by Tarski–Seidenberg), which covers each symmetric power  $\text{Sym}^n(X) := X^n/S_n$  individually. The full bag space  $\coprod_{n \in \mathbb{N}} \text{Sym}^n(X)$  has unbounded ambient dimension across its summands — which no common-ambient tagged union can host — and it is a *first-class object* of the completed state-space category: a countable family of sectors, each symmetric power its own sorted-cone carving (one basic rational piece, `sortedCone`, `mem_sortedCone_iff_monotone`), the space literally the coproduct of its sectors (`bagObj`, `bagIsColimit`, `SuFunctor.StateSpace.FamCompletion`), Polish since every sector is (each clopen in the union). At the Borel stratum this is the bag monad of Dash and Staton [58], and the Bag former of [chapter 12](#) is its System-level lift;
- manifold-flavoured state spaces (Lie groups, Riemannian configuration spaces) enter through their polynomial-data presentations: in this book’s practice they are carved by polynomial data, and every compact smooth manifold is diffeomorphic to a real algebraic set (Nash–Tognoli), so the class loses nothing a certificate could have consumed; all manifolds in this book are second countable;
- record spaces are the exception, handled through the spacetime site and the Fell topology rather than by carving: per window the record carrier of [section 9.4](#) is a compact metrizable hyperspace — Polish outright — with laws pinned by the countable rational determination of [Theorem 9.12](#); for record spaces presented by gluing along finite windows, countable products and countable limits of Polish spaces are Polish, and the cylinder  $\sigma$ -algebra generated by the finite-window restriction maps coincides, in the standard presentations, with the Borel structure of the product realization.

$\Delta(\mathcal{X})$  denotes the set of (countably additive) Borel probability measures on  $\mathcal{X}$ , and  $\Delta_{\leq 1}(\mathcal{X})$  the set of *subprobability* measures (total mass at most 1). Both carry the *weak* (narrow) topology; for Polish  $\mathcal{X}$  this topology on  $\Delta_{\leq 1}(\mathcal{X})$  is metrizable, and for finite  $\mathcal{X}$  it is the Euclidean topology on the subprobability polytope, which is compact. All occurrences of “topologically closed” and “lower semicontinuous” in this stack refer to this topology. The total mass  $\|\nu\| := \nu(\mathcal{X})$  is weakly continuous — and that continuity is why the weak topology, rather than its locally compact cousin, is the topology of record: under the *vague* topology (tests vanishing at infinity, on a locally compact carrier) the subprobability space would be compact at the price of letting mass leak silently in limits —  $\delta_n \rightarrow 0$  vaguely on  $\mathbb{N}$  — which is exactly what liveness constraints cannot survive: for every  $p > 0$  the set  $\{\nu : \|\nu\| \geq p\}$  is weakly closed because mass is weakly continuous, and is not vaguely closed. And the choice is forced, not preferred, by the promotion itself — mechanized pointfree, at record-space generality, with opens and half-spaces first-class (module `Pointfree.Valuation.Topology`): along the promotion that adjoins the cemetery outcome the narrow topology is exactly the induced topology, its mass-monotone half-spaces the cemetery coordinate’s lower half-spaces (`induced_promote_eq`, the promotion a homeomorphism, `promoteHomeomorph`), so the stack’s polarity holds by construction — evaluations lower semicontinuous, mass upper semicontinuous, liveness walls closed (`lowerSemicontinuous_eval`, `upperSemicontinuous_mass`, `isClosed_liveness_wall`) — and the Borel reading enters afterwards, as a bridge rather than a foundation: on second-countable carriers every Borel measure is  $\tau$ -additive [83, 414O], which is what lets the measure-theoretic dialect of this section stand as a corollary of the topological one. Partiality therefore enters this universe explicitly, as the cemetery outcome of the promotion above, never implicitly as a topological limit. Every  $\nu \in \Delta_{\leq 1}(\mathcal{X})$

<sup>4</sup>A subspace of  $\mathbb{R}^n$  is Polish if and only if it is  $G_\delta$  (Alexandrov). A locally closed set is the intersection of an open and a closed set, hence  $G_\delta$  — and, being locally compact and second countable, Polish twice over; per-sector local closedness is the certificate, and the disjoint union inherits both properties with every summand clopen.

with  $\nu \neq 0$  has a unique *ray decomposition*  $\nu = \gamma\mu$  with  $\gamma = \|\nu\| \in (0, 1]$  and  $\mu = \nu/\|\nu\| \in \Delta(\mathcal{X})$ ; the zero measure is the bottom element, written  $0$  or  $\perp$ .

*The object.* A *credal body* on  $\mathcal{X}$  is a subset  $S \subseteq \Delta_{\leq 1}(\mathcal{X})$  such that

(C1)  $0 \in S$ ;

(C2)  $S$  is topologically closed;

(C3)  $S$  is *radial* (downward-scaling-closed):  $\nu \in S$  and  $\lambda \in [0, 1]$  imply  $\lambda\nu \in S$ .

(Given (C3), axiom (C1) says exactly that  $S$  is nonempty.) Convexity is deliberately not a condition: closedness is the whole of the regularity demanded, and it is exactly the body-side face of lower semicontinuity, through the correspondence below. The convex credal bodies are a distinguished subclass, catalogued with their relatives in Remark 9.20.

*Grade and radial structure.* For  $\mu \in \Delta(\mathcal{X})$  the *grade* of  $\mu$  in  $S$  is  $g_S(\mu) := \max\{\gamma \in [0, 1] : \gamma\mu \in S\}$ , the largest admitted scaling of  $\mu$ ; the maximum is attained, because  $\{\gamma : \gamma\mu \in S\}$  is a closed subset of  $[0, 1]$  containing  $0$ . By (C3) the intersection of  $S$  with the ray through  $\mu$  is exactly the closed segment  $[0, g_S(\mu)] \cdot \mu$ : a credal body is determined by its grade function.

*The radially maximal slice.* The *radially maximal slice* of  $S$  is the set of its measures of maximal admitted scale:

$$S^{\max} := \{\nu \in S : \lambda\nu \notin S \text{ for every } \lambda > 1\} = \{g_S(\mu)\mu : \mu \in \Delta(\mathcal{X}), g_S(\mu) > 0\}.$$

The zero measure is never radially maximal, so  $S^{\max} = \emptyset$  if and only if  $S = \{0\}$ , the *bottom credal body*.

*Remark 9.16* (Why the lower envelope reads the maximal slice). Axioms (C1) and (C3) are not optional conveniences: they are what makes the belief correspondence below exact, and the bottom credal body  $\{0\}$  the lawful denotation of infeasibility — the empty set is not a credal body at all, and nothing in this universe ever denotes it. But the axioms force  $0 \in S$ , so for any nonnegative gamble  $h$  the naive infimum over all of  $S$  is degenerate:  $\inf_{\nu \in S} \int h d\nu = 0$  identically, attained at  $\perp$ . The operative lower envelope therefore reads the radially maximal slice, and the upper envelope is unaffected on the gambles that matter: for  $h \geq 0$  and  $S \neq \{0\}$ ,  $\sup_{\nu \in S} \int h d\nu = \sup_{\nu \in S^{\max}} \int h d\nu$ , since every nonzero admitted measure is dominated along its ray by its slice representative. (For gambles taking negative values the supremum must be taken over all of  $S$ .) Degenerate case, with the two sets kept straight: at bottom the *body* is the nonempty set  $\{0\}$  while the *slice* is empty, so with  $\inf \emptyset = +\infty$  the lower prevision of every gamble degenerates to  $+\infty$  while the upper prevision is  $0$ . *Ex falso quodlibet* is the wrong lesson here: the bottom body is not a contradiction from which every verdict follows but a lawful denotation with a definite, one-sided verdict profile. The upper envelope reads the whole body at bottom as everywhere else, and at  $\{0\}$  it reports the zero measure's own values: every event has upper probability  $0$ , so every safety-shaped verdict — an event upper bound, a down-closed constraint — is satisfied at bottom, at the zero measure rather than vacuously; equivalently, since the axioms force  $0 \in S$  for every credal body,  $\{0\}$  is the least body and lies inside every safety face's body. The lower envelope's  $+\infty$ , by contrast, is the arithmetic of an empty index set, not a verdict: a liveness-shaped judgment carries an existential commitment — the class stratum below writes it as an escape clause,  $K \cap S(v)^c \neq \emptyset$  — that the bottom body can never witness, because the zero measure never leaves any demand's body. Reading the vacuous  $+\infty$  as satisfaction of every positive lower threshold would decide those judgments in the wrong direction. The mechanized verdict split of Remark 9.23 enforces the discipline: the slice-reading recovery theorems carry the  $\mu \neq 0$  fence (`satCredal_lower_rayC_geC`), the bottom body is the unique credal set with empty slice (`maxSlice_rayC_empty_iff`), and the ray's zero element is the universal refuter of every positive threshold read over the whole set (`not_satCredal_upper_rayC_geC`). The verdict profile of infeasibility is therefore *all safety, no liveness*: an infeasible denotation satisfies every safety constraint and witnesses no liveness demand — the sound behaviour for the denotation of infeasibility — and the presented class stratum builds the profile in structurally: every presented atom's liveness family carries the properness demand, so the infeasible system belongs to no presented atom's class (`top_notMem_modelClass`; Remark 9.31), the refusal located in the liveness conjunct alone while the safety containment holds throughout.

**Definition 9.17** (Beliefs on the probability face; the radial compile). Adopt the standing structure of Definition 9.15, together with the arithmetic conventions  $e^{-\infty} := 0$  and  $-\log 0 := +\infty$ . The *radial compile* of a belief  $f: \Delta(\mathcal{X}) \rightarrow [0, \infty]$  is

$$S(f) := \{\gamma\mu \mid \mu \in \Delta(\mathcal{X}), \gamma \in [0, e^{-f(\mu)}]\} \subseteq \Delta_{\leq 1}(\mathcal{X}),$$



and the *belief of* a credal body  $S$  is  $f_S(\mu) := -\log g_S(\mu)$ . The two directions are mutually inverse bijections between the lower semicontinuous beliefs on  $\Delta(\mathcal{X})$  and the credal bodies on  $\mathcal{X}$  — lower semicontinuity on the belief side corresponds exactly to closedness on the body side, with no further condition on either — under which the grade dictionary reads  $g_{S(f)} = e^{-f}$ , and inclusion of credal bodies corresponds anti-monotonically to the pointwise order on beliefs:  $S(f) \subseteq S(h)$  iff  $f \geq h$ . The everywhere-infinite belief compiles to the bottom credal body  $S(\infty) = \{0\}$  — never to the empty set — so infeasibility (the sum of two contradictory beliefs, say) denotes a lawful credal body and the correspondence has no excluded cases.

*Remark 9.18* (Mechanization status of the compile correspondence). Kernel-checked at the *general* carrier — subprobability Borel measures over an arbitrary topological measurable record space, under the narrow topology,  $[\text{Nonempty } \mathcal{X}]$  the one standing hypothesis (the bottom body  $\{0\}$  needs a direction to point at) — in `SuFunctor.Universe.General`: closedness of the compile is *equivalent* to lower semicontinuity of the belief, both directions (`isClosed_compile_iff`), and the image characterization closes the bijection there — every nonempty closed radial set is the compile of exactly one lower semicontinuous belief (`exists_lsc_compile_eq_iff`), the ray gauge attained ray by ray (rays are compact for *any* carrier), its belief reconstructing the body (`compile_rayPenalty`), the compile injective (`compile_injective`), the reconstruction law `rayPenalty_compile` now unconditional — an upgrade over the finite-stratum form, which consumed a citizenship hypothesis — and upper semicontinuity of the gauge (`upperSemicontinuous_rayGauge`) the grade-side face of closedness. The proof shape is the pointfree one, which is why no metrizable, no sequences, and no  $\liminf$  bookkeeping appear anywhere: mass is the narrowly continuous observation, direction is continuous wherever mass is bounded away from zero, and the grade side carries the left-limit content algebraically (`le_gradeN_of_forall_lt`) — the topological leg of the correspondence is a theorem *past* the Polish form, not a Polish-stratum concession. The finite stratum remains kernel-checked as the exact reference case (`SuFunctor.Universe.compile`, `SuFunctor.Universe.VietorisStratum`, `SuFunctor.Universe.ImageChar`), and the convex substratum carries a second, exact-rational and exponential-free mechanization (`QtslAux.Imprecise.PenaltyCredalIso`, in the multiplicative grade coordinates  $g = e^{-f}$ : exponential-convexity of the belief corresponds exactly to convexity of the body — the harmonic-mean inequality in cross-multiplied form — the ray-segment description carries attainment of the grade as data, and radially there a theorem, mixing with the zero measure being downward scaling: one place where convexity genuinely pays). The grade dictionary for the composition law — the grade of an additive composite is the *product* of the grades, through any abstract exponential — is proved in the same module. The mechanizations agree on their shared strata through the seam isomorphism of Remark 9.48.

**Example 9.19** (Conditional beliefs, independence beliefs, and what composition does not assume). Two epistemic states arise so often in modelling practice that their compiles are worth stating once, on a record space that factors,  $\mathcal{X} = A \times B$ , with  $X$  and  $Y$  the coordinate observables of a joint law  $\mu \in \Delta(\mathcal{X})$ . Neither is a probability distribution about anything: each is a condition *about* joint laws — exactly the shape of epistemic content that forces the belief reading in the first place [54].

*Conditional beliefs.* The belief “ $X$  given  $Y$  has conditional law  $k$ ” —  $k$  a probability kernel from  $B$  to  $A$  — compiles, crisply, to the set of joints whose disintegration along  $Y$  agrees with  $k$  at  $\mu_Y$ -almost every  $y$ : inconsistency 0 there,  $+\infty$  elsewhere, the body the radial closure of that set. Richardson’s weakening [209] grades the wall instead of asserting it: with a confidence weight  $\beta > 0$ , the belief is the scaled conditional relative entropy

$$f(\mu) = \beta \int_B \text{KL}(\mu(\cdot | y) \| k(\cdot | y)) d\mu_Y(y) = \beta \text{KL}(\mu \| \mu_Y \times k),$$

where  $\mu_Y \times k$  is the joint with  $Y$ -marginal  $\mu_Y$  and conditional  $k$ : each conceivable joint is charged the divergence of its *actual* conditional from the believed one, at a rate set by the confidence. Since  $\mu \mapsto \mu_Y \times k$  is affine and  $\text{KL}$  is jointly convex and lower semicontinuous, these beliefs are convex and *lsc* — they are the per-edge terms of the probabilistic-dependency-graph inconsistency measure, the family whose classical verification the Richardson chart of Remark 9.20 records — and the crisp wall is their  $\beta \rightarrow \infty$  limit. (Kernel-checked at the finite stratum, `SuFunctor.Universe.EpistemicStates`: affinity is `condTarget_mix` — proved, not assumed — joint convexity of the divergence is `klq_mix_mix_le`, lower semicontinuity `lowerSemicontinuous_condPenalty`, and the zero set is the agreement of the disintegration with  $k$  at every  $y$  of positive mass, `klq_condTarget_eq_zero_iff` — the almost-everywhere guard is load-bearing in exactly that disintegration-ratio form, while the equivalent joint-coordinate form absorbs it.)

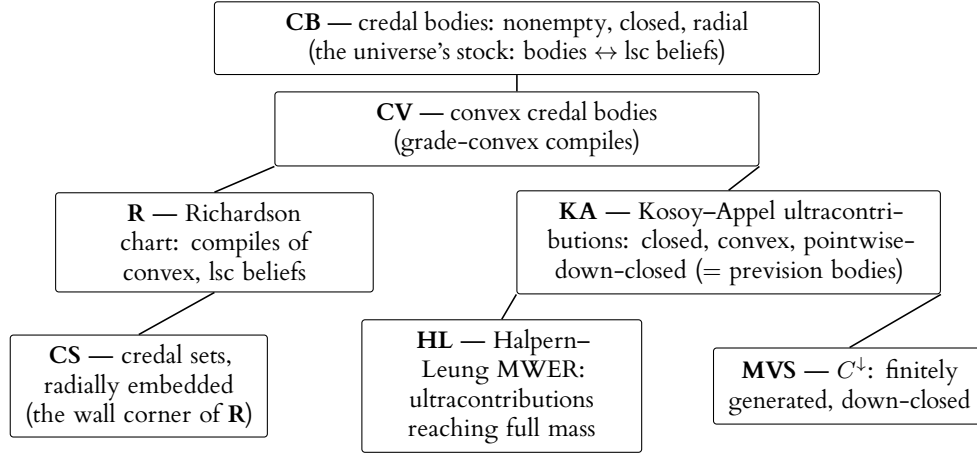


Figure 9.2: The atlas of credal-body subclasses (Remark 9.20) as a Hasse diagram, higher = larger. Every edge is a strict inclusion — placement a general-carrier theorem, strictness a two-point rational witness — and every pair not joined by a path is incomparable, both directions witnessed over the same two-point outcome space; the receipts are the modules cited in the remark. The credal-set class enters by radial embedding, and precise models are its singleton corner.

*Independence beliefs.* The belief “ $X$  and  $Y$  are independent” is, in graded form, the functional proportional to the *mutual information*,

$$f(\mu) = \beta I_\mu(X; Y) = \beta \text{KL}(\mu \parallel \mu_X \otimes \mu_Y),$$

zero exactly on the product laws, so the crisp face has body the radial closure of the product laws. The belief is lsc — KL is jointly lsc and marginal-product formation is weakly continuous; on finite record spaces  $I_\mu$  is continuous outright — but *nothing here is convex*: a mixture of two product laws is not a product law (mixing  $\delta_{(0,0)}$  with  $\delta_{(1,1)}$  in equal parts manufactures perfect correlation from two deterministic independencies), so the crisp independence body is closed, radial, and non-convex, and  $I_\mu$  is neither convex nor concave in the joint. Independence beliefs are the ordinary inhabitant of  $\mathbf{CB} \setminus \mathbf{CV}$  in the atlas below: the universe’s stock demands closedness and radially but not convexity because one of the most common beliefs in the modelling vocabulary would otherwise not denote. (Kernel-checked, same module: zero-iff-product is `mutualInfoQ_eq_zero_iff`, continuity outright `continuous_mutualInfoQ`, the two-delta witness `Witness.not_penaltyConvex_indepPenalty`, and the membership itself `indepBody_credalBody_not_convex` — stated in the atlas modules’ own credal-body vocabulary.)

*What composition does not assume.* Conjunction of beliefs is addition (the grade product, in the multiplicative chart), and a belief that reads only its own subsystem’s marginal constrains only that marginal. So when two Systems sit side by side with no wiring between them, the composite body is the set of *every* joint whose marginals lie in the two bodies — the whole Fréchet class of couplings — and not the independent coupling: absent an explicit causal dependency, the universe does not assume independent noise, because sums of marginal-reading beliefs cannot manufacture a correlation constraint. Independence is itself a belief, asserted when it is asserted — gradedly by the mutual-information belief above, crisply by its wall — and the default composite prices every coupling the marginals admit, which is the sound direction for certification: a verdict that survives the Fréchet class survives whatever the unmodelled world correlates. (Kernel-checked, same module: the composite body is the Fréchet class, `compile_marginalWallSum` with `marginalSum_eq_zero_iff`, and the correlated coupling of two uniform marginals witnesses that the composite never implies independence, `Witness.frechet_does_not_imply_independence`.)

### 9.5.3 The Atlas of Credal-Body Subclasses

*Remark 9.20* (The atlas of credal-body subclasses). The literature’s imprecise-probability objects sit inside the credal bodies as distinguished subclasses, and the inclusions between them are part of this universe’s content — Figure 9.2 draws the poset. Read through the compile, the same atlas is the chart of full subcategories of the thin category of beliefs drawn in [54], with the explicit transforms mechanized

here. The inclusion spine is kernel-checked at the *general* carrier — subprobability measures over an arbitrary record space, each placement below a theorem (`SuFunctor.Universe.General.CredalPoset`, `SuFunctor.Universe.General.Richardson`) — while the finite-rational stratum remains the reference case for the *separations*: every strict edge and every incomparability a two-line rational witness over a two-point outcome space, and the outer class containing all of them (`SuFunctor.Universe.CredalPoset.house_includes_all`). Two closure notions must be kept apart first: *radiality* ((C3): closed under scaling  $\nu \mapsto \lambda\nu$ ,  $\lambda \in [0, 1]$ ) and *down-closure* in the pointwise order on measures ( $\nu' \leq \nu$  setwise implies membership). Down-closure implies radiality; the converse fails — a single ray  $[0, 1] \cdot \mu$  is radial and, for non-degenerate  $\mu$ , not down-closed. With that distinction in hand:

- *precise models*: the rays  $[0, 1] \cdot \mu$  — closed, radial, convex; the compiles of the beliefs vanishing at a single law (Remark 9.22);
- *credal sets*, in the sense of Cozman’s credal-network literature [50]: nonempty closed convex sets  $M$  of *full-mass* laws, entering as the radial closure  $[0, 1] \cdot M$ , with  $S^{\max} = M$  — and closedness of the radial closure is proved *through the dictionary*: the radial embedding of a closed convex full-mass set is the compile of its direction wall (`scaleSegment_eq_compile_dirWall`), so the body-side regularity is Remark 9.18’s closedness theorem applied, not a separate argument. A credal set enters the universe twice over: as its *wall* — the indicator belief  $\chi_M$  of Remark 9.22, the possibilistic reading — and as its *Hausdorff-KL embedding*  $\mu \mapsto \min_{\nu \in M} \text{KL}(\mu \parallel \nu)$ , the graded reading of the same set: zero exactly on  $M$ , relative-entropy-graded off it — the set-valued extension of the Gibbs chart of the grades paragraph above, with the direction of the divergence *forced* rather than chosen: under the epistemology’s multiplicative-weight update [54] (add the expected surprisal  $E_\mu(-\log L)$ , renormalize), the wall reading can only delete states, the reversed-divergence reading collapses to the vacuous belief, while the Hausdorff-KL reading conditions each member and charges it the surprisal of its own marginal likelihood — likelihood updating of weighted sets of probabilities in the sense of Halpern and Leung [108], whose zero set is the maximum-likelihood update of Gilboa and Schmeidler [92] (Dempster conditioning, on belief-function credal sets), and whose graded part retains exactly the marginal-likelihood record that element-wise conditioning [50] forgets (kernel-checked at the finite stratum, including the failure of the element-wise square: `SuFunctor.Universe.HausdorffKL`);
- *Richardson’s universe* [209]: bodies compiled from beliefs both *convex* and *lower semicontinuous* — with the provenance stated carefully, because the class is this universe’s reading of the programme, not a definition of Richardson’s: the probabilistic-dependency-graph programme never axiomatizes a codomain class at all — it proves convexity of its inconsistency measure as a lemma and never mentions semicontinuity — so the reading takes both properties as definitional, assuming them of the programme’s beliefs. The assumption is verified where checkable: the per-edge conditional relative entropies and their nonnegative combinations (the inconsistency measure itself) are jointly convex and lower semicontinuous, classically; the full  $\gamma$ -weighted scores, which add signed conditional-entropy terms, are convex exactly in the programme’s own weight regime (edge confidence at least  $\gamma$  times the structural weight — the programme’s convexity analysis states this bound, and outside it a two-outcome double-edge graph gives an exact rational counterexample). Neither half of the reading is redundant. Convexity of the belief is sufficient, not necessary, for convexity of the body —  $e^f$  convex is the exact condition, and on a two-point record space the belief  $\log(2 - \mu\{a\})$  is strictly concave along the simplex while its exponential is affine, a kernel-checked witness with its exact rational comparison (`QtstAux.Imprecise.PenaltyCredalIso`). And the lsc half is not implied by the convex half: a bounded, two-valued convex belief’s body can fail to be topologically closed, the failure a jump at the simplex boundary — a convex function is automatically continuous on the relative interior, so boundary-lsc is exactly what convexity leaves unsecured (`exists_penaltyConvex_not_closed`), and the same witness compiles to a body that no convex lsc belief compiles, so the two-property class is strictly smaller than the convexity-only reading would have been (`exists_penaltyConvex_compile_not_richardson`). The lsc half is thereby the half that makes the class a subclass of the credal bodies at all; the convex half is what places it under the convex bodies. The placement is a general-carrier theorem, exponential-free: additive convexity of the belief implies the exact grade-convexity criterion by weighted arithmetic-geometric means (`IsRichardsonBody.isConvexCredalBody`);
- the *convex credal bodies*: exactly the compiles of beliefs with  $e^f$  convex — at the general carrier the criterion is carried division-free, as the cross-multiplied harmonic form (`GradeConvex`), with the compile of any grade-convex belief a convex body (`convexBody_compile_of_gradeConvex`) —

and the subclass on which single-shot linear tests — prevision-shaped functionals, in the sense of Goubault-Larrecq’s prevision domains [96, 98] — are a complete instrument, by Fenchel–Moreau duality;

- *homogeneous ultracontributions* (Kosoy and Appel [143]): nonempty topologically closed *down-closed* convex sets of subprobability laws — inside the convex bodies, since down-closure implies radiality (`IsUltra.isConvexCredalBody`, at the general carrier). Every prevision-shaped body — a set cut out by a family of one-sided gamble constraints, the hyperspace object of Goubault-Larrecq’s superlinear-prevision domains — is an ultracontribution at the general carrier (`IsPrevisionBody.isUltra`); on the finite stratum the inclusion is an *equality* (`isPrevisionBody_iff_isUltra`): one poset node, two presentations, infra-Bayesian induction and the domain-theoretic prevision school meeting in one theorem — with the general converse Hahn–Banach-shaped, a named successor rather than a silent assumption;
- the  $C^\downarrow$  *monad* of Mio, Sarkis, and Vignudelli [182]: nonempty *finitely generated* down-closed convex sets — inside the ultracontributions wherever the setwise order of measures is narrowly closed, a hypothesis the mechanization carries explicitly rather than silently: hulls of finite generating families are compact for *any* carrier (the weight-simplex route), so finite generation implies closedness under exactly that named order-closedness (`isClosed_ptwiseDownHull`, `IsMVS.isUltra`) — free at the finite stratum, dischargeable on metrizable carriers, and not free in general;
- the semantics of Halpern and Leung’s minimax weighted expected regret [108]: topologically closed down-closed convex sets of subprobability laws *each containing at least one full-mass law* — the ultracontributions whose maximal slice reaches the probability face, which is condition-for-condition Kosoy and Appel’s *homogeneous ultradistribution* — although neither cites the other — the subprobability encoding being Halpern and Leung’s own.

Every class above is a class of credal bodies, so the present universe includes them all; the chain  $C^\downarrow \subseteq$  ultracontributions  $\subseteq$  convex bodies  $\subseteq$  credal bodies is strict at every step already over a two-point outcome space; the credal-set and precise classes enter through radial closure rather than down-closure (their radial closures are not down-closed in general); and the regularity that closes the whole atlas at the top is closedness alone — the lsc face of the compile, with no convexity anywhere in the outer class.

#### 9.5.4 Envelopes: Previsions and the Degenerate Faces

From either presentation the operators of record are the envelope expectations, split across the slice exactly as Remark 9.16 forces.

**Definition 9.21** (Lower and upper previsions of a credal body). Adopt the standing structure of Definition 9.15. A *gamble* is a Borel function  $h: \mathcal{X} \rightarrow (-\infty, +\infty]$  bounded below; its expectation under  $\nu \in \Delta_{\leq 1}(\mathcal{X})$  is well-defined in  $(-\infty, +\infty]$ . For a credal body  $S$  with radially maximal slice  $S^{\max}$ , the *lower prevision* and *upper prevision* of  $h$  under  $S$  are

$$\underline{E}(h) := \inf_{\nu \in S^{\max}} E_\nu(h), \quad \overline{E}(h) := \sup_{\nu \in S} E_\nu(h),$$

with  $\inf \emptyset = +\infty$  (so the bottom credal body — the nonempty set  $\{0\}$ , whose maximal slice is empty — has vacuously infinite lower previsions). For events we write  $\underline{P}(A) := \underline{E}(1_A)$  and  $\overline{P}(A) := \overline{E}(1_A)$ . The lower prevision must read the maximal slice (Remark 9.16); the upper prevision is defined over all of  $S$ , and for nonnegative gambles and  $S \neq \{0\}$  the suprema over the two index sets agree (the sets themselves do not). Full boundedness is required exactly where the two envelopes are *combined* — imprecision gaps and the attitude blends below — since those expressions are undefined when an envelope is infinite.

A specification’s probability operator  $P_{\geq p}[\varphi]$  denotes the lower probability  $\underline{P}(\varphi) \geq p$  — certification pessimism, always. Three faces of one object, and the flat operator survives as a face rather than a casualty:

*Remark 9.22* (The degenerate faces, stated exactly). The two flat verdict operators survive inside the credal stack as compiles of indicator-type beliefs.

1. *Precise face.* For  $\mu_0 \in \Delta(\mathcal{X})$ , the belief vanishing at  $\mu_0$  and infinite elsewhere is lower semicontinuous, and its compile is the single ray  $S = [0, 1] \cdot \mu_0$  with  $S^{\max} = \{\mu_0\}$ . The maximal slice is the singleton: both envelope operators then read  $\mu_0$  alone, and the denotation is an ordinary (precise) probability model. It is the slice, not  $S$  itself, that is a singleton — a credal body is never a one-point set except at bottom.
2. *Possibilistic face.* Let  $B \subseteq \mathcal{X}$  be a closed set of allowed records and  $D_B := \{\mu \in \Delta(\mathcal{X}) : \mu(B) = 1\}$  the distributions supported on  $B$  — convex, weakly closed, the closed convex hull of the point masses on  $B$ . The possibilistic belief (0 on  $D_B$ ,  $\infty$  elsewhere) is convex lower semicontinuous, and on this face upper probabilities are Boolean:  $\bar{P}(A) = 1$  if  $A \cap B \neq \emptyset$  and 0 otherwise, so  $\bar{P}(\text{bad}) = 0$  is the behavioural verdict “bad is forbidden.” The zero-set is all of  $D_B$ , not merely the point masses — with convexity no longer a membership condition, the point-masses-only belief is also a lawful member, whose body is the union of the rays through the point masses; when  $B$  contains at least two points that body is nonconvex and the belief sits strictly above  $\chi_{D_B}$  in the order (for empty or singleton  $B$  the two coincide), and the face that reproduces classical possibility theory is  $D_B$ .

The classical imprecise-probability world embeds by the same device one level up: an ordinary credal set  $M \subseteq \Delta(\mathcal{X})$  (nonempty, convex, closed, all elements of full mass) corresponds to the indicator belief  $\chi_M$ , whose compile is the radial closure of  $M$  with  $S^{\max} = M$  — the atlas of Remark 9.20 places this and its relatives. Strictly graded beliefs — the soft, dependency-graph-typed models — are exactly the credal bodies whose maximal slices contain submass measures.

*Remark 9.23* (The envelope calculus: the ray and its maximal slice). A precise law  $\mu$  denotes as its ray  $[0, 1] \cdot \mu$  — the credal body, its entry into the universe — and the wiring stratum computes with the ray’s *radially maximal slice*, which for  $\mu \neq 0$  is exactly the singleton  $\{\mu\}$  (`maxSlice_rayC`). That the device is faithful to the denotation is kernel-checked, hypotheses included, in module `SuFunctor.Adequacy.Envelope`: the two verdict readings agree kind by kind — a lower-kind (slice-reading) verdict against the ray is the verdict at  $\mu$  (`satCredal_lower_rayC`), and an upper-kind (whole-set) verdict agrees between ray and slice for every *down-closed* constraint (`DownClosed`; event upper bounds qualify, with no hypothesis on  $\mu$ ). The boundary is part of the theorem, not a footnote: at  $\mu = 0$  the slice is *empty* — the bottom credal body, per Remark 9.16 — and ray and slice agree as raw constraints *only* there (`rayC_eq_singletonC_iff`): divergence is governed by  $\mu \neq 0$ , not by submass. Mixing the two readers is refuted as a theorem, not discouraged as a convention: reading a lower-kind atom with the whole-set reader refuses every positive threshold, for every  $\mu$ , the ray’s zero element being the universal refuter (`not_satCredal_upper_rayC_geC`). For the squares’ consumers the hypothesis discharges for free: a compiled stochastic frame’s law has mass one, hence is nonzero. The set-valued extension is kernel-checked in module `SuFunctor.Adequacy.EnvelopeUnion`. For a *family* of laws the set-valued presentation is the union of the members’ rays: radial and, for a closed family, itself a credal body, though in general *not convex*. Whether the closed convex hull is taken is a modelling fact, not a convention: an adversary with strategy access over the family can *mix*, and for such faces the hull is the achievable set — the correct denotation — while for a parameter fixed once and unknown the bare union is the correct denotation, and taking the hull would add mixtures the family cannot realize. The slice of the union always sits inside the family (`maxSlice_unionRays_subset`, hypothesis-free), and for a family of equal nonzero mass the slice is exactly the family (`maxSlice_unionRays`), equal mass being spent on cross-ray domination. Fixtures witness the hypothesis’s necessity: a *nonempty* all-zero family collapses its union to the bottom object with an empty slice, exactly as the single-ray boundary predicts, and the empty family yields the empty union, which sits below even bottom.

### 9.5.5 Gluing: The Imprecise Extension Problem

A System’s mechanisms hand the semantics *local* credal data — interval kernels, rate intervals, disturbance sets — on finite windows; the denotation must be a global object on records whose restrictions reproduce the local data. This is the Kolmogorov extension problem, transposed, and its imprecise answer stratifies along the record direction:

- *Existence and exactness, kernel-presented case.* When local data arrive as compact credal sets of transition kernels in the record direction — the shape discrete-time mechanism presentation produces — the glue exists with no topological hypotheses beyond that compactness, is computed by iterated marginal extension, and reproduces the local models on the nose: the Marginal Extension Theorem of de Cooman and Hermans [49] for imprecise probability trees, with backwards recursion as



its computational face. The on-the-nose verdict carries its scope wherever this book leans on it: discrete-time record posets, finite or discrete state, compact credal local sets, and Williams-coherent globals. Continuous-state and continuous-time mechanisms hand the semantics local data through their own doctrine presentations; their extension theory is named debt in [section 9.13](#), not a corollary of the kernel case.

- *Uniqueness fails — by design.* Many coherent global models share the same finite-window restrictions. The canonical choice is a universal property, not an isomorphism: the *natural extension* — the least-committal coherent glue, pointwise-smallest lower expectation at [Definition 9.24](#)’s stated scope, largest compatible set of models (convex as a theorem, not by fiat; see the definition). The slogan this makes precise carries through everything in [section 9.9](#): the model constrains the distributions; it does not select one.
- *Beyond the record direction, the flavour of coherence is a real choice.* For parallel cover shapes on infinite spaces, Walley-coherent independent products can fail to exist (Miranda and Zaffalon [121]); adopting Williams coherence repairs existence (De Bock’s independent natural extension), and this universe adopts Williams coherence as its default — a genuine doctrinal selection, recorded here, with no consequence at finite state where the flavours coincide.

**Definition 9.24** (Coherent lower prevision; natural extension). Let  $L$  be the linear space of bounded Borel gambles on  $\mathcal{X}$ . A functional  $\underline{P}: L \rightarrow \mathbb{R}$  is a *coherent lower prevision* when, for all  $h, g \in L$  and real  $\lambda \geq 0$ :

$$(W1) \quad \underline{P}(h) \geq \inf h;$$

$$(W2) \quad \underline{P}(\lambda h) = \lambda \underline{P}(h);$$

$$(W3) \quad \underline{P}(h + g) \geq \underline{P}(h) + \underline{P}(g).$$

Its conjugate upper prevision is  $\overline{P}(h) := -\underline{P}(-h)$ . An *assessment* is a pair  $(K, c)$  with  $K \subseteq L$  and  $c: K \rightarrow \mathbb{R}$  assigning claimed lower bounds — the local data a System’s mechanisms hand the semantics. Its *compatible set* is

$$M(K, c) := \{P \in \Delta(\mathcal{X}) : E_P(h) \geq c(h) \text{ for all } h \in K\},$$

and  $(K, c)$  *avoids sure loss* when  $M(K, c) \neq \emptyset$ . The *natural extension* of an assessment avoiding sure loss is  $\underline{E}_{K,c}(h) := \inf_{P \in M(K,c)} E_P(h)$ ; equivalently — exactly, for finite  $\mathcal{X}$ , by linear-programming duality — it is computed by finite positive combinations of assessment deficits. Universal property, with its scope stated in the same breath:  $\underline{E}_{K,c}$  is a coherent lower prevision dominating the assessment, and at finite  $\mathcal{X}$  it is the pointwise-smallest such functional outright — at infinite  $\mathcal{X}$  the countably additive envelope displayed here is least among the *countably-additive-envelope* lower previsions dominating the assessment, the classical unrestricted minimum being represented by finitely additive probabilities, per the agreement scope below; dually  $M(K, c)$  is the largest compatible set of models outright — the comprehension takes every compatible model, and no convexity clause appears. Its convexity is a *theorem*: each constraint  $E_P(h) \geq c(h)$  is affine in  $P$ , so the compatible set is an intersection of half-spaces with the simplex (and closed whenever the assessment’s gambles are continuous — expectation against a merely Borel gamble is not a weakly continuous functional, so closedness is a hypothesis at the standing generality, not a freebie). Least-committal lower prevision and largest compatible set are the same universal property read on the two sides of the envelope correspondence: the natural extension adds no commitment beyond what the assessments force.

*Remark 9.25* (Mechanization status of the envelope package; the partiality departures). Kernel-checked in this repository, at the finite-rational stratum (module `QtslAux.Imprecise.Credal`): the envelope face of the coherence axioms over credal sets presented as finite families — attainment of both envelopes, superadditivity of the lower and subadditivity of the upper, positive homogeneity, and monotonicity. Kernel-checked at the general carrier — an arbitrary measurable record space, countably additive models, bounded measurable gambles under the Bochner expectation (module `QtslAux.Imprecise.NaturalExtensionGeneral`): envelopes of arbitrary nonempty families of probability measures as coherent lower previsions (no convexity or closure of the family consumed); the compatible set, avoidance of sure loss, and the natural extension as the envelope over the compatible set, with the universal property at exactly the stated scope of [Definition 9.24](#) — least among the countably-additive-envelope lower previsions dominating the assessment (`natExtG_le_lowerEnvG`), the dominating-family hypothesis itself certifying avoidance of sure loss (`avoidsSureLossG_of_dominating`), so no separate feasibility hypothesis enters — together with the largest-credal-set dual, convexity of the compatible set in mixture form, and the definition’s closedness parenthetical



as a theorem: the compatible set is narrowly closed whenever the assessment's gambles are continuous. One storey up, where denotations are credal bodies rather than subsets of  $\Delta(\mathcal{X})$  (section 9.6), the same universal property is available with no convex vocabulary at all: an assessment's constraints have walls, the natural extension is the *join of those walls* in the pointwise order, and its body is the *largest compatible body* — the radial compile of the largest compatible set of laws. For assessment-shaped (gamble-bound) constraints this body is convex by the affinity theorem above, read through the compile; for structured constraints the same join is the right object and is genuinely non-convex — an independence assertion's body is a credal body and not a convex one (the kernel-checked witness of the epistemic-states module: the diagonal mixture of two deterministic independencies is perfectly correlated), which is why citizenship in this universe is lower semicontinuity and never a shape axiom. The distinction is invisible at the lower-prevision stratum by construction — expectation is affine in the model, so a body and its closed convex hull have the same lower envelope on every gamble — and taking the hull is exactly *demonic-choice* semantics: the correct reading where an adversary may mix admitted laws, and a canonical representative rather than a definitional ingredient everywhere else. The classical “largest credal set” formulation is thus recovered, not presupposed: it is the demonic representative of the body-level universal property. The point-free face of envelope coherence — the superlinear class closed under lower envelopes, over gambles in sublevel presentation with the Choquet pairing — is kernel-checked in the point-free tower (`Pointfree.Prevision`). Kernel-checked at the finite stratum, where the claims themselves are finite (finite record spaces, real scalars; module `QtSLAux.Imprecise.NaturalExtension`): the sup-of-deficit-combinations display as exact linear-programming duality, alongside the sure-loss alternative; and the finite envelope theorem — every coherent lower prevision on a finite record space is the lower envelope of its credal set, in both the attained form and the envelope display — the separating engine, one invocation powering the package, being sublinear Hahn–Banach extension. At infinite  $\mathcal{X}$  the classical envelope theorem represents coherent lower previsions by *finitely additive* probabilities; the countably additive envelope is this book's definition, and the identification of the two representations is used only inside the finite-state agreement domain — beyond it, named debt. *Partiality departures, computed*: over the maximal slice of a submass-admitting credal body, (W2) and (W3) persist verbatim, but (W1) weakens to  $\underline{E}(h) \geq \min(\inf h, 0)$  and constant-additivity fails by exactly the leaked mass — deliberate, and why the axiomatic notion is stated on the full-mass stratum, where the classical theory is recovered exactly. The leak-aware safe-side accounting is kernel-checked in this repository (module `QtSLAux.LeakAware`): the mass-split union bound, the per-element and envelope forms of the safe-side subtraction (for a partial model  $P(\text{safe}) = P(1) - P(\text{bad})$ , not  $1 - P(\text{bad})$ ), the full-mass recovery of the familiar  $1 - V(x_0)$  floor, and a total-leak counterexample certifying that the naive floor is *unsound* for submass models.

### 9.5.6 Agreement, and the Attitude Operator

**Agreement.** The scope first: the agreement theorems of this paragraph are proven for finite state spaces in discrete time; their continuous-state and continuous-time lifts are named debt, and certificate soundness (which needs only a per-element argument, section 9.12) does not wait on them. Within that scope, the global object admits three constructions that are not obviously equal: the game-theoretic upper expectation of Shafer and Vovk [218] (an infimum over hedging supermartingales), the axiomatic most-conservative extension of T'Joens, De Bock, and de Cooman [229], and the measure-theoretic envelope over all compatible precise processes. The agreement theorems state that the three coincide on monotone limits of finitary gambles and on bounded-below Borel path functionals — and the specification logic's query class sits inside that domain by construction. This book works with the game-theoretic presentation; the reason is an adapter fact, stated with the rest of the adapter material where it belongs (section 9.12): certificate rows enter the defining infimum *definitionally*.

**The attitude operator.** Certification is pessimism ( $\underline{E}$  of goodness-valued functionals; equivalently the  $\bar{E}$  of the loss). Design exploration wants the other end too, and occasionally a weighted blend — Hurwicz's optimism–pessimism criterion [119], transposed: for  $\alpha \in [0, 1]$  and a *bounded* path functional  $h$ ,  $H_\alpha(h) = \alpha \bar{E}(h) + (1 - \alpha) \underline{E}(h)$ , stated for goodness polarity so  $H_1$  is pure best-case optimism and never a guarantee quantity. The endpoints compose along records for the natural-extension denotations mechanism presentation produces (the marginal-extension tower property); interior  $\alpha$  does not, and the failure is an exact-rational fact — a two-stage interval-kernel chain realizes root-blend  $\frac{15}{32}$  against stagewise-blend  $\frac{14}{32}$ , and a second instance reverses the sign, so composed interior- $\alpha$  scores are neither sound bounds nor reliable estimates. The design consequence is a restriction the grammar adopts wherever

the operator appears:  $H_\alpha$  is a *root-only* query operator with no congruence rule, evaluated by the two passes that do compose and mixed once at the root, and any verdict at  $\alpha > 0$  renders as a design score, never as a guarantee. The structural derivation sits beside the kernel-checked counterexample: an interior blend mixes a refinement-monotone and a refinement-antitone quantity, so it is not monotone in the universe's refinement order and therefore not an entailment-shaped operator at all — the root-only restriction is the order speaking. Attitude lives in the query. Systems carry none.

## 9.6 The Hyperspace Storey

The universe of [section 9.5](#) is a space of functions with a pointwise order. Everything the specification logic will ask of the universe — connectives, entailment under a context, case analysis, and the composition calculus's fibres — lives one storey up, on *sets* of denotations, and this section fixes that storey: the hyperspace, the passage from heights to bodies, and the two facts that organize all later use — from this storey up the order of the universe is inclusion, and Systems enter as singleton elements.

**From heights to bodies.** A belief  $\sigma$  on the probability face compiles to its *body*: the closed radial subset of the subprobability carrier admitting each law  $\mu$  at every scale up to its grade,  $S(\sigma) = \{\gamma\mu : \gamma \leq e^{-\sigma(\mu)}\}$  — Definition 9.17's compile, kernel-checked at the general carrier, with the finite stratum carrying both exact presentations (Remark 9.18). Under the compile the pointwise order on the radial stock becomes inclusion of bodies, belief addition becomes the fibrewise grade product, and the height calculus of [section 9.5](#) is available, on that stock, as a calculus of closed radial sets: one structure with two interchangeable presentations, heights for arithmetic, bodies for topology. The *hyperspace* of the storey is the space of closed subsets of the subprobability carrier under the Fell topology of [section 9.4](#) — compact with no hypotheses on the carrier, and the home of every body, since bodies are closed by definition. And the storeys compose: the record storey already did its work in the Fell topology, so a window's record space arrives *compact* metrizable ([section 9.4](#)), and the subprobability carrier over it is compact Hausdorff (on a compact carrier the weak and vague readings coincide, and the carrier is weak-star closed in the unit ball — Alaoglu, standard); on the finite stratum it is the simplex, compact on sight. On a compact Hausdorff carrier Fell and Vietoris are *one* hyperspace topology — the Fell form agrees with the nonempty-compacts Vietoris presentation, kernel-checked at exactly those hypotheses (`fell_eq_vietoris`) — and the nonempty bodies are compact, so that presentation is the exact stratum's home for the class machinery below. The machinery itself no longer needs the compactness mediation: crescents, their two faces, local closedness, the constructible algebra, and the judgment transport are kernel-checked at the general carrier directly, over the Fell hyperspace of an arbitrary topological measurable record space (`SuFunctor.Universe.General.CrescentStratum` — both crescent faces ride the hit-open subbasis Fell and Vietoris share: the containment face closed because its complement is a hit of an open, the escape faces hit-opens outright), with the compact/Vietoris presentation surviving as the exact-collapse special case. For a carrier that arrives noncompact, the one-point-compactification mediator,  $CL(Y) \cong \{B \in \mathcal{K}(Y^+) : \infty \in B\}$  (Remark 9.11), still carries the Fell form into the compact presentation where the exact stratum wants it.

**The order at this storey is inclusion.** On bodies the universe's order reads  $S(f) \subseteq S(h)$  iff  $f \geq h$  (Definition 9.17), and from here up — bodies in classes, classes in classes, fibres over the wiring base — every comparison this universe supports *is* an inclusion. This is the form of the order the specification logic's judgments will consume when later chapters interpret the logic here: not a new relation at each storey, but one order — pointwise on beliefs, exchanged by the compile with inclusion at the hyperspace, read off the fibre carriers of the composition construction unchanged ([section 9.9](#)).

**Systems are singleton elements.** A System enters the storey as the singleton  $\{S(\sigma)\}$  of its body — an *ideal point* of the hyperspace. The name records a theorem, not a convention — singletons are not spec-definable (no formula of the spec syntax denotes a one-point class: `singletons_not_definable`, with its upper-open companion), so a precise refinement can never be pinned by Spec syntax and selection data is certificate content, never formula content. And the embedding leaves no residue: membership of a body in a class is inclusion of its singleton in the class — the fibre order of [section 9.9](#), at one window (`idealPoint_le_iff`) — so “Systems are singleton elements of the hyperspace” is the exact sense in which one order serves every boundary: body above body, singleton inside class, class inside class.

## 9.7 The Fibre of Specs: Constructible Classes in the Hyperspace

A Spec needs more than one body at a time. The logic conjoins Specs, disjoins them, asks whether one entails another under a context, and wires the Systems they govern into larger Systems. Each of those acts consumes classes of bodies rather than single bodies, and the classes must survive all of them at once. This section fixes the classes. The atoms of the stratum denote *crescents*: one closed containment face, cut by finitely many open escape faces. Compound Specs denote finite unions of crescents, which live in the *finitary constructible algebra* of the hyperspace. Entailment denotes inclusion of classes, and a finite blocking criterion decides it. Every demand the logic places on this structure is discharged as a theorem, and the section ends where the theorems currently end: at the algebra’s boundary, where the attitude classes of [section 9.5](#) provably live outside it, and where a completion question stands open.

What a logic may ask of its classes was settled long before this book. Categorical logic, since Lawvere, reads predicates as objects of fibres over a base of contexts and states what each connective demands [155, 156]. Substitution must act functorially along the maps of the base. Entailment must be the order of each fibre, and posets suffice for a proof-irrelevant sequent logic. Conjunction and disjunction must denote fibrewise lattice structure, preserved by reindexing, with the distributive law available to the context-carrying elimination rules. Quantifiers, where a logic has them, must be adjoints to substitution, subject to Frobenius reciprocity and the Beck–Chevalley condition. Jacobs gives the systematic dictionary between fragments of logic and the fibred structure each fragment demands [124]; Pitts surveys the same territory [199]; Melliès and Zeilberger reconstruct Lawvere’s setting bifibrationally [179]. The Spec connectives speak the *coherent* fragment: finite conjunction, finite disjunction, top and bottom; no negation, no implication, no quantifier former. Its categorical model theory is classical [169], and the match is designed rather than residual. Coherent sequents are the sequents the blocking criterion below decides with finite certificate data — domination rows, selections, choice bytes — and the same fragment, in its implicational dress, is the one the Geolog engine’s chase evaluates ([chapter 5](#)). Certifiability chose the fragment; the tradition then says what the fibre owes.

The debts are paid in order. Atoms and their classes come first, together with the polarity discipline that replaces any monotonicity intuition a reader might import. The disjunction theorem then removes the one degree of freedom an interpreter might believe remains. The blocking criterion decides entailment. Three refusals close the account: negation, refused by the carrier; quantifiers, declined at this stratum while the genuine adjoint lives one storey down; and attitudes, refused by a theorem. Beneath the atoms, the graded layer explains why graded judgments demand no graded classes, and a final subsection states the completion question the refusals leave open. On evidence, one standing note: the coherence claims of this section cite receipts from multiple proof-checking kernels (Lean 4, SafeVerify, and nanoda; [section 9.13](#) states what each checks), the receipts are re-runnable, and the claims that do not yet have them are labelled rather than blended in. This section is where that discipline proved its worth — the design-record instance after Theorem 9.28 tells the story.

### 9.7.1 Atoms: Crescents

**Definition 9.26** (Spec atom, and its crescent). A *spec atom* is a pair  $(u, \mathcal{D})$ : a *safety face*  $u$ , one member of the universe, the atom’s floor; and a finite *liveness set*  $\mathcal{D}$  of members, its escape demands. The atom’s class in the hyperspace is its *crescent*,

$$\text{cres}(u, \mathcal{D}) = \{K \mid K \subseteq S(u)\} \cap \bigcap_{v \in \mathcal{D}} \{K \mid K \cap S(v)^c \neq \emptyset\},$$

the bodies contained in the safety face’s body that escape every liveness demand’s body. The kernel form is `SuFuctor.Universe.crescentClass`.

The two kinds of face quantify in opposite directions, and the opposition is the atom’s content. The containment face demands that *every* admitted law stay within the floor’s body; it is the safety half of the atom, and the pessimistic pass of [section 9.5](#) discharges it, by domination against the worst case. Each escape face demands that *some* admitted law leave the demand’s body; it is a liveness conjunct, and the optimistic pass discharges it, by a witness at the best case. One atom thus couples a  $\forall$ -polarized face with finitely many  $\exists$ -polarized ones. This is the liveness repair of [section 9.5](#) at the right altitude, and the leak asymmetry established there governs how partiality counts on each face.

Topologically, a crescent is one Vietoris-closed containment face cut by finitely many lower-open escape faces, hence a *locally closed* subset of the hyperspace (`isLocallyClosed_crescentClass`, in the locally-closed layer at the general carrier). The shape is kindred to the hit-and-miss generators of [section 9.4](#) — one closed demand, finitely many open ones, the cover-modality geometry — though the two families are not one: the capacity generators hit basis opens and miss compact closures, while a crescent’s closed face is a containment. The kinship is what matters. The spec logic’s atoms, the carrier’s determination principle, and the checker’s test regions speak one geometry, at two floors of the same tower.

The transports between an atom’s presentation and its class are exact: a body lies in the crescent with empty liveness set precisely when its belief dominates  $u$  in the universe’s order, and inclusion between safety-only crescents reverses the pointwise order on their floors. The kernel forms are `compileNC_mem_crescentClass_iff` and `compile_subset_compile_iff`.

**Graded reach-avoid: the atoms, in control-theoretic dress.** The atoms are a familiar object of control theory wearing graded clothing. Call a spec atom with one *genuine* liveness demand — liveness set  $\{\ell, \top\}$ , the properness mark carried along structurally — a *graded reach-avoid* atom. Its judgment demands that the system dominate the safety floor  $u$  (graded avoidance: the floor prices every admitted law rather than merely forbidding a region) and dip under the mark  $\ell$  (graded reach: some admitted law strictly undercuts the demanded bar); the kernel form is `mem_modelClass_reachAvoid_iff`. With wall data on both faces — a crisp safe set  $S$  and a crisp target  $T$  — the judgment becomes literally the classical possibilistic reach-avoid demand, *exclude  $S^c$ , keep some state of  $T$  possible* (`coupledSat_crispRA_iff`), the specification studied as a decision problem since [228] and still the benchmark object of certificate-based stochastic verification [35, 244]. The generalization is strict already over a one-point carrier, and at the algebra level rather than atom by atom: no finite  $\vee/\wedge$  combination of crisp reach-avoid classes separates the constant systems 1 and 0 — every wall admits both or excludes both — while a graded floor does (`exists_genClosure_reachAvoid_not_crispGenerated`). The stratum is then *exactly* the completion of graded reach-avoid under the calculus’s two connectives: a class is constructible if and only if it is a finite disjunction of finite conjunctions of graded reach-avoid atoms, kernel-checked at both strata (`genClosure_reachAvoid_iff_exists_specDnf` on an arbitrary carrier; `genClosure_reachAvoidCres_iff_exists_specDnf` in the hyperspace, with constructibility of the generated algebra as `isConstructibleUnion_of_genClosure_reachAvoidCrescentsLsc`). The decomposition inside a single atom is conjunctive, exactly: a liveness *set* is the intersection of the singleton-liveness atoms sharing its floor (`modelClass_eq_iInter_reachAvoid`, `crescentClass_eq_iInter_reachAvoid`), and the disjunctive reading is refuted outright — one unsatisfiable demand empties an atom but not the union of its singletons (`exists_modelClass_ne_iUnion_reachAvoid`). Disjunction of liveness demands, for its part, never needs the class level at all: within one atom it is mark-join (`dip_sup_iff`), so the  $\vee$  the completion spends is spent entirely on unions of classes. Two boundary readings earn the “exactly.” The empty disjunction is generated — the false Spec, the empty class — while the empty conjunction is deliberately not: no constructible class is trivially true, because every presented atom carries the properness demand (Remark 9.31), and that same mark is what closes the completion using *binary* conjunction alone, since every liveness set is nonempty and its intersection is anchored at the atom’s own  $\top$ -mark. For a reader arriving from the reach-avoid literature the translation reads: a Spec of this logic is a finite monotone combination of reach-avoid demands whose avoidance is priced rather than binary and whose reach is a strict undercut of a graded bar — and the wall fragment ([section 9.12](#)) is the crisp sublanguage through which checked verdicts embed.

## 9.7.2 Classes: the Constructible Algebra

**Definition 9.27** (Spec class, and the constructible sets). The *spec class* of a compound Spec in disjunctive normal form — a finite family of atoms  $(u_i, \mathcal{D}_i)_{i < n}$  — is the union of its disjuncts’ crescents,  $\text{dnf}((u_i, \mathcal{D}_i)_i) = \bigcup_{i < n} \text{cres}(u_i, \mathcal{D}_i)$ . A subset of the hyperspace is *constructible*, in the finitary sense, when it is a finite union of locally closed sets (`SuFunctor.Universe.IsConstructibleUnion`). The classes of the stratum are the constructible sets.

Two boundaries in that definition are deliberate. The notion is finitary rather than spectral, and the mechanization defines it directly instead of reusing a library notion: on a connected compact Hausdorff hyperspace the spectral-space constructible algebra, generated by compact open sets, degenerates to the trivial one, while the finitary algebra is the one the spec syntax generates. And the family is closed under the operations the calculus performs: finite unions by definition, finite intersections because crescents

intersect to crescents (`crescentClass_inter_crescentClass`) and intersection distributes over union by the powerset frame law (`inter_dnfClass_append`). Top and bottom are the trivial atom's class and the empty disjunction. Systems enter the stratum as the ideal points of [section 9.6](#): the singleton class  $\{S(\sigma)\}$  of a System's body, spec-undefinable, with membership read as inclusion.

What makes this algebra *the* fibre, rather than one candidate among several, is transport. The composition calculus of [section 9.9](#) reindexes classes along wirings, and the transport dictionary makes reindexing *preimage* at the body level (`compile_pullPen`). Preimage respects both kinds of face — the pullback of a pointwise join is the join of the pullbacks, so atom-level transport commutes with atom conjunction definitionally — and distributes over the constructible unions (`transportClass_union`), which, for subobject fibres, is the coherent-category axiom that finite unions be pullback-stable. So the fibres are posets under inclusion, the one order of the universe from the hyperspace storey up; the connectives denote fibrewise lattice structure; and reindexing preserves it by variance rather than by axiom. The adjudicated boundary of the transport claim — atom-level transport against image transport, exact along surjections, genuinely divergent off them — is [Remark 9.49](#), and the double Grothendieck assembly of [section 9.9](#) packages the whole, every obligation kernel-checked, with the window-indexed families of these classes as its fibre data. The definite article is earned on two sides: this is the least family that contains the atoms and holds the connectives, and no larger family currently carries the assembly's kernel-checked functoriality. Whether a larger one can is the closing subsection's question.

### 9.7.3 Polarity, not Monotonicity

Entailment denotes inclusion, and inclusion is an order; a reader may therefore expect the classes to sit monotonically in the universe's *other* order, refinement — every class an up-set or a down-set of better-and-worse systems. They do not, and the failure is structural. A containment face is downward closed: shrink a body inside the floor and it stays inside. An escape face is upward closed: enlarge a body that meets an escape region and it still meets it. A crescent with both kinds of face genuinely present is the intersection of a down-set with finitely many up-sets, so it is neither, and unions inherit the mixture. Refining a System preserves its safety verdicts and threatens its liveness verdicts; coarsening does the opposite. Satisfaction is not monotone along refinement in either direction, and no repair should be attempted, because the mixture is what a coupled safety–liveness demand *means*.

What composes is polarity. Every literal of a presented Spec carries a face. Each face is discharged by one envelope pass alone: the floor by the pessimistic pass, each escape by the optimistic pass, and the passes compose along records separately (the marginal-extension tower property of [section 9.5](#)). What a literal returns is a verdict — one Boolean, with the face's certificate behind it — and verdicts are data. The connectives combine finished verdicts and never reach inside a face, so a compound judgment survives wiring face by face even though no monotonicity survives at all. In brief: *a Spec mixes verdicts*. The one operator in this book that mixes the passes themselves, before any verdict is taken, cannot live in this stratum; that refusal now has a theorem, and it is the subject of the attitude subsection below.

### 9.7.4 The Disjunction Theorem

Any interpretation of the logic into this stratum will read formulas as constructible classes, entailment as inclusion, and conjunction as intersection. One choice appears to remain: which class interprets  $\Phi_1 \vee \Phi_2$ . The rule catalogue's disjunction elimination, in its context-carrying core form, demands of an interpretation  $J$  of  $\Phi_1 \vee \Phi_2$  (with  $A = \llbracket \Phi_1 \rrbracket$ ,  $B = \llbracket \Phi_2 \rrbracket$ ):

$$\forall \gamma, \psi: \quad \gamma \cap A \subseteq \psi \longrightarrow \gamma \cap B \subseteq \psi \longrightarrow \gamma \cap J \subseteq \psi.$$

**Theorem 9.28** (Union is forced). *Over inclusion-ordered class fibres, on any carrier:*

1. *contextual disjunction elimination is valid for all contexts  $\gamma$  and conclusions  $\psi$  if and only if  $J \subseteq A \cup B$ ;*
2. *adding soundness of the two introduction rules ( $A, B \subseteq J$ ), the conjunction holds if and only if  $J = A \cup B$  exactly;*
3. *every strict inflation is refuted uniformly: if  $A \cup B \subset J$  strictly, there exist an explicit context and conclusion —  $\gamma$  any class meeting  $J$  off the union,  $\psi = A \cup B$  — under which both premises of the elimination rule hold and its conclusion fails.*



The proof is elementary (instantiate  $\gamma$  at the full class and  $\psi = A \cup B$  for the forcing direction; distribute intersection over union for soundness) — elementary the way a door with one key is easy to open: the fibre order leaves the interpretation of  $\vee$  *no freedom at all*. The tradition states the context-carrying elimination rule as a demand an interpretation must satisfy; over these fibres the demand has one solution, so the rule is not checked against a chosen semantics — it chooses the semantics. The kernel forms are `SuFunctor.Universe.ctxOrE_valid_iff`, which depends on no axioms whatsoever, pure constructive set logic; `orI_ctxOrE_valid_iff`; and `exists_ctxOrE_countermodel_of_ssubset`, the latter two within the development’s standard three axioms (section 9.13 names them).

Three consequences follow, in ascending order of consequence for this book.

*First, the availability problem is real, in-universe.* Locally closed classes are not closed under union, and not merely in general topology. Two spec-atom classes of the shape above — one closed atom, one escape atom, over a two-point outcome space — have a union that fails local closedness in the hyperspace, kernel-checked with an explicit convergence witness (`exists_union_crescentClass_not_isLocallyClosed`). A stratum of locally closed classes *alone* cannot hold its own disjunctions; and by clause (3) of the theorem, any attempt to stay locally closed by interpreting the join as a locally-closed *envelope* of the union manufactures its own countermodel. The constructible closure escapes both refutations, and it is not a closure operator at all: it enlarges the admitted classes until the union is native, adding no points to any join. On the constructible stratum the rule holds with equality and no side conditions. The class of the join is literally the union (`dnfClass_append`), conjunction distributes (`inter_dnfClass_append`), the conclusion is constructible (`isConstructibleUnion_dnfClass_append`), and contextual elimination is an instance of clause (1) (`ctxOrE_dnfClass`).

*Second, the design-record instance.* An earlier design of this semantics made convexity an admissibility condition of denotations. It stated no semantics clause for spec disjunction — but inside a universe of convex denotations the only candidate join of two spec denotations is their envelope, the greatest convex lower semicontinuous minorant, whose body is a hull; and a hull inflates the union whenever the disjuncts are not nested, so clause (3) manufactures the refutation. Concretely, over one probability coordinate  $x$ , the walls  $A = [0, \frac{1}{3}]$  and  $B = [\frac{2}{3}, 1]$  with context  $\gamma = \{\frac{1}{2}\}$  refute contextual elimination against the envelope reading: both premises hold vacuously, since the context meets neither wall, and the conclusion fails because the hull swallows the context. The refutation, found in this development’s design record against the reading that admissibility condition forced, is thereby an *instance of a kernel-checked schema* rather than a fact about convexity: convex hull, locally-closed envelope, and any future closure operator fail the same rule by the same mechanism, and plain union is the reading left standing. The repair cost nothing that was true — every verdict of the wall fragment transports to the class stratum through the crescent embedding unchanged. And the refuted candidate is itself mechanized at the standing generality rather than surviving as prose (`SuFunctor.Universe.HullClosure`): over complete second-countable metric carriers, binary hulls of closed sets of laws are narrowly closed with no closure bar (`isClosed_pairHull`, `isClosed_binaryHull` — the mechanism is the cone’s own, with no linear-space analogue: a fixed limit law manufactures its compactness, tightness of the one measure plus positivity of the decomposition, no cancellation between arms), and the quasiconvex coproduct is *levelwise exact*: its sublevels are the plain hulls of the sublevel unions, with none of the  $\varepsilon$ -regularization the general-locally-convex literature would demand (`sublevel_qCoproduct_eq`, `greatestness` as `qCoproduct_le_min` and `le_qCoproduct_of_forall_mixConvex`). The join the elimination rule forbids is thus refuted as a fully mechanized object, at full strength: the hull inflates the union not because its topology misbehaves — it provably does not, on this cone — but because inflation is what a hull *is*.

*Third, the two disjunctions separate, and the calculus’s  $\vee$  is the union one.* At the functional level the pointwise minimum is the lattice join of demands, and against single-functional conclusions it validates the elimination rule (the height lattice is a frame). The class stratum’s grammar is strictly richer — conclusions may be classes — and against class-valued conclusions the minimum reading breaches: the body of  $\min(\sigma_x, \sigma_y)$  for two point-mass walls contains the two-point *hedger*  $\{x, y\}$ , a body committed to the union without committing to either disjunct, so the containment class of the min strictly inflates the union of the containment classes (kernel-checked: `boxClass_union_ssubset`, with the resulting elimination countermodel at an entailment-expressible conclusion, `exists_ctxOrE_countermodel_body_reading_carrier`). The functional-level minimum survives as what it is, the join of the height lattice, sound for the single-conclusion fragment; the specification logic’s disjunction is the class union, where the richer grammar’s elimination rule demands it. What the union reading returns is worth naming: the *disjunction property* at the system boundary.



Membership of a body in a joined class is membership in one disjunct's class (`mem_dnfClass_append_iff`), so a System inside a disjunction is inside a disjunct, and a certificate of the membership carries the choice of disjunct as one byte of certificate data — restoring, at the semantic boundary, the constructive reading the functional level provably lacks.

*Remark 9.29* (The three layers of the elimination rule). The rule's status is settled at three independent layers, and the layers do not lean on one another. *Proof theory*: the context-carrying form is not derivable from the context-free core — every core rule is sound over every lattice interpretation, and the modular lattice  $M_3$  refuses the distribution the derivation would need; a kernel-checked interpretation argument, untouched by any semantics. *Chart semantics*: inside the convex chart the rule is refuted (the design-record instance above), which retrospectively explains the underderivability, since every other core rule is convex-chart-sound and a derivation would have transferred a chart-validity the rule lacks. *Class semantics*: over the constructible stratum the rule is valid with equality, and inflation-of-union is provably the only failure mode (Theorem 9.28). The catalogue's decision to carry the context-carrying form as a core rule is thereby vindicated three ways: primitive because underivable, valid because union, and necessarily primitive because its validity is what the convex chart cannot supply.

### 9.7.5 Inclusion, Negation, and Properness

Inclusion between constructible classes — the relation the logic's entailment sequents will denote — is decided by a finite *blocking criterion*, kernel-checked in the locally-closed layer (`coupled_union_refinement_iff`). A DNF-to-DNF inclusion holds iff every *selection* (one demand chosen per conclusion disjunct) is *blocked* by a premise-side domination. That is a quantifier alternation over finite index sets whose leaves are pointwise domination rows between members — certificate-shaped throughout: domination rows plus, per disjunction, the choice bytes above. The criterion is sound and complete on the stratum with no satisfiability side conditions; inconsistent premises self-block, so *ex falso* is derived rather than assumed. (At the presented stratum the properness reading below sharpens the boundary behaviour: every presented atom carries the properness demand, so the everywhere-infinite member belongs to *no* presented class at all, `top_notMem_modelClass`, rather than sitting vacuously in conclusions.) Two calibration facts bound the completeness claim: the naive rule pair that reads only per-disjunct domination without selections is incomplete, witnessed by a floor-raising inclusion, and refinement of the liveness sets is essential (`exists_naive_cover_incomplete`, `exists_essential_floor_raise`). The packaging of the criterion as a two-species sequent calculus — membership sequents and inclusion sequents with structural rules and admissible cut — is an obligation recorded in the ledger; the criterion the calculus would organize is kernel-checked.

*Remark 9.30* (Negation, and a boundary the class algebra makes easy to trip). The constructible family is closed under complement (the complement of a locally closed set is the union of an open and a closed set), so the class *algebra* is Boolean, and a future editor might read that as licence to complete the logic with a negation-former. The boundary stands, and its provenance is the carrier rather than the algebra: on submass carriers the mass identity fails —  $\mu(A)$  does not determine  $\mu(A^c)$  when mass may leak — so the complement of a wall is not the wall of a complement, event-level duals do not lift to denotation-level duals, and the typing rejections the grammar carries are theorems of this universe, not conservatism. Set-theoretic availability of complements at the class stratum is not semantic availability of a negation-former one floor down; the orientation-paired judgment kinds (upper and lower probability atoms) are row-level de Morgan bookkeeping, the sound residue of the failed duality, and they lose exactness only where double negation genuinely loses mass.

*Remark 9.31* (The properness presupposition, adjudicated). One semantic decision at this stratum was long held open: whether membership of a body in an atom's class *presupposes properness* — so that membership additionally asserts that the belief is finite somewhere, the everywhere-infinite denotation belongs to no atom's class, and *ex falso* inverts at the system boundary — or whether bare safety exists, with vacuous membership quarantined by explicit properness certificates where a rule needs them. The presentation stratum decides it, structurally, for the properness reading (*option A*): every presented atom's liveness set contains the properness demand — the mark  $\top$ , whose escape face a body meets exactly when it is proper — so an empty liveness set is unrepresentable rather than merely forbidden, the vacuous atom is  $(0, \{\top\})$  (exactly the decoration unit of [section 9.9](#)), the inconsistent system belongs to no presented atom's class, and *ex falso* inverts at the system boundary by construction. The inversion is located, not blanket. An atom's

safety containment holds at the bottom body for every floor (the axioms force  $0 \in S(u)$ , so  $\{0\} \subseteq S(u)$ ), and what refuses is exactly the escape clause — the existential conjunct every liveness demand carries, which the zero measure, a member of every demand’s body, can never witness. An infeasible system thus satisfies every safety face and escapes no liveness demand (the envelope stratum’s verdict profile, Remark 9.16); its exclusion from every presented class is the conjunction of those two one-sided verdicts — properness being the liveness demand every presented atom is guaranteed to carry — not a satisfaction collapse in either direction. The decision cost no existing result. The blocking criterion, the disjunction theorem, and the class algebra were stable under both readings — under the properness reading each liveness slot simply carries one more demand, which is literally what the presentation’s atoms now do — and the general hyperspace machinery underneath (crescents at arbitrary liveness sets) is untouched, which is why every verdict transported unchanged.

### 9.7.6 The Declined Quantifier

One demand of the dictionary this stratum deliberately declines: there is no quantifier former on the classes. Binding lives elsewhere in the universe. Records carry the variables (section 9.3); the certificate stratum’s shape-indexed families live on the functorial plates of Remark 9.42; and the substitution that quantifiers would be adjoint to belongs to the base — marginalization of joints along wire maps (Definition 9.41).

The existential the tradition asks for does exist, kernel-checked, one storey below the classes. Hiding a wire is the lower-semicontinuous envelope of inf-projection along the visible marginal, and against every citizen of the universe it is adjoint to belief pullback on the nose:  $\varphi \leq \text{hide}_p \sigma$  iff  $p \circ \varphi \leq \sigma$  (`le_infProj_iff`, `le_hide_iff`; Remark 9.43). Read in the entailment order, which opposes the pointwise order (section 9.6), this is Lawvere’s  $\exists_p \dashv p^*$ .

Frobenius reciprocity holds in the lax direction the formalities supply. For a citizen  $\psi$  of the visible carrier,  $\text{hide}_p \sigma \vee \psi \leq \text{hide}_p(\sigma \vee p^* \psi)$ , and likewise with  $+$  in place of  $\vee$  — either conjunction the storey speaks (`hide_sup_le`, `hide_add_le`). The comparison is strict: a kernel-checked witness on the escaping-mass family exhibits citizen beliefs for which conjoining-then-hiding and hiding-then-conjoining genuinely differ (`exists_lowerSemicontinuous_hide_add_ne`). The mechanism is the envelope step of the repaired existential. The envelope of a wall is the wall of the closure (`lscEnv_charFn`), so hiding adds the boundary values the visible image approaches but never attains, and a belief vanishing exactly at an added point detects the addition. The gap is visible at judgment resolution — a constant specification is dominated by one side and not the other (`exists_judgment_separating_hide_add`) — unlike the envelope itself, which changes no judgment. Under the inf-compactness certificate of Remark 9.43 the certificate survives conjunction with a pulled-back citizen, the reflector idles, and Frobenius reciprocity holds with equality (`hide_sup_comp`, `hide_add_comp`). Where marginalization is exact, quantification commutes with contexts on the nose; off the certificate, the fibred existential is genuinely lax — a fact the logic observes, not a packaging debt.

What is not claimed is the class-stratum packaging of that adjoint: a fibred existential on constructible classes, with Frobenius reciprocity over contexts and Beck–Chevalley squares over the site. At a quantifier-free stratum the Beck–Chevalley condition, governing as it does the commutation of quantifiers with substitution, has nothing to govern. The mechanized seam facts already mark where such a claim would have to be earned: escape witnesses die off the range of a non-surjective transport, and image transport leaves the constructible stratum (the saturation residue of Remark 9.49) — and images are what existentials are made of. The ledger records the obligation under its own name, the image-preimage adjunction on class fibres (section 9.13).

### 9.7.7 Attitude Classes: Outside, by Theorem

The refusals so far were designed; this one is proved. Recall the attitude operator of section 9.5: Hurwicz’s optimism–pessimism blend [119],  $H_\alpha(h) = \alpha \overline{E}(h) + (1 - \alpha) \underline{E}(h)$ , admitted into the grammar as a root-only query with no congruence rule, because interior blends fail the tower property and are not monotone in the refinement order (the exact-rational counterexample and the order-theoretic derivation both live in that section, and the specification language records the restriction, §14.10.18). That settled the operator’s grammar. It left a class-stratum question open. The *attitude class*  $\{K \mid \vartheta \leq H_\alpha(h)(K)\}$  is a definite subset of the hyperspace like any other, and nothing yet stops a reader from asking whether some Spec — perhaps an ingenious compound one — denotes it. The answer is no, and the reason is a

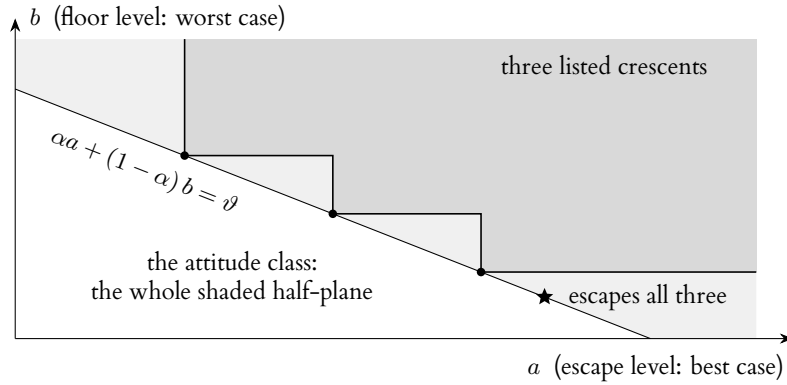


Figure 9.3: Why no finite disjunction reaches a mixed attitude, drawn in the plane of the two envelope values  $(\bar{E}(h), \underline{E}(h))$ . Each admissible chord  $(a, b)$  contributes one crescent, which covers the closed quadrant above and to the right of its corner; chords are demands, so their corners need not be attainable pairs. The attitude class covers the half-plane above the chord line. Finitely many quadrants form a staircase, and the starred model sits on the line beyond every listed escape face: it meets the Hurwicz threshold and escapes every listed crescent (`finite_chords_strict`).

decomposition worth having in its own right.

Fix the exact stratum (finitely presented credal sets over finitely supported laws — the stratum where certificates live), a bounded claim  $h$ , and a threshold  $\vartheta$ . Call a pair  $(a, b)$  an *admissible chord* when  $\vartheta \leq \alpha a + (1 - \alpha)b$ . Each chord names one atom-shaped demand: an escape face at  $a$  — some admitted law prices  $h$  at  $a$  or better, the optimistic pass — and a floor face at  $b$  — every admitted law prices  $h$  at  $b$  or better, the pessimistic pass. The Hurwicz threshold holds precisely when some admissible chord's demand holds, because the attained pair  $(\bar{E}(h), \underline{E}(h))$  is itself an admissible chord (`hurwicz_ge_iff_exists_chord`); at the level of classes, the attitude class equals the union, over the admissible chords, of their crescents (`hurwiczSuperlevel_eq_iUnion_crescents`). An attitude class is therefore a disjunction of Specs — an *infinitary* one, indexed by a half-plane of chords.

Infinitary, and irreducibly so. For a genuinely mixed attitude,  $0 < \alpha < 1$ , no finite subfamily of chords yields the class: against any finite list of admissible chords, a two-point model sits on the Hurwicz line beyond every listed escape face, meeting the threshold while escaping every listed crescent (`finite_chords_strict`). Figure 9.3 shows the geometry. In the plane of the two envelope values, one chord's crescent covers a closed quadrant; the attitude class covers the half-plane above a line of strictly negative slope; and finitely many quadrants make a staircase, which never covers a sloped line. Since every Spec denotes a finite union of crescents, no Spec denotes an attitude class. The grammar's root-only restriction thus reappears at the class stratum as a theorem about denotations: the operator does not merely lack inference rules; the classes it carves lie outside the algebra every Spec inhabits.

The boundary has a one-line grammar: *a Spec mixes verdicts; an attitude mixes envelopes*. Each literal of a Spec consults one envelope pass and returns a Boolean, and the connectives combine finished Booleans, which is why compound judgments transport face by face.  $H_\alpha$  adds the two passes into one scalar before any verdict is taken, and the sum behaves as neither pass: it has no single polarity, so no one face discharges it; it is not monotone along refinement, so it is not an entailment-shaped operator; and it is not stable under composition, because the chord attained at the root re-chooses when kernels chain — which is what the tower failure of section 9.5 says, read through the decomposition.

What survives for certification is the chord itself. A floor certificate at  $b$ , an escape certificate at  $a$ , and the arithmetic check  $\vartheta \leq \alpha a + (1 - \alpha)b$  together discharge the threshold (`le_hurwicz_of_chord`); a certificate pair is a choice of disjunct. A finitely presented certificate layer for attitude thresholds therefore carries its chord as data and never enumerates the class — one more instance of a pattern this universe keeps producing: where a class refuses finite presentation, its certificates do not (compare the disjunction property's choice byte above, and the properness mark).

### 9.7.8 Beneath the Atoms: the Graded Layer

A crescent's faces test bodies, and a body is a belief's shadow. The compile of [section 9.6](#) sends each belief  $\sigma$  to the closed radial set of laws it admits at each grade. A *wall* — a  $\{0, \infty\}$ -valued belief — casts a crisp shadow, the set of laws it admits outright; a graded  $\sigma$  knows strictly more, namely how expensively each law is admitted. The atoms of this section consult shadows. So a fair question about the whole construction is whether graded beliefs eventually demand graded classes — some second fibre, richer than the constructible one, where soft judgments would live. They do not, and the exact stratum now carries the reason as a pair of theorems.

At the exact stratum a convex prevision arrives finitely presented: finitely many laws  $M_i$  with signed rational costs  $c_i$ , pricing a claim  $f$  at  $\max_i (E_{M_i}[f] - c_i)$  — the penalty-function representation (so named in that literature) of convex risk measures [79], restricted to finite presentations (penUpperE, with penLowerE its conjugate). The wall tier is the zero-cost face, definitionally (penUpperE\_zero\_cost), and the precise tier is the one-member face. The first theorem says the grades never leave the reach of the classes: the scalar is the finite Legendre transform of its own threshold profile — the map sending each affordability threshold  $t$  to the best expectation among members costing at most  $t$  — and the profile's data at each threshold is an ordinary credal set (penUpperE\_eq\_sup\_levelProfile). A graded judgment therefore unfolds into a family of possibilistic judgments, one crescent face per threshold, and the scalar folds back by the transform. The second theorem says composition respects the unfolding: belief kernels compose by an exact intertwining square whose cost law — costs integrate along the record — is forced by countermodels on either side (penUpperE\_compFam; the unweighted and the worst-case aggregations each break the square on a two-state witness).<sup>5</sup> These are theorems of the exact stratum — finitely supported laws, rational costs, where certificates live; the general-carrier belief calculus of [section 9.5](#) stands beneath them unchanged. The atoms stay possibilistic; the grades thread through them one threshold at a time; and the classes of this section carry the soft tier without gaining a single new former.

### 9.7.9 The Completion Question

Two pressures point at one enlargement. The attitude classes ask for unions of crescents over a compact index; the graded layer, read at every threshold at once, asks for level families of the same shape. Both land in the  $\sigma$ -completion of the constructible algebra: classes presented as directed unions of crescents over declared index families, with the index carried as data. Nothing wild lives there. Each candidate class is assembled from the same atoms as every Spec, and its certificates are already finite objects — a chord, a threshold, a pair of faces.

What is not settled is transport, and the chord decomposition says why the question has teeth. The constructible fibre's functoriality is a theorem because preimage acts on finitely many faces at once. An enlarged fibre would transport index families, and the transport is pointwise in the index: under composition the attained chord re-chooses, which is the tower failure again, now as a design constraint rather than a pathology. A completion that demanded one uniform chord would refute itself at the first kernel chain. The candidate discipline, stated here as design and not as artefact: admit closed classes that declare semicontinuity certificates for both envelope passes, and transport them only along wirings carrying both certificates — the two-pass evaluation of [section 9.5](#) made compositional. Whether the resulting fibre keeps the composition calculus's kernel-checked functoriality, and whether it is the largest fibre that does, are open questions; the closing ledger names them ([section 9.13](#)). Until they close, this book's judgments live on the constructible stratum, where every demand of this section is discharged by a theorem.

## 9.8 The Fibre Sheaf: Spec Classes over the Site

The class stratum of [section 9.7](#) was built at one window. Records live over a site, so spec classes do too, and this section fixes how: the classes of [section 9.7](#), window by window, assemble into a presheaf whose

<sup>5</sup>The tier's calculus holds a few further facts a practitioner will want. Convexity replaces coherence: a mixture of claims is overpriced by the mixture of prices (penUpperE\_mix\_le), while positive homogeneity genuinely fails off the calibrated face, by a one-point witness (penUpperE\_not\_posHomogeneous) — the tier is convex-prevision, strictly wider than Walley coherence. And the point-mass family with a pointwise gate prices claims as a max-plus contraction (penUpperE\_dirac\_family), the idempotent-integration reading that connects this layer to tropical dynamic programming.

sheaf-theoretic behaviour is governed by the carrier results of [section 9.4](#) — with one structural surprise, itself kernel-checked, about *which* window-indexed families are admissible.

**Window beliefs, and the variance of extension.** Over a window  $W$  and an instantaneous carrier  $X$ , the trajectory space is the record carrier of [section 9.4](#), taken at state fibre  $X$ :

$$\text{Traj}(W, X) := \text{CL}(\mathring{S}(W) \times X^+)_{\text{Fell}},$$

closed subsets of the stage-interior-times-state ambient under the Fell topology, with  $X^+$  the carrier with its cemetery adjoined —  $\mathbb{H}(W)$  itself when  $X$  is the window’s own state fibre. Functions  $W \rightarrow X$  embed by *graph-then-closure*,  $f \mapsto \text{cl} \{(t, f(t))\}$ : on the speed-bounded class the graph is already closed and the closure does nothing — the graph reading of [section 9.3](#), recovered as the special case — while at a jump the graph is *not* closed, and the closure charges the jump instant with both one-sided values: the closed-graph doctrine again, a jump multivalued at the moment it happens. The embedding is accordingly not injective on jump functions — two functions differing only in which one-sided limit they assign at the jump instant close to the same set — and that quotient is intended, not a defect: the closed set remembers every value the instant is honestly charged with and forgets the convention that chose among them. Window beliefs are beliefs over this carrier: functionals carried, in the working presentation of [section 9.5](#), on subprobability joints — probability joints over the cemetery-completed carriers — at the standing generality of [Definition 9.15](#); these are laws of exactly the kind the capacity presentation of [section 9.4](#) pins by countably many rational observations. The scalar stock of that generality is kernel-checked as stated — subprobability Borel joints over arbitrary measurable carriers, with pushforward, belief transport, weak continuity, and the lower-semicontinuity closure and attainment package (`SuFunctor.Plain.MeasJoint`; the composition tier below carries the details) — with the finitely supported joints of the exact stratum its kernel-checked sub-instance along the seam (`toMeas_push`); nothing in the stock reads how the carrier is presented, so it serves the Fell carrier and the function presentation alike. The window *assembly* is kernel-checked at both joint strata over the *function presentation* (windows as subsets of a stage type, trajectory carriers the graphs before closure): the extension calculus and the persistence discipline of this section replay verbatim over subprobability Borel joints, with measurability as the morphism datum — the presented carriers get the product  $\sigma$ -algebra, under which clipping and pointwise state transport are measurable outright, and the window-compositor exchange is one application of the base-free naturality core with both comparison squares discharged definitionally (`SuFunctor.Site.GeneralWindow`: `penExtM` with its presheaf laws, both exchange theorems, and the closure package for persistent families) — and the exact assembly is the general one’s kernel-checked sub-instance along the seam (`penExtM_comp_toMeas`, `IsPersistentM.seamRestrict`). The same assembly restated over the Fell carrier itself — Borel-on-Fell as the measurability datum, the carrier section’s clip as the restriction — is an open obligation of the closing ledger, not a receipt in hand; the restriction leg is not part of the debt, its measurability being a corollary of the landed Fell-continuity of clip-to-open ([Remark 9.11](#)). The descent results below were never joint-stratum-bound at all: their ambient is an arbitrary topological space (the class-descent module is fibre-class-blind), so no stratum scoping applies to them. Two boundaries are real and worth naming. The double-categorical *total* of the site-indexed decoration functor over a wiring base stays at the exact stratum — the finite state-family presentation discharges its comparison squares definitionally, and a measurable wiring base owes its own transport data first, the same residue one tier up as in [Remark 9.45](#). And the matching-fork witness below is stated at the exact stratum because nothing more is needed there: its trajectory carriers are finite, and over finite carriers every presentation collapses into every other — a finite Hausdorff ambient is discrete, so every graph is closed, graph-then-closure is the plain graph embedding and is injective, and the two joint strata literally coincide, the seam embedding being a bijection (`toMeas_surjective` with `toMeas_injective`) — so the fork refutes the strict discipline at the general stratum, and on the Fell carrier, in the same breath. Which *evaluation-level* claims complete across the seam by density — and which provably do not — is [Remark 9.33](#). For nested windows  $V \subseteq W$ , trajectories *clip* ( $\text{Traj}(W, X) \rightarrow \text{Traj}(V, X)$ ): intersection with the subwindow’s open band — the carrier section’s Fell-continuous restriction, at fixed state fibre the pure clip of  $\rho_u$ ), and the clip agrees with the function presentation through the embedding: clipping a closed graph over an open sub-band is the closure of the restricted function’s graph, because closure-then-intersect is relative closure over an *open* set — one more place the interior coverage earns its keep. Beliefs therefore *extend*: a subwindow demand  $u$  is read over the superwindow as “evaluate  $u$  on the clip” (`penExt` := precomposition of the joint pushforward along clipping; `SuFunctor.Site.penExt`, on the presented carriers). The variance is forced: a belief reads whole-window joints, and only the small-to-big



direction has a canonical trajectory map — a closed set clips canonically and extends non-canonically, so there is no restriction of beliefs. Their *bodies* restrict — that is the compile trade: heights extend along the site, bodies clip along the site, and the two motions are exchanged by the compile isomorphism. Both motions are used below, each on its own side.

*Remark 9.32* (Relations first; functions as a dense locus). The carrier just fixed takes closed *relations* as its primary objects, and the function layer enters as a distinguished locus rather than as the ambient notion — in three precise senses. The locus is *dense*: wherever the stage interior has no isolated points (every nondegenerate classical window), each nonempty basic hit-and-miss class contains the graph of a finitely supported single-valued function on a sub-timeline — one point in each hit region, off the miss compact, shared stages separated within the open regions — so every closed relation is a Fell limit of function graphs. Its calculus is a *restriction*: the window assembly above, kernel-checked today on the function presentation, is designed as the relational assembly read along graph-then-closure, and the relational form itself is the closing ledger’s Fell-carrier obligation, not a receipt in hand. And the correspondence-theoretic apparatus arrives with no added hypotheses: for a closed-valued map into the compact state fibre, lower hemicontinuity is continuity for the hit face and upper hemicontinuity is continuity for the miss face — compactness of the fibre is what turns a closed complement into a miss parameter — so hemicontinuity is the slice-wise reading of the topology the carrier already wears, not a regularity assumption on a new kind of object.

*Remark 9.33* (Density completion: what the exact stratum settles). The exact stratum invites a completion argument: finitely supported joints are narrowly dense in the subprobability carrier, so should not every claim kernel-checked at the exact stratum extend to the general carrier by taking limits? The answer has a shape, and both halves of it are kernel-checked (`SuFunctor.Plain.SeamDensity`).

First, vocabulary: “exact” names finite *support*, not rational *data*. The exact stratum’s weights are arbitrary reals in  $[0, 1]$ ; the data a certificate checker consumes — rational weights, admitting exact arithmetic — is a further sub-stratum. Its density step is kernel-checked: rounding weights down to multiples of  $2^{-n}$  converges narrowly, mass only dropping, so wherever the seam is dense, the dyadic-rational seam is already dense (`Joint.dense_ratSeam`). Density of the seam itself is the classical statement of [192, Thm. II.6.3], and is kernel-checked here over separable pseudometrizable carriers (`Joint.denseRange_toMeas`, `Joint.denseRange_toMeas_of_pseudoMetriizable` — covering the chapter’s Polish standing generality), with the truncation instance over  $\mathbb{N}$  proved by an independent second route (`Joint.denseRange_toMeas_nat`); the transport package is stated against an explicit density hypothesis, so each transport theorem applies at exactly the carriers where density is on record.

Second, the completion itself is one-sided, and the direction is forced by the universe’s own membership hypothesis. Lower semicontinuity makes sublevel conditions closed, and closed sets absorb dense subsets; so *upper* bounds transport — a lower semicontinuous belief bounded on the seam by an upper semicontinuous one is so bounded everywhere (`MeasPen.le_of_seam_le`) — suprema are seam-determined (`MeasPen.iSup_seam_eq`), and narrowly continuous beliefs agreeing on the seam agree everywhere (`MeasPen.eq_of_seam_eq`). *Lower* bounds and infima do not transport, and the failure is semantic rather than a mechanization debt, witnessed at a carrier where density holds: the drop wall — zero at the geometric joint over  $\mathbb{N}$ , infinite elsewhere — is lower semicontinuous (its one proper sublevel set is a closed singleton), sits at infinity on the entire seam, and has infimum zero on the carrier (`dense_seam_inf_gap`). A wall may open exactly at an infinitely supported joint, and no net of finitely supported joints sees it open. So a verdict computed on the exact stratum is exact for suprema and upper bounds and can be unboundedly optimistic for infima and lower bounds; the checker-facing consequence is that safety ceilings certified on rational data complete to the full carrier, while worst-case floors need either upper semicontinuity or the general carrier itself.

What remains constitutively exact-stratum is computation, not semantics: exact rational comparison and decidability are finite-data notions, which is why the checker face lives there — and at finite outcome carriers even the carriers coincide, the seam embedding being a bijection (`toMeas_surjective`).

**The presheaf of classes, and the fork.** The canonical window-restriction of a *class* is the preimage under extension: a spec class  $\Phi_W$  over  $W$  restricts to the subwindow demands it certifies,  $\Phi_V = \text{penExt}^{-1} \Phi_W$  — the presheaf of spec classes over the site. A window-indexed family can then satisfy two disciplines.



**Definition 9.34** (Matching and persistent families). Let  $\Phi = (\Phi_i)_i$  be a window-indexed family of spec classes over a window system  $U$ , with extension maps  $\text{penExt}_{ij}$  for  $U_i \subseteq U_j$ .

- *Matching families* (strict sections):  $\Phi_i = \text{penExt}_{ij}^{-1} \Phi_j$  for every  $U_i \subseteq U_j$  (`SuFunctor.Site.IsMatching`)
- *Persistent families* (lax sections): extension carries each window's members into every superwindow's class,  $\text{penExt}_{ij}[\Phi_i] \subseteq \Phi_j$  (`SuFunctor.Site.IsPersistent`)

The fork between the two disciplines is settled by a theorem, and it is the one place the site dimension exerts force on the composition calculus:

**Theorem 9.35** (The matching fork). *Matching families are not closed under the window-wise composition of decorations: there exist matching families  $\Phi, \Psi$  over a two-window system whose componentwise composite (the window compositor of Definition 9.44, applied per window) is not matching. Persistent families are closed under every operation of the decoration calculus — tight transport, cell transport, composition, and the unit.*

The witness is small enough to describe: over a stage line with one extra time point, take the two *support walls* through the shared middle boundary — one demanding the extra point read true, the other false. Each wall generates a matching family; their composite at the superwindow is the zero-support wall (the demands contradict through the shared middle), which is clip-invariant — so the composite's subwindow component, being empty, fails the matching equation against it. Composition manufactures window-local content (here: infeasibility) that strict sections cannot carry (kernel-checked, with the witness exactly as described: `SuFunctor.Site.MatchingFork.matching_not_closed_under_compositor`, against the closure theorem `IsPersistent.image2_boxplusW`). The fibre objects of the construction are therefore the *persistent families* — the lax sections, equivalently the families closed under reading-small-demands-large, the least-committal completion of window-local spec data. The pattern recurs across this universe's gluing theory: record carriers glue exactly (the carrier sheaf, at its stated hypotheses), assessments complete least-committally (the natural extension, adjoint to restriction), and here admissibility itself completes least-committally — in each case the naive expectation locates laxity in the composition operation, and in each case the operations stay exact while the completion happens one construction over. (The gluing clause of that pattern carries the carrier proposition's own hypotheses — directed interior covers, fixed fibre — and the adjunction runs with the natural extension on the left: least among the coherent globals whose restrictions dominate the assessment, at the universal property's stated scope.)

**Class descent, and ideal points as global members.** Over a directed open cover, the descent homeomorphism of Theorem 9.9 induces an order-isomorphism of class fibres,

$$\text{Set}(\mathbb{H}(W)) \cong \text{Set}\left(\varprojlim_i \mathbb{H}(W_i)\right),$$

carrying constructible classes to constructible classes in both directions — the site's fibre stratum and the glued fibre stratum have the same constructible layer (kernel-checked: `SuFunctor.Site.classDescent`, `isConstructibleUnion_classDescent_iff`, with the transport lemmas `IsLocallyClosed.classExt` and `IsConstructibleUnion.classExt` for the free contravariant direction). And the ideal points of the fibre sheaf are exactly the global members: compatible singleton-sections of the point presheaf — per window the clip of one body, compatible by the clip calculus — biject with closed members of the glued ambient (`SuFunctor.Site.bijective_toProjLim`, the descent homeomorphism's underlying bijection consumed whole). A System over the glued window is its compatible family of window bodies — on the nonempty-body stratum the compiled-System scope of section 9.5 draws — and membership in a persistent family is window-local along pullback (`clip_mem_classExt_iff`).

*Remark 9.36* (The full fibre column, and the localic reading). Assembled, the fibre over a window is a four-storey tower, each storey kernel-checked at the strata named in its own section: the *record locus* (closed subsheaf of the Fell carrier at the classical stratum, section 9.4); *laws* over it (the subprobability stratum of section 9.5, pinned by the countable rational determination of Theorem 9.12); the *hyperspace* of denotation bodies; and the *constructible classes* of section 9.7 — crescents in cover-modality shape, closed under the connectives, ordered by inclusion, with membership and inclusion decided by the blocking criterion. The design-normative reading of every storey above the  $\omega$ -semialgebraic state spaces is pointfree — the record storey as the Vietoris powerlocale of the compactified ambient object (the carrier section's own localic reading), the law storey as the valuation locale of the record locale, the hyperspace storey as its Vietoris

powerlocale, each presented by generators and relations (the cover-modality presentation of [233]) so that the whole column externalizes objectwise over the site by geometricity.

That reading is no longer unmechanized design. The point-free library carries a kernel-checked tier of its own, stated on frames — no points, no countability, no  $\sigma$ -algebra. Checked there: the *presented hyperspace frame* — the free frame on hit-and-miss data, quotiented by the least nucleus forcing the transposed hit-and-miss laws — with the Choquet–Matheron determination on its  $\nabla$ -generators, so that the classical theory is recognized as having been pointfree all along, now as a kernel theorem at the localic stratum (Remark 9.13); the *valuation layer* of the law storey, frame-general — valuations with modularity load-bearing, pushforward functorial and the Dirac unit natural (the presheaf structure the storey needs), the dagger promotion chart driven by modularity alone, and the narrow topology *forced*, not chosen: it is the lower topology of the promoted carrier pulled back along the chart (Pointfree.Valuation.Basic, .Promotion, .Topology, .Functor); and the *two-tier tie* — restriction of Borel measures to opens is Scott-continuous exactly on the  $\tau$ -additive stratum (Fremlin’s 414O supplies that stratum under second countability; the Dieudonné measure fences it), continuous from the weak into the narrow topology by the portmanteau open half, natural in pushforward, with the measure-level dagger chart a homeomorphism carrying Prokhorov compactness to the subprobability carrier, the escape phenomenon witnessed without points (a free-ultrafilter valuation refusing Scott continuity against a directed family of opens), and  $\tau$ -additivity re-read as order-continuity into the measure algebra (.Bridge, .BridgeContinuity, .BridgeFunctor, .EscapeWitness, Pointfree.Measure.DaggerChart, Pointfree.MeasureAlgebra.Basic).

What remains argued design is the column reading itself: the identification of each presented frame with the powerlocale of the storey below it (equivalently, that spatialization returns the classical Fell hyperspace — the determination theorem is proven at arbitrary relation sets exactly so that nothing kernel-checked gates on this identification); the geometricity by which the column externalizes over the site; and the prevision-side completion — every Scott-continuous superlinear prevision the lower envelope of its body of subprobability valuations, the double-powerspace reading of [97] — of which the kernel now holds the sublevel-presented gambles, the Choquet pairing, closure of the superlinear class under lower envelopes, and the cheap half (every prevision below the envelope of its body; Pointfree.Prevision.Basic), the representation converse cited.

The boundary, stated exactly: part of the mechanized tier now *is* the localic reading — the determination theorem consumes the presented frame, and the valuation layer is stated on arbitrary frames outright — while the spine this chapter’s sections consume stays deliberately point-set, exact on compact windows, where continuous valuations and Radon measures agree, and the two tiers exchange theorems through the kernel-checked bridge rather than by analogy. Everything classical is stated so as not to obstruct the localic reading; the localic reading now answers with theorems of its own.

## 9.9 Composition: the Double Grothendieck Construction

Chapter 13 catalogues the composition patterns a modeller wields — fusion, pools, dispatch, delayed trace. This section is the stratum underneath: the single categorical construction in which any wiring of open Systems composes, lawfully, so that pattern-level soundness claims have one place to discharge. The route is direct: span-valued lax double functors on the interfaces-and-wirings base, consumed by the double Grothendieck construction at its fully mechanized stratum, with every obligation — the functors’, the totals’, and the projections’ — kernel-checked. Two functors share the base and the machinery, in a division of labour stated where each is defined: the *Spec* functor carries arbitrary classes and the site dimension (the window-indexed persistent families of section 9.8 — the fibre of Specs); the *coupled-DNF* functor carries the constructible presentations of section 9.7 as fibre data, refines the Spec functor along a kernel-checked comparison cell, and its total is the development’s authoritative SingleSiteSemanticUniverse — the double category of System Specs, whose site-indexed culmination, the chapter’s headline object, is stated whole in section 9.10. Two boundaries of that opening are doing work. The operation is total on composable pairs; the conditions that live deliberately *outside* the functor — causality, simultaneity discipline, Zeno exclusion — are side-condition oracles recomputed on each composite, because they could not be object-level: two individually causal couplings can close an algebraic loop, so composition does not preserve them. And the fibres are *Specs*, not System entities: a System enters the total through its denotation, as an ideal point of a fibre, and the entity-level stratum (simulation witnesses as fibre morphisms, refinement as a fibred structure over this one) is a design layer above the total, recorded in the ledger, not part of it. Figure 9.4

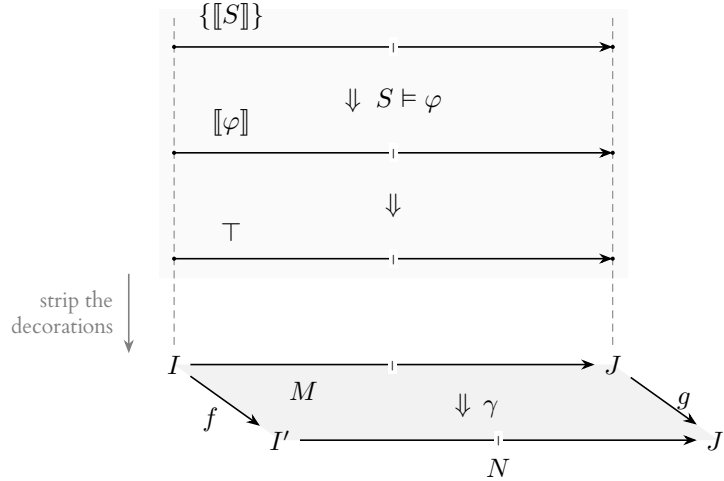


Figure 9.4: The two storeys of the composition construction. On the floor lies the wiring base: interfaces  $I, J$ , wire maps  $f, g$  (tight), wiring patterns  $M, N$  (loose, slashed), and a pattern cell  $\gamma$ ; tight morphisms compose strictly, loose arrows up to the invertible structure cells. Hovering above the arrow  $M$  sits its fibre, drawn upward in the plane the dashed verticals span: each storey is  $M$  again, now carrying a decoration, and height orders the decorations by how much they assert. A System  $S$  enters at the top, as the singleton  $\{[S]\}$  of its own denotation — its ideal point, the sharpest decoration  $S$  satisfies; a Spec  $\varphi$  enters as the class  $[\varphi]$  it denotes over the wiring; the bottom storey  $\top$  admits every belief and demands nothing. The judgment  $S \models \varphi$  of this chapter’s opening, unfolded to  $[S] \in [\varphi]$ , appears as the globular cell between the top two storeys: an inclusion of decorations over a fixed pattern is a cell of the kernel-checked judgment layer (`SuFunctor.QtslSem.SpecJudg: SpecJudgLayer, cell0fSubset`), and the layer is thin, so the cell carries no data beyond the judgment it witnesses. The tower descends from sharp to weak: every class includes into  $\top$ , so  $S \models \top$  comes for free as the composite cell, and  $S \models \varphi$  is the storey that costs something. Judgment cells also compose loosewise, along the wiring: an initial condition enters as one more decorated arrow and conjoins through the compositor, which is monotone (`wComp_mono`), while an undemanded face costs nothing (`wUnit_transparent`); this is how component judgments assemble the composite judgment at the composite wiring. Stripping the decorations lands every storey on  $M$  and every cell shown upstairs on the identity cell of  $M$ . The total of Theorem 9.46 refines this picture: its objects fix a designated Spec at each interface, and its strict projection to the floor lifts uniquely — a discrete double opfibration. The designations at the objects are chaining contracts, matched on the nose when wirings compose; entailment lives upstairs, in the fibre order, as drawn.

draws the destination: the wiring base on the floor, and above one wiring its fibre of decorations, where a System’s ideal point, a Spec’s class, and the judgment between them all live.

The ambient notions come first, at reference width — this chapter is the book’s authority for them.

### 9.9.1 The Ambient Notions

**Definition 9.37** (Pseudo double category). A *pseudo double category*  $\mathbb{D}$  consists of: a category  $\mathbb{D}_0$  of *objects* and *tight morphisms*; for each pair of objects, a collection of *loose arrows*  $m: X \rightharpoonup Y$ ; for all loose arrows  $m: X \rightharpoonup Y, n: X' \rightharpoonup Y'$  and tight morphisms  $f: X \rightarrow X', g: Y \rightarrow Y'$ , a collection of *cells*  $\alpha: m \rightarrow n$  over  $(f, g)$ ; tight-wise identity cells and tight-wise composition of cells; loose identities  $\text{id}_X: X \rightharpoonup X$  and loose composition  $m \odot n$ , with loose-identity cells on tight morphisms and loose-wise composition of cells; and invertible globular structure cells — associators  $a_{m,n,p}: (m \odot n) \odot p \rightarrow m \odot (n \odot p)$  and unitors  $\lambda_m, \rho_m$  — subject to: the tight-wise category laws (loose arrows and cells form a category  $\mathbb{D}_1$  whose source and target functors land in  $\mathbb{D}_0$ ); functoriality of the loose identity and of loose composition with respect to tight-wise composition (the interchange law); tight-wise naturality and invertibility of the structure cells; and the pentagon and triangle laws. Equations between cells whose tight boundaries agree only up to the category laws of  $\mathbb{D}_0$  are read modulo the canonical transport of cells along those laws. The notion is

Grandis and Paré’s weak double category [100, 101]; the tight/loose terminology follows Shulman [219]. When the structure cells are identities,  $\mathbb{D}$  is a strict double category.

*Remark 9.38* (Mechanization of the ambient notions). Kernel-checked, in the double-category library of the proof-artefact repository: the data and the eighteen axiom clauses as `PseudoDoubleCategoryStruct` and `PseudoDoubleCategory` in module `DoubleCat.Basic` — cells dependently indexed by their tight boundaries, with the boundary identifications this definition reads silently carried by an explicit transport operator (`Cell.cast`), and every axiom whose two sides have propositionally-but-not-syntactically equal boundaries stated up to that transport. Lax double functors (Definition 9.39) are `LaxDoubleFunctor` in `DoubleCat.Functor.Lax`, with pseudo-ness as an invertibility property. The library further carries, at the same kernel grade: tight and loose transformations and modifications, assembling the functor double category; the loose bicategory bridge, through which the library inherits bicategorical coherence rather than re-proving it; companions, conjoints, and equipments, with spans the first genuine equipment (and a proof that the double category of commutative squares is not one); and the span and matrix models, including the dependent-span model (`DoubleCat.DepSpan`) on which the Grothendieck construction below is stated.

**Definition 9.39** (Lax double functor). Let  $\mathbb{A}$  and  $\mathbb{B}$  be pseudo double categories. A *lax double functor*  $F: \mathbb{A} \rightarrow \mathbb{B}$  consists of a functor  $\mathbb{A}_0 \rightarrow \mathbb{B}_0$  on objects and tight morphisms; an action on loose arrows ( $m: X \multimap Y$  to  $Fm: FX \multimap FY$ ) and on cells (preserving tight boundaries strictly, functorially for tight-wise composition); a *compositor* family of globular cells  $Fm \odot Fn \rightarrow F(m \odot n)$ ; and a *unit comparison* family  $\text{id}_{FX} \rightarrow F(\text{id}_X)$  — subject to naturality of both families in cells and tight morphisms respectively, associativity coherence of the compositor against the associators, and the two unit-triangle coherences against the unitors.  $F$  is *pseudo* when compositor and unit comparison are invertible, and *strict* when they are identities.

## 9.9.2 The Wiring Base

The base of the construction is the interfaces-and-wirings layer, itself kernel-checked, in a presentation chosen so that the transport identities a decoration tier consumes hold definitionally. The semantic description first, then the presentation.

**Definition 9.40** (Wire types, interfaces, and state spaces). Fix a set  $\mathbb{T}$  of *wire types*, and for each  $t \in \mathbb{T}$  a nonempty Polish space  $\text{El}(t)$  of states of that type, per the standing structure of Definition 9.15. An *interface* is a finite set of wires typed over  $\mathbb{T}$ ; interfaces and type-preserving maps form the slice category  $\text{Wire} = \text{FinSet}/\mathbb{T}$ , which has all finite colimits, computed as in `FinSet` with the induced typing — any two wires ever identified by a zigzag of type-preserving maps have equal types, so quotients are type-coherent automatically. A *wiring pattern* from  $I$  to  $J$  is a cospan  $I \rightarrow A \leftarrow J$  in `Wire`; its legs may be non-injective, which is how a pattern identifies wires — all identification of wires lives in the legs (equivalently, in the quotient forming the apex), never in the tight direction. Patterns compose by pushout of apexes.

The paradigm pattern is a plugging: to wire a controller’s output into a plant’s input, take the cospan whose apex is the quotient of the disjoint union identifying exactly those two wires. The legs stay injective — the identification lives in the apex — and composing two pluggings by pushout chains them through the shared middle interface.

**Definition 9.41** (State spaces, joints, and marginalization). For an interface  $I$ , the *state space* is the product  $\mathcal{X}(I) := \prod_{x \in |I|} \text{El}(\tau_I(x))$  of its wires’ state spaces — a Polish space, finite discrete when every type is finite. A type-preserving map  $\varphi: I \rightarrow J$  induces the continuous *restriction*  $\varphi^*: \mathcal{X}(J) \rightarrow \mathcal{X}(I)$ , contravariantly functorial, and  $\mathcal{X}(-)$  carries the finite colimits of `Wire` to limits: restriction along a pattern’s pushout legs is a homeomorphism  $\mathcal{X}(A+_JB) \cong \mathcal{X}(A) \times_{\mathcal{X}(J)} \mathcal{X}(B)$  onto the agreement subspace. *Joints* over  $I$  are elements of  $\Delta_{\leq 1}(\mathcal{X}(I))$  — subprobability laws, read per section 9.5 as probability joints over the cemetery-completed carriers, so that partiality composes along the cemetery coordinate; the *marginalization* induced by  $\varphi$  is the pushforward  $\varphi_{\#} := (\varphi^*)_*: \Delta_{\leq 1}(\mathcal{X}(J)) \rightarrow \Delta_{\leq 1}(\mathcal{X}(I))$ , functorial and weakly continuous — the functor part of the Giry monad [95], and that sentence is kernel-checked verbatim at this generality: pushforward of subprobability Borel joints is total, functorial along measurable maps — measurability binding exactly at functoriality, the stratum’s constitutive fact — mass-exact, weakly continuous along continuous maps, and literally `Measure.map` under the coercion (`SuFunctor.Plain.MeasJoint`: `push_id`, `push_push`, `mass_push`,

continuous\_push, push\_toMeasure). When  $\varphi$  is a coprojection,  $\varphi_{\#}$  is marginalization of a joint onto a factor, which is the sense in which every  $\varphi_{\#}$  below is read.

*Remark 9.42* (Two strata: Polish semantics, exact-rational certificates). Stated once, here, for the whole composition tier. The *semantic* stratum — every definition of this section — lives at the standing generality of Definition 9.15: Polish record spaces carrying the weak topology on their subprobability measures (the vague topology’s role, and why the weak one is the topology of record, are settled there), with the Borel  $\sigma$ -algebra underneath. Finiteness, where it genuinely binds in a theorem, enters as a hypothesis, never as a regime of the space. The *certificate* stratum is exact-rational certification over the locally closed  $\omega$ -semialgebraic carriers of section 9.2, and what is finite there is the *presentation*, not necessarily the space: in dimension zero distributions are rational mass tables and sublevel checks are linear programs; in positive dimension the finite object is a vector of rational polynomial coefficients per piece, countable piecing is expressed by functorial plates at the hypernet stratum — one finite object denoting a whole shape-indexed family, naturally in the index — and discharged shape-uniformly at the certificate stratum through pooled aggregate coordinates, so that a certificate discharges polynomial inequalities over an uncountable carrier without enumerating it; count-budget rows survive as optional tightenings, not as the price of admission, and the standing convention below keeps the non-uniform residue. What connects the strata is the standing convention: data outside the exact fragment — non-semialgebraic carriers, non-polynomial scores (the logarithms constituting relative-entropy beliefs included) — ride the wire through outward rational bounds, which enlarge the credal body, the sound direction.

*Remark 9.43* (Hiding a wire: inf-projection, the reflector, and the tightness certificate). The composition calculus marginalizes *joints*, never beliefs (the compositor below); but the assembled stack does eventually hide wires — a demand read over visible coordinates with the hidden ones existentially absorbed — and the semantic content of hiding is the inf-projection of a belief along the pushforward. Its status at the standing generality is fixed by three kernel-checked facts (SuFunctor.Universe.Marginalization). First, the boundary is real: inf-projection does not preserve lower semicontinuity — the witness is escaping mass along the hidden coordinate, a closed family of near-unit point masses whose image under the visible marginal is not closed (exists\_lowerSemicontinuous\_not\_lowerSemicontinuous\_infProj) — so the naive hidden-wire denotation can exit the universe. Second, the repair is judgment-invisible: hiding taken as the lower-semicontinuous envelope of the inf-projection changes no judgment against any citizen of the universe — the envelope is a reflector, and domination against a hidden-wire system never evaluates the infimum at all, resolving upstairs as ordinary domination through the inf-pullback adjunction (le\_lscEnv\_iff, le\_hide\_iff, le\_infProj\_iff) — and the relaxation only lowers the system side, the sound direction. Third, a certificate turns the naive formula exact: for beliefs whose sublevels are tight — equivalently compact, the Prokhorov reading, mechanized as consumption (isCompact\_sublevel\_of\_isTightMeasureSet) — sublevels of the inf-projection are *exactly* the pushforward images of sublevels, the infimum is attained wherever finite, and the inf-compact class is closed under marginalization (sublevel\_infProj\_eq, exists\_infProj\_attained, isCompact\_sublevel\_infProj, lowerSemicontinuous\_infProj, with the coordinate instantiation sublevel\_infProj\_margFst\_eq). The calculus’s generator- and moment-row certificate species are, read at this stratum, inf-compactness certificates; and one coercive summand puts a whole additive composite in the class — a one-line consequence of nonnegativity, the composite’s sublevels sitting closed inside the summand’s compact ones — so tightness enters as a certificate on the operand that needs it, never as an axiom of the universe.

**The mechanized presentation of the base.** The kernel-checked form of the wiring base presents patterns by their *state families*: the base double category has as objects interfaces (bundles of wires over a fixed state type — the untyped stratum, the typed slice being orthogonal bookkeeping), as tight morphisms wire maps acting on states by restriction, as loose arrows patterns presented as type-valued families  $M: \mathcal{X}(A) \rightarrow \mathcal{X}(B) \rightarrow \text{Type}$  — for a cospan, the family of apex states restricting to the given boundary states — with apex state space the total  $\text{Tot } M = \sum_{x,y} M(x,y)$ , loose composition the  $\Sigma$ -pullback (exactly the agreement subspace of Definition 9.41), and cells acting on apex states backward (a map of patterns pushes apex wires forward, hence pulls apex states back over the boundary restrictions). In this presentation the wire category’s laws are definitional, every boundary transport vanishes, and the comparison identities of iterated pullbacks — the entire calculus a decoration tier consumes — hold by reflexivity (kernel-checked: DoubleCat.WireSpan, a full PseudoDoubleCategory instance; the associativity-transport identity of the total-state calculus depends on no axioms at all). Variable identification stays out of the tight direction by construction: tight morphisms act on states by restriction and never quotient anything, and



all identification lives in the pattern families.

### 9.9.3 The Decoration Functor

**Definition 9.44** (The decoration functor: Specs over the wiring base). Fix a stage type with a window system  $U$  (the site dimension, entering through the decoration per [section 9.8](#)) and a state type. The *decoration functor* assigns:

- to each interface  $A$ , the *fibre of Specs*: persistent window-indexed families of classes of beliefs on joints over trajectories in  $A$ 's state space, ordered componentwise by inclusion — such a family is precisely what a Spec denotes, and the fibre is named accordingly;
- to each tight morphism (wire map), the componentwise image of belief transport along the induced trajectory restriction — covariant on classes, because states restrict contravariantly;
- to each pattern  $M$ , the decorations: persistent window-indexed families of classes of beliefs on joints over apex trajectories ( $\text{Traj}(W, \text{Tot } M)$ ), with boundary parties supplied by the indexing;
- to each cell, the componentwise image of belief transport along the backward apex-state map;
- as compositor — the *belief laxator*  $\boxplus$ , the name later chapters use for this operator — the componentwise elementwise *pushforward-then-add*: for decorations  $\mathcal{U}$  over  $M$  and  $\mathcal{V}$  over  $N$ ,

$$\mathcal{U} \boxplus \mathcal{V} := \left( \left\{ (\pi_M)_\# \circ u + (\pi_N)_\# \circ v \mid u \in \mathcal{U}_W, v \in \mathcal{V}_W \right\} \right)_W,$$

where  $\pi_M, \pi_N$  are the joint-to-component projections of the glued apex: the joint over the composite is marginalized to each component's trajectory space, each component belief is precomposed with its marginal map, and the results are added in  $[0, \infty]$  — the beliefs are never themselves marginalized; the joint is;

- as unit, the componentwise singleton of the vacuous belief (zero everywhere) on identity-pattern trajectories — the *austere* unit: everything permitted, no residue.

These data satisfy every obligation of a span-valued lax double functor on the wiring base — functoriality of both transports, naturality of compositor and unit, associativity and both unit coherences — and persistence of every result is the site-direction naturality package (extension commutes with state-side transport, with the window compositor, and fixes the unit).

*Remark 9.45* (Mechanization status of the decoration functor). Kernel-checked, end to end, at three strata of the semantic-universe strand — two nested, one general. The single-window (plain) stratum: carriers and transports in `SuFunctor.Plain.Penalty` (Joint — finitely supported subprobability joints over an arbitrary carrier, partiality native —, `Pen`, `Pen.pull`), the functor with all obligations in `SuFunctor.Plain.SpecFunctor`, its acceptance battery in `SuFunctor.Plain.Battery`. The site-indexed stratum, which is the carrier of the construction's site dimension: window carriers and the extension calculus in `SuFunctor.Site.WindowPenalty` — at both joint strata: the same calculus over subprobability Borel joints, with its persistence closure package and the seam theorems, is `SuFunctor.Site.GeneralWindow` ([Remark 9.33](#)); both are stated over the function presentation of the trajectory carriers — graphs before closure, per [section 9.8](#) — with the Fell-carrier restatement the closing ledger's obligation —, the functor `siteSpecFunctor` with all obligations proved componentwise in `SuFunctor.Site.SiteFunctor`, the battery in `SuFunctor.Site.Battery`, and the matching fork in `SuFunctor.Site.MatchingFork`. The componentwise reduction is itself the structural finding: the seven obligations never see the site — each reduces, window by window, to the single-window coherence core, because trajectory transport is definitionally functorial and the base's transport calculus is reflexivity-valued — so the site dimension's entire force lands in fibre admissibility ([Theorem 9.35](#)), never in the functor laws. The *general* stratum carries the same single-window calculus over subprobability Borel joints on arbitrary measurable carriers: the scalar stock whole — pushforward, belief transport, the compositor over arbitrary marginalization pairs, weak continuity, the lower-semicontinuity closure package with wall-grade attainment on nonempty compacts, and the compactly-supported subclass closed under transport (`SuFunctor.Plain.MeasJoint`) — the coherence *cores* base-free, with the wiring base's comparison identities carried as commuting-square hypotheses (the finite tier's cores are the all-squares-by-reflexivity special case); the tight-tier functor over measurable carriers with the acceptance battery transposed row by row, refutation pins included



(`SuFunctor.Plain.GeneralFunctor: measSpecObj`). The residue: the double-categorical *total* at the general stratum is not claimed — the cores consume comparison squares the exact base discharges by reflexivity, and a measurable wiring base carrying its own transport data is the remaining object, a design artefact in its own right, recorded in the ledger.

#### 9.9.4 The Construction

The double Grothendieck construction — in the covariant, span-valued form Patterson [193] used to rebuild decorated cospans [80], the strict-tight face of the double fibrations of Cruttwell, Lambert, Pronk, and Szyld [52], whose discrete case is Lambert’s [149] — at the dependent (span-of-types) stratum, assembles the decoration functor into the *double category of System Specs*: objects are interfaces with a designated Spec; tight morphisms are wire maps with fibre transport; loose arrows are patterns with a decoration; cells are pattern cells with transported decorations; loose composition pairs the base composition with the compositor. Open Systems live in this total as embedded ideal points of its fibres — the paragraphs below — never as a separate object class. The projection to the wiring base strips the decorations, and it is *strict* — its compositor and unit comparison are identities.

**Theorem 9.46** (The assembled total, and its projection). *The total of the decoration functor under the double Grothendieck construction is a pseudo double category, and its projection to the wiring base is a strict double functor that is a discrete double opfibration: tight morphisms out of a fixed source and cells out of a fixed loose arrow both lift uniquely.*

*Remark 9.47* (Mechanization status of the construction). Kernel-checked, and the division of labour is where the proof burden sits: the Grothendieck construction, the pseudo double category instance on the total (all eighteen axioms), the strict projection, and the generic discrete-opfibration theorem are proved once, for an arbitrary span-valued functor on an arbitrary base (`DoubleCat.Grothendieck`, `DoubleCat.Opfibration`; the general split-opfibration development — `opcartesian` lifts, `opcleavages` compatible with source and target, split laws as theorems about the canonical lifts, and the hom-level isomorphism between fiberwise transformations and strictly projection-compatible functors — is kernel-checked at the same stratum, choice-free); the construction then consumes the generic theorems by instantiation (`SuFunctor.Site.SiteSpecTotal`, `siteSpecTotal_pseudoDoubleCategory`, `siteSpecProj_isPseudo`, `siteSpecProj_discreteOpfibration`) — a one-line consumption in each case, because the generic lifting calculus never inspects fibre internals: the window index travels inside the fibre objects, invisible to it. The variance is forced, not chosen: beliefs push forward totally along wire maps, but cannot be pulled back without a lossy infimum — the cartesian (negative) half, preimage-transport lifts assembling the image-preimage adjunction on class fibres, is a named obligation of the ledger. One generality note: span-of-categories generality is consumed nowhere in the construction — the Spec fibres are dependent families of types, exactly the stratum at which every general theorem above is kernel-checked, so no gap opens between the stated and the mechanized generality. The entity-level fibres (fibre *categories* of Systems and simulation witnesses) are the refinement layer above this total — design, recorded in the ledger, with the general machinery it would consume already kernel-checked, and with a prose-verified construction retained in the workshop record as its design basis; the paragraph closing this section says why that layer is expected to be lighter than a second Grothendieck construction.

#### 9.9.5 The Authoritative Total: Presentations as Fibre Data

The class-level construction above carries arbitrary classes; the authoritative total of the spec tier sharpens the fibres to the class stratum’s own normal form, carried as data. The *coupled-DNF decoration functor* assigns to each interface the coupled-DNF presentations over its state space — finite families of spec atoms, each a safety floor with a finite liveness family containing the properness demand (the option-A discipline: the vacuous atom is  $(0, \{\top\})$ , and an empty liveness family is unrepresentable rather than merely forbidden) — and to each wiring pattern the presentations over its apex. Transport is *atom-level*, disjunct by disjunct — pull the floor and each mark, the canonical fibre operation, judgment-invariant along surjections (Remark 9.49). The compositor is atom-level pairing — floors compose by pushforward-then-add, liveness families by pulled union — and the unit is the vacuous atom itself: the austere unit and the properness demand are the *same atom*, which is exactly why the unit laws close on the nose — every transport fixes the properness mark, and every atom’s own marks absorb it. The plain double Grothendieck total of this functor is the development’s `SingleSiteSemanticUniverse`:

objects are interfaces with a designated presentation, loose arrows are patterns with an apex presentation, and the strict projection to the wiring base is a discrete double opfibration by the same one-line consumption of the generic theorem (kernel-checked, module `SuFunctor.SingleSiteSemanticUniverse: dnfSpecFunctor` with the full obligation package, `SingleSiteSemanticUniverse`, `singleSiteUniverseProj`, `singleSiteUniverseProj_isPseudo`, `singleSiteUniverseProj_discreteOpfibration`). The relation to the class-level functor is refinement, never replacement, and it is a theorem: componentwise, on the surjective sub-base, the image-level transport of a presentation's model class is exactly the model class of the atom-level transport cut to the transport's range — the transport structure theorem of Remark 9.49 instantiated at the wiring base (`specMap_modelClass_eq`), the inclusion half being the comparison cell in the inclusion-ordered fibres (`specMap_modelClass_subset`, with its cell-component companions). Away from the surjective stratum the two genuinely part company, and the atom-level operation is the one under which the fibres keep their constructible presentations: at the finite stratum, through the seam of Remark 9.48, every fibre presents a constructible hyperspace class under semicontinuity of the presented data alone (`Universe.SpecDnf.isConstructibleUnion_probRead` — the finiteness that constructibility consumes is carried by the presentation, by construction). Convexity plays no role at this tier: admissibility of a fibre is constructibility of its presented class, and atoms, presentations, and transport are all indifferent to convex structure. One packaging distinction, stated precisely because it is easy to misread as a gap in the mathematics: the window direction of the site dimension is carried, in full, by the Spec functor above (window-indexed persistent families, section 9.8), composition across distinct clocks is likewise built (the cross-clock paragraph below), and the cross-site direction enters the base as profile-indexing (the profiled-base paragraph below); what the presentation-level total instantiates today is its fibres *per carrier*, so the re-instantiation of the coupled-DNF fibres with window-indexed presentation families — a packaging step, not a new composition theory — is an obligation recorded in the ledger (section 9.13).

**Systems in the total: ideal points, and the order as the fibre carriers' order.** A System enters the class-level total as an ideal point of its interface's fibre — the persistent family of singletons of its window denotations, which is a global member by descent (section 9.8) — and the two structural facts that make the embedding harmless are both battery theorems: the embedding is *monoidal* for composition (singleton decorations compose to the singleton of the composite: `singleton_boxplusW`, `transport_singletonW` — ideal points compose to ideal points, so composing Systems and then embedding agrees with embedding and then composing), and membership agrees with singleton inclusion under the componentwise inclusion order the fibre carriers wear (`wIdealPoint_le_iff`, window by window). One boundary transfers with the claim rather than being smoothed over: monoidality is a class-level, safety-face fact — at the coupled membership relation of the presentation fibres, composing two members of a paired atom's class need not produce a member, one component's liveness dip being destroyed by the other's cost already along identity transports (`exists_coupledSat_add_fails`, kept in the tree as the design boundary); the safety face composes unconditionally (`add_comp_safety_le`), and coherence of the atom pairing is a statement about presentations, never about membership of composed models. One precision, so the categorical claim is not stronger than the machinery: that inclusion order is *external* structure on the fibre carriers — the construction's Grothendieck data are dependent families of types, and no order is part of them — so “the order is the fibre order” is shorthand for a statement about the carriers the fibres happen to be, not about the discrete opfibration. The order does, however, carry a kernel-checked double category of its own: the thin *judgment layer* over the same wiring base, whose loose arrows are decorated patterns and whose cells over identity boundaries are furnished by inclusions of decorations (`DoubleCat.JudgLayer`, kernel-checked at the single-window stratum; instantiated at this tier in `SuFunctor.QtslSem.SpecJudg: SpecJudgLayer`, `cellOfSubset`, with thinness as `cell_subsingleton`) — the upstairs plane of Figure 9.4, and the shape in which the judgment  $S \models \varphi$  composes as a cell. The order-enriched refinement of the total itself, in which the designated fibres and the inclusion cells would live in one structure, remains an obligation recorded in the ledger. With that precision, the one-universe thesis acquires its structural form at the present grade: one fibration over the wiring base, Specs as fibre objects, Systems as ideal points inside those fibres, the universe's order read off the fibre carriers, and the base direction reserved for wiring.

### 9.9.6 The Composition Law

The law itself — pushforward-then-add, lifted to classes — carries two theorems, each with its kernel receipts, and their conjunction is the exact statement of what composing stochastic wirings means.

*Coherence (the law is lawful).* Composition of decorations is associative and unital on the nose modulo the base’s transport, with no renormalization anywhere, because no axiom needs one: the unit is austere and composes transparently (`wUnit_transparent`), the composite class is exactly the image of the pairs — nothing is forgotten that a coherence law could detect (`mem_wComp_iff`; nonempty inputs give nonempty composites, `wComp_nonempty`) — and the element-level coherence calculus reduces through the pushforward calculus to the commutative-monoid laws of  $[0, \infty]$ . The battery fixes the design against its named neighbours, as refuting theorems rather than advice: a *taxed* compositor (any constant surcharge on composites) breaks unit transparency (`taxedBoxplusW_breaks_unit_transparency`); a *funnel* compositor (forgetting the pair content) breaks the unit law against the empty class (`funnelCompW_breaks_unit_coherence`); and composing decorations along mismatched middle interfaces is not wrong but *unrepresentable* — a type error at the total, pinned as a build-time refusal.

*Containment, not identity (the law is more committal than the glue).* Read through the compile isomorphism, the composite belief’s body is contained in the natural-extension glue of the components’ bodies over the shared interface — always, with no topological hypotheses — and the containment is strict off the crisp locus: in grade coordinates the law is the *product* triangular norm while the glue is the *minimum*, and

$$\pi_1 \circ f_1 + \pi_2 \circ f_2 = \max(\pi_1 \circ f_1, \pi_2 \circ f_2) + \min(\pi_1 \circ f_1, \pi_2 \circ f_2),$$

so the two sides agree at a joint precisely when the joint is crisp — one side at 0, or one side at  $\infty$  (per the identity’s reading in [section 9.5](#): the min-term is not a cancellative gap at infinite heights) — and the crisp locus, where every joint is crisp, is exactly the equality locus. There deterministic intuition is exactly right that composition is conjunction: on indicator beliefs both compute intersection of behaviours. Off the crisp locus a compact exact-rational counterexample closes the question permanently: two single-edge dependency-graph models disagreeing about a shared binary variable (relative-entropy beliefs toward  $(\frac{2}{3}, \frac{1}{3})$  and  $(\frac{1}{3}, \frac{2}{3})$ ) admit the witness  $\frac{9}{10} \cdot \text{uniform}$  inside the glue but outside the composite — the two decisive comparisons being  $81 > 80$  for exclusion and  $729 \leq 800$  for inclusion — and both sides are already compact, convex, and scaling-closed, so no closure rescues equality. Containment theorem, crisp-locus characterization, and strict witness are kernel-checked in this repository at both strata: at the finite stratum (module `QtslAux.Imprecise.Containment`; the irrational glue grade enters through the square dictionary and an exact rational split, and the two decisive comparisons are hand-checkable arithmetic), and at the general stratum (module `QtslAux.Imprecise.ContainmentGeneral`) — the containment over arbitrary carriers, an order-pure argument consuming no tightness anywhere (`sublevel_compositeG_subset_glueSetG`, `certificate_transfer`), instantiated at pushforwards of laws with the glue closed (`sublevel_pmComposite_subset_glue`, `isClosed_pmGlueSet`), the crisp locus characterized exactly at the denotation level (`forall_sublevel_eq_iff`, through the exact-scale identity `ENNRat.add_eq_max_iff`: a sum equals the join precisely when the lesser term vanishes or the greater is infinite), and strictness witnessed on a compact carrier, the finite counterexample above entering as the loaded instance — finite discrete carriers are already of the standing generality (`binaryComposite_ne_glue`, `sublevel_ssubset_glue_witness`). The direction of the containment is the sound one — certificates bound every element of the glue, hence a fortiori the composite — and the gap is conservatism, never unsoundness; by the wall-blindness lemma of [section 9.12](#), no wall-fragment verdict can even observe it, so the conservatism costs only the graded successor, and there per-slice.

**Inconsistency and feasibility.** Because beliefs add, two wirings can compose to the everywhere-infinite belief — which compiles not to the empty set but to the bottom credal body, a lawful decoration: composing contradictory constraints *should* denote infeasibility, and it does so first-class, with the boundary verdicts of [section 9.7](#) adjudicating which classes an infeasible composite inhabits — every safety containment, no liveness escape, hence no presented atom’s class (Remark 9.16 and Remark 9.31). Properness of a composite has an existential characterization with a constructive witness: the composite belief is proper iff some pair of component joints, each of finite inconsistency, agrees over the shared interface, and a compatible pair yields a witnessing joint by exact conditional-product gluing — a characterization, not a decision procedure: what the theorem supplies is the shape of the check and the witness format. The coupling ingredient — two laws with equal pushforwards over the shared interface admit a coupling concentrated on agreement, at standard-Borel strength — is kernel-checked in this repository (`QtslAux.Imprecise.CouplingGlue`); the packaged equivalence at both strata, with the finite conditional product as an exact rational formula, is kernel-checked beside it (`QtslAux.Imprecise.FeasibilityGlue`). Soundness of the additive law needed no check at all — every joint the composite admits has its component marginals inside the components’

certified credal sets, a one-line theorem (the dropped summands are nonnegative) — and the kernel carries the two near-misses that guard it: signed scores break it, and sub-unit confidence weights break it, each falling exactly on a field the wire format retains (nonnegative codomain; unit weights or the rebated conclusion; `signed_relaxation_fails`, `subunit_weight_fails`). The algebraic lemma is checked twice over, on the exact stratum and space-free (arbitrary joint models, finiteness surviving only in the contributor index — arity, not state; the two marginal-compatibility modules above).

*Remark 9.48* (The seam between the two mechanized stocks). The composition tier’s exact stratum (finitely supported subprobability joints over arbitrary carriers) and the judgment tier’s finite stratum (laws as real weight functions over a finite outcome type) are two presentations of one object, and the identification is kernel-checked rather than asserted: an equivalence between the two carriers respecting mass, zero, and the probability slice, an order isomorphism of the belief stocks by precomposition, and the judgment bridge — the fibre-order membership judgment of the total transported onto the judgment tier’s belief type, where the crescent machinery of [section 9.7](#) consumes it (`SuFunctor.Plain.Seam: jointEquivBel, jointEquivProb, penEquiv`, and the transported membership `iff`). A second seam runs upward, to the general stratum: the exact joints embed in the subprobability Borel joints as weighted point masses, respecting mass, zero, and pushforward — the restriction theorem: finite pushforward *is* general pushforward read along the embedding — injectively over measurable-singleton carriers, and at the belief and class levels the finite decoration functor is the restriction of the general one along the seam (`SuFunctor.Plain.GeneralSeam: Joint.toMeas, toMeas_push, toMeas_injective, pull_toMeas, image_pull_toMeas`). The three mechanized stocks — weight-function laws at the finite stratum, finitely supported joints at the exact stratum, subprobability Borel joints at the general stratum — thus form a kernel-checked nested ladder, and every statement this chapter makes across stocks factors through these seams.

### 9.9.7 Transport, Clocks, and Profiles

*Remark 9.49* (Constructibility under transport: adjudicated). The construction’s fibre objects are, in the kernel, persistent families of *arbitrary* classes of window beliefs; the stratum of [section 9.7](#) distinguishes the constructible classes inside them, and whether the construction’s operations keep the semantics on the constructible stratum is now adjudicated rather than open, with the answer fixing a design decision (kernel-checked, module `SuFunctor.Universe.TransportCitizenship`). The transport dictionary is the keystone: the body of a transported belief is the *preimage* of its body (`compile_pullPen`), so at the body level transport is preimage — which preserves local closedness and constructibility outright — and the coupled membership relation is transport-invariant along surjections (`coupledSat_pullPen_iff`; off the surjective stratum the invariance fails in both directions — dips can sit outside the range, and safety can be violated there). At the class level the admitted operation is *atom-level* transport — pull the atom’s data, form its crescent — and not image transport of classes: for surjective wire maps the two agree up to an exact structure theorem (the transported crescent is the pulled atom’s crescent intersected with the saturated classes — those equal to the preimage of their image — both inclusions: `transportClass_crescentClass`), while a non-surjective counterexample shows raw image transport escaping the pulled atom’s crescent (escape witnesses die off-range: `exists_transport_not_subset_pulled_crescent`). The compositor has its own dictionary — the body of a sum is the fibrewise radial grade product (`compile_add_radProd`) — and union-distribution receipts close the question against the forcing schema of [section 9.7](#): the atom-level operations distribute over the constructible unions, transport laxity is licensed in the sound direction, and no closure operator re-enters through transport. Class descent preserves constructibility in both directions independently ([section 9.8](#)). The packaging is the authoritative total above: the decoration functor re-instantiated with DNF-presented constructible fibres, atom-level transport and composition built on exactly these element cores, and the comparison cell supplied by the structure theorem; the composite-atom liveness set is carried by the compositor itself (pulled union, properness mark absorbed). Strata, stated plainly: everything this remark names is kernel-checked at the finite stratum (`SuFunctor.Universe.TransportCitizenship`) *and* at the general carrier (`SuFunctor.Universe.General.TransportGeneral`, transports the continuous maps with Borel codomain): the transport dictionary rests on mass exactness and scaling linearity of the pushforward (`compile_pullPen`), the coupled judgment is transport-invariant at surjective *pushforwards* (`coupledSat_pullPen_iff`, with both off-surjective failure directions kernel-checked rather than parenthetical), and the structure theorem splits where its hypotheses genuinely bind — the forward inclusion under surjectivity, the reverse under a named image-closedness supply (the price compactness paid silently at the finite stratum), equality under a closed map. Two hypotheses earn their own mention: bare surjectivity



of a wire map does not yield a surjective pushforward at the general carrier (a measurable-selection matter; a measurable section suffices, kernel-checked as `probPush_surjective_of_section`), and ideal-point constructibility wants a  $T_1$  hyperspace, available over locally compact ambients. Two residues remain, recorded in the ledger — closedness of the saturation locus (which would upgrade image transport itself to an admitted class operation on the surjective stratum) and constructibility of non-surjective images.

**Composition across clocks.** Heterogeneous-time composition — a discrete-time controller against a continuous-time plant, distinct clocks per wire — enters the construction as content of the decoration, not as new machinery; three facts make the seam tractable, the first two kernel-checked, the third proven in adversarially reviewed prose in the development’s workshop record. First, a declared clock morphism (nominal rate with a jitter band, and its hold) enters the composite’s belief content as one *wall on the joint record space* — the indicator of the admissible co-realizations, one shared realization per clock instance — and spec-to-spec reasoning across the seam is decided by the same blocking criterion as within one site, with the sampler’s wall joined into the premise floor as one more conjunct: exact at both the membership and the class level, kernel-checked (`SuFunctor.Universe.DemonicClock: cross_site_union_refinement_iff` — the criterion on the joint floor — with the lower-semicontinuity supply that keeps the adversary class closed, `lowerSemicontinuous_selSup`; criterion and supply are carrier-general, and the wall species with the class-level exactness are kernel-checked at the general carrier too (`SuFunctor.Universe.General.DemonicGeneral: probWall` with its citizenship characterization, `crescentClass_inter_wall`, `cross_site_union_refinement_iff_probWall`), so the sentence holds at every stratum; the sampler-relation stock and its exact composition laws, with prevision transport and certificate soundness across the seam, in `DoubleCat.CrossSite`). Second, transporting behaviour across a relational (jittered) seam is a genuine functor once it acts on *bodies* — all realizations at once, no choice — with composition preserved on the nose: the record-tier no-go (a functional transport would have to choose realizations; kernel-checked as `exists_no_functional_transport`) is evaded one storey up, where the Egli–Milner lift of closed relations composes exactly, the middle witness being an intersection rather than a selection (`SuFunctor.Universe.DemonicClock: emLift_comp`, axiom-free, following the endofunctor theorem of Abbadini, Bezhanishvili, and Carai [7]; what that theorem’s non-allegory caveat actually marks is *meets* — the lift of an intersection of relations can be strictly below the intersection of the lifts, kernel-checked as a one-bit fixture). Third — the prose-proven fact — everything genuinely lossy in crossing a seam is one mechanism in three costumes: an envelope or meet taken *before* a transport forgets shared-realization correlation. The sound discipline is uniform: conjoin walls and compose relations first, envelope once, last. The remaining laxity of the cross-site glue is thereby located, not diffuse, with its species named: the per-component exchange between coarsening and marginalization is *exact* for the pushforward of a single law (naturality, consuming continuity only — no measurability clause anywhere in the proof, records transporting by the carrier’s clip-and-push legs); at the credal species the same exchange is lax in general and strict under a fine-lift (surjectivity) hypothesis on the transports — satisfied by the paradigm coarsenings, where every coarse record is some fine record’s restriction. Under that hypothesis the single inclusion of the coarsening theorem is entirely the glue-gap — direct images under-distribute over meets — and in every case the certified direction is sound. On a pipeline that observes the discipline — and whose exchange satisfies the fine-lift hypothesis — the wall’s width is the seam’s entire price: relation composition is exact, the body-tier lift is exact, the exchange is strict, so what a jittered clock charges is the relaxation of the realization wall itself — wider realization classes, weaker walls, more admitted joints — which the order then treats like any other imprecision. Off the discipline, the ledger’s envelope and meet gaps charge too; the discipline is how one avoids paying them.

**The site in the base: profiles.** The construction is *parametric* over state vocabularies and *indexed* over site-profiles, and the two are one design’s two faces. Parametric: no functor law of the decoration tier inspects the state vocabulary, so instantiating at a clocked vocabulary — realized records carrying their clock — costs no new proof, and the fibres of the indexed object compute to the fibres already built, definitionally (`spacetimeSpecFunctor_fibre_eq` is a `rfl`). Indexed: parametricity alone would leave the site data ambient-in-the-states — a total that *contains* multi-site content while *knowing* nothing of sites — so the site assignment enters as object data of a profiled base (`SuFunctor.Spacetime.Base`): objects carry a wire-to-clock profile, tight morphisms preserve it — along any raw wire map into a profiled interface there is exactly one source profile making it a tight (`profileLiftHom`, `profile_unique_of_hom`): interfaces over sites, as a fibration fact — and states are the fused states the profile constrains. The indexing is by profiles rather than by single sites because it has to be: a mixed-boundary device (`mixedInterface`) has no object over a single-site

base, and the family-of-one-site-totals reading of “indexed” is refuted under jitter by the relational no-go (`exists_no_functional_transport`, the cross-clock paragraph above). The profile-forgetting embedding (`profiledEmbed`) composed with the decoration functor then yields the iterated double Grothendieck total `SemanticUniverse` — Specs over interfaces over site-profiles — whose strict projection is a discrete double opfibration by one line of the generic theorem (`universeProj_discreteOpfibration`): the lifting calculus never inspects fibres, so the site grading costs it nothing. The two faces stay bridged in both directions: each one-site base sits at the constant profile, with tights, states, and patterns corresponding by equivalences on the nose (`constHomEquiv` and companions — per-site conservativity), and cross-site seams enter the *fibres* as wall conjuncts, the preceding paragraph verbatim, inside the indexed object. One comparison direction is genuinely lax and deliberately so: `profiledEmbed`’s compositor is invertible (a constrained middle state is a fused middle with a membership proof, and proofs are irrelevant) but its unit comparison is not — the profiled identity embeds as the fused identity pinned to the profile, a genuine sub-pattern of the fused diagonal — so forgetting the site discipline is a lax operation because the discipline is information. The outer storey is what makes “which site is this wire on” a question the base can answer and the grammar can enforce.

### 9.9.8 The Monoidal, 2-Functorial, and Certificate Layers

**The monoidal and 2-functorial layers.** The juxtaposition tensor on the wiring base, and with it the symmetric monoidal structure of the assembled total, ride a hypothesis package on the decoration functor — fiberwise tensors with coherence data, plus two families of mediating cells against the compositors — whose status this book reports with its receipts and its history, in [section 9.13](#): the package’s axioms are mechanized as named mixins whose satisfaction and refutation are decidable over cyclic-group graded models, the dependency table among them machine-checked, and the package’s original version omitted an independent associator-interchange axiom, witnessed by a finite graded model. The 2-functorial layer — maps of decoration data inducing maps of assembled totals, with an isomorphism of hom-categories onto the strictly projection-compatible functors — is kernel-checked at the construction’s own stratum (the hom-level isomorphism with explicit inverse and both roundtrip laws, choice-free), with its 2-categorical superstructure at prose grade. Neither layer is consumed by anything the earlier sections claim.

**Certificates: fibrewise morphisms, not a second tower.** An earlier design sketched certificates-over-Systems as a second Grothendieck construction over this one, iterated since opfibrations compose. The tower collapses: in inclusion-ordered fibres a certificate is a *fibrewise morphism* — a witnessed inclusion of an ideal point, or of a smaller spec class, into a larger one — and fibrewise morphisms already compose, vertically by transitivity, along tight and cell transport by monotonicity, and horizontally along the compositor by witness-carrying morphisms, so no second certificate storey is needed: the certificate layer is the order-enriched refinement of the total named above (fibre 2-cells as inclusions), not a new fibration. The collapse is sharper than a tidiness claim, in two kernel-checked ways (module `SuFunctor.Universe.CertificateLayer`). First, certificate transport needs no side conditions: the transported coupled judgment factors through the fibrewise infimum (`fibInf`, empty fibres reading  $\top$  — the right Kan extension along the carrier map) at *arbitrary* carrier maps (`coupledSat_comap_iff_fibInf`), so transported model classes are preimages and spec-to-spec transport is unconditionally monotone (`SpecDnf.comap_modelClass_mono`) — the opfibration line above, beliefs pushing forward totally while pulling back only through a lossy infimum, upgraded from variance remark to mechanism: the lossy infimum reappears exactly once, as the reflection every transported judgment factors through, and surjectivity’s entire load rests on ideal points, whose off-range reflection a non-surjective transport rewrites to  $\top$  (`exists_comp_notMem_modelClass_comap`). Second, the iterated tower is not merely unnecessary but unbuildable in plain form: a second-storey certificate functor would owe a *total* composition laxator, and there are component certificates whose composite target holds no certificate at all (`exists_coupledSat_add_fails`) — the fibrewise picture survives precisely because a fibre-hom may fail to exist without breaking any structure; where the con-witness is supplied the composite exists (`coupledSat_comp_add_of_dips`), and on the safety fragment composition is witness-free. A companion witness locates why the fibres are presentations: two presentations of the same model class whose composites with one fixed atom differ (`exists_boxplus_not_semantically_monotone`) — the compositor is monotone in the presentation order (`dnfBoxplus_mono`) but does not descend to semantic classes, so a composite certificate names the disjuncts it composes and carries the composite dip family. The assume-guarantee calculus of Capucci and Myers [37] is the worked instance of this collapse, not of the iterated construction: their Boolean stratum is the



fibrewise-morphism picture verbatim — a certified lens is a base loose arrow together with fibre-morphism data over its source — and their quantitative construction is itself one double Grothendieck over ordered fibres of bundle predicates, never iterated, with slack composition recovered here as the derived hom-action of the compositor rather than postulated as data. The boundary: everything in this paragraph holds per site-fibre; the site dimension — Specs over interfaces over site-profiles — is not collapsed, and keeps the iterated double Grothendieck of the profiled-base paragraph above. What remains, recorded in the ledger, is nesting (a System’s entire state as one variable of another), which is the same move aimed at the fibre. Nothing above depends on this paragraph.

## 9.10 The Semantic Universe, Assembled

The storeys are all built; this section states the building whole. The definition below is the chapter rendered as a single mathematical object: each clause names the storey it consumes and the section that built it.

**Definition 9.50** (The Semantic Universe). Fix a set of sites (clocks) and, for each site, a vocabulary of realized records carrying that site’s stamp (section 9.1–section 9.3). The **Semantic Universe** is the iterated double Grothendieck total

$$\mathbb{S}\mathbb{U} = \int (E \circ \mathcal{D}) \quad \text{— Specs over interfaces over site-profiles —}$$

the total, under the covariant span-valued double Grothendieck construction of section 9.9, of the composite of the profile-forgetting embedding  $E$  of the profiled wiring base into the fused wiring base with the coupled-DNF decoration functor  $\mathcal{D}$ . Unpacked, outermost layer first:

- *The outer indexing: site-profiles.* An object of the profiled base is an interface — a bundle of wires — together with a *site-profile*: an assignment of a site to each wire. Underneath every site sits the causal order (section 9.1), from which the site’s windows and coverage derive and on which its duration and tangency are declared (section 9.2); the profile is how that temporal geometry reaches the grammar of interfaces. Tight morphisms preserve profiles, and uniquely so — along a raw wire map into a profiled interface there is exactly one source profile making it profile-preserving — so the tight category of the profiled base is *discretely fibred* over site-profiles. This is the *first* Grothendieck integration: interfaces, integrated over site-profiles, as a fibration fact. The indexing is by profiles rather than by single sites (the mixed-boundary and clock-jitter arguments of section 9.9).
- *The middle base: interfaces and wirings.* Beneath the profile lies the fused wiring base of section 9.9: interfaces over the fused record vocabulary, wire maps as tight morphisms, wiring patterns as loose arrows — presented by their state families, with apex states restricting to boundary states — and pattern cells acting on apices. The embedding  $E$  forgets the profile. It is lax and not pseudo: its compositor is invertible, but its unit comparison embeds the profiled identity as the fused identity *pinned to the profile* — a genuine sub-pattern of the fused diagonal (section 9.9).
- *The fibres: presented constructible classes.* Over an interface, the decoration functor  $\mathcal{D}$  assigns the coupled-DNF presentations of constructible classes of belief bodies over that interface’s states. This clause compresses the tower: a record is an observation history over a window (section 9.3); the record carrier over each window is a compact Fell hyperspace in which records enter as graphs (section 9.4); a System denotes a belief — a lower semicontinuous functional on laws, its radial body a closed set of laws (section 9.5); the hyperspace of bodies makes Systems singleton elements (section 9.6); a Spec denotes a constructible class of bodies, a finite union of crescents closed under the connectives (section 9.7); and classes are carried window by window over the site, gluing as persistent families (section 9.8). The fibre holds *presentations*, not bare classes, because composition consumes presentations: the compositor is monotone in the presentation order and does not descend to semantic classes (section 9.9).
- *The second integration.* The double Grothendieck construction integrates these fibres over the profiled base — Specs over interfaces, stacked on interfaces over site-profiles. The two integrations compose into one total; naming them as they stack is the definition’s phrase “Specs over interfaces over site-profiles.”

Concretely, in the assembled total:

- an *object* is a site-profile, an interface over it, and a designated coupled-DNF presentation of a constructible class over the profile's fused states — which site each wire is on, what may be plugged where, and what is demanded of whatever is plugged;
- a *tight morphism* is a profile-preserving wire map together with the transport of the designated presentation — and the transport is not a choice: the projection below makes it unique;
- a *loose arrow* is a profiled wiring pattern with an apex presentation — a way of coupling two boundaries through a shared body of joint states, demands included;
- a *cell* is a pattern cell with transported decorations, and *loose composition* pairs the base composition of patterns with the compositor of presentations.

The strict projection  $\pi: \mathbb{S}\mathbb{U} \rightarrow (\text{profiled base})$  strips the decorations, and it is a *discrete double opfibration*: tight morphisms out of a fixed source and cells out of a fixed loose arrow lift uniquely. That is what the construction buys. Composition of Systems and Specs is never a design decision at use time — demands push forward totally along wirings, with no choices left to make and no side conditions owed to the site grading — and every pattern-level composition law of [chapter 13](#) has one place to discharge.

*Remark 9.51* (Mechanization status of the Semantic Universe). Kernel-checked, end to end, by one-line consumption of the generic Grothendieck theorems of [section 9.9](#) (the lifting calculus never inspects fibre internals, so the site grading costs it nothing): the iterated total is `SemanticUniverse` in module `SuFunctor.Spacetime.Universe` — the double Grothendieck total of `spacetimeSpecFunctor`, which is the landed decoration functor composed along the profile-forgetting embedding (`profiledEmbed`), with the pseudo double category instance (`semanticUniverse_pseudoDoubleCategory`) and the strict projection `pseudo` and a discrete double opfibration (`universeProj_isPseudo`, `universeProj_discreteOpfibration`). The profiled base and its unique profile lifts are `SuFunctor.Spacetime.Base` (`profileLiftHom`, `profile_unique_of_hom`); the embedding's laxity is by semantics, its faithfulness on tights a theorem (`profiledEmbed_faithful_tight`); and the fibres of the iterated object are the landed fibres *definitionally* (`spacetimeSpecFunctor_fibre_eq` is a `rfl`) — the construction's parametricity paying inside the indexed object. Per-site conservativity is packaged and pseudo: each one-site base sits inside the profiled base at the constant profile with tights, states, and patterns corresponding on the nose (`constProfile`, `constProfile_isPseudo`), and reading a presentation at one site is restriction along that packing (`siteRead`). Cross-site seams enter the fibres as wall conjuncts, exactly as in [section 9.9](#) (`SpecDnf.mem_modelClass_withWall`). The fused total — the double category of System Specs over the fused base, the development's `SingleSiteSemanticUniverse` — is the middle integration's total, and the headline object refines it by composing the site dimension into the base rather than leaving site data ambient in the states. What remains at design grade is unchanged from the composition section's own ledger: the monoidal layer's hypothesis package and the 2-categorical superstructure ([section 9.13](#)); nothing in this definition consumes either.

## 9.11 Adequacy: the Big Tent, Squared

A universe that claims to be a big tent owes each resident framework a receipt: a proof that compiling the framework's models into the toolkit and denoting them here agrees with interpreting them by the framework's own textbook semantics and *including* that semantics into this universe. The receipt has a fixed genre, the *adequacy square* ([Figure 9.5](#)), built so that squares are cheap to mint and impossible to hollow. Its *corners* are the literature's object (restated with the literature's own well-formedness fields, and tied where possible to a mechanized probability library's construction of the same object), the shipped compiled row grain (sparse conventions included, exactly as the emitters produce it), and this universe at the per-element stratum. Its *edges* are the shipped compile, the framework's own law, and the inclusion of precise models into the imprecise universe. The square's statement is an *invertible globular cell* in the thin double category of per-element semantic constraints — loose arrows are constraints on subprobability mass functions, loose composition is conjunction, cells are pointwise entailments — between denote-after-compile and include-after-interpret. The one contentful ingredient per square is a *feeding lemma*: the compiled constraint diagram pins the framework's law, exactly (finite-horizon Kolmogorov-extension content, spent once per engine).

Everything else is uniform packaging (`SuFunctor.Target`: the constraint double category;

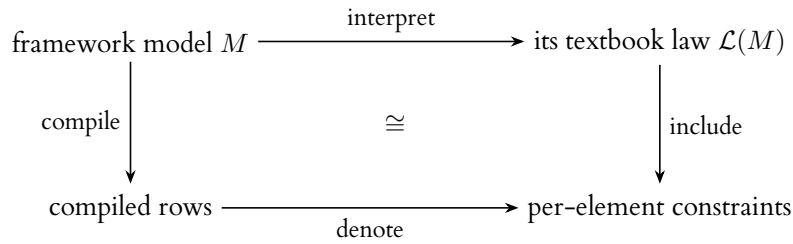


Figure 9.5: The adequacy square, schematically. The left edge is the shipped compile, exactly as the emitters produce its rows; the top edge is the framework’s own semantics; the right edge is the inclusion of precise models into this universe; the bottom edge reads the compiled rows at the per-element stratum. The square’s statement is that the two composites agree — an invertible globular cell — and the feeding lemma is the one contentful ingredient: the compiled constraints pin the law, exactly.

`SuFunctor.Adequacy.Inclusions`: the inclusion library — singleton inclusion for precise frames, point-mass inclusion for deterministic ones, constraint-list conjunction with its definitional unit and cons laws, and the invertible-cell packager that turns a pointwise iff into the square).

Seven squares are kernel-checked at this writing, and each is stated in full — receipts included, most at the working scale of an exam problem with its sketch — in the section of the modelling catalogue (chapter 12) that presents its System-former or, in the delay line’s case, the wiring pattern that replaces one; the composition patterns that ride the squares cite them in place (chapter 13). Adequacy lives with the entry it certifies, and what this section keeps is the genre and the division of labour. The *Markov chain* square (`SuFunctor.Adequacy.Dtmc`; section 12.11) is the engine: one induction carries all the stochasticity, the compiled constraints force total mass one — a precise frame never draws on the universe’s native partiality, which is the “inclusion into the precise face” clause made a theorem — and a connector ties the exact-rational law to the mechanized probability library’s own construction. The *Bayesian network* square (`SuFunctor.Adequacy.Pgm`; subsection 12.20.1) is the same engine transposed from a time-line to a topological order, which is exactly what it was minted to demonstrate: nothing in the genre presumes time, because the genre is *parametric* over its trajectory carrier rather than indexed over the site — a square consumes only the carrier’s product structure, never an order on its coordinates — and where the site direction genuinely enters is exactly the two named gaps at this section’s close, not a hidden assumption inside any square. The *Markov decision process* square, fixed-policy face (`SuFunctor.Adequacy.Mdp`; section 12.13), is the first to reuse an engine wholesale — the chain square instantiated at the closed loop, no new induction, every stochasticity fact already spent — and the *demonic* (open-policy) face of the same class (`SuFunctor.Adequacy.MdpDemonic`, on the set-valued envelope vocabulary of `SuFunctor.Adequacy.EnvelopeUnion`; same section) is the first genuinely imprecise square: a set-valued bottom edge, one singleton inclusion per policy, reconciled with the credal reading through the radially maximal slice, with its degenerate cases mechanized. The *delay-line* square (`SuFunctor.Adequacy.DelayLine`; section 12.6) leaves the stochastic engines — the first spatially extended cell, making delay differential equations a wiring pattern rather than a formalism — and the *heat-on-torus* square (`SuFunctor.Adequacy.HeatTorus`, with the marginal-ring contrast in `SuFunctor.Adequacy.MarginalGlue`; section 12.5) is the first where the cover tier genuinely bites, the ring-closing wall carrying the mesh’s holonomy obligation. The seventh square (`SuFunctor.Adequacy.HeatTorusInit`; same section) ports the mesh square’s initial face as an exposed boundary trace along the window’s own filter, and, beside the squares, the set stratum of the same mesh is adjudicated rather than left to intuition (`SuFunctor.Adequacy.SolutionSetMesh`; same section), the first consumer of the record carrier’s equalizer-descent form: closed sets of records glue by exact descent while the point stratum’s holonomy wall goes wall-invisible there, and the missing datum is branch pairing — a named successor, not any pointwise constraint.

The squares also feed the universe this chapter assembles: the wiring stratum (`SuFunctor.Adequacy.SemanticUniverseWiring`) ties the stochastic squares to the coupled membership relation of the semantic universe’s fibres, per square, with no new induction — the squares’ feeding lemmas are consumed whole. Systems enter the fibres through one door, the radial wall of the precise constraint (`rwall_singletonC`): a law’s entry *is* its ray, the body of the precise belief in that law, and the judgment bridge is ray-shaped — a law’s entry meets a constraint’s wall atom exactly when the law satisfies the constraint

(`SuFunctor.Universe.RadialUniverse`; the bridge in three forms, unconditional at full mass). The wiring squares themselves are kernel-checked on the *slice* presentation of that entry (`idealPoint`: the wall vanishing at the law's joint alone, across the injective scalar seam `SubPMF.toJoint`), and the two presentations are verdict-equivalent against every radial constraint at every mass — the slice is the ray's radially maximal slice, the reconciliation the envelope stratum proves and the radial module sharpens: judged identically, the slice is nevertheless *not itself a member* of the universe (no belief compiles to a pinned point, `compile_ne_singleton`), so it is a computation device of the wiring stratum, never a denotation. Compiled diagrams become members of the fibre as single-disjunct DNF presentations (`diagramDnf`) whose safety face is the wall of the conjunction's image and whose liveness set is exactly the properness demand; and the membership bridge (`coupledSat_idealPoint_cwall_iff`) carries all four stochastic squares through one equivalence (`idealPoint_mem_diagramDnf_iff`) — membership of the law's entry in the compiled presentation's model class is equivalent to the law being the classical semantics, for the chain, the graphical model, the fixed-policy decision process, and, at the set stratum, the demonic face. Properness is free at every law's entry, the bottom law included — the lawful denotation of infeasibility is proper, and a jointly unsatisfiable compiled conjunction denotes the bottom body, never an exclusion of the inconsistent world (`contradictory_composite_denotes_bot`); no law enters as the inconsistent system, and the inconsistent system belongs to no presented diagram's model class. Two boundaries: the wiring is *per-carrier* — the fibre's record space is supplied directly, and the construction of the wire object from the compiled system is a separate, named stratum (the record-space tie below) — and the constraint walls are exact-stratum objects; the topological reading is the envelope stratum's successor, not built here.

Two disciplines govern every square, present and future. *Horizon*: the stochastic squares are stated at every finite horizon, with the  $\sigma$ -additive infinite-horizon extension (Ionescu–Tulcea) a named separate stratum, never smuggled; the differential squares are stated over open time windows, their record-tier content pointwise. *Envelope*: precise frames include by the exact singleton at the constraint stratum, and the reconciliation with the credal reading — the ray's maximal slice is the singleton, verdicts agree kind by kind, with the zero-law boundary and the down-closure hypothesis stated — is the kernel-checked bridge of Remark 9.23, composed once per square rather than re-proven; deterministic frames include their solution sets at the possibilistic stratum, with per-solution point-mass faces beside the set-level square. Beyond the seven kernel-checked squares, the future roster's cost is set by engine reuse and left-edge availability. Probabilistic logic programs (finite groundings, exact-rational) and declared-scheduler automata would ride the existing engines; continuous-time chains would split into an exact jump-chain face and a separately-stated holding-time stratum; input boxes and mixture formers would ride the set-valued envelope stratum the demonic square opened; stochastic-differential frames wait on their own analytic left edges; and the deterministic differential squares' finite/certificate and enclosure strata are un-mechanized successors with cost estimates recorded — estimates, not available squares. The two ties that would knit the squares to the record carrier and the site direction are the named gaps of the ledger: the *record-space tie* (the squares' trajectory carriers, supplied directly today, identified with the record loci of section 9.4 over the construction's base) and *horizon-pasting* (sequential composition of squares along the site direction, which is the construction's compositor read at the adequacy stratum).

## 9.12 The Wall Fragment: How Checked Verdicts Embed

The universe is now complete on its own terms: beliefs and their order, bodies and their hyperspace, constructible classes, fibres over the wiring base. This section is the adapter — everything about how the fragment the shipped checkers actually certify embeds into that universe is collected here, after the universe and before the receipts, so that no definition above owes anything to the checker and every checker fact below is an instance, not an axiom.

**Proposition 9.52** (The restriction theorem: walls). *Call a belief a wall when it is the indicator of a closed set  $C \subseteq \Delta_{\leq 1}(\mathcal{X})$  — zero on  $C$ , infinite off it; convexity is not part of the notion (the mechanized wall takes arbitrary closed sets), though every wall the shipped grammar denotes is in fact convex — half-spaces and their intersections — as a theorem, not a requirement. Every judgment kind of the shipped specification grammar denotes a wall at its carrier: the checker's valuation spaces are countable and discrete, where every event is clopen and both threshold polarities close, so the probability atoms  $P_{\geq p}[\varphi]$  and  $P_{\leq p}[\varphi]$  denote closed half-spaces of laws and sequent validity an intersection of such. (Over a general record space of the standing structure, event evaluation  $\mu \mapsto \mu(A)$  is not weakly continuous for arbitrary Borel  $A$ , and threshold sets close by the Portmanteau polarity discipline: lower bounds*

$\mu(A) \geq p$  give closed sets for closed events, upper bounds  $\mu(A) \leq p$  for open events; the grammar's event classes are chosen to obey it.) On walls the universe's order is equivalent to the classical verdict:  $\sigma \geq \chi_C$  iff every law admitted at finite height lies in  $C$ . The shipped checker's verdicts are comparisons in the universe's order, restricted to walls; nothing re-adjudicates under the graded reading.

**Proposition 9.53** (The bridge theorem: tolerance slices). *For any  $\sigma, \varphi: \Delta_{\leq 1}(\mathcal{X}) \rightarrow [0, \infty]$ ,*

$$\sigma \geq \varphi \iff \{\sigma \leq s\} \subseteq \{\varphi \leq s\} \text{ for every tolerance } s \in [0, \infty).$$

(Left to right is monotonicity of sublevels. Right to left, at each  $\mu$ : if  $\sigma(\mu) = \infty$  the inequality at  $\mu$  is automatic; otherwise instantiate  $s = \sigma(\mu)$ .) A graded comparison is therefore equivalently a family of crisp containments indexed by a rational tolerance — each slice a comparison of today's wall shape about a derived object — and this slice family is the graded order's certification chart: a producer may certify slices incrementally, while the composed claim remains the single domination statement.

**Why the universe grades at all, and what grading costs the checker.** The motivation is a muteness fact about the wall fragment: an indicator belief  $\chi_C$  and a graded ramp that vanishes at the centre of  $C$ , prices deviation within it, and is infinite off it receive *identical verdicts* from every wall — nothing expressible in the crisp fragment distinguishes a denotation from its indicator collapse, although the two describe different models. The graded order makes the landscape itself the subject. The cost: certifying  $\sigma \geq \varphi$  off the wall fragment is a named successor obligation for the certificate calculus, not shipped machinery. For belief data declared by rational polynomials over the law polytope — the finite-stratum carrier  $\Delta_{\leq 1}(V)$  is a rational polytope — domination is a single polynomial-nonnegativity obligation over a semialgebraic set, the certificate species this book's rule catalogue already discharges elsewhere, re-aimed; equivalently, by Proposition 9.53, per-slice containment rows with one added rational tolerance field (rational tolerances suffice: each real sublevel is the intersection of the rational ones above it). Until that obligation ships, the chapter's discipline is the claim ledger's: the wall fragment is shipped; the graded fragment is the semantics, argued and costed.

**Wall-blindness.** The wall fragment cannot see the gap between composition and conjunction (section 9.5's one inequality), a two-line fact: a finite sum in  $[0, \infty]$  is finite iff every summand is, so  $\{\sigma_1 + \sigma_2 < \infty\} = \{\max(\sigma_1, \sigma_2) < \infty\}$ ; and a wall comparison reads only the finiteness set ( $\sigma \geq \chi_C$  iff  $\{\sigma < \infty\} \subseteq C$ ). Sum-composition and join-composition therefore give identical verdicts against every wall — every exactness question the graded universe opens, it opens at the graded successor, and there as quantified conservatism in the safe direction, never as unsoundness. A companion invisibility governs convexity: against a convex Spec, a System and its biconjugate (greatest convex lower semicontinuous minorant) receive identical verdicts, so system-side non-convexity is invisible to single-shot linear tests while remaining composition-visible — the checker sees the convex reading, composition sees the territory, which is this book's untrusted-producer/trusted-checker split reappearing inside the semantics; a Spec whose content is not convex-expressible is certified through its own sublevel presentation, per-tolerance slice rows, rather than through an instrument that cannot see it.

**Where certificate rows enter the defining object.** This book works with the game-theoretic presentation of the glued global object (section 9.5) because certificates plug into it *definitionally*: the game-theoretic upper expectation is an infimum over bounded-below supermartingales dominating the target functional, and exhibiting a certificate  $V \geq 0$  whose floor rows dominate the indicator of the bad event and whose generator rows make it a supermartingale against the local credal data is exhibiting one element of the index set of the defining infimum — under one bookkeeping hypothesis that makes the membership literal: the limit-inferior condition is checked for the certificate's process stopped at the début of the bad region, and that stopped-process discipline is a per-mechanism-profile obligation, carried with the profile. With it,

$$\overline{P}(\text{bad}) \leq V(x_0), \quad \underline{P}(\text{safe}) \geq \underline{E}(1) - V(x_0),$$

a coherent lower-prevision bound, witnessed — and for full-mass denotations the familiar  $\underline{P}(\text{safe}) \geq 1 - V(x_0)$ ; the subtraction is the leak asymmetry of section 9.5, with its kernel receipts. The scheduler-uniformity a reader trained on nondeterministic model checking will look for is already inside the quantifier order: certificate rows carry the credal parameters symbolically over their declared regions, so a checked row holds for every element of every local credal set at once; fixing any compatible precise process makes



$V$  an ordinary supermartingale; the classical Ville–Doob inequality gives the per-element bound; and the bound passes to the envelope last. One quantifier, one direction, no interchange — and the checker verifies membership (sign conditions on polynomial rows), never the infimum itself; no completeness claim is made anywhere in the stack.

**The anchor: where a sequent verdict lives.** Every certificate in this book ultimately discharges into a checker verdict, and the verdict object is deliberately small: a sequent  $\Phi \vdash \Psi$  over an ambient hypernet (chapter 10), judged semantically valid over  $H$  when every admissible valuation of  $H$ ’s wires satisfying  $\Phi$  satisfies  $\Psi$  — the satisfaction clauses and the validity statement are stated once, with their mechanized counterparts quoted beside them, in chapter 15. The answer to *where* the verdict lives in this universe is exact and short: sequent validity is the wall-fragment restriction of the universe’s order, instantiated at the hypernet’s valuation space. It consumes no site structure — by design — and it consumes composition structure by inheritance; both statements are made precise below.

**The carrier.** Fix a hypernet  $H$ . A valuation assigns to each typed wire an exact tensor value of the wire’s declared type, and the valuation space  $\text{Val}(H)$  is the product of those countable discrete spaces — Polish by the standing structure. Admissibility and satisfaction each consult finitely many coordinates, so the three sets a verdict is about — admissible valuations, premise satisfiers, conclusion satisfiers — are cylinders over finitely many discrete coordinates, hence clopen. No temporal structure enters anywhere: a valuation is a single observation — a record with one stage — and the satisfaction clauses consume neither duration nor tangency. That is the site contract’s second freedom (*the checker never sees the site*) made visible in the semantics itself rather than promised at the format layer.

**The face.** Let  $B$  be the set of valuations the hypernet’s applications admit and the premise allows — closed, by the cylinder observation. Its possibilistic belief (Remark 9.22) is a wall, and under the order three readings of the verdict coincide:

$$\Phi \vdash \Psi \text{ semantically valid over } H \iff B \subseteq \{v \models \Psi\} \iff \bar{P}(\{v \not\models \Psi\}) = 0,$$

the checker’s quantifier, the behavioural superset test, and the degenerate face’s Boolean upper probability — equivalently, the order  $\sigma_{\text{scene}} \geq \sigma_{\text{sequent}}$  between the two walls, which is Proposition 9.52 at this carrier. The chain is kernel-checked at the exact stratum, stated as its two equivalences and their composite: the superset reading (`semanticallyValid_iff_subset`), the Boolean upper probability (`upperProb_eq_zero_iff`), and the composed face theorem (`semanticallyValid_iff_face`, module `SuFunctor.Plain.CheckerFace`). The corners agree too, and the agreement is where naive readings crack: when no admissible valuation satisfies  $\Phi$ , the sequent is vacuously valid, the wall is everywhere infinite, the compile is the bottom credal body — and the bottom body upper-bounds every event by zero (Remark 9.16), so the vacuous implication of the checker’s quantifier is the safety half of the denotation stratum’s bottom verdict, one fact in two dialects — itself kernel-checked as a named corollary of the face theorem: the compile of an unsatisfiable region is the bottom credal object (`suppIn_empty`), and the corner discharges through it (`face_ex_falso`). The agreement is lawful because sequent validity is an upper-envelope verdict throughout: the reading that refuses at bottom — the liveness reading, with its existential conjunct — is not one a sequent ever speaks. What the verdict deliberately does not exercise: the prevision pair never separates (nothing weighs valuations), partiality never bites (there is no mass to leak), and no belief grades (the face is crisp).

**No site structure by design; composition structure by inheritance.** The absence of site structure from the anchor is design (the second-freedom reading above), and the inclusion theorem preserves it: the anchor face factors through the one-stage stratum. Composition structure the anchor inherits: hypernets form a decorated-cospan double category, read through Patterson’s double-categorical lens [193] — loose arrows are cospans of wire contexts, the legs labelling the input and output wires, decorated by a wiring diagram on the apex — the anchor face is the object part of a lax double functor from that double category into the double category of System Specs (an open hypernet goes to its possibilistic coupling, the wall of its admissible-valuation set), packaged in the toolkit’s Lean tree as `hnetInclusion` (module `SuFunctor.HypernetDouble`): `opcartesian` lifts preserved on the nose, the compositor invertible exactly on the seam-disjoint stratum that content addressing guarantees for encoded hypernets, the unit comparison genuinely lax by conformance-pinning. The substitution laws of chapter 10 (Theorem 10.4) are exactly its



functoriality — with the determinism hypothesis of law (ii) deleted along the way, since the possibilistic face composes for the full calculus, `cell` included.

**What sits above the anchor.** The temporal and probabilistic instruments of this chapter are exercised by the *rules*, not by the verdict. A generator row’s meaning as supermartingale membership lives in the rule lemma that emitted it (the certificate paragraph above); the reading of an accepted sequent as a graded-order bound on a System’s denotation is the graded fragment, the named successor of Proposition 9.53. The factorization is the architecture’s own: the checker verifies membership — satisfaction over valuations, sign conditions on rows — and the deep semantic content lives in soundness lemmas quantified over every hypernet and every admissible valuation, one lemma per rule.

*Remark 9.54* (Mechanization status of the wall fragment and the anchor). Kernel-checked, over the general carrier of the semantic-universe strand — the statements are order-pure, and the mechanization consumes nothing of the record space (`SuFunctor.Universe.General.CheckerAnchor`): the wall restriction of the order (`sat_wall_iff`, `sat_wall_wall_iff` — Proposition 9.52) and the tolerance-slice bridge (`sat_iff_sublevel_subset` — Proposition 9.53). Kernel-checked, for the inclusion’s combinatorial core at the aligned stratum — the pushout-normal form where seam wires are literally shared — against the calculus’s own hypernet definitions (module `Qtsl.HypernetCospan`): loose composition of well-formed open hypernets with every boundary clause proved (`OpenHypernet`, `glue`), the admissibility split of a glued net — a valuation of the composite is admissible iff both parts’ applications hold, determinism nowhere (`glue_admissible_iff`) — and the unit and associativity laws on the nose (`unitOpen`, `glue_assoc`); the full-generality rename transport is kernel-checked in the same tree alongside the inclusion’s three other transformation laws (`theta_rename`, module `SuFunctor.HypernetInclusion`), and the double-category-library instantiation of the decoration functor is the packaged inclusion itself (`hnetInclusion`, module `SuFunctor.HypernetDouble`). Kernel-checked, in this repository’s certificate-calculus soundness development (the `Qtsl` package under `lean/qtsl`, whose per-rule statements [chapter 15](#) quotes beside each rule): the valuation, satisfaction, and validity definitions the anchor instantiates, and the theorem that every certificate object of the calculus witnesses its sequent’s semantic validity — total over the rule catalogue, by structural recursion with one lemma per rule (`ProofObject.sound`, with its checker-level face `check_sound`, module `Qtsl.Soundness`). The HOL4 development behind the certified checker (this repository’s `hol` tree) takes hypernet-level soundness as its trusted-statement target, and its composition toward the end-to-end theorem is in progress: demonstrated route, assembling now, and this section will say “machine-checked end to end” about that side only when it is.

## 9.13 Coherence Receipts

A chapter that promises “proven mathematically coherent” owes the reader its evidence classes, claim by claim. This book’s organizing principle — powerful untrusted producers, small trustworthy checkers — applies to its own metatheory, and this section is the application. Three evidence classes, fixed here and used in every mechanization remark of the chapter: *kernel-checked* means the named statements are theorems of this repository’s proof-artefact development; *prose-proven* means proven in adversarially reviewed prose in the development’s workshop record, not mechanized; *open obligations* are design with named obligations, stated in the future tense. In the claim-ledger vocabulary used across this book’s public surfaces, kernel-checked and prose-proven claims are [shipped] at their stated strata, open obligations are [argued], and this chapter contains no [aspirational] claims. Every kernel-checked claim lives in the toolkit’s proof-artefact repository, sorry-free at this writing, with the axiom census re-emitted at every build (each named theorem depends on at most propositional extensionality, choice, and quotient soundness; the forcing iff of [section 9.7](#) and the relational-transport core of [section 9.9](#) depend on no axioms at all). Two independent instruments keep the class checkable beyond the producing kernel: the repository’s `lean-cone` tool re-derives, for any named constant, a statements-only audit surface — the transitive signature cone, topologically sorted, elaborated against the real development — so the correspondence between this chapter’s prose and the tree’s statements is itself checkable without trusting either; and the compiled proof objects are re-checked by proof-checking kernels independent of the one that produced them (`SafeVerify`, and `nanoda`), so “kernel-checked” receipts are receipts from multiple kernels, not one.

**Kernel-checked.** By strand, one bullet per artefact layer, each named beside its statement in the body of the chapter:

- the double-categorical ambient: pseudo double categories with dependent cells and a cast calculus; lax double functors with the full coherence package; transformations and modifications assembling the functor double category; the loose bicategory bridge; companions, conjoints, and equipments; the span, matrix, and dependent-span models; the double Grothendieck construction at the dependent stratum, complete — total, strict projection, the split double opfibration development with canonical lifts and split laws as theorems, the discrete-opfibration interface, and the hom-level isomorphism between fiberwise transformations and strictly projection-compatible functors, choice-free; and the monoidal hypothesis package as decidable mixins over cyclic-group graded models, dependency table machine-checked;
- the wiring base in state-family presentation, with its reflexivity-valued transport calculus;
- the decoration tier, plain, site-indexed, and presentation-level: carriers, belief transport, the spec-class functor with all obligations at both strata, the coupled-DNF functor with atom-level transport, the atom-pairing compositor, and the option-A unit absorption, the assembled totals — the site-indexed class total and the authoritative `SingleSiteSemanticUniverse` — with strict projections and discrete double opfibrations by the generic theorems, the image-level comparison cells with their structure theorem, the constructible-fibre reading across the seam, the acceptance batteries (unit transparency, no funnel, associativity modulo base transport, ideal points monoidal, judgment as fibre order), and the refuting fixtures (taxed and funnel compositors, the unrepresentability pin, the matching fork);
- the record carrier: the Fell topology with hypothesis-free compactness; clip and proper-pushforward continuity with the seam fixture; descent as a homeomorphism onto the projective limit, with the nondirected and Vietoris sharpness fixtures; the record locus (single-timeline stratum) as a closed subsheaf with the speed-row and directedness fixtures and the Lipschitz-arc reading; the one-point compactification correspondence; the capacity presentation (basis, second countability, measure determination, separation, the frame form, and the two boundary fixtures); and class descent with constructibility in both directions;
- the universe and hyperspace storeys: carriers, membership closure, generation by rational rows, by tents, and by gamble rows at the general carrier, the pointwise order with its wall restriction and the tolerance-slice bridge, the order-level connective faces (conjunction's universal property; the functional-level minimum with its universal property and hedging witness), the coupled membership machinery with the blocking criterion (sound and complete, ex falso derived) and its incompleteness fixtures, the crescent and constructible-class calculus with the compile characterizations (the bijection with the credal bodies included), and the ideal-point/definability package;
- the transport stratum, at both the finite and the general carrier: the transport dictionary (body of a transported belief as body preimage), invariance of the coupled membership relation along surjections, the surjective structure theorem with its saturation condition, the non-surjective counterexample, the compositor's radial grade-product dictionary, and the union-distribution receipts;
- the disjunction stratum: the forcing iff (axiom-free), the countermodel schema, exactness and constructibility of the join, the disjunction property, the locally-closed-insufficiency witness, and the hedger fixtures separating the functional and class readings;
- the imprecise-probability strand of this repository: the belief-credal compile isomorphism (exact-rational, exp-free, with the concave-belief witness and the composition grade dictionary); the finite envelope-coherence package (attainment, super- and subadditivity, homogeneity, monotonicity); the additive law's domination and sublevel lemmas with their two fencing counterexamples, at the finite-rational stratum and space-free; the standard-Borel coupling-on-agreement theorem; the composition and tower laws of credal envelopes; the Hurwicz machinery with its composite evaluations; the belief-kernel tier at the exact stratum — tilted envelopes over signed costs with the credal zero-cost face and the one-member precise face, the positive-homogeneity failure witness, the finite Legendre transform of threshold profiles, the exact intertwining square with its two cost-law countermodels and the once-drawn diagonal's strict gap, the mixed-contraction identity with the strict exchange witness, wing collapse at prevision equivalence, and the Hurwicz chord decomposition with the certificate-pair rule and the finite-chords strictness witness; the coherent lower-prevision

- package with the finite envelope theorem and the natural extension's universal property, LP duality, and sure-loss alternative, with its four-mutant battery; the composition law's containment in the least-committal glue, with the crisp-locus iff and the strict rational witness; the feasibility (properness) equivalence at both strata with the exact conditional-product witness and its agreement-clause mutant; the leak-aware safe-side bound for submass models, envelope form and precise recovery, with the total-leak counterexample; and the credal-body poset of Remark 9.20 — placements, strict edges, and incomparability witnesses across the atlas's classes, the convexity adjudication trio (sufficiency, non-necessity, the boundary-jump closedness failure), and the prevision-ultracontribution identification;
- the generalization-column ledger items: the causal-diamond interior coverage (thickness iff and pair-filteredness at bicontinuous-poset generality), clock-map functoriality and cover-preservation with the homogenization core, and the open-quantum bounded-generator transport theorem with GKSL trace-annihilation and the Heisenberg trace adjunction;
  - the quantum strand of the causal-diamond column, at the finite-dimensional stratum throughout: the records-imply-consistency theorem at hypotheses weaker than projections, its additivity book-keeping, the load-bearing-hypothesis witnesses, the pure-state converse via branch projectors, a Minkowski instantiation, and the weakly-consistent-without-records witness; state-body gluing failure on the spacelike pair-cover with entanglement load-bearing (rigidity at pure marginals as the contrast), refusal of the failing cover by the derived topology, the general repair theorem (interior attainment implies separation), and the gap located at the linear rather than the multiplicative span; transport of diamonds, interiority, coverage, nets, separation, and determinacy along chronology-respecting order isomorphisms, with cover-preservation in the site-category vocabulary; the state presheaf with its separation and non-separation witnesses, the record shadow into probability rows with its strict credal band, and the unique-density-state representation theorem; Cauchy determinacy at both grades from complete initial data plus a reversible mechanism, its reversibility and exhaustiveness witnesses, and the frozen-shadow tie; the order-superposition probes (no records on a coherently switched index; pointer orthogonality classicalizes it); and the assembled locally covariant kinematic fragment with its two-qubit inhabitant;
  - the envelope reconciliation (ray versus singleton, with its boundary and refutation fixtures) and the seam isomorphism between the composition tier's and the judgment tier's carriers;
  - the cross-site stratum: the wall-conjunct criterion instances, the relational no-go kernel, the Egli-Milner lift with exact composition and the meets fixture, and the transport laws;
  - the site dimension in the base: the profiled base with its discrete fibredness over profiles, the profile-forgetting embedding (compositor invertible, unit comparison deliberately lax), the iterated total — Specs over interfaces over site-profiles — with its strict projection a discrete double opfibration, definitional fibre computation at the clocked instantiation, and the constant-profile equivalences of per-site conservativity;
  - the temporal-type-theory inclusion at the classical instantiation: the interval base site with the translation-quotient functor from the spec tier's windows (discrete-fibration lifts in both directions, covers corresponding to covers), the trajectory identification, the indicator calculus with behaviour-subtype spec objects, the judgment bridge (satisfaction as behaviour membership), strict tight intertwining, the compositor on indicators as span composition, and the semantically silent class-level laxity;
  - the certificate layer: the fibrewise-infimum transport dictionary at arbitrary carrier maps (safety adjunction, liveness transfer, the keystone factorization, transported model classes as preimages), unconditional transport monotonicity at both the semantic and the presentation order, the surjective dictionary re-derived as a one-line corollary, witness-gated composite certificates on the compositor, and the two fences (non-surjective ideal-point transport; presentation-sensitivity of the compositor);
  - the adequacy strand: the constraint double category, the inclusion library, the chain, Bayesian-network, and fixed-policy decision squares with mass theorems and library connectors, the demonic decision square with its set-valued bottom edge and the union-of-rays slice stratum, the delay-line square (delay differential equations as a wiring pattern, two-sided by the method of characteristics), the heat-on-torus square (the cover tier's cyclic mesh, with the holonomy wall and the frustrated-ring contrast) with its initial-face port (trace atoms, the conditional feed discharge, derived wrap-

compatibility), the solution-set mesh (exact equalizer descent consumed at the set stratum, the wall-invisible  $\mathbb{Z}$ -orbit holonomy witness), and the universe wiring (ideal points across the scalar seam, presented diagrams, the judgment bridge feeding the stochastic squares through the coupled judgment, and the singleton-versus-ray refutation);

- the checker anchor’s substrate: the certificate calculus’s valuation, satisfaction, and validity definitions with the per-rule soundness development, total over the catalogue (`ProofObject.sound` and `check_sound`, module `Qtsl.Soundness` of the `Qtsl` package), and the hypernet inclusion’s combinatorial core at the aligned stratum (open hypernets over the calculus’s own definitions, loose composition with well-formedness closure, the admissibility split, units and associativity on the nose, module `Qtsl.HypernetCospans`);
- the pointfree strand and the general stratum, landed as one campaign: valuations on frames with the promotion homeomorphism and the polarity package, the Borel bridge with  $\tau$ -additivity as its exact hypothesis class, bridge continuity into the pointfree carrier, and the free-ultrafilter escape witness; the classical subprobability carrier with the measure-level dagger chart, compactness by two independent routes, and metrizable; the general compile dictionary — closedness iff lower semicontinuity, the image characterization, unconditional reconstruction — with the general-carrier atlas placements, the grade-convexity criterion, the named order-closedness hypothesis of the finite-generation theorem, the Radon-functional tie at locally compact second-countable Borel carriers, and the general wall restriction and tolerance-slice bridge; subprobability Borel joints with the Giry-functor package, base-free coherence cores, the tight-tier general functor with its transposed battery and refutation pins, and the upward seam from the exact joints; the marginalization triple (escaping-mass witness, judgment-invisible reflector, tightness-certificate exactness) and the hull-closure package (closed binary hulls, levelwise coproduct exactness); the general-stratum composition containment with its crisp-locus characterization; the completed state-space category (all countable coproducts, disjoint and universal, with the bag space as the coproduct of its sorted-cone sectors); and the campaign’s continuation at the judgment and class tiers: the general connective faces, closure, and gamble-row generation (directed unconditionally; countable under second countability, discharged for compact metrizable record spaces), the general crescent and constructible stratum, the general transport dictionary with its named surjectivity and image-closedness supplies, the general demonic wall species with the cross-site criterion’s class exactness, and the general natural-extension package (envelope coherence over arbitrary nonempty families, the universal property at the definition’s stated scope with the dominating-family hypothesis self-certifying avoidance of sure loss, and closedness of the compatible set under continuous assessment gambles).

**Incorporated from the workshop.** A named minority of this chapter’s mechanized claims were, until their landing, *workshop-checked*: sorry-free kernel theorems whose modules lived in the development’s workshop tree pending incorporation, graded for a reader of this repository alone as adversarially reviewed prose with a mechanization attested. That set — the natural-extension package (universal property, linear-programming duality, sure-loss alternative, finite envelope theorem); the leak-aware safe-side package with its total-leak fixture; the containment theorem’s kernel form (crisp locus and strict rational witness); the packaged feasibility equivalence at both strata; and the site-column ledgers’ discharged items (interior-coverage pair-filteredness and the thickness equivalence at bicontinuous-poset generality, clock-map functoriality and cover-preservation, the homogenization core of time-inhomogeneity, and the open-quantum hypotheses’ transport theorem at the finite-dimensional stratum) — has been incorporated whole, as modules `QtslAux.Imprecise.NaturalExtension`, `QtslAux.Imprecise.Containment`, `QtslAux.Imprecise.FeasibilityGlue`, `QtslAux.LeakAware`, `QtslAux.DiamondSite`, and `QtslAux.OpenQuantum`, with axiom receipts emitted by the same build target as the rest of the package. As predicted at the staging, no claim of this chapter changed shape in the landing — only its evidence class; the workshop-checked class stands empty at this writing.

**Why the receipts are the doctrine: the phantom axiom.** The monoidal package’s original axioms do not imply the associator-interchange compatibility square: in graded models over an abelian group every coherence axiom linearizes, derivability becomes exact linear algebra, and the decision procedure exhibited a package satisfying every stated axiom, mediating cells invertible, whose associator-interchange square fails at explicit small integers. The repair is additive and costs every concrete instance nothing; the

audited dependency table is itself the exhibit — the new axiom is retained with its independence proven in both directions by machine witness, and a third candidate axiom turned out to be *derivable*: the audit removed an axiom thought necessary in the same pass that added the one that was missing. The lesson is drawn narrowly: in coherence axiomatics, a machine-decidable dependency table is the reliable form of the claim. That is why this chapter’s coherence claims cite kernel receipts, and why the disjunction stratum’s own repair went through the same door: the failure was found by countermodel, generalized into a kernel-checked schema, and only then written into prose.

**Prose-proven, adversarially reviewed.** The localic reading of the fibre column (the powerlocale and valuation-locale identifications, geometricity of the layer-by-layer presentation) — though no longer with the mechanized tier confined to its point-set shadow: the capacity presentation’s determination theorem and the valuation layer beneath it are kernel-checked at the localic stratum (Remark 9.13, Remark 9.36); the site-column mathematics not yet in the kernel (the causal-diamond column’s manifold-level instantiation of the clock-map hypotheses; the open-quantum column’s design hypotheses); the 2-categorical superstructure of the change-of-base layer; the span-of-categories generality of the Grothendieck construction (consumed by nothing in the shipped construction); and the classical agreement theorems’ prose legs at their stated finite-state, discrete-time scope; the collapse theorem for the certificate storey (fixed site, ordered fibres, the comma reading of the iterated construction), whose instantiation at the landed fibres is the kernel-checked certificate layer above; and the hypernet inclusion’s three theorems at full generality — the decorated-cospan double category of raw wiring diagrams, the well-formed genus as a sub-double category closed under composition, and the strict double-functorial inclusion via the possibilistic coupling — with their combinatorial core kernel-checked at the aligned stratum. An entity-level decoration functor — windowed System fibres, coupling categories, five prose-verified axioms — remains in the workshop record at this grade as the design basis of the certificate layer’s refinement stratum; the construction’s definition is the kernel-checked functor pair of section 9.9, and no claim of this chapter leans on the entity-level construction.

**Open obligations, with names.** The remaining gaps, so the reader need not hunt for them — and there are several, not one. The graded order’s checker obligation: certifying belief domination off the wall fragment (polynomial nonnegativity over the law polytope, or equivalently slice-containment rows with a tolerance field), the [argued] line of Proposition 9.53, with the wall fragment shipped. The site-indexed re-instantiation of the presentation fibres (the packaging step of section 9.9), and the two transport residues of Remark 9.49: closedness of the saturation locus, and non-surjective image constructibility. The cartesian half of the projection (preimage-transport lifts; the image-preimage adjunction on class fibres) and the two-sided form with the site as a visible index. The record-space tie and horizon-pasting for the adequacy squares (section 9.11). The Fell-carrier form of the window assembly (section 9.8): the extension calculus, its presheaf laws, both exchange theorems, and the persistence closure package — kernel-checked today over the function presentation of the trajectory carriers, graphs before closure under the product  $\sigma$ -algebra — restated over the record carrier itself, with Borel-on-Fell as the measurability datum and the carrier’s Fell-continuous clip as the restriction (the embedding being graph-then-closure, so that the presented calculus becomes the restated one’s image along the presentation); the clip’s own measurability is already a corollary of its landed continuity. The pair-sequent packaging of the blocking criterion. The  $\sigma$ -completion of the class fibre (section 9.7): attitude superlevels and threshold families as unions of crescents over declared index families, with transport carrying semicontinuity certificates for both envelope passes. The body-level representation of non-radial members (the classes quantify over bodies, which are radial objects; the graded universe’s two-axis members await their own body construction there). The hybrid (multivalued-instant) event strata — the shadow of the compact parameter class closed as a continuous image, and the alternation-bounded classes closed by hit-face arithmetic — plugging into the kernel-checked generic local-locus interface; should a clause family ever certify same-instant sequencing, the lens adjoins an event face by the same con-coupling that adjoined the measure face, and nothing below re-plumbs. The record locus over general causal supports (Definition 9.8) — where the single-timeline slicing argument does not bundle: over chronologically thick supports, two-point specimens refute closedness of the full record class (a spacelike stage-pair Fell-converging onto a null pair awakens a zero-duration constraint no approximant felt; a merging spacelike pair defeats single-valuedness), so what is owed is the factored replacement. Closedness of the interior-pair locus — speed bounds over pairs in the interior of comparability, which push-up identifies with chronology — by the same violating-pair margin argument, consuming two site hypotheses: chronology open in the stage topology, and



the two-stage duration upper semicontinuous on it (on a globally hyperbolic instantiation: openness of  $I^+$ , with the interval topology the manifold topology, and continuity of the Lorentzian distance). The plug of that locus into the kernel-checked local-locus interface, whose directedness step is unchanged. Closedness of the single-timeline-support sub-locus, where chain supports are Fell-stable because causality is closed. And the zero-duration-pair and single-valuedness conditions as certified  $G_\delta$  strata inside the closed locus, carried in the manner of the Zeno stratum rather than by the topology. The outer storey of the background-independent column (section 9.2): morphisms beyond isomorphisms and coverage for order-indexed sites — transport along chronology-respecting order-isomorphisms is kernel-checked above — together with the quantum column's named residues: relative Cauchy evolution and the stress-energy tensor, where the kinematic fragment fences off dynamics proper; non-surjective embeddings between different spacetimes, with functoriality of the assembled fragment along them; and the reading of the fragment's record shadow into the belief storey at a constant site-profile. The prevision-sheaf face of the fibre column and the fully typed display of the monoidal package's batched discharge rows, with the mechanical comparison identifying the Grothendieck-route coherences with the classical decorated-cospan components — and, for the hypernet inclusion of section 9.12, the full-generality rename transport and the double-category-library instantiation of its decoration functor. The logic-site contract's remaining transports: the analytic half of the extension-theorem audit (objectwise-classical, per window class), the probabilistic time-inhomogeneity lemma's projective-limit step, and the open-quantum event/record transport theorem. The HOL4 certified checker's end-to-end composition over the anchor's validity statement: demonstrated route, assembling now. And nesting — a System's entire state as one variable of another, named in section 9.9 — together with the library-grade generic form of the certificate storey's collapse (the ordered-fibre double Grothendieck with its comma isomorphism), whose instantiation at the landed fibres is kernel-checked above. Each entry above is stated in the future tense, deliberately, and nothing downstream in this book consumes any of them ahead of their receipts.



# Hypernets: Primitive Functions

DRAFT

Three strata of this book each need, at some point, a notion of *computation*. A System (chapter 12) declares mechanisms whose fields — drift laws, guard conditions, reset maps, neural controllers — are computational objects. A specification (chapter 9) asks whether observables computed from a trajectory stay inside a region, and a certificate about it carries polynomial content of its own. The rule catalogue (chapter 15) states every inference rule over “the ambient hypernet” — a typed computation graph that each rule reads facts out of. If those three strata each invented their own computational syntax, the toolkit would own three evaluators, three serialization schemes, and three subtly different answers to “is the function the certificate reasons about the function the system runs?”

It owns one. The shared object is the *hypernet*: a typed, exactly representable computation graph built from ten primitive boxes, with one evaluator, one canonical serialization, and one content digest. Everything in the toolkit that computes — an ODE right-hand side, a ReLU network, a guard observable, the goal polynomial of a Lyapunov certificate — either *is* a hypernet or elaborates into one by a definitional translation. This chapter defines the object once, so that the rest of the book can consume it by reference: the representation and its well-formedness discipline (section 10.1); the ten boxes themselves (section 10.2); the completion theorem that fixed their number at ten (section 10.3); the plate extension, under which one finite graph denotes a function of variable-dimensional state — symbolic extents, not an eleventh box (section 10.4); the single-genus design rule and the view/elaboration discipline it imposes on every other computational syntax (section 10.5); the substitution operation under which hypernets compose, with its coherence and digest laws (section 10.6); the precise sense in which Systems, specifications, and rules all consume the same genus (section 10.7); and the kinship between the graph-side contraction box and the relational engine’s contraction rows (section 10.8). A closing ledger (section 10.9) states which claims in this chapter are machine-checked, which are proven in prose with executable witnesses, and which remain in the design tense.

This chapter is also the toolkit’s *Primitive Catalogue*. Each of its thirteen rows — the ten boxes, the container, the substitution operation, and the well-formedness preflight — carries an anchored entry inside the narrative below, stating the row in compact metanotation and quoting its producer API, TypeScript then Rust. The metanotation statements are typeset by hand for notational quality; their mechanized ground truth is the catalogue’s per-box typing rules and well-formedness predicates, realized twice and in agreement (section 10.9). The API blocks are generated, never hand-quoted: signatures are re-extracted from the toolkit sources at every regeneration, and every construction snippet is quoted verbatim from the conformance corpus whose emitted canonical bytes are verified byte-identical against the reference implementation’s golden vectors.

A reading map, since the chapter doubles as reference material: on a first pass, read the representation (section 10.1), the table that opens section 10.2, the completion theorem of section 10.3 (a self-contained story), and the single-genus rule of section 10.5; return to the per-box catalogue entries as later chapters cite them, and leave the plate extension (section 10.4) until the token-population models of section 12.18 send you back for it. The mathematics is newest in the plate extension and in section 10.7, where the chapter’s compositional claims assemble into one machine-checked inclusion theorem (Theorem 10.6).

## 10.1 The intermediate representation

A hypernet is a dataflow graph over *tensor-typed wires*. A tensor type  $\langle s, \delta \rangle$  pairs a shape  $s$  (a finite list of extents; scalars have the empty shape  $[]$ ) with a dtype  $\delta$  — Boolean, integer, or exact rational  $\mathbb{Q}$  in the configurations this book uses. Exactness is a design commitment, not an optimization default: constants

enter as exact rationals,<sup>1</sup> the reference evaluator computes in  $\mathbb{Q}$ , and any change of numeric representation must pass through an explicit box with a declared rounding mode. There is no ambient floating-point semantics for a certificate to be quietly wrong about.

**Definition 10.1** (hypernet). A hypernet consists of: a *wire-type table* assigning each wire a tensor type; a pool of simplicial complexes (the payloads of the branching box, section 10.2.4); a finite sequence of *applications* of the ten primitive boxes of section 10.2, each consuming named input wires and producing named output wires; and an *interface*: an ordered list of graph inputs and an ordered list of graph outputs. It is *well-formed* when three conditions hold:

- **linearity** — every wire is produced by at most one application (graph inputs by none) and consumed by at most one;
- **acyclicity** — applications are listed in static-single-assignment order: every consumed wire is a graph input or the output of an earlier application;
- **type consistency** — each application satisfies its box’s typing rule (stated per box below).

All three conditions are decidable by direct inspection of the ground data, and both realizations of the IR — the Lean structure, which carries the three conditions as fields, and the reference implementation, which checks them — enforce them at the same strength.

### the well-formedness preflight

Primitive Catalogue entry H-WF

The row (Computer Science Metanotation)

$$\frac{\text{linearity}(H) \quad \text{acyclicity}(H) \quad \text{every application satisfies its box's typing rule}}{\text{wf}(H)} \text{WF}$$

All three conjuncts are decidable predicates over the ground data of Definition 10.1; the preflight operation decides them and returns the (possibly empty) list of violations, each carrying its geometry — the offending wire, application index, or typing fact. It is a producer-side courtesy: the checker replays the same conditions independently and believes only its own verdict.

## API

Producer surface of the mtk toolkit for this catalogue entry: the TypeScript arrangement (npm package @maitria/mtk, hypernet namespace) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this block; each construction is quoted verbatim from the toolkit’s hypernet conformance corpus, code whose emitted canonical graph bytes are verified byte-identical against golden vectors generated by the reference implementation (the corpus’s fixture graphs — an all-ten-primitives graph, an identity, a substitution composite, and the fifteen-application drift of this chapter’s worked example — are the receipts). This API constructs and emits; it never checks — the checker re-derives every claim independently. Later entries of this catalogue cite this note rather than repeating it.

Courtesy check: the returned list is every violation found, and empty means the linearity, SSA, and per-primitive typing conditions hold and every pooled complex validates — but the checker’s replay is the authority, never this preflight. The corpus construction validates the worked example’s fifteen-application drift graph.

**TypeScript** `hypernet.Hypernet.preflight`

signature — `ts/src/hypernet.ts`

| `preflight(): ReadonlyArray<Violation>`

construction — `ts/tests/hypernet-native.test.ts`

<sup>1</sup>An IEEE float read from a model file denotes a dyadic rational and is stored as one, losslessly; the ACAS Xu import of section 10.9 carries every published network weight this way.

```
const h = hn.unwrap(hn.Hypernet.fromCanonicalBytes(hypernetGolden("g3_cartpole_drift")));
assert.equal(h.preflight().length, 0);
assert.equal(h.appCount, 15);
```

**Rust** `mtk_catalogue::hypernet::Hypernet::preflight`

signature — `crates/mtk-catalogue/src/hypernet/wf.rs`

```
pub fn preflight(&self) -> Vec<Violation>
```

construction — `crates/mtk-catalogue/tests/golden_hypernet.rs`

```
let h = Hypernet::from_canonical_bytes(&golden("g3_cartpole_drift")).unwrap();
assert!(h.preflight().is_empty());
assert_eq!(h.apps.len(), 15);
```

Linearity is the load-bearing choice. In most graph formats, sharing a value is implicit (draw two edges out of a node) and discarding one is invisible (draw none). Here both are rows: fan-out is the copy box, fan-in is the stack box, discard is the delete box, and exchange is the swap box. The payoff is that structure becomes syntax — how many times a value is used, and where, is readable off the application list without any analysis pass, which is what lets the substitution operation of section 10.6 repair seams mechanically and lets well-formedness be preserved by a proof rather than a linter.

A well-formed hypernet is a *string diagram*: boxes are the typed, multi-input multi-output operations, wires carry the content, and two parallel wires between the same boxes are genuinely distinct. Combinatorially, string diagrams over a symmetric monoidal signature correspond to typed hypergraphs with interfaces, and the directed, Frobenius-free fragment is carved out by exactly two conditions — monogamy and acyclicity (Bonchi, Gadducci, Kissinger, Sobociński, and Zanasi [27]; the companion paper [26] treats the unrestricted setting) — which are the linearity and SSA conditions above under other names. “Hypernet” names this object with its interface and canonical serialization made first-class.<sup>2</sup>

Two near neighbours locate the design choice. Keeping monogamy makes an application list a term of the free symmetric monoidal syntax on the box signature, in which sharing, discarding, and ordering are rows. Dropping the consumer half of monogamy would give the *gs-monoidal* setting of term graphs, in which fan-out is absorbed into the wiring [84]; dropping the producer half as well would give hypergraph categories [26], in which a twice-produced wire asserts an equation rather than carrying a value. The calculus stays in the first setting because its consumers are certificates: how many times a value is used, and where, must be readable off the bytes without an analysis pass. The choice costs plumbing rows in the graphs. It does not cost composition: gluing two hypernets along a shared interface inserts no structural row at all, a fact section 10.6 states and the Lean tree checks.

Serialization is part of the definition’s job description, because certificates refer to computations by content. Every well-formed hypernet has a canonical byte serialization — wire ids assigned by first appearance, complex pool deduplicated, evaluator hints and debug names excluded — and its *digest* is the SHA-256 hash of those bytes. Two design consequences are worth stating before the boxes: equal digests certify equal computation graphs, not merely equal functions (two different graphs computing one function deliberately do *not* share a digest; extensional equality of programs is not decidable, and the digest never pretends otherwise); and every certificate row that cites a computation cites it by digest, so “which function is this claim about?” has a content-addressed answer.

## the container

Primitive Catalogue entry H-GRAPH

### The row (Computer Science Metanotation)

$$H = \langle W, \mathcal{K}, A, in, out \rangle \qquad \text{digest}(H) = \text{SHA-256}(\text{canon}(H))$$

The record of Definition 10.1: wire-type table  $W$ , complex pool  $\mathcal{K}$ , SSA application list  $A$ , interface  $(in, out)$ . The canonical serialization `canon` assigns wire ids by first appearance, deduplicates the pool, and excludes evaluator hints and debug names; equal digests certify equal graphs, never merely equal functions.

<sup>2</sup>The name comes from the graph-machine school that coined it [8, 89], whose hypernets are likewise monogamous directed hypergraphs. There the interface is read off vertex degrees; here it is declared data, and the canonical serialization is new.

## API

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue's first entry](#).

The trio quoted here is the container's content-address surface — strict decode, canonical bytes, SHA-256 digest; a graph is closed by `HypernetBuilder::finish` (TS `Builder.finish`), which runs the full preflight as a belt. The construction quoted below is the corpus's decode-re-encode-digest loop over the four committed reference vectors.

### TypeScript `hypernet.Hypernet`

signature — `ts/src/hypernet.ts`

```
static fromCanonicalBytes(bytes: Uint8Array): Result<Hypernet, CanonRefusal>

canonicalBytes(): Result<Uint8Array, CanonRefusal>

digest(): Result<HypernetDigest, CanonRefusal>
```

construction — `ts/tests/hypernet-native.test.ts`

```
const h = hn.unwrap(hn.Hypernet.fromCanonicalBytes(golden));
const re = hn.unwrap(h.canonicalBytes());
assert.equal(toHex(re), toHex(golden), "native re-encode");
assert.equal(hn.unwrap(h.digest()), manifest.fixtures[name]!.sha256, "native digest");
```

### Rust `mtk_catalogue::hypernet::Hypernet`

signature — `crates/mtk-catalogue/src/hypernet/canon.rs`

```
pub fn from_canonical_bytes(bytes: &[u8]) -> Result<Hypernet, CanonRefusal>

pub fn canonical_bytes(&self) -> Result<Vec<u8>, CanonRefusal>

pub fn digest(&self) -> Result<HypernetDigest, CanonRefusal>
```

construction — `crates/mtk-catalogue/tests/golden_hypernet.rs`

```
let bytes = golden(name);
let h = Hypernet::from_canonical_bytes(&bytes)
    .unwrap_or_else(|e| panic!("{name}: decode refused: {e}"));
let re = h.canonical_bytes().unwrap();
assert_eq!(re, bytes, "{name}: re-encode differs from golden bytes");
assert_eq!(
    h.digest().unwrap().to_hex(),
    manifest_digest(name),
    "{name}: digest differs from manifest"
);
```

## 10.2 The ten primitive boxes

The calculus has ten primitive boxes. Constructor order in the mechanized catalogue and tag order in the wire format agree (tags 0–9), and the order below follows them. Throughout,  $\langle s, \delta \rangle$  is a tensor type,  $\mathbb{B}$  abbreviates the Boolean scalar type  $\langle [], B \rangle$ , and  $k::s$  is the shape  $s$  with a new leading axis of extent  $k$ .

|   | box                    | signature                                                                                                 | one-line semantics                 |
|---|------------------------|-----------------------------------------------------------------------------------------------------------|------------------------------------|
| 0 | <code>const</code>     | $\emptyset \rightarrow \langle s, \delta \rangle$                                                         | emit a literal tensor              |
| 1 | <code>contract</code>  | $\langle s_1, \delta \rangle, \dots, \langle s_k, \delta \rangle \rightarrow \langle s_0, \delta \rangle$ | sum of products over bound indices |
| 2 | <code>cast</code>      | $\langle s, \delta \rangle \rightarrow \langle s, \delta' \rangle$                                        | change dtype, declared rounding    |
| 3 | <code>cell</code>      | $\langle [d], \mathbb{Q} \rangle \rightarrow \mathbb{B}, \dots, \mathbb{B}$                               | one-hot region membership          |
| 4 | <code>reshape</code>   | $\langle s, \delta \rangle \rightarrow \langle s', \delta \rangle$                                        | same entries, new shape            |
| 5 | <code>broadcast</code> | $\langle s, \delta \rangle \rightarrow \langle s', \delta \rangle$                                        | replicate along new axes           |
| 6 | <code>copy</code>      | $\langle s, \delta \rangle \rightarrow \langle s, \delta \rangle^k$                                       | explicit fan-out                   |
| 7 | <code>delete</code>    | $\langle s, \delta \rangle \rightarrow \emptyset$                                                         | explicit discard                   |
| 8 | <code>swap</code>      | $\tau_1, \tau_2 \rightarrow \tau_2, \tau_1$                                                               | exchange two wires                 |
| 9 | <code>stack</code>     | $\langle s, \delta \rangle^k \rightarrow \langle k::s, \delta \rangle$                                    | explicit fan-in: new leading axis  |

Nine of the ten are deterministic, total, single-valued operations. `cell` is the deliberate exception: *the calculus's only source of disjunction*. Every case split a hypernet can express — the two sides of a ReLU, the five advisory regions of a collision-avoidance policy — is some `cell` application's one-hot output, and every downstream consumer of a case split can be traced to it. A checker that must reason by cases knows, syntactically, where all the cases come from.

### 10.2.1 Sources and the multilinear engine: `const` and `contract`

`const` takes no inputs and produces one wire of a declared type, carrying a literal value: network weights, gain matrices, sign vectors, the all-ones vector that turns a `stack` into a sum (section 10.3). All numeric literals are exact rationals. Constants are applications like everything else — a hypernet's parameters are part of its graph, hence part of its digest.

**const**

Primitive Catalogue entry H-CONST

The row (Computer Science Metanotation)

$$\frac{v \text{ a literal of type } \langle s, \delta \rangle \text{ (row-major, size}(s) \text{ entries, exact)}}{\text{const}_v : \emptyset \rightarrow \langle s, \delta \rangle} \text{const}$$

Semantics:  $\text{out} = v$ . Tag 0; the only zero-input box.

## API

Producer API of the `mtk` toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue's first entry](#).

The exact-rational constant is quoted; the Boolean flavour (payload of bits) is constructed in the same corpus fixture, a few lines below the quoted one.

**TypeScript** `hypernet.Builder.constant`

signature — `ts/src/hypernet.ts`

```
constant(
  ty: TensorType,
  entries: ReadonlyArray<QLike> | ReadonlyArray<boolean>,
): Result<Wire, BuildRefusal>
```

construction — `ts/tests/hypernet-native.test.ts`

```
const c1 = hn.unwrap(b.constant(hn.rat([2]), ["3/2", "-7/3"]));
```

**Rust** `mtk_catalogue::hypernet::HypernetBuilder::constant`

signature — `crates/mtk-catalogue/src/hypernet/graph.rs`

```
pub fn constant(&mut self, ty: TensorType, value: ConstPayload) -> Result<Wire, BuildRefusal>
```

construction — crates/mtk-catalogue/tests/golden\_hypercent.rs

```
let c1 = b
  .constant(
    TensorType::rat([2]),
    ConstPayload::Rat(vec![q(3, 2), q(-7, 3)]),
  )
  .unwrap();
```

**contract** is the entire multilinear engine: the  $n$ -ary generalized Einstein summation, with its index structure carried as explicit data rather than inferred from a notation string.

**Definition 10.2** (contraction pattern). A contraction pattern for inputs of ranks  $r_1, \dots, r_k$  and output rank  $r_0$  consists of a finite index set  $I$ , an extent  $E_i \in \mathbb{N}$  for each  $i \in I$ , a binding  $\beta_m: \{0, \dots, r_m-1\} \rightarrow I$  for each input  $m$ , and a binding  $\beta_0: \{0, \dots, r_0-1\} \rightarrow I$  for the output; type consistency requires every bound axis's extent to equal its index's extent. Writing  $\nu\beta$  for the coordinate tuple obtained by composing an index assignment  $\nu \in \prod_{i \in I} \{0, \dots, E_i-1\}$  with a binding  $\beta$ , the semantics is

$$\text{out}[c] = \sum_{\nu: \nu\beta_0=c} \prod_{m=1}^k \text{in}_m[\nu\beta_m].$$

Matrix product is a contract ( $ai, ij \rightarrow aj$  in einsum notation); so are dot products, traces, elementwise products, tensor products, and axis sums. Two structural facts about this box do quiet work later. First, a rank-0 input has no axes to bind but still contributes its single entry as a factor to *every* term of the sum — scalars multiply in; they do not add in. Second, and in the same direction: every term of every output entry contains a factor from *every* input wire. **contract** is purely multiplicative across wires, and section 10.3 turns that observation into a theorem about what the box cannot do.

**contract**

Primitive Catalogue entry H-CONTRACT

The row (Computer Science Metanotation)

$$\frac{\beta = (I, E, \beta_0, (\beta_m)_{m=1}^k) \text{ a contraction pattern (Definition 10.2)} \\ \text{each bound axis's extent equals its index's extent}}{\text{contract}_\beta: \langle s_1, \delta \rangle, \dots, \langle s_k, \delta \rangle \rightarrow \langle s_0, \delta \rangle} \quad \text{contract}$$

Semantics: Definition 10.2's sum of products,  $\text{out}[c] = \sum_{\nu: \nu\beta_0=c} \prod_m \text{in}_m[\nu\beta_m]$ . Tag 1; the entire multilinear engine, index structure as explicit data.

## API

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue's first entry](#).

**TypeScript** `hypernet.Builder.contract`

signature — ts/src/hypernet.ts

```
contract(
  inputs: ReadonlyArray<Wire>,
  bindings: {
    extents: ReadonlyArray<number>;
    inBindings: ReadonlyArray<ReadonlyArray<number>>;
    outBinding: ReadonlyArray<number>;
  },
): Result<Wire, BuildRefusal>
```

construction — ts/tests/hypernet-native.test.ts



```
const d = hn.unwrap(
  b.contract([x, c1], { extents: [2], inBindings: [[0], [0]], outBinding: [] }),
);
```

**Rust** `mtk_catalogue::hypernet::HypernetBuilder::contract`

signature — `crates/mtk-catalogue/src/hypernet/graph.rs`

```
pub fn contract(
  &mut self,
  inputs: &[Wire],
  extents: &[u64],
  in_bindings: &[Vec<u32>],
  out_binding: &[u32],
) -> Result<Wire, BuildRefusal>
```

construction — `crates/mtk-catalogue/tests/golden_hypernet.rs`

```
let d = b
  .contract(&[x, c1], &[2], &[vec![0], vec![0]], &[])
  .unwrap();
```

## 10.2.2 Representation moves: cast, reshape, broadcast

**cast** converts dtype at fixed shape, carrying an explicit rounding mode. It is the only box at which numeric representation changes, which localizes every exactness argument: a hypernet with no cast (or only exact ones, such as the Boolean-to-rational conversion that turns an indicator into a gate) computes in  $\mathbb{Q}$  end to end, and one carrying rounding casts declares, in its syntax, where error can enter.

**cast**

Primitive Catalogue entry H-CAST

The row (Computer Science Metanotation)

$$\frac{\rho \text{ a declared rounding mode for } \delta \rightarrow \delta'}{\text{cast}_{\delta \rightarrow \delta', \rho} : \langle s, \delta \rangle \rightarrow \langle s, \delta' \rangle} \text{ cast}$$

Semantics: entrywise conversion under  $\rho$ ; exact casts (the Boolean-to-rational indicator gate among them) introduce no error. Tag 2; the only box at which numeric representation changes.

## API

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue's first entry](#).

**TypeScript** `hypernet.Builder.cast`

signature — `ts/src/hypernet.ts`

```
cast(inp: Wire, to: Dtype): Result<Wire, BuildRefusal>
```

construction — `ts/tests/hypernet-native.test.ts`

```
const ci = hn.unwrap(b.cast(bc, hn.INT));
```

**Rust** `mtk_catalogue::hypernet::HypernetBuilder::cast`

signature — `crates/mtk-catalogue/src/hypernet/graph.rs`

```
pub fn cast(&mut self, inp: Wire, to: Dtype) -> Result<Wire, BuildRefusal>
```

construction — `crates/mtk-catalogue/tests/golden_hypernet.rs`

```
let ci = b.cast(bc, Dtype::INT).unwrap();
```

**reshape** re-indexes a tensor’s entries into a new shape of the same total size — row-major order, no data motion in the semantics. **broadcast** replicates along new or singleton axes, under the trailing-axis compatibility rule familiar from array programming; the result is constant along each broadcast axis. Both are plumbing; they let contract patterns stay simple by normalizing operand shapes, and their semantics are entry-relabelling and entry-duplication, contributing no arithmetic of their own.

**reshape**

Primitive Catalogue entry H-RESHAPE

**The row (Computer Science Metanotation)**

$$\frac{\text{size}(s') = \text{size}(s)}{\text{reshape}_{s'} : \langle s, \delta \rangle \rightarrow \langle s', \delta \rangle} \text{ reshape}$$

Semantics: the same entries under row-major re-indexing; no data motion. Tag 4.

**API**

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue’s first entry](#).

**TypeScript** `hypernet.Builder.reshape`signature — `ts/src/hypernet.ts`

```
| reshape(inp: Wire, newShape: ReadonlyArray<number>): Result<Wire, BuildRefusal>
```

construction — `ts/tests/hypernet-native.test.ts`

```
| const r = hn.unwrap(b.reshape(t, [3, 1]));
```

**Rust** `mtk_catalogue::hypernet::HypernetBuilder::reshape`signature — `crates/mtk-catalogue/src/hypernet/graph.rs`

```
| pub fn reshape(&mut self, inp: Wire, new_shape: &[u64]) -> Result<Wire, BuildRefusal>
```

construction — `crates/mtk-catalogue/tests/golden_hypernet.rs`

```
| let r = b.reshape(t, &[3, 1]).unwrap();
```

**broadcast**

Primitive Catalogue entry H-BROADCAST

**The row (Computer Science Metanotation)**

$$\frac{s \leq s' \text{ (trailing-axis compatible)}}{\text{broadcast}_{s'} : \langle s, \delta \rangle \rightarrow \langle s', \delta \rangle} \text{ broadcast}$$

Semantics: replication along the new and singleton axes; the result is constant along each broadcast axis. Tag 5.

**API**

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue’s first entry](#).

**TypeScript** `hypernet.Builder.broadcast`signature — `ts/src/hypernet.ts`

```
| broadcast(inp: Wire, toShape: ReadonlyArray<number>): Result<Wire, BuildRefusal>
```

construction — ts/tests/hypernet-native.test.ts

```
| const bc = hn.unwrap(b.broadcast(r, [3, 2]));
```

**Rust** mtk\_catalogue::hypernet::HypernetBuilder::broadcast

signature — crates/mtk-catalogue/src/hypernet/graph.rs

```
| pub fn broadcast(&mut self, inp: Wire, to_shape: &[u64]) -> Result<Wire, BuildRefusal>
```

construction — crates/mtk-catalogue/tests/golden\_hypernet.rs

```
| let bc = b.broadcast(r, &[3, 2]).unwrap();
```

### 10.2.3 Structural plumbing: copy, delete, swap

**copy** duplicates one wire into  $k$  wires of the same type. Under linearity it is the only way a value acquires multiple consumers, so sharing is always a visible row — when the cartpole example of section 10.5 uses the pole-angle sine in three monomials, the graph says so with a single ternary copy.

**copy**

Primitive Catalogue entry H-COPY

The row (Computer Science Metanotation)

$$\frac{k \geq 1}{\text{copy}_k: \langle s, \delta \rangle \rightarrow \langle s, \delta \rangle^k} \text{ copy}$$

Semantics:  $\text{out}_i = \text{in}$  for every branch. Tag 6; under linearity, the only way a value acquires a second consumer.

## API

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue's first entry](#).

**TypeScript** hypernet.Builder.copy

signature — ts/src/hypernet.ts

```
| copy(inp: Wire, k: number): Result<ReadonlyArray<Wire>, BuildRefusal>
```

construction — ts/tests/hypernet-native.test.ts

```
| const ys = hn.unwrap(b.copy(y, 2));
```

**Rust** mtk\_catalogue::hypernet::HypernetBuilder::copy

signature — crates/mtk-catalogue/src/hypernet/graph.rs

```
| pub fn copy(&mut self, inp: Wire, k: u32) -> Result<Vec<Wire>, BuildRefusal>
```

construction — crates/mtk-catalogue/tests/golden\_hypernet.rs

```
| let ys = b.copy(y, 2).unwrap();
```

**delete** consumes one wire and produces nothing. A value nobody reads must be discarded by a row, never dropped by omission; the substitution operation of section 10.6 leans on this (“the discard is a row, never an absence”) when it repairs the seam between two graphs.

**delete**

Primitive Catalogue entry H-DELETE

**The row (Computer Science Metanotation)**

$$\frac{}{\text{delete}: \langle s, \delta \rangle \rightarrow \emptyset} \text{ delete}$$

Semantics: no outputs; the input's consumption is the entire content. Tag 7.

**API**

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue's first entry](#).

**TypeScript** `hypernet.Builder.delete`signature — `ts/src/hypernet.ts`

```
| delete(inp: Wire): Result<undefined, BuildRefusal>
```

construction — `ts/tests/hypernet-native.test.ts`

```
| hn.unwrap(b.delete(cells[0]));
```

**Rust** `mtk_catalogue::hypernet::HypernetBuilder::delete`signature — `crates/mtk-catalogue/src/hypernet/graph.rs`

```
| pub fn delete(&mut self, inp: Wire) -> Result<(), BuildRefusal>
```

construction — `crates/mtk-catalogue/tests/golden_hypernet.rs`

```
| b.delete(cells[0]).unwrap();
```

**swap** exchanges two wires, of possibly different types. It is the symmetry of the monoidal structure as an explicit application — pure renaming, no computation — so that wire order, like sharing and discarding, is syntax rather than convention.

**swap**

Primitive Catalogue entry H-SWAP

**The row (Computer Science Metanotation)**

$$\frac{}{\text{swap}: \tau_1, \tau_2 \rightarrow \tau_2, \tau_1} \text{ swap}$$

Semantics:  $(\text{out}_1, \text{out}_2) = (\text{in}_2, \text{in}_1)$  — pure renaming, types possibly distinct. Tag 8.**API**

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue's first entry](#).

**TypeScript** `hypernet.Builder.swap`signature — `ts/src/hypernet.ts`

```
| swap(in1: Wire, in2: Wire): Result<readonly [Wire, Wire], BuildRefusal>
```

construction — `ts/tests/hypernet-native.test.ts`

```
| const [s1, s2] = hn.unwrap(b.swap(d, ys[0]!));
```

**Rust** `mtk_catalogue::hypernet::HypernetBuilder::swap`

signature — `crates/mtk-catalogue/src/hypernet/graph.rs`

```
| pub fn swap(&mut self, in1: Wire, in2: Wire) -> Result<(Wire, Wire), BuildRefusal>
```

construction — `crates/mtk-catalogue/tests/golden_hypernet.rs`

```
| let (s1, s2) = b.swap(d, ys[0]).unwrap();
```

### 10.2.4 Branching: `cell`

**cell** is the case-analysis box. Its payload is a simplicial complex  $K$  in ambient dimension  $d$ : exact-rational vertex coordinates, a downward-closed and conformal list of simplices, and a list of *cells*, each a pair  $(S, F)$  of a simplex  $S$  and a proper subset  $F \subsetneq S$  of its vertices (the *extension set*). A point  $x \in \mathbb{Q}^d$  lies in the cell  $(S, F)$  when  $x$  is an affine combination  $x = \sum_{v \in S} \lambda_v v$  with  $\sum_v \lambda_v = 1$ ,  $\lambda_v \geq 0$  on the core vertices  $S \setminus F$ , and  $\lambda_v \leq 0$  on  $F$ . With  $F = \emptyset$  this is the closed simplex itself; each extension vertex opens the region outward through the facet opposite it. The sign split of the line — the instance that gates a ReLU below — takes the path complex with vertices  $-1, 0, 1$ : the cell pairing the edge  $[0, 1]$  with extension set  $\{1\}$  covers the nonpositive half-line, the cell pairing  $[-1, 0]$  with  $\{-1\}$  covers the nonnegative half-line, and the two overlap only at the origin.

The box’s typing: input one wire  $\langle [d], \mathbb{Q} \rangle$  — a point of the ambient space — and output one Boolean scalar per listed cell. Its semantics: the outputs are one-hot, with the hot output naming a cell that contains the input point. Where the point lies in several cells at once — on a shared boundary — *any* of the containing cells is an admissible choice. The hypernet’s semantics at such a point is the set of behaviours under all admissible choices; a rule or certificate that cites a `cell` output must hold under every one of them, and the reference evaluator’s tie-break (lowest cell index, documented as such) is a pinned representative, not a semantic commitment. Boundary nondeterminism is thereby owned by the proof obligations, not hidden in an evaluator convention.

One box in this role is a deliberate economy. A ReLU is copy, then `cell` over the sign split of the line (two cells: nonnegative and nonpositive), then a gating contract of the indicator against the value; an argmax-style advisory head is one `cell` over the fan of dominance cones in difference coordinates. Piecewise structure of any shape enters through this single door, and “where could this computation branch?” is answerable by listing its `cell` applications.

**cell**

Primitive Catalogue entry H-CELL

The row (Computer Science Metanotation)

$$\frac{K \text{ a valid simplicial complex in ambient dimension } d \quad \text{with cells } (S_0, F_0), \dots, (S_{n-1}, F_{n-1})}{\text{cell}_K : \langle [d], \mathbb{Q} \rangle \rightarrow \mathbb{B}, \dots, \mathbb{B} \quad (n \text{ outputs})} \text{ cell}$$

Semantics: the outputs are one-hot; a hot index  $j$  is admissible exactly when the input point lies in cell  $(S_j, F_j)$ , and on a shared boundary *every* containing cell is admissible — the nondeterminism belongs to the proof obligations, never to an evaluator convention. Tag 3; the calculus’s only source of disjunction.

## API

Producer API of the `mtk` toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue’s first entry](#).

**TypeScript** `hypernet.Builder.cell`

signature — `ts/src/hypernet.ts`

```

cell(complexIdx: number, inp: Wire): Result<ReadOnlyArray<Wire>, BuildRefusal>

export interface SimpC {
  readonly ambDim: number;
  /** Vertex coordinates, `ambDim` exact rationals each. */
  readonly vertices: ReadOnlyArray<ReadOnlyArray<QLike>>;
  readonly simplices: ReadOnlyArray<ReadOnlyArray<number>>;
  readonly cells: ReadOnlyArray<SimpCCell>;
  readonly hint?: string;
}

```

construction — ts/tests/hypernet-native.test.ts

```
const cells = hn.unwrap(b.cell(sigma, p));
```

**Rust** mtk\_catalogue::hypernet::HypernetBuilder::cell

signature — crates/mtk-catalogue/src/hypernet/graph.rs

```
pub fn cell(&mut self, complex_idx: u32, inp: Wire) -> Result<Vec<Wire>, BuildRefusal>
```

construction — crates/mtk-catalogue/tests/golden\_hypernet.rs

```
let cells = b.cell(sigma, p).unwrap();
```

### 10.2.5 Fan-in: stack

**stack** takes  $k$  wires of one common type  $\langle s, \delta \rangle$  and produces a single wire of type  $\langle k::s, \delta \rangle$ , with semantics

$$\text{out}[i, \text{idx}] = \text{in}_i[\text{idx}], \quad i = 0, \dots, k-1.$$

It is copy's dual — fan-in as explicit structure, exactly as fan-out is explicit copy — and its characteristic idiom is the *sum*: to add wires  $u$  and  $v$ , stack them and contract the stacked wire against the all-ones vector,

$$u + v = \text{contract}(\text{stack}(u, v), 1_2),$$

binding the new leading axis. Subtraction is the same idiom with a  $[1, -1]$  sign vector, which keeps imported model weights byte-faithful: negation lives in a separate constant, never in a rewritten payload. **stack** adds no disjunction (its facts are entry equalities, structural like copy's), and it is the newest member of the calculus — the next section is the story of why it had to exist.

**stack**

Primitive Catalogue entry H-STACK

The row (Computer Science Metanotation)

$$\frac{k \geq 1 \quad \text{all inputs of one type } \langle s, \delta \rangle}{\text{stack}_k : \langle s, \delta \rangle^k \rightarrow \langle k::s, \delta \rangle} \text{ stack}$$

Semantics:  $\text{out}[i, \text{idx}] = \text{in}_i[\text{idx}]$ . Tag 9; copy's dual, and the  $\Sigma\Pi$  boundary's visible seam.

## API

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue's first entry](#).

**TypeScript** hypernet.Builder.stack

signature — ts/src/hypernet.ts

```
stack(inputs: ReadOnlyArray<Wire>): Result<Wire, BuildRefusal>
```

construction — ts/tests/hypernet-native.test.ts



```
| const t = hn.unwrap(b.stack([s1, s2, ys[1!]!]));
```

**Rust** `mtk_catalogue::hypernet::HypernetBuilder::stack`

signature — `crates/mtk-catalogue/src/hypernet/graph.rs`

```
| pub fn stack(&mut self, inputs: &[Wire]) -> Result<Wire, BuildRefusal>
```

construction — `crates/mtk-catalogue/tests/golden_hypernet.rs`

```
| let t = b.stack(&[s1, s2, ys[1]]).unwrap();
```

## 10.3 Why ten: a completion theorem

The calculus was not designed to have ten boxes. It had nine — `stack` was absent — until compiling real networks forced the question of what the nine could not say, and the answer turned out to be a theorem rather than an inconvenience.

**Proposition 10.3** (the packing gap). *In the nine-box fragment (without `stack`), no wire can carry the sum of two independently produced wires: for any two wires  $u, v$ , no later wire has an entry whose expansion — read with  $u$  and  $v$  as opaque sources, whatever produced them — contains both a monomial with  $u$ -factors but no  $v$ -factors and a monomial with  $v$ -factors but no  $u$ -factors; in particular, no entry expands to  $u + v$  itself.<sup>3</sup>*

*Proof sketch.* Only `contract` combines values from more than one data wire — `cast`, `cell`, `reshape`, `broadcast`, `copy`, and `delete` are single-input, `const` is zero-input, and `swap` is a renaming. But every output entry of a `contract` is a sum of monomials each containing a factor from *every* input wire (Definition 10.2: the product runs over all  $m$ ) — the box is purely multiplicative across wires, and a constant input multiplies into every term rather than adding. So the first application to consume both  $u$  and  $v$  already violates the required shape, and single-input boxes only relabel or duplicate entries thereafter; induction along the SSA order closes the argument.  $\square$

The practical blast radius of this small proposition was total: no bias can be added to a linear layer ( $x - 1$  is already inexpressible), and after a ReLU layer splits into per-scalar gate indicators, nothing can reassemble the gated values into the next layer’s input — so no network with two or more neurons could be expressed at all, even though each individual matrix product and each individual ReLU could. Of the two sub-gaps, one (additive constants) admitted a workaround — an augmented-input convention carrying a constant-one component — at the price of an interface wart; the other (cross-wire packing of independently produced values) admitted none.

The repair is the tenth box. `stack` closes both gaps at once: sums become the `stack-and-ones` idiom of section 10.2.5, gate vectors assemble from per-scalar `cell` outputs, and with it every piecewise-polynomial dataflow — every ReLU network included — becomes expressible. Alternatives were weighed and declined: a vectorized `cell` (fixes gates, not bias, and complicates the format); a polynomial node promoted to primitive rank (collapses the `einsum` discipline — section 10.5 explains why the polynomial belongs at the view layer instead); a `contract` variant that also sums over a choice index (semantically equivalent to `stack-then-contract`, but it muddies the `einsum` reading, whereas `stack` keeps *product* and *sum* orthogonal). `stack` entered the calculus the same day the gap was written down.

Two properties of the repair deserve the emphasis of a design principle. First, `stack` introduces no new disjunction — the invariant that `cell` is the calculus’s only branching source survives the extension untouched. Second, the division of labour it induces is legible in compiled graphs: products form at `contract` boxes, cross-wire sums form at `stack-plus-ones` boxes, and the boundary between the  $\Sigma$  and the  $\Pi$  of a polynomial is a visible seam in the graph. The worked example of the next section shows that seam carrying exactly one `stack`.

What the ten boxes still deliberately cannot say, so that no reader over-extrapolates the completion: there is no transcendental function box, and there is no division box. An oscillator enters through the standard cylinder lift — carry  $s = \sin \theta$  and  $c = \cos \theta$  as state variables with the algebraic invariant  $s^2 + c^2 = 1$  —

<sup>3</sup>The expansion stratum, not run-time values, is where the claim holds at this generality: two `copy` descendants of one wire are produced by distinct applications, yet twice their common value is one `contract` against a constant. Read over expansions with  $u$  and  $v$  held opaque, the statement needs no independence hypothesis at all, and that is the form the mechanization proves (section 10.9).

rather than through a sin node; a rational right-hand side clears its mass matrix into implicit polynomial form. Both exclusions keep the genus inside piecewise-polynomial dataflow over  $\mathbb{Q}$ , which is where exact certification machinery has traction. The exclusion is a fact about boxes, not about reach: the admissible-symbol discipline of section 10.5 extends to a register of *certified function aliases* through which sine and cosine, rational powers, logarithm, and exponential become available anyway — as piecewise-polynomial approximations with declared domains and accuracy budgets, receipt-checked in exact rational arithmetic — without the wire format ever learning their names. Should a modelling frame ever genuinely force a quotient node, the procedural path is the one stack took: a written inexpressibility finding, an explicit decision, and enactment at the calculus layer — never a side-channel evaluator bolted onto a view.

## 10.4 Plates: symbolic extents

Token-population systems — coloured Petri nets first among them (section 12.18) — have variable-dimensional state: the number of tokens changes as the system runs, and is canonically unbounded across a run even though it is finite at every instant. A certificate about such a system is a function on markings — multisets of token colours — and a naive syntax would need one computation graph per token count: a countable family of artefacts where one certificate was wanted. The calculus closes this gap with *plates*: symbolic extents, not an eleventh box.

**The data.** Two additions to Definition 10.1, both container-level. The container gains an ordered finite list of *plate variables*  $\iota_1, \dots, \iota_r$  (serialized de Bruijn by position; debug names excluded from canon, like wire names). And the extent alphabet in wire shapes and contraction-pattern index extents becomes  $E ::= n \mid \iota$  — a literal or a declared plate variable, with no extent arithmetic ( $\iota + 1$  is not an extent; products of plates arise as multiple axes on one wire, never as extent products). A *plated hypernet* is the same record over the extended alphabet plus the plate list; a plated hypernet with zero plates is a hypernet, byte for byte. For an extent assignment  $\sigma : \{\iota_1, \dots, \iota_r\} \rightarrow \mathbb{N}$ , *instantiation*  $\text{inst}_\sigma$  replaces every symbolic extent by its value, yielding an ordinary hypernet; everything downstream — evaluator, canonical serialization, the whole checker stack — consumes instances and only instances. The plate is marked on *wires* (a per-token scalar is a wire of shape  $[\iota]$ ; a token family of  $d$ -dimensional colours,  $[\iota, d]$ ; an attention matrix,  $[\iota, \iota]$ ), never as a membrane around a subgraph: scope would be degenerate — plate variables are container-level — and what a membrane would add beyond axes is per-index variation of the body, which is exactly what the equivariance theorem below must forbid.

**How the boxes lift.** One principle, ten instances: a box lifts along plate axes by whiskering — the lifted application is the original box applied per index tuple, well-typed exactly when the box’s typing clause holds over the extended alphabet, with extent equality read *syntactically* (literal against literal numerically;  $\iota$  against  $\iota$  by name; a  $\iota$ -against-literal match is refused — conservative, never unsound, since schematic typing must hold at every  $\sigma$ ). *contract* admits symbolic index extents (a bound symbolic index is a pool over the instance’s extent; a free one maps over it); *broadcast* may add or extend plate axes under the trailing-axis rule, the membrane-crossing law in wire clothing (a plate-free wire broadcast along  $\iota$  is constant along the index); *cast*, *copy*, *delete*, *swap* are pointwise; *stack* keeps literal arity (a plate-sized family is never  $\iota$  separate wires — it is one wire carrying the  $\iota$  axis, from birth); *reshape* is refused on symbolic-extent wires in this edition (symbolic *totalSize* equality is decidable polynomial identity, but *contract* carries the demand natively, so nothing routes through it); and *const* — the load-bearing refusal — takes all-literal shapes only, forced rather than chosen: a literal tensor has finitely many entries.

**Three theorems, nearly free from one syntactic fact.** The language has no token literals: no expression for “the  $i$ -th token,” no order or successor on the index sort, no per-index branching (a *cell* reads colours, never indices), and *const* cannot utter a plate. Three consequences make the notation functorial rather than decorative.

- **Schematic well-formedness.** A well-formed plated hypernet instantiates to a well-formed ordinary hypernet at *every* extent assignment: instantiation is a homomorphism on extents, every typing clause is syntactic extent equality plus extent-free conditions, and both survive substitution.

- **Equivariance for free.** The instance family is natural in bijections of each plate’s index set: every expressible plated computation is equivariant under the product of symmetric groups, *as a theorem of the grammar* rather than a proof obligation on artefacts — plate axes are born only from the interface, broadcast, and contract, all equivariant, and the mutant that adds an index-literal box breaks the theorem at a two-token transposition, kernel-visibly. A corollary worth the emphasis: *causal masking is unwritable*. A causal mask is a function of the index order, and the syntax has no order to consult — the transformer this calculus can express is inherently the exchangeable one.
- **Descent.** An output wire carrying no plate axis is invariant, hence descends to the multiset quotient: the family assembles into one function on the bag space  $\bigsqcup_{n < \omega} \text{Sym}^n(B)$  — with one plate per place, precisely the coloured-net marking space of section 12.18. One finite syntax object denotes a function of the whole variable-dimensional state.

Semantically the plate is a functorial box in the sense of Melliès [178] — the strong monoidal functor  $(-)^{\otimes I}$ , drawn as axes instead of a membrane — and at infinite index it is the plate over a Kolmogorov product in the sense of categorical probability [43], the unique morphism whose finite marginals are the body’s instances. The index category is deliberately the groupoid of finite sets and bijections, not all functions: aggregation is not functorial under instance insertions [222], and equivariance is exactly the functoriality that survives — which is the symmetry the marking space quotients by.

**The count wire.** The const refusal bans  $\iota$  in a literal’s *shape*, never the scalar. So the token count is three applications — const 1, broadcast to  $[\iota]$  (the vacuous instance of the trailing-axis rule), contract binding the index — and each instance evaluates to  $n$ . Uncoloured counter places need nothing else: classical place-invariant certificates are degree-one polynomials in count wires, and count-bound rows ( $0 \leq k_P \leq M_P$ , linear conservation laws) attach to this wire. The idiom is a natural packaged symbol for the register of section 10.5 — expansion as above, projection lemma trivial — so certificates may cite one symbol while the wire format never learns it exists.

**The certificate stratum never sees a plate.** A plated hypernet with invariant outputs, built from contract pools, is a presentation of an aggregate polynomial in finitely many *pooled coordinates*: token counts  $\vec{k}$  and per-place power-sum pools  $\vec{s}$  (a sum-pool is a contraction row whose bound index is the plate — the relational rendering of section 10.8 applies unchanged, side conditions included). Obligations about all markings at once are ordinary polynomial sequents over  $(\vec{k}, \vec{s})$  with hypothesis rows — coupling enclosures  $l_j k_P \leq s_j^P \leq u_j k_P$  from the colour boxes, count rows, integer rounding at count atoms — so obligation count and dimension are set by the net and the feature dictionary, never by any marking bound. The plate joins “the checker never sees a probability” and “never sees a marking”: it compiles away through the pooled normal form, and what reaches the kernel is hypothesis-carrying polynomial rows of the standard shape. How the per-transition rows of a token-population system collapse to one row per Petri transition — and which pooled aggregators can carry progress claims, not merely safety — is taken up with the modelling frame itself (section 12.18).

**Worked size.** A one-layer transformer encoder with degree-2 polynomial attention — a published, empirically competitive attention class [159], of which the plate-expressible fragment is precisely the exchangeable subclass — over a two-dimensional colour box, with residual connection, invariant quadratic readout, and count wire, is *nineteen applications*, of which the count wire is three; its pooled normal form is a per-token quadratic form against pooled third moments, an aggregate polynomial over fifteen pool coordinates plus the count. An executable exact-rational fixture pins the transcription against a per-token symbolic computation at  $n \in \{1, 2, 3\}$ , the collapse identity symbolically, and invariance under all  $n!$  permutations.

**Status.** The type layer and the three theorems are mechanized in the Lean catalogue (lean/qts1, modules Qts1/Plate/{Syntax, TPlate, Eval, TNat, TDesc, Probes}): symbolic shapes and their instantiation homomorphism, the lifted per-box typing table with the schematic well-formedness theorem across all ten boxes, a reference evaluator with the diagonal-action equivariance theorem, the unwritability of causal masking (stated as a refutation: the one-application index-literal extension falsifies equivariance at  $n = 2$ ), descent to Multiset-indexed denotations, and a symbolic count-wire evaluation theorem, under a three-mutant battery at preregistered loci. The certificate-stratum row family this section’s pooled normal

form calls for — plate declarations, per-token feature dictionaries, aggregate wires as contraction rows, fold certificates with per-transition footprint rows, and the idempotent-gap refusal — is landed theory vocabulary, and the producer emits it: the semiconductor tutorial (chapter 17, section 17.5) is its first exercising campaign, and the engine-side evaluator family for the theory's demanded side conditions now discharges them in that campaign's production loop, decided at the coloured place/transition fragment with the remainder refused by name (section 17.5.8). The hypernet *wire-format* extent tag itself — symbolic extents on general wires, beyond the certificate rows — remains design with its pieces priced; plate-free serialization is unaffected by construction (zero-plate graphs are byte-identical today).

## 10.5 One genus: views elaborate in

A toolkit that certifies computations cannot afford two kinds of computation. The design rule entered the development log as a maxim the day the System layer adopted hypernets for its dynamical content: hypernets are to be re-used for differential-equation specifications and their kin. The maxim had an edge because the project had already lived through the failure once, when terms grew into a separate genus from hypernets and were soon demanding einsum machinery of their own.

The rule this chapter takes from that entry, stated positively:

**The single-genus rule.** Hypernets are the one genus of computational content. Every other computational syntax — a polynomial vector field written as a coefficient table, a term inside a proof obligation, a guard expression, a PDE operator — is a *view*: surface syntax equipped with a definitional elaboration into the ten boxes. A view's meaning *is* the value of its elaboration; no view acquires its own evaluator, its own serialization, or its own contraction machinery.

The precedent for the pattern is einsum itself. Einsum notation is not a second contraction engine anywhere in the toolkit; it lowers to explicit contraction structure (patterns here; contraction rows in the relational engine, section 10.8), and the summation it denotes is definitional — evaluated only where a checker needs values. Views generalize this: convenient syntax on the outside, one calculus on the inside, and the translation is part of the definition rather than an implementation detail.

The disease the rule prevents has a concrete signature, worth naming because it is how single-genus erodes in practice: a producer grows an ergonomic node — say, a polynomial represented as its monomial table — that is not among the ten boxes; artefacts begin to carry it *instead of* applications; and some downstream consumer, finding no applications to read, grows an evaluator against the ergonomic node directly. At that moment there are two genera, two evaluators, and a fresh supply of questions about whether they agree. The repair — performed on the polynomial node itself, and now the standing template — is *demotion to view* under the admissible-symbol discipline:

- an **admissible packaged symbol** is a named node with a typing rule, admitted alongside (never into) the core ten;
- it carries a **definitional expansion** into core boxes — the node *serializes as* its expansion, so the wire format and the digest scheme never learn the symbol exists;
- a once-proven **projection lemma** establishes that the symbol's declared facts follow from its expansion's facts, so rules may cite the symbol soundly;
- and a **candidacy rule** bounds the admission: the expansion's disjunctive content must come only from `cell` (most packaged symbols are disjunction-free; a packaged ReLU inherits its single case split from the `cell` inside its expansion, and nothing else is allowed to smuggle new branching into the calculus).

Four packaged symbols currently ride this register, each with its expansion and projection lemma mechanized: `relu` (expansion: copy, sign-split `cell`, gating contract); `ppoly`, the monomial-table polynomial view whose expansion is the scheme of the worked example below; `pderiv`, the formal partial derivative on dense coefficient tables (expansion: one constant shift-and-scale matrix and one contract); and `facet`, restriction of a coefficient table to a face (expansion: one basis-vector constant and one contract). None is an eleventh primitive; all four erase at serialization time.

Three audit questions probe any proposed addition, and the calculus answers all three by construction: *a second evaluator?* — no; views elaborate, and the one exact evaluator computes. *A second digest scheme?* — no; the canonical serialization hashes expansions, so a view and its elaboration cannot drift apart behind distinct content addresses. *A second contraction path?* — no; the contraction pattern is the only index-binding construct, and any syntax “asking for its own einsum” is asking, by definition, to violate the rule.

## Beyond exactness: certified function aliases

One register remains, opened by a question the packaged symbols cannot answer. Modelling frames keep asking for  $\sin$ ,  $\cos$ ,  $t^{p/q}$ ,  $\ln$ , and  $\exp$  — in drift coefficients, rate laws, guard surfaces — and no piecewise polynomial *equals* any of them, so no definitional expansion of the packaged-symbol kind can exist. What can exist, and what exact numerics has shipped for decades under the name of certified function libraries [185], is an approximation with a proof: a piecewise-polynomial table  $\hat{f}$  over exact rationals, a declared domain  $[a, b]$ , and a declared budget  $\varepsilon$ , together with a receipt — exact-rational inequalities a small checker re-derives from the table bytes alone — establishing  $|\hat{f} - f| \leq \varepsilon$  on the domain. Remainder brackets of the Maclaurin kind make such receipts pure rational arithmetic (the checker compares coefficients against a factorial tail bound; it never needs to know that  $e$  exists), and implicit relations extend the same discipline to functions the brackets do not reach directly: a logarithm table is certified through the residual  $|\exp(\hat{w}(t)) - t| \leq \eta t$  against exponential brackets, a rational-power table through the purely polynomial residual  $|\hat{w}^q - t^p| \leq \eta$  — so no logarithm and no fractional power is ever enclosed by anything but polynomial comparison.

A *certified function alias* packages exactly this: a name and pinned parameters selecting one table, hence one hypernet (the pieces enter through `cell`, the piece polynomials through the `poly` scheme, the one-hot blend through `contract`); the budget  $(\varepsilon; [a, b])$ ; and the receipt's citation. The alias is admitted alongside the calculus in the same sense as the packaged symbols — the wire format and the digest scheme never learn its name; artefacts carry the expansion — and the three audit questions above pass by the same erasure argument. What changes is the epistemic contract: a packaged symbol's projection lemma says *equal*; an alias's receipt says *within  $\varepsilon$ , on  $[a, b]$*  — and the domain is a precondition, owed a containment argument by any consumer whose wires might leave it, the same obligation shape as a jump's reset containment. Downstream, the distinction costs nothing; a drift built from an alias *is* piecewise-polynomial dataflow, its proof obligations are polynomial rows like any other, and the transcendental fact lives entirely in the receipt. A claim over such a system may then be read at either of two strengths: against the table as the model itself, with  $\varepsilon$  a declared modelling tolerance of the same kind as float rounding (the reading under which nothing downstream changes at all); or against the band  $\hat{f} \pm \varepsilon$  as a differential inclusion, whose verdict covers the true function by containment at the price of one rational widening term per obligation row.

The family is closed under differentiation —  $\sin' = \cos$ ,  $\exp' = \exp$ ,  $\ln' = t^{-1}$ ,  $(t^{p/q})' = \frac{p}{q} t^{p/q-1}$  — so an alias's derivative data is a pointer to a sibling alias, never new machinery. The cylinder lift met earlier in this chapter remains the better encoding where it applies (an autonomous oscillator's sine and cosine carried as state, exactly, with their algebraic invariant); the alias route serves the cases the lift cannot reach — forced and parametric arguments, logarithms and powers of state variables, rate laws. The toolkit carries certified tables for the whole family named above. The exponential, the logarithm, and the rational powers arrived first, as quantile transforms for its distribution generators, verified by the residual disciplines above; sine and cosine followed as forward tables on a declared symmetric domain, their error bounded absolutely against the same factorial-tail brackets the remainder arguments above consume — absolutely of necessity, since both functions vanish inside every interesting domain, leaving a relative bound no positive floor to stand on. One more fact about the trigonometric pair is worth the ink:  $\pi$  is irrational, so periodic folding is not exactly available in  $\mathbb{Q}$ , and a range-reduced table would smuggle an approximation of  $\pi$  into its own argument, outside the certificate. The declared domain is therefore data; coverage beyond it is refused, never clamped, and range reduction is the consumer's burden, priced where the consumer knows its own modulus.



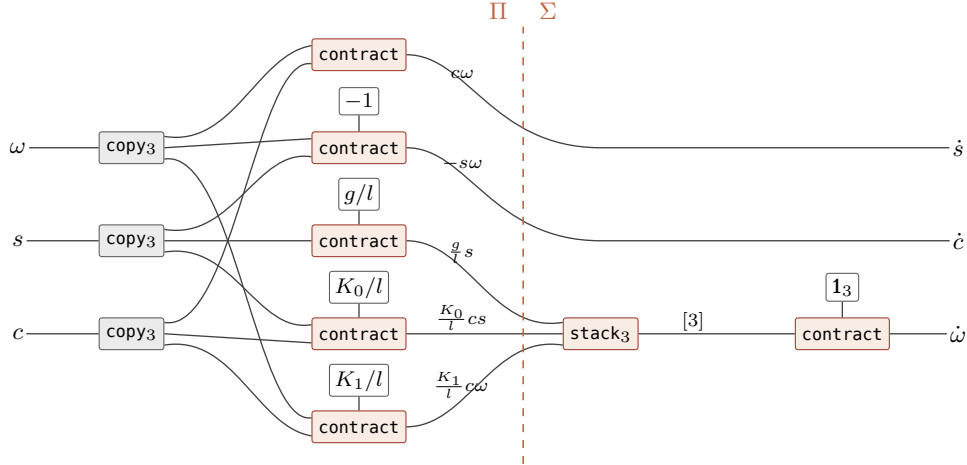


Figure 10.1: The closed-loop cartpole drift of the worked example, depicted as a hypernet: fifteen applications, three inputs, three outputs. Structural plumbing (copy, grey) makes every act of sharing a visible row; constants (white) are applications like everything else; every monomial is one contract (peach); and the calculus’s single cross-wire sum — the three-term right-hand side of  $\dot{\omega}$  — crosses the  $\Sigma\Pi$  boundary (dashed) as one stack contracted against the all-ones vector.

### Worked example: a closed-loop drift, in fifteen applications

The example that carried the design discussion is small enough to print whole. Take the cartpole’s closed loop on the cylinder lift  $(s, c, \omega)$  — sine and cosine of the pole angle carried as state, per the no-transcendentals discipline — with a linear feedback law  $a(s, \omega) = K_0 s + K_1 \omega$  folded in, plant half-length  $l = \frac{1}{2}$ , gravity  $g = \frac{49}{5}$ , and exact rational gains  $K_0 = -\frac{161753}{8148}$ ,  $K_1 = -\frac{44746}{9799}$ :

$$\dot{s} = c\omega, \quad \dot{c} = -s\omega, \quad \dot{\omega} = \frac{g}{l}s + \frac{K_0}{l}cs + \frac{K_1}{l}c\omega.$$

Its hypernet — three input wires, three output wires, every scalar a rank-0 exact rational — is fifteen applications:

| stage                    | applications                                                                          | count |
|--------------------------|---------------------------------------------------------------------------------------|-------|
| fan-out                  | copy each of $s, c, \omega$ into its uses                                             | 3     |
| constants                | const $-1, \frac{g}{l}, \frac{K_0}{l}, \frac{K_1}{l}, 1_3$                            | 5     |
| monomials                | contract: $c\omega; (-1)s\omega; \frac{g}{l}s; \frac{K_0}{l}cs; \frac{K_1}{l}c\omega$ | 5     |
| the $\Sigma\Pi$ boundary | stack the three $\dot{\omega}$ monomials into shape $[3]$                             | 1     |
| the sum                  | contract against $1_3$ , binding the $[3]$ axis                                       | 1     |

Every monomial is contract-native; the single cross-wire sum in the system — the three-term right-hand side of  $\dot{\omega}$  — is the single stack (Figure 10.1). The table is not hand-lore: the polynomial view elaborator emits this graph, its fifteen applications digest to a pinned golden (69b6f4c6...), and the elaborator’s output is already in canonical form, so the digest doubles as a regression witness for the elaboration itself. Neither is the figure hand-lore: it is emitted, deterministically, from those same canonical bytes by the depiction tool beside the producer surface (hypernet-tikz), so the pinned digest is the drawing’s witness too — only the semantic names ( $s, c, \omega, g/l$ ) arrive as declared labels, since the canonical form deliberately carries no name metadata. The same scheme compiles the goal polynomial of the matching Lyapunov certificate — one more stack, one more ones-contract — which is the single-genus rule doing exactly its job: system dynamics and certificate content share one representation, so one recomputation can cross-check a digest cited by both.

## 10.6 Substitution: how hypernets compose

Hypernets compose by *substitution*: plugging one graph’s outputs into another’s inputs. The operation is defined once, at the graph layer, and everything that composes computations — nesting a component’s



dynamics inside a larger mechanism, elaborating a view in context, merging two boxed computations into one — consumes the same definition.

The data of a substitution: an outer hypernet, an inner hypernet, a *port map*  $\varphi$  matching some of the outer graph’s inputs to inner outputs (type-exact; several outer inputs may consume one inner output), and an *exposure set* of inner outputs to remain outputs of the composite — by default the unmatched ones, so nothing is silently dropped and nothing matched is silently re-exposed. The construction concatenates the inner application list, a block of *seam repairs*, and the redirected outer list, then rennumbers into canonical form. The seam repairs are where the linearity discipline pays: an inner output consumed by  $n$  ports and exposures gains exactly one consumer at the seam — a *delete* when  $n = 0$  (the discard is a row, never an absence), nothing when  $n = 1$ , a *copy* when  $n \geq 2$  — and consumer counts change nowhere else.

**Theorem 10.4** (substitution laws). *For well-formed hypernets and valid port structures: (i) the composite is well-formed; (ii) on the deterministic fragment, evaluation of the composite equals the composition of evaluations — inner outputs feeding matched outer inputs, unmatched interface passing through; (iii) substitution is unital (substituting an identity hypernet, in either direction, returns the other graph up to canonical renumbering) and associative — and both laws hold on the nose for the canonical forms, hence as equalities of digests, not merely up to isomorphism.*

The proofs are elementary but not careless; the one choice that genuinely matters is *where the repair blocks live*. Placing each seam’s repairs at that seam — between the inner and outer blocks it mediates — makes the two association orders of a three-graph chain produce byte-identical canonical forms; the tempting alternative (all repairs at the end) satisfies (i) and (ii) equally well and breaks (iii) at the digest layer, leaving associativity only up to a permutation that content-addressing does not forgive. Coherence at the byte level is a design property, chosen because certificates compare bytes.

Clause (ii) as stated is a floor, and the machine-checked form is stronger. In the toolkit’s Lean tree, composition at the aligned stratum — the pushout-normal form in which the seam wires of two composable open hypernets are literally shared names, the form every composite of content-addressed hypernets reduces to — carries the law family with no determinism hypothesis anywhere (module `Qtsl.HypernetCospan`): gluing preserves well-formedness wholesale — linearity, acyclicity, type consistency, and every boundary clause (`glue`); a valuation of the composite is admissible exactly when every application of each part holds (`glue_admissible_iff`); and unit and associativity hold on the nose (`glue_unit_left`, `glue_unit_right`, `glue_assoc`) — clause (iii)’s shadow in pushout-normal form.

The relational law subsumes the deterministic clause: where the parts’ admissible relations are graphs of functions, “every application holds” is exactly “evaluation of the composite equals the composition of evaluations” — and `cell`’s one-hot disjunction, the calculus’s one nondeterministic citizen, passes through the conjunction untouched. The composition law never needed determinism; clause (ii) names the fragment where “evaluation” is a function because that is the fragment the evaluator runs.

Read through that normal form, the seam repairs acquire a precise status: they are a *change of interface*, never composition overhead. The bridge between the two presentations is itself machine-checked (module `Qtsl.SubstitutionBridge`): substitution factors as a gluing against a small structural hypernet — one *delete* or *copy* row per seam wire whose consumer count changes — that mediates between the inner graph’s interface and the interface the port map asks for, and the seam-repair block is that mediator’s application list, verbatim. Gluing along an interface both sides already agree on inserts nothing; the rows appear exactly when the interface moves.

For certificate consumers, the digest story splits into three claims of different strengths, and keeping them separate is what makes “digest coherence” a tool rather than folklore:

**Remark 10.5** (digest verdicts). *(a) Determination — a theorem.* The composite’s digest is a function of the parts’ canonical bytes and the port structure: the operation reads nothing else. Equal-digest parts yield an equal-digest composite, so a certificate may cite the parts and the port row and thereby pin the whole. *(b) Coherence — a theorem, conditional on a discipline.* The unit and associativity equalities of Theorem 10.4(iii) are digest equalities *because* normalization is part of the operation; producers must emit canonical form or normalize before digesting, and the elaborators do (their outputs are fixed points of renumbering, checked). *(c) Homomorphism — refused.* There is no operation on digest *strings* computing the composite’s digest from the parts’ digests: SHA-256 composes with nothing, by design. The checker’s path is recomputation —

possess the parts' bytes, recompute the substitution, compare the claimed digest — and every consumer in the toolkit uses that pattern rather than assuming an algebra of hashes.

One boundary deserves stating before anything leans on it: elaboration and substitution commute *semantically* (elaborating a polynomial composite equals substituting the elaborations, as functions; proven, with executable witnesses), but the two routes produce *different graphs*, and different graphs never share a digest. Expanding a polynomial before elaborating re-associates its monomials; substituting factored graphs keeps the factoring. A certificate relating the two routes therefore carries a semantic linkage argument, never a claimed hash equality — the digest refuses extensionality on purpose, and this is the first place the refusal does real work.

## substitution

Primitive Catalogue entry H-SUBST

### The row (Computer Science Metanotation)

$$\frac{G, H \text{ well-formed} \quad \varphi: \text{in}(G) \rightarrow \text{out}(H) \text{ type-exact} \quad E \subseteq \text{out}(H) \text{ (default: the unmatched outputs)}}{\text{subst}(G, \varphi, H, E): \text{in}(H) + (\text{in}(G) \setminus \text{dom } \varphi) \rightarrow \text{out}(G) + E} \text{ SUBST}$$

Semantics and laws: Theorem 10.4 — composition of evaluations on the deterministic fragment; unit and associativity on the nose, as digest equalities. The seam repair is forced by linearity: an inner output with  $n$  consumers at the seam gains a delete ( $n = 0$ ), nothing ( $n = 1$ ), or a copy ( $n \geq 2$ ), and consumer counts change nowhere else. The operation normalizes its result (the canonical renumbering); the identity hypernet is its unit.

## API

Producer API of the mtk toolkit — the TypeScript arrangement and the Rust catalogue crate, one implementation underneath. The extraction, corpus, and receipt disciplines governing this block are stated once, at [this catalogue's first entry](#).

The normal form (renumber) and the unit (the identity hypernet) are quoted alongside `substitute`. The corpus construction exercises a fan-out seam, a delete seam, an explicit exposure, and a pass-through port; the composite's bytes are verified against the reference operation's output.

### TypeScript `hypernet.substitute`

signature — `ts/src/hypernet.ts`

```
export function substitute(
  outer: Hypernet,
  portMap:
    | ReadonlyMap<Wire, Wire>
    | ReadonlyArray<readonly [outerInput: Wire, innerOutput: Wire]>,
  inner: Hypernet,
  opts?: {
    expose?: ReadonlyArray<Wire>;
    normalize?: boolean;
  },
): Result<Hypernet, SubstRefusal>

export function renumber(h: Hypernet): Result<Hypernet, SubstRefusal>

export function identity(types: ReadonlyArray<TensorType>): Hypernet
```

construction — `ts/tests/hypernet-native.test.ts`

```
const comp = hn.unwrap(
  hn.substitute(
    outer,
    [
      [u, os[0]!],
      [v, os[0]!],
    ],
  ),
)
```

```

    },
    inner,
    { expose: [os[1]!] },
  ),
);

```

**Rust** `mtk_catalogue::hypernet::substitute`

signature — `crates/mtk-catalogue/src/hypernet/subst.rs`

```

pub fn substitute(
    outer: &Hypernet,
    port_map: &BTreeMap<Wire, Wire>,
    inner: &Hypernet,
    expose: Option<&BTreeSet<Wire>>,
    normalize: bool,
) -> Result<Hypernet, SubstRefusal>

pub fn renumber(h: &Hypernet) -> Result<Hypernet, SubstRefusal>

pub fn identity_hypernet(types: [&TensorType]) -> Hypernet

```

construction — `crates/mtk-catalogue/tests/golden_hypernet.rs`

```

let port_map: BTreeMap<u32, u32> = [(u, os[0]), (v, os[0])].into();
let expose: BTreeSet<u32> = [os[1]].into();
let comp = substitute(&outer, &port_map, &inner, Some(&expose), true).unwrap();

```

## 10.7 The shared core: Systems, specifications, rules

The claim this chapter exists to underwrite: the three strata that consume computation all consume *this* genus, so the chapter’s material is common infrastructure, not one stratum’s private concern.

### Systems: mechanism content is boxed hypernets

A System (chapter 12) is a structured presentation — mechanisms wired together — and the mechanism calculus itself deliberately contains *no computational primitives at all*. Its vocabulary identifies variables, grades stochastic couplings, sequences operating modes, nests components; none of these operations computes a value. Wherever a mechanism has dynamical content — a drift field, a jump reset, a guard, a neural feedback law — that content is a hypernet, referenced by digest.

The embedding has a precise grammar: a well-formed hypernet with  $m$  inputs and  $n$  outputs becomes a mechanism whose ports are its wire types and whose causal orientation is fixed (inputs consumed, outputs read out) — a *deterministic directional mechanism*, sitting inside the System layer as what string-diagram practice calls a functorial box [178]: a membrane with hypernet calculus inside and mechanism calculus outside. Composition respects the membrane — merging two boxes joined by a directed wire is computed by the substitution operation of section 10.6, inside — and the membrane is selective in both directions. Relational variable-identification, graded stochastic couplings, and mode-switching edges cannot cross into the box: they are not computations. Acyclicity ends at the membrane in the other direction: inside, SSA order is law; outside, feedback loops are the System layer’s bread and butter, closed through state across box boundaries. A control loop is a cycle of mechanisms whose every computational fragment is internally acyclic — which is how the toolkit gets both certified computation and real feedback without either discipline compromising the other. (In the vocabulary of Theorem 10.6 below: a functorial box packages a well-formed, deterministic loose arrow of the double category of open hypernets, and the membrane is that fragment’s boundary.)

### Specifications: one digest, one referent

A specification constrains observables of a trajectory; a certificate discharges the constraint with computational content of its own — a Lyapunov function’s polynomial, a barrier’s gradient bounds, the observable defining a guard surface. Both sides of that transaction are hypernets, and both cite digests under the one canonical serialization. The consequence is a sentence this book will rely on repeatedly, in both directions:

*the guard the certificate reasons about is the guard the system runs*, and the checker verifying the certificate can confirm as much by recomputing a hash — no semantic argument, no trusted translation step, no “the spec model and the implementation model surely agree.” Where a certificate’s content is produced by elaboration from a view, digest determination (Remark 10.5a) extends the same guarantee through the elaboration: cite the view’s parameters and the elaborator’s version, or cite the elaborated digest directly; either way the referent is pinned by content.

## Rules: the ambient hypernet

Every inference rule in the catalogue of chapter 15 is stated over an ambient hypernet: the rule’s variables are wires, its premises and conclusions constrain the values those wires can take, and its soundness theorem quantifies over *every* hypernet and every admissible valuation. The primitive boxes supply the base facts — each application contributes its defining equations (a contract’s sum-of-products identity, a stack’s entry equalities, a cell’s one-hot disjunction) as the synthesized facts rules build on. The economy of section 10.2 becomes a proof-theoretic economy here: nine box families contribute equations, exactly one contributes a disjunction, so case analysis in the rule layer is confined to cell-shaped splits, and a packaged symbol admitted under the candidacy rule of section 10.5 cannot enlarge that confinement.

This triple consumption is the chapter’s placement argument. Systems, specifications, and rules each depend on the hypernet genus; the genus depends on none of them. Read this chapter once, and the later chapters’ references — “dynamical content, by digest,” “the ambient hypernet,” “the certificate cites the graph” — resolve to one object with one definition.

The placement argument also has a mathematical face, and the face is machine-checked.

**Theorem 10.6** (possibilistic inclusion). *Open hypernets — hypernets with declared input and output rosters — form a double category: objects are wire rosters, tight arrows are roster inclusions, loose arrows are hypernet-decorated cospans in the sense of Fong [80], read at double-categorical strength through Patterson’s lens [193], and loose composition is gluing at the shared seam. Sending an open hypernet to its possibilistic content — the set of valuations its applications admit — extends to a lax double functor from this double category into the semantic universe of chapter 9, and the functor:*

1. *satisfies its unit, composition, and rename-transport laws as exact equalities, with no determinism hypothesis anywhere;*
2. *preserves the canonical opcartesian lifts on the nose;*
3. *has its composition comparison invertible precisely on seam-disjoint gluings — which is every gluing of encoded hypernets, since content addressing makes wire names rigid designators: distinct off-seam wires never collide, and no capture-avoiding renaming ever intervenes;*
4. *identifies two hypernets precisely when they admit the same valuations.*

Every clause is a named artefact in the toolkit’s Lean tree. The packaged functor is `hnetInclusion` (module `SuFunctor.HypernetDouble`), built from four transformation laws proved exactly, one by one, in `SuFunctor.HypernetInclusion`: unit, composition with no determinism hypothesis (section 10.6’s relational law is this clause read at the substitution seam), rename transport at full generality, and the vacuous-diagram law (`theta_unit`, `theta_comp`, `theta_rename`, `vacDnf_dnfPull`). Lift preservation is `hnetInclusion_pushGLoose`. The laxity is located, not hidden: clause 3 is `hnetInclusion_laxComp_isInvertible`, while the unit comparison is genuinely one-way — typing lives in the pattern carrier, and forgetting it is a constraint, not a defect. Clause 4 (`austereDnf_eq_iff`) says the inclusion is deliberately non-faithful, and makes its kernel a theorem rather than a folklore caveat: it is the extensionality refusal of section 10.1 standing exactly where it belongs — two graphs computing one relation share a meaning and never a digest; syntax and its content addressing live in the double category, and the semantic universe sees the relation.

What Systems, specifications, and rules consume is therefore not merely one data structure but one semantics: composing graphs and then taking meaning equals taking meaning and then composing relations.

## 10.8 Relational kinship: contraction rows

The toolkit meets tensor contraction twice: as the `contract` box here, and as *contraction rows* in the relational engine, where an `einsum` expression lowers to stored index structure — an index map and an output set as data — with the summation definitional, evaluated only where a checker needs values. The two are one definition at two strata, and it is worth seeing the shared skeleton once. Over a semiring  $\mathbb{S}$ , with a finite family of input axes  $A(i)$  and a map  $a$  from input-axis positions into a finite index set, contraction along  $a$  with output set  $O$  sends a (lazily formed) product tensor to

$$\left( v: \prod_{m \in O} B(m) \right) \mapsto \sum_{v': \prod_{j \notin O} B(j)} T(i \mapsto (v \cup v')(a(i))),$$

where the extent family  $B$  is induced on the image of  $a$  by the input extents, and is data only on the output axes no input hits (the broadcast axes, of which more below). Definition 10.2 is this formula with  $\mathbb{S} = \mathbb{Q}$ , the index set  $I$  playing the target of  $a$ , the input bindings playing  $a$ , and the output binding enumerating  $O$ ; the box evaluates it eagerly on wire values, the relational rendering stores it as rows and lets the checker fold the sum on demand. The design move is also shared: *einsum lowers to contraction rows* in the engine for the same reason *views elaborate into hypernets* here — surface notation is never given its own machinery.

Two deltas between the strata, stated so the kinship is not overclaimed. First, the relational side imposes a side condition the wire format does not: every index hit by no input must belong to the output set, so the case that would silently scale a result by an extent is *unrepresentable* in rows, while in the IR the same case is representable and simply carries its summed semantics — deterministic, if rarely what anyone meant; `einsum` lowerings never emit it on either side. Second, broadcasting: the relational rendering folds it into contraction itself (output axes hit by no input are the broadcast axes, along which the result is constant), while the IR keeps a separate broadcast box — in a dataflow graph the degenerate contraction would cost a pattern where a relabelling suffices. Same semantics; different economies; one calculus.

## 10.9 Receipts

What stands behind this chapter, sorted by strength of evidence, per the claim discipline used throughout the book.

**Machine-checked and running.** The ten-primitive calculus exists twice, in agreement: as a mechanized catalogue (the primitive type, per-box typing rules, and the well-formedness trio as decidable predicates over ground data, with well-formedness carried as proof fields of the hypernet structure) and as a reference implementation (the same typing and well-formedness checks, the exact- $\mathbb{Q}$  evaluator, a float64 shadow evaluator carrying a conservative accumulated-rounding bound, and the canonical serialization with SHA-256 digests). An ONNX importer compiles all 45 networks of the ACAS Xu collision-avoidance benchmark [134] (architecture 5–50×6–5, ReLU) to well-formed hypernets — 2,172 applications and 2,767 wires per network, exact dyadic weights — with committed digest goldens (45 networks × 3 head variants, all 135 distinct) and an oracle gate: on 47 published verification instances, exact and shadow evaluation agree within the carried bound (largest observed gap  $1.9 \times 10^{-16}$ ), and verdicts agree with an independent floating-point runtime 47/47. A property battery (well-formedness of random SSA slices; digest sensitivity — a one-ulp weight corruption flips the digest, a clean recompile reproduces the golden; evaluator agreement against an independent forward pass) runs green in seconds. The substitution operation and canonical renumbering are implemented, with the coherence and digest laws of section 10.6 each exercised by executable digest-equality witnesses (among them the deliberate digest-*inequality* of the two elaboration routes). The polynomial-view elaborator is pinned to the fifteen-application worked table by its committed digest golden, and the four packaged symbols' expansions live in the mechanized catalogue with their projection lemmas. The composition layer above the catalogue is machine-checked in the same tree: the aligned-stratum gluing with its relational law and on-the-nose unit and associativity (`Qtsl.HypernetCospan`), the reduction of the port-map substitution to that gluing (`Qtsl.SubstitutionBridge`), the four transformation laws of the possibilistic inclusion (`SuFunctor.HypernetInclusion`), and their packaging as a lax double functor into the semantic universe (`SuFunctor.HypernetDouble`) — sorry-free, axiom census the standard classical base. The plate extension (section 10.4) is machine-checked at the same grade in `Qtsl/Plate/{Syntax, TPlate, Eval, TNat, TDesc, Probes}`: symbolic shapes and instantiation, schematic well-formedness across all ten boxes, evaluator equivariance under the diagonal symmetric-group action, the causal-mask refutation,

descent to multiset-indexed denotations, and the symbolic count-wire theorem, under a three-mutant battery at preregistered loci (index-literal box, widened broadcast slot, deleted contract-binding clause — each refused at its predicted theorem); an exact-rational fixture additionally pins the nineteen-application polynomial transformer against per-token symbolic computation at  $n \in \{1, 2, 3\}$ .

**Proven in prose, with executable witnesses; mechanization under way.** Proposition 10.3 (the completion theorem behind stack) and Theorem 10.4 (well-formedness preservation, the merge law, unit and associativity on the nose) are proven at full detail in the development record’s design notes, and every claimed equality is additionally checked by the executable witnesses above. The proposition is, further, machine-checked in the toolkit’s Lean tree (lean/qtsl/, module Qtsl.PackingGap): a sorry-free Lean development, its axiom census the standard classical base, mechanizes it over the formal-expansion support model the proof sketch itself runs on — assuming no well-formedness, quantifying over all wires rather than only later ones, and holding the two tracked wires opaque whatever produced them — together with a positive control deriving the forbidden configuration once stack is added, so the nine-box hypothesis is itself proven necessary. Of Theorem 10.4 in the port-map presentation given here, the linearity clause of well-formedness preservation is machine-checked end to end, and the laws themselves are machine-checked in pushout-normal form (the aligned-stratum gluing of section 10.6: well-formedness closure, the relational composition law with the determinism hypothesis deleted, unit and associativity on the nose). The bridge — that the port-map, seam-repair, and renumbering construction of this chapter reduces to that normal form — is machine-checked in the same tree (Qtsl.SubstitutionBridge): under the port-validity hypotheses and the no-taps interface discipline, the raw composite is field-for-field the aligned gluing of the padded inner hypernet, the seam-repair block taken verbatim as the structural interpolant, and the redirected outer hypernet (substituteRaw\_eq\_glue), so substitution is the canonical renumbering of an aligned-stratum glue (substitute\_eq\_glue\_renumber) — the structural reshuffle of the prose statement absorbed into the choice of interface lists — and the relational composition law applies to the substitution itself (substituteOpen\_admissible\_iff); sorry-free, axiom census the standard classical base. The presentation of this chapter and the normal form it reduces to are thereby machine-checked as one construction.

**Design tense.** Higher-order hypernets (function-space wires — wanted by no modelling frame currently in scope, and deliberately unbuilt until one forces them); the block-ordering convention that would sharpen the commutation of substitutions into *disjoint* port sets from convention-conditional to unconditional; and the plate producer surface (the wire-format extent tag with its backward-compatibility golden, builder support, and the packaged count symbol — the pieces named in section 10.4’s status paragraph). All have written design records; nothing stated above depends on any of them.



# Hypernets: Markov Kernels

DRAFT

Chapter 10 fixed a deterministic calculus: ten primitive boxes, one evaluator, one canonical serialization, one content digest. Stochastic models need kernels: a diffusion with jumps needs a jump law, a stochastic Petri-style transition needs a firing law, and a Markov chain simply *is* its transition kernel. This chapter is about how the toolkit says “Markov kernel” without adding a primitive. A *Markov-kernel hypernet* is an ordinary well-formed hypernet together with a designated roster of *noise ports*: inputs read as fresh uniform draws on  $[0, 1]$ . Its denotation is the Markov kernel obtained by pushing the uniform reference measure forward through the deterministic semantics — every kernel is presented by an explicit deterministic dilation with its ancilla exposed. Randomness enters at the interface, never at a box; the wire format, the evaluator, and the digest scheme never learn that the extension exists. A reader who only wants to know which named distributions the toolkit provides can go straight to the coverage dashboard (section 11.4).

## 11.1 Kernels as dilations

Three consequences of this reading come essentially for free. *Normalization is a theorem, not a constraint*: every box is total and the reference measure is a probability, so nothing expressible at this level loses mass. *Discard is marginalization*: deleting an output wire marginalizes the kernel. *Copy is sharing, never resampling*: fan-out of a noise-derived value is an explicit copy row — both consumers see one realization — while an independent resample is a second noise port, so the two are different graphs with different digests. The last point is a central subtlety of the probabilistic string-diagram literature (tossing a coin and copying the outcome is not tossing two coins [25, 84]), and the linearity discipline of the deterministic calculus settles it structurally, with no new axioms; Figure 11.1 draws the two graphs.

All three consequences are machine-checked in the toolkit’s Lean tree at the finite-outcome stratum, where a stage is a measurable outcome map from the noise cube  $[0, 1]^N$  to a finite value set. Total mass one is the theorem `sum_noiseDenotFun`; pushing the denotation forward along a projection is `noiseDenot_comp`, with `discard` as the special case that forgets a coordinate. For composition, the law runs in the positive direction and in the negative one: with disjoint noise rosters, a sequential composite denotes the Kleisli composite of the stage denotations (`noiseDenot_seqComp`), while a shared roster provably departs from that composite (`copy_ne_kleisli`), so the disjointness hypothesis is content rather than bookkeeping. All of these live in the namespace `SuFunctor.Stochastic`, sorry-free.

The evaluator never samples: given noise values as ordinary rational inputs, the exact evaluator runs unchanged. Simulation is the evaluator plus an external random-number driver; certificates consume the kernel’s polynomial data and the reference measure, never samples.

## 11.2 Jump structure: mass, intensity, thinning

Continuous time asks for more than normalized kernels. Jump structure contributes a jump measure and a killing rate to a process’s extended generator [60, 69], and what must be tracked is *mass*: rates scale it, killing removes it, and staying put carries it without moving. The kernel vocabulary for this tier is a small generator language over normalized kernels — jump, stay, kill, scale, superposition, and the empty generator — in which every term carries a rational *intensity majorant*, computed by the typing derivation: scale multiplies it, superposition adds it. State-dependent intensities  $\lambda(x)$  recover from a scalar majorant  $\Lambda$  by thinning: run the clock at rate  $\Lambda$  and, at each ring, branch to the real jump with probability  $\lambda(x)/\Lambda$ , staying put otherwise.

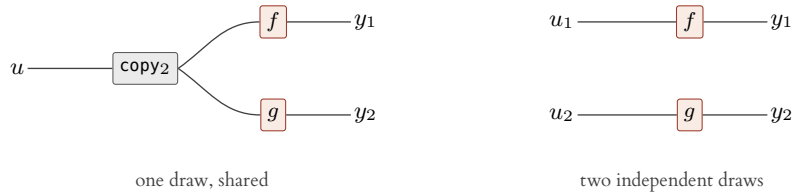


Figure 11.1: Copy is sharing, never resampling. Fanning a noise port out through an explicit copy (left) hands both consumers one realization; an independent redraw is a second noise port (right). The two are different graphs with different digests, and their denotations provably differ (`copy_ne_kleisli`).

Killing is deliberately distinct from the mass-preserving discard, and stay is deliberately distinct from zero: a stay contributes nothing to the generator but carries mass in the grading, which is what thinning and simulation budgets need to see.

This vocabulary is machine-checked too — at the finite-state stratum, in the same sorry-free module family as the dilation theorems. Every term elaborates, by structural induction, to a *thinning normal form* (`ThinningForm`): a majorant together with one normalized kernel into the state space extended by two tagged outcomes, stay and kill. A sub-stochastic mass deficit is therefore unrepresentable rather than merely discouraged. The elaboration cannot emit sub-mass, and the declared intensity distributes exactly over the named channels (`elaborate_mass_partition`, with `outflow_le_intensity` giving the majorant its meaning). Elaboration itself preserves the extended generator (`generator_elaborate`). The normal form is deliberately non-unique, and its two semantics-preserving moves are themselves theorems: a majorant may be inflated with the slack paid to stay (`generator_dilate`), and self-jump mass may be reclassified as stay (`generator_absorbDiagonal`) — killing mass may not, which is the kill/stay separation made mechanical. Composition is machine-checked at the same stratum (module `SuFunctor.Stochastic.JumpAlgebra`): the normal forms carry the term vocabulary’s composite algebra — superposition, thinning-scale, coordinate extension, and parallel interleaving, with the intensity majorants as its grading — and elaboration is a homomorphism of that graded algebra, definitionally for the module operations (`elaborate_superpose` and its kin close by `rfl`) and in the thinning quotient for interleaving (`generator_elaborate_interleave`); sequential composition of jump kernels is no part of this tier’s algebra by design — successive jumps compose through time, and the Kleisli law for discrete steps is section 11.1’s. The same tagged form recovers the jump-diffusion generator of chapter 12. That chapter’s legacy encoding reads a reset mixture’s mass deficit as killing, in Davis’s convention; rewriting each deficit into an explicit kill outcome leaves the legacy transfer increment unchanged, and the full generator is recovered with nothing sub-stochastic remaining (`generator_eq_stepMix`). Every well-typed term thus declares a finite intensity majorant, the same bounded-intensity condition the non-Zeno arguments elsewhere in this book compose with.

One consumer is worth naming here: the supervisors of the composition inventory (section 13.11) are ordinary jump-only Systems — policy memory is marking, and a policy update’s stochastic content is a firing kernel of this tier — so control through enablement adds no kernel vocabulary of its own.

### 11.3 Packaged generators

Two families of packaged generators round out the vocabulary, both in the packaged-symbol discipline of chapter 10 (a definitional expansion, a projection lemma, and a candidacy rule — never a new box).<sup>1</sup> *Distribution generators* name the classical laws: each is an alias for a certified piecewise-polynomial quantile table applied to a noise port, carrying an explicit accuracy budget, so that a model that draws from a named distribution elaborates into the same hypernet language every other certificate in this book speaks. Section 11.4 is their coverage dashboard. The *stochastic-matrix generator* covers the finite-state case directly: a kernel between finite-extent wires given by an explicit tensor of transition probabilities, whose key conformance obligation is row-stochasticity — mass one from each source state — making finite Markov chains and the jump chains of continuous-time models first-class citizens rather than encodings.

<sup>1</sup>Two senses of *generator* meet in this chapter: the analyst’s (section 11.2 presents extended generators) and the algebraist’s (a packaged generator is a named symbol with a definitional expansion). The word is inherited from both literatures; the toolkit keeps the senses apart by type.

A status note, in the spirit of the preview note at the front of this volume. The deterministic substrate is chapter 10's and is shipped. The denotational core of the noise-port reading — normalization, discard as marginalization, the Kleisli composition law and its shared-roster failure — is machine-checked at the finite-outcome stratum (section 11.1), and the jump vocabulary's semantics — the intensity grading, the thinning normal form, its soundness, and the thinning-quotient moves — are machine-checked at the finite-state stratum (section 11.2). The kernel-language *surface* has meanwhile settled: the certificate theory (theories/Qtsl.geolog) carries it as normative schema. A kernel citation there names its hypernet by digest together with its noise roster and three tagged outcome channels: *next*, *stay*, and *cemetery*, the wire format's name for the kill outcome. With every unit of mass explicitly named by one of the three channels, a sub-stochastic deficit is unrepresentable, and a pure kill is a weight-one cemetery branch.

A transition carries a *clock* with three wings, and the wing decides when the kernel is consulted. A *racing* clock declares its intensity majorant. At a finite majorant, its polynomial presentation elaborates definitionally into the thinning normal form of section 11.2; the infinite majorant instead marks the instant-fire boundary case, contributing no generator term. A *demonic* clock has no timing law — whether and when its kernel is consulted belongs to a scheduler over which every judgment quantifies — but carries a strictly positive dwell floor in the system definition: the scheduler may fire the transition only after that much contiguous dwell inside its guard region. A row with floor  $d$  therefore fires at most  $T/d + 1$  times by horizon  $T$  under every resolution, and the demonic wing cannot produce Zeno behaviour at all. An untimed Petri net transcribes with unit dwell floors: the transcription is its *discrete-time unfolding*, embedded in the continuous-time frame. It is faithful for ordering-and-counting properties, which are all an untimed net ever asserted. A *forced* clock fires at exactly its declared positive dwell of contiguous in-guard time: deterministic delay, completing the triple — exponential, immediate, deterministic — from which deterministic and stochastic Petri nets are built, so watchdogs, timeouts, and protocol timers are native rows rather than encodings. Preemption is by guard reset (whatever fires first wins, and the forced clock resets with its guard). A same-instant tie between two forced transitions resolves through the demonic scheduler's seat: the classical deterministic-net literature needs an ad-hoc tie-breaking convention exactly where an imprecise-probability semantics has a seat already waiting. With both dwell floors strictly positive, the one syntactically visible source of Zeno behaviour in the vocabulary is an unbounded chain of instant firings — precisely what the crossing-budget certificates price.

The stochastic-matrix generator is carried with row-stochasticity among its named conformance obligations. The certified quantile tables behind the dashboard's certified rows exist today as test-suite artefacts, and the dashboard section states, row by row, what each status rests on and what would move it. This chapter still owes the settled surface its presentation: a coming revision states it at the deterministic calculus's level of rigour, with its soundness against the semantic universe, and a section on firing kernels alongside.

## 11.4 Distribution generators: a coverage dashboard

The toolkit's stochastic vocabulary needs named random variates the way its deterministic vocabulary needs elementary functions, and it admits them under the same discipline: never as new primitives. A *distribution generator* is a packaged symbol whose expansion is a certified piecewise-polynomial quantile table applied to uniform noise ports, so a model that draws from a named distribution elaborates — before anything is checked — into the same hypernet language every other certificate in this book speaks (chapter 10). Checkers and certificate formats learn nothing new; being wrong in a table costs a rejected certificate, never a false verdict.

Each generator registers with an explicit budget  $(\varepsilon; \delta_{lo}, \delta_{hi})$ : quantile sup-error at most  $\varepsilon$  on the core  $[\delta_{lo}, 1 - \delta_{hi}]$ , in the family's standard scale, with the clipped tail mass declared per side. The working target is  $\varepsilon \leq 2^{-23}$  — single-precision machine epsilon — with default clips of  $2^{-30}$ ; heavy-tailed families declare conspicuously larger honest clips rather than pretending to a tail no polynomial reaches, and multiplicative families state  $\varepsilon$  relatively. Verification never encloses an inverse transcendental: each certificate drives the *forward* CDF through polynomial enclosures, piece by piece in a rescaled frame, and closes with a residual row, a derivative-floor row, and one rational comparison. A verification class grades what forward material that takes: **V0** nothing (the construction is exact), **V1** polynomials only, **V2** the mechanized elementary enclosures (exponential, sine, cosine, the error function, and now the incomplete gamma), **V2q** certified quadrature, **V3** an enclosure family built specially for the row that needs it (to date one exists: the incomplete beta, whose tail lemma is its own). Every certified row also ships a refusal battery in the sense

the Experiments chapter (chapter 2) makes precise: tampered coefficients, shaved residuals, dishonest tail declarations, partition gaps, and dropped budget-composition terms must each be refused by a named row.

**Enclosures are certified valid unconditionally.** The core range bounds where the *tightness* claim lives, not where validity ends: a generator’s certificate is a bundle of unconditional theorems, and no row’s truth is scoped to the core. The enclosures are also certified to be *tight* to single-precision machine epsilon within their core range (the working target; a row that declares a different  $\varepsilon$  says so in the dashboard), and outside the core they degrade honestly, in two genre-specific ways, because the two kinds of table stand on different input axes. A quantile table’s axis carries the probability measure itself, so the table is total on  $[0, 1]$ : its end cells clamp to exact rational constants, and the clamp-honesty rows certify that the mass beyond each clamp is at most the declared  $\delta$  — a checked row, never trust. Monotonicity keeps even the off-core bracket better than trivial: beyond the core the true quantile stays pinned between the certified core-edge value and the support boundary, a certified inference wherever the monotonicity rows are claimed, so the generated law couples to the named law within  $\varepsilon$  except on an event of probability at most  $\delta_{lo} + \delta_{hi}$ . A function table — a certified polynomial approximant of a plain function on a declared domain, the wrappers among them — stands on an axis that carries no measure, so there is no tail mass with which to pay for a clamp: off its declared domain such a table simply makes no claim, and an off-domain evaluation degrades to the trivial bracket — nothing is known about this number — which is still a valid statement, never an assertion of impossibility. Refusal, where it appears in this discipline, is a judgment about claims rather than about numbers, and it lives at production and certification time: a producer declines to emit a table it cannot cover, and a checker refuses a tightness claim over domain a table does not cover. The dashboard’s  $\times$  against the unbounded uniform is different in kind from all of this: there the family itself admits no quantile function, an impossibility of the mathematics rather than a limit of a table’s coverage.

**The census.** Coverage is measured against an external standard the toolkit does not own: Crooks’ *Field Guide to Continuous Probability Distributions* [51], whose numbering organizes some 170 named forms into on the order of a hundred sections related by an explicit lattice of limits, reparametrizations, and transforms. The Field Guide’s own section numbers are the census: every numbered form appears in exactly one row below (block rows name the sections they cover; synonyms resolve to the section they cite — Lorentzian to Cauchy, Fisk to log-logistic, fatigue-life to Birnbaum–Saunders). The *route* column says how the family is reached — *exact* (arrow algebra over existing generators: order statistics, sums, products, mixtures, folds), *composition* (certified wrappers over the two quantile roots), *own table* (a directly-tabled quantile where that is cheaper than composition), or *dispatch* (a parameter-time classification to a member family) — with the verification class in parentheses. The *status* column is the dashboard: *certified* rows carry receipts as described above; *routed* rows have a fixed construction and verification class but no table yet; *out of scope* rows are deliberate, each with the honest reason, the priced alternative, and the condition that would reopen it. Two roots (the standard exponential and standard normal quantiles) and three wrappers (rational powers, logarithm, exponential) generate everything except the own-table rows; location-scale families register one standard table each, extended by exact affine arrows.

Each row carries a mark in the same vocabulary as the QTSL rule catalogue’s coverage dashboard:  $\checkmark$  certified,  $\sim$  routed (construction and verification class fixed, no table yet),  $\times$  out of scope;  $\checkmark \sim$  marks a row certified for its named part with the residue routed.

**Shared infrastructure (roots, wrappers, constants).**

| object                                                                                                                | status |                                                     |
|-----------------------------------------------------------------------------------------------------------------------|--------|-----------------------------------------------------|
| exponential quantile root (the registered standard exponential)                                                       | ✓      | certified — $\varepsilon \leq 4.3 \times 10^{-8}$   |
| error-function forward enclosure (mechanized; soundness stated against the Gaussian integral, standard axioms only)   | ✓      | certified                                           |
| probit quantile root (the registered standard normal)                                                                 | ✓      | certified — $\varepsilon \leq 2.9 \times 10^{-8}$   |
| rational-power wrapper (reciprocal and square root included)                                                          | ✓      | certified — per-instance $\varepsilon \leq 2^{-24}$ |
| logarithm wrapper (verified through the exponential's forward enclosure — no logarithm enclosure exists or is needed) | ✓      | certified — $\varepsilon \leq 2 \times 10^{-9}$     |
| exponential wrapper (sinh, cosh derived exactly)                                                                      | ✓      | certified — relative-error native                   |
| certified constants ( $\ln 2$ , $\pi$ , $2/\sqrt{\pi}$ )                                                              | ✓      | certified — relative half-width $\leq 2^{-55}$      |
| composition algebra: exact budget arithmetic per arrow, with a route audit that refuses any dropped term              | ✓      | certified                                           |

**Zero shape parameters (Field Guide ch. 1–4).**

| §   | family                                        | route            | status |                                                                                                                         |
|-----|-----------------------------------------------|------------------|--------|-------------------------------------------------------------------------------------------------------------------------|
| 1.1 | uniform (half uniform; degenerate point mass) | exact (V0)       | ✓      | certified — exact                                                                                                       |
| 1.— | unbounded uniform                             | —                | ✗      | out of scope — improper (not normalizable), so no quantile function exists; set-valued premises cover the intended uses |
| 2.1 | exponential (anchored, standard)              | composition (V2) | ✓      | certified — $\varepsilon \leq 2^{-24}$                                                                                  |
| 3.1 | Laplace                                       | composition (V2) | ✓      | certified — $\varepsilon \leq 2^{-24}$                                                                                  |
| 4.1 | normal                                        | composition (V2) | ✓      | certified — $\varepsilon \leq 2.9 \times 10^{-8}$ at unit scale (exact affine scaling)                                  |

## One shape parameter (ch. 5–9).

| §    | family                                                                           | route            |   | status                                                                                                                                                |
|------|----------------------------------------------------------------------------------|------------------|---|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5.1  | power function (generalized Pareto, $q$ -exponential reparametrizations)         | composition (V1) | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                |
| 5.4  | wedge (ascending, descending)                                                    | exact (V0)       | ✓ | certified — exact                                                                                                                                     |
| 5.5  | Pareto (Lomax 5.6 = shifted)                                                     | composition (V1) | ✓ | certified — $\varepsilon \leq 2^{-23}$ (moderate core; the wide core is deep-tail-dominated and its excluded mass is declared)                        |
| 5.7  | exponential ratio (uniform-prime 5.8; Lomax(1))                                  | composition (V1) | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                |
| 6.1  | log-normal (Gibrat)                                                              | composition (V2) | ✓ | certified — relative $\varepsilon \leq 2^{-23}$                                                                                                       |
| 7.1a | gamma, integer shape (Erlang)                                                    | exact (V2)       | ✓ | certified — $\varepsilon \leq 2^{-24}$ (three-port pin; per-port tables pre-tightened so sums stay under target)                                      |
| 7.1b | gamma, half-integer shape                                                        | exact (V2)       | ✓ | certified — declared per alias — the squared-clamp amplification is stated at its measured value (e.g. $3.5 \times 10^{-7}$ at one degree of freedom) |
| 7.1c | gamma, general rational shape                                                    | enclosure (V2)   | ✓ | certified — $\varepsilon \leq 2^{-23}$ at the pinned fractional shapes (measured $2^{-27}$ -class)                                                    |
| 7.3  | chi-square (scaled; Porter–Thomas)                                               | exact (V2)       | ✓ | certified — $\varepsilon \leq 2^{-24}$ at even $k$ ; odd $k$ declared per alias                                                                       |
| 8.1  | gamma-exponential<br>(chi-square-exponential;<br>generalized Gumbel)             | composition (V2) | ✓ | certified — $\varepsilon \leq 2^{-23}$ (integer shapes); half-integer members inherit the squared-clamp declaration                                   |
| 8.5  | Gumbel                                                                           | composition (V2) | ✓ | certified — $\varepsilon \leq 2^{-26}$                                                                                                                |
| 8.7  | BHP                                                                              | composition (V2) | ✓ | certified — $\varepsilon \leq 2^{-23}$ at the irrational shape $\pi/2$ (a bracketed-shape pin plus an exact shape-transfer argument)                  |
| 8.8  | Moyal                                                                            | composition (V2) | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                |
| 9.1  | Pearson VII, half-integer $m$<br>(Student $t$ , integer $\nu$ ;<br>Breit–Wigner) | composition (V2) | ✓ | certified — declared per floor — the heavy tail's excluded-event mass is priced exactly, and the sup concentrates in near-floor tail events           |
| 9.4  | Student $t_3$ (the odd- $\nu$ pattern)                                           | own table (V2)   | ✓ | certified — $\varepsilon \leq 2^{-26}$                                                                                                                |
| 9.6  | Cauchy (Lorentzian; $t_1$ )                                                      | own table (V2)   | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                |



**Two shape parameters (ch. 10–12).**

| §     | family                                                                                                                                                        | route               |   | status                                                                                                                                                                                                                                       |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10.1  | unit gamma                                                                                                                                                    | composition (V2)    | ✓ | certified — $\varepsilon \leq 2^{-23}$ (relative $\leq 2^{-38}$ at the standard pins); half-integer members declare their relative error                                                                                                     |
| 10.2  | uniform product                                                                                                                                               | exact (V0)          | ✓ | certified — exact                                                                                                                                                                                                                            |
| 11.1  | Amoroso (Stacy = anchored)                                                                                                                                    | composition (V1+V2) | ✓ | certified — $\varepsilon \leq 2^{-23}$ (integer and half-integer pins; both signs of the power parameter — the inverse-Amoroso branch included)                                                                                              |
| 11.7  | half-normal                                                                                                                                                   | exact (V2)          | ✓ | certified — $\varepsilon \leq 2.9 \times 10^{-8}$ at unit scale; no clipped mass at the support edge                                                                                                                                         |
| 11.8  | chi (Rayleigh, Maxwell, Nakagami)                                                                                                                             | composition (V2)    | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                       |
| 11.13 | inverse-Amoroso block: inverse gamma and exponential, Lévy, scaled/inverse chi and chi-square, inverse Rayleigh, Maxwell, half-normal, Nakagami (11.13–11.23) | composition (V1+V2) | ✓ | certified — $\varepsilon \leq 2^{-23}$ (clean pins); the heavy members (Lévy and kin) are declared per alias at every floor                                                                                                                  |
| 11.24 | Fisher–Tippett block: Weibull, reversed Weibull, Fréchet, and their generalized forms (11.24–11.29)                                                           | composition (V1+V2) | ✓ | certified — $\varepsilon \leq 2^{-23}$ (Weibull pins); the heavy-tailed Fréchet members declare their clip masses                                                                                                                            |
| 12.1a | beta, integer shapes (PERT; Pearson XII)                                                                                                                      | exact (V0)          | ✓ | certified — exact                                                                                                                                                                                                                            |
| 12.1b | beta, half-integer shapes                                                                                                                                     | composition (V2)    | ✓ | certified — declared per pin — the ratio route states its amplification (it cannot reach the working target at any floor); correctness rests on exact anchors, and tight-absolute consumers take the incomplete-beta root at the same shapes |
| 12.1c | beta, general rational shapes                                                                                                                                 | enclosure (V3)      | ✓ | certified — $\varepsilon \leq 2^{-30}$ at the five full-depth pins (vs the $2^{-23}$ target), $\delta = 2^{-31}$ per side                                                                                                                    |
| 12.5  | central beta                                                                                                                                                  | exact (V0)          | ✓ | certified — exact                                                                                                                                                                                                                            |
| 12.6  | arcsine (centered arcsine)                                                                                                                                    | own table (V2)      | ✓ | certified — $\varepsilon \leq 2^{-24}$ ; exact end cells, no clipped mass                                                                                                                                                                    |
| 12.8  | semicircle                                                                                                                                                    | own table (V2)      | ✓ | certified — $\varepsilon \leq 2^{-24}$ ; exact end cells, no clipped mass                                                                                                                                                                    |
| 12.9  | Epanechnikov, biweight, triweight (12.9–12.11)                                                                                                                | exact (V0/V1)       | ✓ | certified — exact                                                                                                                                                                                                                            |

**Two shape parameters (ch. 13–16).**

| §    | family                                                | route                                            |   | status                                                                                                                                                                                                                                 |
|------|-------------------------------------------------------|--------------------------------------------------|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 13.1 | beta prime (inverse Lomax;<br>13.1–13.4)              | exact +<br>composition (V1<br>int / V2 half-int) | ✓ | certified — $\varepsilon \leq 2^{-23}$ (integer shapes;<br>the $\gamma=1$ heavy tail declares its clip mass);<br>half-integer members declare per floor,<br>cross-anchored against the integer route                                   |
| 13.3 | $F(k_1, k_2)$                                         | composition (V1<br>even / V2 odd)                | ✓ | certified — $\varepsilon \leq 2^{-23}$ (even degrees of<br>freedom); odd-parity members declare<br>per floor (mixed parity with an even<br>denominator is measurably tightest)                                                         |
| 14.1 | beta-exponential (hyperbolic-sine;<br>Nadarajah–Kotz) | composition (V2)                                 | ✓ | certified — $\varepsilon \leq 2^{-31}$                                                                                                                                                                                                 |
| 14.2 | exponentiated exponential                             | composition<br>(V1+V2)                           | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                 |
| 15.1 | beta-logistic (Fisher $z$ )                           | exact (V2)                                       | ✓ | certified — $\varepsilon \leq 2^{-23}$ (integer shapes —<br>the logit-chain identity is<br>distribution-level exact); half-integer<br>members declare their chain budgets                                                              |
| 15.2 | Burr II (reversed Burr II)                            | composition<br>(V1+V2)                           | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                 |
| 15.4 | central logistic                                      | exact (V2)                                       | ✓ | certified — $\varepsilon \leq 2^{-24}$ (= the logistic pin's<br>exactly: order-statistic gates are<br>1-Lipschitz in the sup norm)                                                                                                     |
| 15.5 | logistic                                              | exact (V2)                                       | ✓ | certified — $\varepsilon \leq 2^{-24}$                                                                                                                                                                                                 |
| 15.6 | hyperbolic secant                                     | own table (V2)                                   | ✓ | certified — $\varepsilon \leq 2^{-26}$                                                                                                                                                                                                 |
| 16.1 | Pearson IV                                            | quadrature (V2q)                                 | ✓ | certified — the composed sup error is<br>stated per pin with the tangent-stage<br>Lipschitz amplifier explicit; heavy<br>polynomial tails make deep clamps<br>expensive and the two-floor<br>presentation applies (both floors stated) |

**Three and more shape parameters (ch. 17–20).**

| §     | family                                                                                                                                        | route                        |   | status                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 17.1  | generalized beta                                                                                                                              | composition (V1+V2)          | ✓ | certified — $\varepsilon \leq 2^{-39}$ (standard pins); the $\beta=1$ member degenerates to its source exactly                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 17.2  | Kumaraswamy                                                                                                                                   | composition (V1)             | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 18.1  | generalized beta prime (transformed beta)                                                                                                     | composition (V1+V2)          | ✓ | certified — $\varepsilon \leq 2^{-35}$ (standard pins)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 18.3  | Burr block: Burr XII (Singh–Maddala), Dagum, paralogistic (18.3–18.6)                                                                         | composition (V1)             | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 18.7  | log-logistic (Fisk)                                                                                                                           | composition (V1)             | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 18.8  | half-Pearson VII (half-Cauchy 18.9)                                                                                                           | composition / own table (V2) | ✓ | certified — the algebraic member $m=\frac{3}{2}$ at $\varepsilon \leq 2^{-23}$ (provably the only polynomial-route shape); half-Cauchy takes the exact tangent-table fold ( $\varepsilon \leq 2^{-23}$ , zero low-side clip); general half-integer $m$ as a half-normal $\times$ chi-ladder product net, declared per member (heavy-ratio tail geometry)                                                                                                                                                                                                                                                                                                                                                       |
| 18.10 | half generalized Pearson VII; half-Laha                                                                                                       | member routes                | ✓ | certified — heavy-tailed per the Pearson VII geometry: the inverse gamma chain's clamp mass is the declared tail mass and the product amplification by the clamp value is stated per pin                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 19.1  | Pearson family                                                                                                                                | dispatch                     | ✓ | certified — exact — a classification, not a table: parameters dispatch to a member family at registration time, in exact rational (and quadratic-extension) arithmetic                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 19.2  | $q$ -Gaussian                                                                                                                                 | dispatch                     | ✓ | certified — exact — a reparametrization, not a table: $q > 1$ dispatches to Student $t$ , $q < 1$ to central beta, $q \rightarrow 1$ to normal                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20.—  | grand-unified block: Halphen, Sichel, Libby–Novick, Gauss- and confluent-hypergeometric, Perks, Appell beta, Laha, and kin (20.2, 20.5–20.18) | quadrature (V2q)             | ✓ | certified — $\varepsilon \leq 2^{-23}$ -class per member at declared cores ( $2^{-24}$ – $2^{-28}$ for the quadrature members; normalization constants — Bessel- $K$ and hypergeometric values — pinned by bracketed quadrature to $10^{-13}$ – $10^{-15}$ width, no special function ever named; the generalized members cross-anchored two ways, direct quadrature vs. power-wrapper pushforward, at $4.5 \times 10^{-9}$ sup gap); heavy-tailed members declare a core-depth frontier ( $2^{-23}$ -class at moderate cores, deeper-core cost stated); the Perks family is certified across its full rational-shape domain (all four discriminant regimes; $2^{-29.9}$ – $2^{-30.7}$ at the full-depth pins) |
| 20.3  | inverse Gaussian (reciprocal inverse Gaussian 20.4)                                                                                           | own table (V2)               | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

**Miscellanea (ch. 21).**

| §     | family                                                                              | route                    |   | status                                                                                                                                                                                                                                             |
|-------|-------------------------------------------------------------------------------------|--------------------------|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 21.1  | Bates (Irwin–Hall 21.9; triangular 21.22; uniform difference 21.23)                 | exact (V0/V1)            | ✓ | certified — exact                                                                                                                                                                                                                                  |
| 21.2  | beta-Fisher–Tippett (exponentiated Weibull)                                         | composition (V1+V2)      | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                             |
| 21.3  | Birnbaum–Saunders (fatigue-life)                                                    | composition (V1+V2)      | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                             |
| 21.4  | exponential power, general $\beta$ (Box–Tiao; Subbotin)                             | enclosure (V2)           | ✓ | certified — $\varepsilon \leq 2^{-23}$ (the classical members anchor: $\beta=1$ Laplace exactly, $\beta=2$ normal against the error function)                                                                                                      |
| 21.6  | generalized Pearson VII                                                             | member routes            | ✓ | certified — as 18.10 (the symmetric double: sign cell-mixture takes the max of two mirror branches, so the half core’s declaration carries over)                                                                                                   |
| 21.8  | K (generalized K 21.5)                                                              | exact (V2)               | ✓ | certified — $\varepsilon \leq 2^{-23}$ (integer shapes); half-integer members inherit the squared-clamp declaration                                                                                                                                |
| 21.10 | Johnson SU (SB, SL variants)                                                        | composition (V2)         | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                             |
| 21.12 | log–Cauchy                                                                          | composition (V2)         | ✓ | certified — relative $\varepsilon \leq 2^{-23}$ (the log-normal precedent); a family with no moments of any order — only the relative bound is claimed, the astronomical absolute error is stated, and the excluded tail mass is declared per side |
| 21.13 | Meridian                                                                            | exact (V1 (+cell))       | ✓ | certified — $\varepsilon \leq 2^{-23}$ (moderate core)                                                                                                                                                                                             |
| 21.15 | noncentral chi-square block: noncentral chi 21.14, Rice 21.18, noncentral $F$ 21.16 | exact + composition (V2) | ✓ | certified — $\varepsilon \leq 2^{-23}$                                                                                                                                                                                                             |
| 21.17 | pseudo–Voigt                                                                        | exact (V2)               | ✓ | certified — $\varepsilon \leq 2^{-23}$ (the mixture takes the larger component budget, inheriting its Cauchy component’s declaration exactly)                                                                                                      |
| 21.19 | slash                                                                               | composition (V2)         | ✓ | certified — $\varepsilon \leq 2^{-23}$ (moderate core); the full heavy core’s error is declared at its measured value                                                                                                                              |
| 21.20 | stable (Holtsmark 21.7; Landau 21.11)                                               | composition (V2)         | ✓ | certified — $\varepsilon \leq 2^{-23}$ at the certified pins (Holtsmark $2^{-24.3}$ ; Landau $2^{-23.6}$ — the $\alpha = 1$ seam member); the angular tables are direct mid-slack residual rows, one transcendental bracket per pin                |
| 21.21 | Suzuki                                                                              | exact (V2)               | ✓ | certified — $\varepsilon \leq 2^{-23}$ (small shapes); wide-support members declare the log-normal scale amplification                                                                                                                             |
| 21.24 | Voigt                                                                               | exact (V2)               | ✓ | certified — $\varepsilon \leq 2^{-23}$ at the half-scale witness pin; the unit pin states its summed budget (component errors add across the convolution sum)                                                                                      |

**What the statuses rest on.** Status column as of July 27, 2026; it will move with each preview. Unlike the tables of the Experiments chapter, this section’s rows are not yet re-derived by the experiments repository’s make targets: the dashboard re-derives from the toolkit workshop’s census of record, and the certificates behind the certified rows live at this date in the toolkit repository as ordinary test-suite artefacts (accept fixtures at the declared budgets plus their refusal batteries). Migrating them onto the experiments repository’s reproduction spine — regenerated as the same sealed containers that chapter’s fluid-drift certificates already ride, re-verified by the blessed checkers, one make target for the family — is named successor work, and this section’s tables will then re-derive from committed receipts like every table of the Experiments chapter. Per the preview note at the front of this volume, until an artefact appears in the public repositories, treat its mention as a statement of what exists, not yet a link.

# Systems: Mathematical Models

DRAFT

Chapter 9 lays out the universe these profiles present into, with its records, its structured time base, its imprecise denotation, and its composition calculus. The statuses below grade each frame against that semantics. The grading has a stated division of labour: most entries give the frame’s own rung in their *Semantics* paragraph — a generator, a path measure, a solution concept — and the bridge lemmas, staged as named obligations and discharged frame by frame, then make up the distance between that rung and the record-space denotation. Where an entry’s semantic content *is* an interpretation into the universe’s objects, as with the Bag former’s carrier colimits or the credal reading of quantification over policies, the entry carries the interpretation explicitly.

Throughout this book a *System* means a structured presentation, a composition term over *mechanisms* and *wirings*. A mechanism is a primitive dynamics declaration: a family of fields over declared state variables together with an interface binding. A wiring identifies variables across interfaces, subject to causality and simultaneity side conditions. A System is never the semantic quotient of the dynamics it presents, and in particular it is never the infinitesimal generator. The generator re-enters one level down, as the certification interface, which compiles mechanism rows into the obligation rows the exact kernel checks. Time is base-universal. Composites live on real time, clocks are internalized as mechanism state, and a discrete-time component taken in isolation is schedule-generic, acquiring its schedule assumptions exactly at the boundary where it composes with continuous neighbours.

One consequence organizes this chapter: **a model type is a profile, not a tag**. Nothing in the theory stores “this model is a stochastic differential equation”. A model’s type consists in *which field families its mechanisms populate*, and a composite may mix profiles freely. Ordinary engineering forces this design. Hybrid stochastic systems must be first-class citizens, and a robot — any robot — is a mixed discrete/continuous composite that must type-check across its own components. A stored per-model doctrine tag fails both requirements, whereas a profile satisfies both by construction. The sections below give the normative inventory of those profiles, at the granularity a modeller shops at. A companion human-facing book, *Mathematical Modelling with MAITRIA*, will treat modelling as a discipline in its own right, while this chapter takes only what a toolkit user needs.

The roster of sections is not ours. It is the list of modelling frames that the Safeguarded AI programme thesis (David “davidad” Dalrymple, *Safeguarded AI: Constructing Guaranteed Safety*, ARIA) [55] sets out in its Appendix A.1 as the frames a unified modelling language should subsume — the same TA1.1/TA1.2 rubric the rest of this book grades itself against. Grading a candidate language against that list is itself programme-native. In a December 2023 programme talk, davidad ran the exercise, scoring the identical list against stochastically and dynamically coloured Petri nets (SDCPN, section 12.18) as the candidate unifier. Two cells came back not-covered in 2023, deterministic and stochastic partial differential equations, and the deterministic half is where the present stack has landed some of its freshest certificates (section 12.5). We read that as evidence that the candidate graded here is not merely re-walking SDCPN’s coverage, and we keep the grading honest for the same reason the 2023 slide did, since the value of the exercise lies in the map of what remains.

The catalogue itself runs wider than the 2023 list. This chapter presents the *Model Types* wing of a two-winged *System Catalogue*, with model types here and composition patterns in chapter 13. The catalogue’s roster also carries the substrate every profile populates (the System core, section 12.1; tick declarations, section 12.2) together with the types in production use beyond the list (clocked circuits, the Bag former, SDCPN, the i.i.d. tier), and one frame beyond the list whose mechanized semantics arrived ahead of any producer surface: the quantum nets of section 12.25. Every entry, on the list or beyond it, follows the same section grammar.

**How to read the sections.** Every section carries the same four rubrics, adapted from the working catalogue's entry format. *The frame* says what the model class is, and gives its bread-and-butter verification question. *Profile* says which field families a model of this type populates: content language, carrier, disturbance semantics, and time semantics. *Semantics* names the bridge-obligation class its satisfaction claims discharge, and every analytic lemma that licenses reading exact algebra as a statement about trajectories or measures is *staged as a named hypothesis*, never silently assumed. *Status* gives the state of four legs, each graded only against a witness (below). Each entry then closes with two blocks in the Rule Catalogue's grammar. The first, a *Computer Science Metanotation* block, gives the entry's *term former*, an inference rule whose premises are the formation data the builders consume, with their sorts, and whose conclusion is a primitive judgment at the sort System (or at the entry's own true sort, where an entry declares vocabulary rather than forming a System). The second, an *API* block, gives the producer surfaces (TypeScript, then Rust) with signatures and constructions extracted from the toolkit sources at every regeneration of this catalogue. Frontier entries (section 12.26) carry neither, since no producer surface exists for them at this preview, and an invented one would defeat the grading. The wiring-pattern entries (delay differential equations, section 12.6; partially observable decision processes, section 12.14) also carry neither, deliberately: each of those frames *is* a composite of other formers, and the absence of a former of its own is the entry's content. The quantum entry (section 12.25) sits between the two conventions: its formation rule stands on mechanized Lean ground truth, so it appears; its API block waits for a producer surface to exist. The four legs —

- **representation:** a model of the frame is declarable as rows of the modelling language in a form a frame-native reader recognizes as *their* model, with the frame's compositional grain preserved rather than flattened by an encoding trick;
- **semantics:** the frame's standard semantics is recovered, either exactly or at a named rung with the bridging hypotheses staged;
- **certification:** at least one of the frame's own bread-and-butter questions compiles to obligations checked by the *unmodified* exact kernel, with a shipped exact-rational artefact;
- **compositional closure:** the frame's systems compose with systems of *other* frames through the patterns of chapter 13, with certificates transported across the composite boundary and witnessed by shipped artefacts rather than design prose. A frame can pass the first three legs and still be flattened at its composition boundaries, and this leg exists to detect exactly that. Where a frame's semantics genuinely cannot instantiate a pattern's composition data (a static distribution has no internal time axis for a supervisory switch to sequence), the omission is earned by a filed argument from the frame's own semantics, never by a bare declaration.

As of this preview, *no frame on the list passes all four legs*, and exactly one (ordinary differential equations) holds a cross-frame composition witness. The statuses below serve as the campaign's map rather than a victory lap. They are re-graded at each preview, and several moved in the week this chapter was drafted. We publish the grades at this granularity because the alternative, a coverage matrix of green ticks, is how unification claims usually go wrong.

**How to read this chapter.** The sections run in parallel, with the same rubrics in the same order, so a reader may take them in any order and sample by interest. A reader new to the catalogue should read two entries in full. Ordinary differential equations (section 12.4) carry the catalogue's simplest square, and discrete-time Markov chains (section 12.11) carry the square that is the engine most later squares reuse. After those two, every other entry is a variation on a theme the reader already knows, each adding one idea. Four entries repay attention even from a reader in a hurry: the delay line (section 12.6), which dissolves an apparently new semantic species into a wiring pattern over inventory the catalogue already has; the demonic faces of sections 12.13 and 12.15, where the adequacy squares first become set-valued; the coloured nets of section 12.18, where the 2023 candidate unifier is itself compiled into the catalogue it was once graded against; and the quantum nets of section 12.25, the one entry whose set-valued face is demanded by a theorem of its own literature rather than chosen by the modeller. The formal blocks closing each entry serve as reference material, so skip them on a first pass, and return to them when declaring a model.



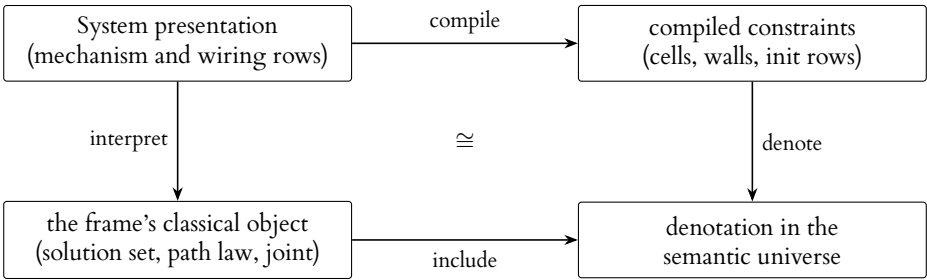


Figure 12.1: The adequacy square that this chapter’s Adequacy paragraphs instantiate. At the two-sided entries the two routes agree by an invertible globular cell; at the martingale–problem entries (stochastic and jump–diffusion) the cell runs one way, from the community’s semantics to the certified verdict. The middle column of the table below grades this square, frame by frame.

**The list at a glance.** The middle column grades each entry’s *adequacy square* (Figure 12.1), the mechanized statement, carried in the entry’s Adequacy paragraphs, that compiling a model and then denoting it agrees with interpreting it classically and then including the classical object. The outer columns grade certification, and dashes mean nothing shipped.

| frame                | adequacy square                                                                                                                                                         | certified today                                                                                                  | principal gap                                      |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| static               | definitional — constant records (stated in the entry, not mechanized)                                                                                                   | —                                                                                                                | a consuming certificate                            |
| ODEs                 | two-sided                                                                                                                                                               | controller certificates; validated tubes; one cross-frame composite                                              | mechanism rows through the ingestion path          |
| PDEs (deterministic) | two-sided, at every mechanism of the former                                                                                                                             | four exact dissipation certificates; bridge lemmas mechanized                                                    | classical existence (non-vacuity, never soundness) |
| SDEs                 | one-directional, at the martingale problem                                                                                                                              | generator certificates with correlated noise, $n \leq 3$                                                         | state-dependent noise at $n \geq 2$                |
| SPDEs                | —                                                                                                                                                                       | moment + probability certificates, noise floors exact; derived covariance barrier uniform over the parameter box | trace-class instances; multiplicative noise        |
| RODEs                | two-sided, per draw and over the box                                                                                                                                    | all three field families, $n \leq 2$                                                                             | the PDE side (RPDEs)                               |
| jump-diffusion       | one-directional; generator splice hypothesis-free                                                                                                                       | —                                                                                                                | certificates exercising multi-leg kernels          |
| DTMCs                | two-sided                                                                                                                                                               | the published-table campaigns                                                                                    | native rows; logic closure                         |
| CTMCs                | two-sided (jump-chain face)                                                                                                                                             | four fixtures, incl. unbounded reach                                                                             | nested temporal operators                          |
| MDPs                 | two-sided, fixed-policy and demonic faces                                                                                                                               | fixed policy                                                                                                     | the policy-quantified reproduction target          |
| Markov automata      | two-sided, fixed-scheduler and demonic faces                                                                                                                            | fixed scheduler                                                                                                  | the scheduler-quantified reproduction target       |
| open games           | —                                                                                                                                                                       | —                                                                                                                | representation and semantics                       |
| mean-field games     | —                                                                                                                                                                       | —                                                                                                                | everything; road through PDEs                      |
| GSHS                 | generator rungs only                                                                                                                                                    | a full-complement hybrid example                                                                                 | guarded composites at $n \geq 2$                   |
| Petri nets           | statics, race, cascade, and set-firing faces (schedulers declared and quantified) at the coloured entry; the untimed interleaving face two-sided at the classical entry | token game run natively; coverability                                                                            | bounded-exploration hypothesis                     |
| Bayesian networks    | two-sided                                                                                                                                                               | exact posteriors, finite case                                                                                    | continuous case; conditioning at scale             |
| SCMs                 | substitution rung                                                                                                                                                       | counterfactual bounds, both directions                                                                           | conditioning; latent confounders                   |
| MRFs                 | —                                                                                                                                                                       | partition-function brackets                                                                                      | continuous case                                    |
| PPLs                 | —                                                                                                                                                                       | the discrete-time slice                                                                                          | conditioning                                       |
| ProbLog              | —                                                                                                                                                                       | —                                                                                                                | weighted-chase wiring                              |
| SBGMs                | —                                                                                                                                                                       | —                                                                                                                | composing two shipped parts                        |

A single content language carries the whole continuous and hybrid family below, a manifest vocabulary of modes with domains, drift, diffusion, multi-driver noise, jumps, and guarded transitions with resets, and the types differ only in which fields are populated. Content-addressed references carry neural and piecewise content (a network controller in the loop, a learned drift) on any of these profiles, and where a polynomial view coexists with network content, their agreement is a recomputation obligation, never a trusted row. Discrete-kernel types present outcome rows, while nets and graphical models carry their own small theories, described in place.

## 12.1 The System Core

**The entry.** The substrate every profile populates: the manifest root, its two strata, and the family view that makes “profile, not tag” an operational sentence rather than a slogan. A System manifest is a composition term — a *mechanism* (the primitive-dynamics stratum: field families over declared state variables plus an interface binding) or a *composite* (the wiring stratum: component Systems plus variable-identification rows and the composite-level descriptors of chapter 13) — the mechanisms-and-wirings reading of systems whose doctrinal home is the double operadic theory of Libkind and Myers [163]. Every field family of a mechanism is individually optional at the type level; the free-form entry point never refuses, because any family combination is a legal *presentation* — conformance to a named profile is a separate, optional check (the presets of the sections below), and a mechanism whose populated set matches no preset is a first-class citizen. That check, in plain words and stated here once for the whole chapter: a manifest *conforms* to a model type when it populates that type’s field families and no others beyond the shared substrate ( $\Sigma$ , below); the preset signatures quoted in each entry’s API block are the mechanized form of exactly that check. No time-semantics tag is stored anywhere: continuous, counter, or hybrid is derivable from which families are populated, and storing it would be exactly the per-model doctrine tag this catalogue rules out.

**Trust posture.** Presets and the preflight walk are courtesies — profile notes and dangling-reference warnings with path geometry, never verdicts; the checker’s replay is the only authority on content. On the wire the manifest is CBOR under a schema-versioned envelope with exact scalars; the committed golden vectors are the cross-language byte authorities, and the TypeScript surface reaches the same Rust implementation in-process, so the two surfaces cannot diverge.

### Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\begin{array}{c}
 \vec{x} : \overrightarrow{\text{StateVar}} \quad \Sigma_m \text{ substrate rows over } \vec{x} \text{ } (\Sigma, \text{ below}) \\
 \Phi \subseteq \mathcal{F} \quad F_\phi : \overrightarrow{\text{Row}}_\phi \quad (\phi \in \Phi) \\
 \hline
 \text{Mechanism}(\vec{x}; \Sigma_m; (F_\phi)_{\phi \in \Phi}) : \text{System} \quad \text{MECHANISM}
 \end{array}$$
  

$$\begin{array}{c}
 S_1, \dots, S_k : \text{System} \quad W : \overrightarrow{\text{WiringRow}} \quad E : \overrightarrow{\text{SwitchEdge}} \quad A : \overrightarrow{\text{ScheduleAssumption}} \\
 \hline
 \text{Composite}(S_1, \dots, S_k; W; E; A) : \text{System} \quad \text{COMPOSITE}
 \end{array}$$

The two primitive System formers. The family alphabet  $\mathcal{F}$  mirrors, group for group, the field families of the manifest schema (the `MechanismInit` groups of the API block below):

|                                                                                                                  |                         |
|------------------------------------------------------------------------------------------------------------------|-------------------------|
| $\mathcal{F} = \{\text{modes, drift, lagrangian, diffusion, noise, noise\_spectrum, jumps, kill, transitions}\}$ | continuous / hybrid     |
| $\cup \{\text{outcomes, init\_dist, rates, scheduler, circuit}\}$                                                | discrete-kernel         |
| $\cup \{\text{tick}\}$                                                                                           | counter time            |
| $\cup \{\text{petri, bag, sdcpn, stochastic\_wires, equations, interventions, exogenous, weighted\_program}\}$   | relational / population |

Each family’s row sort  $\text{Row}_\phi$  is the family’s own, given by the formation rule of the entry that consumes it. Every entry below is a checked specialization of Mechanism: its head form carries the formation data that individuates its type, and optional declarations (marked ?) are consumed without appearing in it. No premise constrains  $\Phi$  — any family set is well-formed formation data, and a mechanism whose populated set matches no preset is a first-class citizen. That the toolkit’s free entry point never refuses is the implementation face of this rule: a forward-compatibility property (families added to  $\mathcal{F}$  extend the schema without invalidating old manifests), not a logical feature. On the wire, either former’s term is carried in

the manifest record  $\langle \text{schema}, v, \text{name?}, \text{term} \rangle$ .  $\text{pop}(m) = \Phi$  is a view, recomputed on demand and never stored. The substrate families, admitted alongside every former's own formation data without being listed:

$$\Sigma = \{\text{state}, \text{params}, \text{interface}, \text{init}, \text{hypernet\_refs}, \text{pinning}\}.$$

## API

Producer surface of the mtk toolkit for this entry: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named  $\text{fx\_}$  or  $\text{f.}$  are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.buildSystem`

signature — `ts/src/system.ts`

```
export function buildSystem(init: MechanismInit & SystemOpts): System

export interface MechanismInit {
  // substrate
  readonly state?: ReadonlyArray<StateVar>;
  readonly jetVars?: ReadonlyArray<JetVar>;
  readonly params?: ReadonlyArray<ParamDecl>;
  readonly interface?: Interface;
  readonly init?: InitRegion;
  // continuous / hybrid
  readonly modes?: ReadonlyArray<Mode>;
  readonly drift?: ReadonlyArray<DriftRow>;
  /** S-ACTION: the Lagrangian family (dynamics derived, never
   * stored; coexisting `drift` rows carry a recompute obligation). */
  readonly lagrangian?: LagrangianDecl;
  readonly diffusion?: ReadonlyArray<DiffusionRow>;
  readonly noise?: ReadonlyArray<NoiseRow>;
  /** S-SPDE: noise-spectrum driver rows (populating either spectrum
   * field creates the family; the SPDE preset refuses a spectrum
   * whose `noiseTail` was never declared). */
  readonly noiseDrivers?: ReadonlyArray<NoiseDriverRow>;
  /** S-SPDE: the spectrum's required tail form – never defaulted. */
  readonly noiseTail?: TailForm;
  readonly jumps?: ReadonlyArray<JumpRow>;
  /** S-JUMPDIFF: killing rows (absent = conservative). */
  readonly kill?: ReadonlyArray<KillRow>;
  readonly transitions?: ReadonlyArray<GuardedTransition>;
  readonly hypernetRefs?: ReadonlyArray<HypernetRef>;
  // discrete-kernel
  readonly outcomes?: ReadonlyArray<OutcomeRow>;
  readonly initDist?: ReadonlyArray<InitDistRow>;
  readonly rates?: ReadonlyArray<RateEntry>;
  readonly scheduler?: SchedulerDecl;
  readonly circuit?: CircuitDecl;
  // tick
  readonly tick?: TickDecl;
  // structured-population / relational
  readonly petri?: PetriDecl;
  readonly bag?: BagDecl;
  readonly sdcpn?: SdcpnDecl;
  readonly stochasticWires?: ReadonlyArray<StochasticWire>;
  readonly equations?: ReadonlyArray<EquationRow>;
  readonly interventions?: ReadonlyArray<InterventionRow>;
  readonly exogenous?: ExogenousTable;
  readonly weightedProgram?: WeightedProgramDecl;
  // i.i.d. tier
  readonly pinning?: PinningDecl;
}
```

construction — `ts/tests/system-native.test.ts`

```

const init = {
  name: "ou-ld",
  state: [{ id: 0, name: "x" }],
  drift: [{ var: 0, poly: mono(1, 0, 1, "-1") }],
  diffusion: [{ var: 0, poly: constant(1, "1/2") }],
  init: { kind: "point", coords: [BIG] },
} as const;
const viaPreset = sys.unwrap(sys.sde(init));
const viaFree = sys.buildSystem(init);
assert.ok(viaPreset.contentEquals(viaFree));
assert.equal(toHex(viaPreset.toCbor()), toHex(viaFree.toCbor()));

```

The quoted corpus lines run the corpus's preset discipline as an assertion: a preset's output is byte-identical to buildSystem's on the same mechanism — presets add checking, never representation.

### Rust mtk\_catalogue::system::SystemManifest

signature — crates/mtk-catalogue/src/system/manifest.rs

```

pub fn free(mechanism: Mechanism) -> Self
pub fn composite(composite: Composite) -> Self

```

construction — crates/mtk-catalogue/tests/builders.rs

```

assert_eq!(known_profile(&drift_only(), Some("ODE")));
let free = SystemManifest::free(drift_only());
// Nothing on the wire stores the classification: same bytes with
// or without the preset having been consulted.
let via_preset = Ode::manifest(drift_only()).unwrap();
assert_eq!(free.to_cbor().unwrap(), via_preset.to_cbor().unwrap());

```

The quoted corpus lines assert the profile-not-tag doctrine on the wire: same bytes with or without the preset having been consulted.

## 12.2 Tick Declarations

**The entry.** The composition-critical counter-time vocabulary. Time is base-universal — composites live on real time — so a counter-time mechanism composed into real time declares its tick as a timer mechanism whose reset parameter is either a point (*exact tick*) or a box (*jitter bracket*). The distinction is part of the model type, not a checker convention, and it has been priced by certificate: a lander free-flight pair certifies the same funnel under both semantics from one value table and one cover, with the cost of clock jitter a factor of 2.66 in certified ticks-to-funnel (1631 against 614), and a factor 2.92 in the certified elapsed bound, since a jittered tick may run a tenth longer than an exact one. Both horizons are upper bounds carried by the certificate, not measured times. A discrete-time component in isolation remains schedule-generic; its schedule class becomes *assumption data* exactly at the discrete/continuous composition boundary, through the hybrid-time pattern of chapter 13.

### Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema's own (crates/mtk-catalogue/src/system/), and the former's mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{\text{period} \in \mathbb{Q}_{>0}}{\text{exact}(\text{period}) : \text{TickDecl}} \text{EXACT} \qquad \frac{\text{lo}, \text{hi} \in \mathbb{Q}}{\text{jitter}([\text{lo}, \text{hi}]) : \text{TickDecl}} \text{JITTER}$$

Not a System former: the conclusion is at the declaration's own sort. A TickDecl is carried either on the mechanism itself (tick  $\in \text{pop}(m)$ , admitted by the counter-time profiles) or in a composite's assumption slot as ScheduleAssumption = ⟨component, tick⟩.

## API

Producer surface of the mtk toolkit for this entry: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.TickDecl`

signature — `ts/src/system.ts`

```
/** S-TICK: tick declaration – exact tick or jitter bracket. Tick
 * semantics is part of the model type, not a checker convention. */
export type TickDecl =
  | { readonly kind: "exact"; readonly period: QLike }
  | { readonly kind: "jitter"; readonly lo: QLike; readonly hi: QLike };
```

construction — `ts/tests/system-native.test.ts`

```
return sys.unwrap(
  sys.dtmc({
    name: "chain-3",
    outcomes: [
      { src: 0, dst: 0, prob: "1/3", reward: "0" },
      { src: 0, dst: 1, prob: "2/3", reward: "1" },
      { src: 1, dst: 2, prob: "1", reward: "5" },
      { src: 2, dst: 2, prob: "1" },
    ],
    initDist: [{ state: 0, prob: "1" }],
    tick: { kind: "exact", period: "1/50" },
  }),
);
```

The quoted fixture declares an exact tick; the composite fixture’s schedule assumption (S-HYBTIME) carries the jitter-bracket form.

### Rust `mtk_catalogue::system::TickDecl`

signature — `crates/mtk-catalogue/src/system/manifest.rs`

```
/// System Catalogue: Tick declaration – the composition-critical
/// counter-time vocabulary. A counter-time mechanism composed into
/// real time declares its tick as a timer mechanism whose reset
/// parameter is a point (**exact tick**) or a box (**jitter
/// bracket**). Tick semantics is part of the model type, not a
/// checker convention (the LunarLander pair is the standing price
/// demonstration: a factor 2.66 in certified ticks-to-funnel).
#[derive(Debug, Clone, PartialEq, Serialize, Deserialize)]
#[serde(rename_all = "snake_case")]
pub enum TickDecl {
  /// Exact tick: reset parameter a point.
  Exact { period: WitRat },
  /// Jitter bracket: reset parameter a box `[lo, hi]`.
  Jitter { lo: WitRat, hi: WitRat },
}
```

construction — `crates/mtk-catalogue/tests/common/mod.rs`

```
let m = MechanismBuilder::new()
  .outcome(OutcomeRow {
    src: 0,
    action: None,
    dst: 0,
    prob: r(1, 3),
    reward: Some(r(0, 1)),
  })
  .outcome(OutcomeRow {
    src: 0,
    action: None,
    dst: 1,
```



```

        prob: r(2, 3),
        reward: Some(r(1, 1)),
    })
    .outcome(OutcomeRow {
        src: 1,
        action: None,
        dst: 2,
        prob: r(1, 1),
        reward: Some(r(5, 1)),
    })
    .outcome(OutcomeRow {
        src: 2,
        action: None,
        dst: 2,
        prob: r(1, 1),
        reward: None,
    })
    .init_dist(0, r(1, 1))
    .tick(TickDecl::Exact { period: r(1, 50) })
    .finish();
presets::Dtmc::manifest(m)
    .expect("fixture conforms to the DTMC profile")
    .named("chain-3")

```

## 12.3 Static Systems

**The frame.** A value chosen once and never revised: the unknown constant of an estimation problem — a parameter picked demonically from a box and frozen for all time — and the token System of a colourless Petri net under the Bag former of section 12.17, where Bag over a point is the naturals with counting semantics. The bread-and-butter question is membership: which values the initial region admits, and what a surrounding composite may conclude from the box that bounds them.

**Profile.** Substrate only — state variables, unboxed or boxed parameters, interface, demonic initial region — with no dynamics family populated: the bottom of the manifest family; every other profile of this chapter populates families beyond it. At any dimension it is the zero-drift degenerate of the drift-only frame; at dimension zero it is the only dynamics-capable reading, because the zero drift vector there *is* the empty vector — a drift roster empty by dimension, not by omission. Time: continuous, trivially; the trajectory never moves.

**Semantics.** One convention is fixed here and consumed by everything downstream: absent dynamics denote a *frozen* carrier, never an unconstrained one. Records are constant paths valued in the initial region; the demonic initial face is the same box convention as the deterministic siblings — the denotation is the union over the region of per-point singletons — so the record set is the initial region, read as paths.

**Adequacy.** Definitional at this frame, and stated rather than mechanized: both routes of the square of Figure 12.1 name the constant records valued in the initial region — compilation contributes the initial row and the frozen-carrier convention, classical interpretation contributes the constant path at each admitted point, and there is no analysis between them to be wrong about. At positive dimension the same record set is carved by the drift-only rows of Problem 12.1 at  $f = 0$ ; at dimension zero the carrier is a single point. No dedicated mechanization is claimed: the frame's load-bearing content is the convention, not a theorem.

**Status.** Vocabulary and remedy, not yet certificate: no shipped certificate consumes the profile at this preview — it enters the inventory as the landing pad other entries point at. Production: presets in both languages construct the manifest (the API block below). The distinguishing shipped artefact is the refusal pairing: a drift-requiring preset meeting a zero-dimensional, substrate-only mechanism refuses with a remedy naming this profile, and the property suite checks that refusal's geometry — family absent, state roster empty, and the named remedy accepting. One name collision is worth disambiguating here: elsewhere in this chapter the graphical-model sub-frames are called static because they have *no internal time axis* (section 12.20); this profile has the time axis, and nothing moves along it.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema's own (`crates/mtk-catalogue/src/system/`), and the former's mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ ,  $\text{pop}$ , and profile conformance are defined in section 12.1.

$$\frac{}{\text{Static} : \text{System}} \text{S-STATIC}$$

The axiom of the manifest family: no dynamics family is consumed, and the formation data beyond the substrate  $\Sigma$  of section 12.1 (state variables, possibly none; parameters; interface; demonic initial region) is empty. Any populated dynamics family is a stray — the preset refuses it at any dimension.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.staticProfile`

signature — `ts/src/system.ts`

```
/** The static profile: substrate only – state space, parameters,
 * interface, demonic initial region; no dynamics families. The
 * initial condition is chosen demonically from the initial region
 * and then never moves: the zero-drift degenerate of the ODE profile
 * at any dimension, the only reading at dimension zero (where the
 * zero drift vector IS the empty vector), the token System of a
 * colourless Petri net under the Bag former, and the
 * unknown-constant System (a boxed parameter frozen for all time).
 * (`static` is a reserved word in module code, so the export carries
 * the profile suffix.) */
export const staticProfile = profileFactory("static");
```

The TS conformance corpus holds no construction through this factory at this preview — byte authority for the profile is the Rust corpus opposite; the factory takes the same init-object vocabulary as every other profile.

### Rust `mtk_catalogue::system::presets::Static::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```
preset!(
    /// System Catalogue: static profile – substrate only (state
    /// space, unboxed or boxed parameters, interface, demonic
    /// initial region), no dynamics families. The initial condition
    /// is chosen demonically from the initial region and then never
    /// moves: the zero-drift degenerate of the ODE profile at any
    /// dimension, and the only dynamics-capable reading at dimension
    /// zero, where the zero drift vector IS the empty vector. Two
    /// standing uses: the token System of a colourless Petri net
    /// under the Bag former (Bag over a point is the naturals with
    /// counting semantics), and the unknown-constant System (a value
    /// picked from a box, frozen for all time).
    Static, "static",
    req = [],
    allowed = [],
    extra = ok
);
```

construction — crates/mtk-catalogue/tests/builders.rs

```
let m = MechanismBuilder::new()
    .init(InitRegion::Box {
        region: BoxRegion { axes: vec![] },
    })
    .finish();
let sys = Static::manifest(m.clone()).unwrap();

// The unknown-constant reading: a boxed parameter axis, frozen
// for all time — substrate-only, so the same profile.
let frozen = MechanismBuilder::new()
    .param(0, Some("theta"), Some([r(1, 2), r(3, 2)]))
    .finish();
Static::manifest(frozen).unwrap();
```

The quoted span is the builders-suite happy path: the zero-dimensional token System (an initial box with no axes — the unique point of the zero-dimensional carrier) and the unknown-constant System (a boxed parameter, frozen for all time). The profile is also a refusal remedy: a drift-requiring preset meeting a zero-dimensional, substrate-only mechanism refuses with a pointer to this preset by name, and the refusal suite checks that the named remedy actually accepts.

## 12.4 Ordinary Differential Equations

**The frame.**  $\dot{x} = f(x)$  on a declared domain; the bread-and-butter questions are invariance, reach-avoidance, and Lyapunov decrease, plus validated enclosure of flows when the question is quantitative.

**Profile.** Drift only — the noise-free, jump-free corner of the manifest family. Carrier: the declared state variables over exact-rational box or simplex domains; polynomial right-hand sides native, non-polynomial ones by rationalization or by certified abstraction (the latter designed, not shipped). Openness, when wanted, enters as demonic input wires, not as a stochastic field. Time: continuous.

**Semantics.** Trajectory semantics — the drift-only specialization of the System semantics. Certificates about flows (tubes, flow maps) are licensed by once-per-format meta-theorems: the kernel checks inclusion-invariance sequents, and the reading of those sequents as statements about solutions is a named, stated obligation rather than kernel code. In the universe's terms: records over the classical site, read at the precise or possibilistic face of the denotation (Remark 9.22) according as inputs are absent or demonic.

**Adequacy.** The drift-only frame carries the catalogue's simplest instance of the square of Figure 12.1, and its content is easy to misjudge as vacuous: what compilation produces is not the textbook solution concept but *forward* constraint rows, and the square is the theorem that the two meet. Stated self-contained, at the working scale of an exam problem:

**Problem 12.1** (adequacy of the drift-only compilation). Fix a continuous  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$  and  $x_0 \in \mathbb{R}^n$ . Records are paths  $y : \mathbb{R} \rightarrow \mathbb{R}^n$ ; only values at  $t \geq 0$  are observable. Say  $y$  satisfies the *compiled rows* when (i)  $y$  is continuous on  $[0, \infty)$  and at every  $t \geq 0$  has the forward derivative  $f(y(t))$  — the one-sided limit through times  $\geq t$  — and (ii)  $y(0) = x_0$ . Say  $y$  is a *classical solution* when  $y(0) = x_0$  and  $y$  is differentiable on  $[0, \infty)$  with  $\dot{y} = y \circ f$  (composition in diagrammatic order: first  $y$ , then  $f$ ) — two-sided at interior times, one-sided only at the corner  $t = 0$ . Show the two sets of records are equal, and show that the continuity clause in (i) cannot be dropped: exhibit a record with the correct forward derivative everywhere on  $[0, \infty)$  that solves nothing.

*Sketch.* Classical solutions satisfy the rows by restriction. Conversely the rows put  $y$  under the fundamental theorem of calculus twice: continuity of  $y$  (the record-class row) makes  $t \mapsto f(y(t))$  continuous, so the forward rows integrate — FTC for right derivatives — to the Picard identity  $y(t) = x_0 + \int_0^t f(y(s)) ds$ , the integral-equation form of the classical theory [46, 111]; then differentiating the integral at interior times gives the two-sided derivative. For the countermodel take  $f = 0$ ,  $x_0 = 0$ , and the step record  $y = k \mathbb{1}_{[t_0, \infty)}$  with  $k \neq 0$  and  $t_0 > 0$ : every forward derivative on  $[0, \infty)$  vanishes — right-continuity is free — but left-continuity fails at  $t_0$ , so the record-class row is load-bearing, not decorative.

In the universe’s terms: the two constraint families — the cell and the initial row — form a loose-arrow diagram at the possibilistic stratum, and the theorem says the diagram is isomorphic, by an invertible globular cell, to the support inclusion of the classical solution set: the same set-valued convention as the delay line (section 12.6), of which this square is the seam — the delay-line composite’s ODE-cell clause is this square’s row form at the field composed with the input signal, and the random-parameter frame (section 12.9) specializes it per draw. Three boundaries are stated rather than smoothed. Existence is not owed: the compiled atoms never exclude the vacuous model, blow-up in finite time ( $y = y^2$ ) can empty the solution set over  $[0, \infty)$  outright, and the constructive properness witness is the equilibrium record at a zero of the drift. Determinism is not a constraint-stratum fact: the atoms denote the solution *set*, and uniqueness enters as a theorem under the frame’s own declared-domain discipline — Lipschitz drift on the domain, solutions confined to it — consumed from the mechanized Grönwall interface, whose hypotheses are the forward rows verbatim: the community’s own uniqueness engine speaks constraint rows, not solution operators. And the connector consumes a completeness hypothesis on the carrier (the two FTC directions integrate), which the frame’s finite-dimensional state spaces satisfy; the bundling, the pinning theorems, and the wiring below do not consume it. The demonic initial face is the box convention of the mechanism declarations: its denotation is the union of the per-point solution sets, the deterministic frame’s shadow of the credal unions the stochastic squares use. The statement and both directions are mechanized, sorry-free, in the toolkit’s Lean tree (`lean/su-functor/`, module `SuFunctor.Adequacy.0de`): the connector is `isOdeSol_iff_integralCurve` — the classical side is the mechanized analysis library’s own integral-curve concept, not a private restatement — with the Picard form as `isOdeSol_iff_integral`, the invertible cell is `ode_adequacy` with the box face `ode_box_adequacy`, the pinning theorems are `odeSols_eq0n_of_lipschitz0nWith` and its global corollary, and the wiring into the semantic universe’s judgment (`su_ode_adequacy`) runs through the same transport layer as the sibling squares, with one possibilistic contrast stated deliberately: the zero law’s ideal point models every compiled presentation — support constraints never force realization, and the demand that the system realize anything at all is judgment-level content.

**Status.** Certification is in place twice over. Closed-loop controller certificates — an inverted-pendulum family, and a ladder re-verification of an aircraft collision-avoidance controller family — check through the unmodified kernel. The validated-enclosure species ladder de-trusts an established rigorous integrator: candidates from any proposer are re-founded on exact fixed-point inclusion checks, so the assurance story never rests on trusting interval arithmetic (the worked example of section 19.1). This frame also holds the inventory’s only cross-frame composition witness at this preview: an ODE plant wired to a discrete-time controller, with the assume-guarantee square transporting the certificate across the boundary and the composite checked by the unmodified kernel — every mutant of the composite refuses. Production: the toolkit’s producer surface constructs presentations in both languages and derives the Lie-derivative row  $\mathcal{A}V = \nabla V \cdot f$  symbolically from the presentation — the exact polynomial whose sign conditions the martingale-problem headlines of the stochastic siblings (sections 12.7 and 12.10) consume verbatim at their pure-drift face; derived and never trusted, the checker re-derives it. The double-integrator plant of the API block below is a committed standalone cross-language byte-identity vector whose conserved-energy zero the kit’s tests pin in closed form in both languages, and the kit is the jump-diffusion assembler at its degenerate face, the delegation warranted by the mechanized pure-drift degeneracy. Open: mechanism declarations exist as staged rows, but the ingestion path from rows to a checked model is not yet exercised end to end — a gap this frame shares with most of the list, named once here and cited below as “the ingestion gap”; and the enclosure ladder’s upper rungs rest on meta-theorems outside the kernel, an explicit trust boundary the species documentation states.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{\vec{x} \text{ declared state variables} \quad D : \overrightarrow{(\text{var}, \text{poly})} \text{ drift rows, each } \text{poly} \in \mathbb{Q}[\vec{x}]}{\text{ODE}(\vec{x}; D) : \text{System}} \quad \text{S-ODE}$$

Drift is the only dynamics family consumed. Openness, when wanted, enters as demonic input wires on *m.interface* — interface wires are substrate, never a stochastic family. At dimension zero the drift family cannot be populated at all — the roster is empty by dimension — and the preset refuses with a remedy naming the static profile (section 12.3).

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named *fx\_\** or *f.\** are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.ode`

signature — `ts/src/system.ts`

```
/** S-ODE: drift only (openness, if any, enters as demonic input wires
 * on the interface). Checks the family signature, then wraps into the
 * same manifest type as every other entry point – presets add
 * checking, never representation. */
export const ode = profileFactory("ode");

/** The derived ODE generator kit: the exact Lie-derivative
 * polynomial  $\dot{V} = \nabla V \cdot f$ . Derived, never trusted – the checker
 * re-derives it. */
export interface OdeGenKit {
  readonly gen: PolyWitness;
}

/** Derive the ODE generator kit from a mechanism declaration and a
 * certificate observable. Admission is the ODE profile's – diffusion,
 * jump, or killing rows refuse naming the stray family (those
 * presentations belong to `sdeGenKit` / `jdGenKit`); a parameter
 * roster is substrate and does not refuse, while drift naming a
 * parameter axis refuses at the arity fence (that presentation
 * belongs to `rodeGenKit`) – and the computed polynomial's identity
 * with the semantic generator at the drift-only face is mechanized
 * in the toolkit's Lean tree, whose ODE certificate theorems consume
 * the derived row's sign conditions verbatim. */
export function odeGenKit(
  init: MechanismInit,
  cert: Polynomial,
): Result<OdeGenKit, Refusal> {
  const raw = native.sysOdeGenKit(
    mechanism(init).nativeHandle,
    polyToNative(cert),
  ) as { kit: OdeGenKit } | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}
```

construction — `ts/tests/system-native.test.ts`

```
return sys.unwrap(
  sys.ode({
    name: "double-integrator",
    state: [
      { id: 0, name: "pos" },
      { id: 1, name: "vel" },
    ],
    interface: {
      inputs: [{ id: 0, name: "u" }],
      outputs: [{ id: 1, name: "y" }],
    },
  })
```

```

    drift: [
      { var: 0, poly: mono(2, 1, 1, "1") },
      { var: 1, poly: mono(2, 0, 1, "-1") },
    ],
  }},
);

```

**Rust** `mtk_catalogue::system::presets::ode::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```

preset!(
  /// System Catalogue: ODE profile — drift only (openness, if any,
  /// enters as demonic input wires on the interface).
  Ode, "ODE",
  req = [Drift],
  allowed = [],
  extra = ok
);

```

construction — `crates/mtk-catalogue/tests/common/mod.rs`

```

let m = MechanismBuilder::new()
  .state_var(0, Some("pos"))
  .state_var(1, Some("vel"))
  .interface(Interface {
    inputs: vec![WireDecl {
      id: 0,
      name: Some("u".to_owned()),
      sort: None,
    }],
    outputs: vec![WireDecl {
      id: 1,
      name: Some("y".to_owned()),
      sort: None,
    }],
  })
  .drift(None, 0, mono(2, 1, 1, r(1, 1)))
  .drift(None, 1, mono(2, 0, 1, r(-1, 1)))
  .finish();
presets::ode::manifest(m)
  .expect("fixture conforms to the ODE profile")
  .named("double-integrator")

```

The construction is the committed double-integrator golden vector (fixture ode, standalone byte authority; the same presentation serves the composite golden as its plant component), and it doubles as the generator kit's conserved-energy closed-form control ( $A(\text{pos}^2 + \text{vel}^2) = 0$ ); the kit's Rust face is `mtk_catalogue::system::ode_gen_kit`.

## 12.5 Partial Differential Equations

**The frame.** Evolution equations  $\partial_t u = F(u, \partial_x u, \partial_x^2 u, \dots)$ ; the bread-and-butter certified questions are energy dissipation and invariance.

**Profile.** A differential-polynomial mechanism over declared jet variables and a spatial domain — a sibling extension at the kernel's own level: integration-by-parts structure enters as exact polynomial-syzygy rows, pointwise nonnegativity as cone rows, and the checker gains nothing beyond jet arithmetic it already has. Time: continuous. Deterministic; the stochastic half is section 12.8's, which adds exactly one family to this profile.

**Semantics.** A solution-concept *run*g — classical, strong, or weak — with the semantic bridge (the finitely many analysis lemmas licensing the algebra-to-solutions reading) entering as named obligations, since largely discharged for this frame (see Status). This obligation discipline was established here first and the rest of the inventory inherits it.



**Adequacy (heat on a cyclic mesh).** Adequacy for a composed PDE says that compiling a spatial mesh of cells and walls, then denoting, agrees with interpreting the equation globally and including its solution set — and the first instance where the mesh is genuinely a *cover* is the heat equation on a circle, cut into three overlapping arcs. The cover is cyclic: the arcs meet in three collars, and the third collar closes the ring through the periodic identification. Cyclic covers are where local-to-global inference gets dangerous — Vorob’ev exhibited pairwise-consistent families of finite marginals over a three-cell ring with no joint law at all [235] — so the instance is chosen to make the mesh earn its consistency rather than inherit it.

**Problem 12.2** (heat on a cyclic mesh). Fix a diffusivity  $D$ , a circumference  $L$ , arc breakpoints  $0 < a < b < L$ , a collar half-width  $0 < \delta < L$ , and an open time window  $T$ . On the universal chart, the three arcs are  $I_1 = (-\delta, a + \delta)$ ,  $I_2 = (a - \delta, b + \delta)$ ,  $I_3 = (b - \delta, L + \delta)$ . A mesh record is a triple of cell fields  $(u_1, u_2, u_3)$ , and the compiled constraints are six: on each arc, the field is continuous, twice differentiable in space, and satisfies  $\partial_t u_i = D \partial_{xx} u_i$  pointwise (the cell rows);  $u_1 = u_2$  on  $I_1 \cap I_2$  and  $u_2 = u_3$  on  $I_2 \cap I_3$  at observed times (the chain walls); and  $u_3(x) = u_1(x - L)$  on the ring-closing collar (the wrap wall). Show that a record satisfies all six constraints if and only if there is a single  $L$ -periodic global solution of the heat equation agreeing with every cell field on its arc. Show moreover that the wrap wall is not implied by the rest: the linear field  $u_i(x, t) = x$  satisfies every cell row and both chain walls on all three arcs, and fails the wrap wall alone.

*Sketch.* ( $\Rightarrow$ ) Paste the two chain seams first: the piecewise field agrees with some cell field on a neighbourhood of every chart point, the walls supplying agreement across the collars, and the heat rows are germ-determined — differential operators are local — so the paste solves on the chart. Then periodize: the wrap wall is consumed exactly once, as the consistency identity of the periodic extension at the ring-closing collar, and locality carries the rows to the whole line. ( $\Leftarrow$ ) Clipping a global periodic solution to the arcs satisfies the cell rows by restriction, the chain walls because all three cells read one field, and the wrap wall by periodicity. No parabolic theory appears in either direction — no semigroup, no regularity, no maximum principle: the square lives at the record stratum, where gluing is descent plus locality.

In the universe’s terms: the six constraint families form a loose-arrow diagram of the thin double category of per-element constraints, walls and cells alike read at the possibilistic stratum, conjoined once at the mesh joint — and composition order is immaterial, since satisfaction of a conjunction reads through membership only. The theorem says the diagram is isomorphic, by an invertible globular cell, to the support inclusion of the global periodic solution set; as in the transport instance, no constraint forces mass, and the demand that the mesh realize anything is judgment-level, discharged constructively by the equilibria. The cyclic cover’s extra obligation is a *holonomy* constraint, carried as an ordinary reusable wall: the linear field — a perfectly good local solution family that fails to descend to the circle — passes every per-cell and pairwise-chain check and is refused by the wrap wall alone, and no global periodic field restricts to it. This is the two rings’ contrast made precise: the frustrated marginal ring fails invisibly, at the measure tier, where consistency of laws carries no global witness and the lawful credal denotation collapses to the bottom object; the heat ring’s obligations are record-tier, local, and visible as walls, its denotation is satisfiable, and local-to-global gluing is a theorem. The statement and both directions are mechanized, sorry-free, in the toolkit’s Lean tree (lean/su-functor/, module SuFunctor.Adequacy.HeatTorus): the ring glue is `heat_mesh_glues`, the invertible cell is `heat_adequacy`, the holonomy witness pair is `chainAtoms_dirac_lin` with `not_wrap_lin`, the interval control (cutting the ring trivializes the obligation) is `heat_chain_glues`, and the marginal-ring obstruction and its credal absorption are `vorobev_no_joint` and `credalCompile_vorobevGlue` in `SuFunctor.Adequacy.MarginalGlue`, with the two rings’ contrast packaged as a single statement, `heat_ring_glues_vorobev_ring_not`. Two strata are named separate, deliberately: with finitely many test rows in place of the pointwise rows, the compiled constraints are implied by solution-hood and do not pin it, the equivalence returning only in the mesh-refinement limit; and the exact-rational certificate stratum carries the dynamics as outward-rounded enclosures, never equalities —  $e^{-\lambda t}$  is transcendental at rational times, so the only exactly-propagated mode is the conserved mean. Neither stratum is mechanized; both carry recorded cost estimates — estimates, not available squares. The sign of  $D$  is consumed by neither direction of the square: certified decay, and with it the direction of well-posedness, is enclosure-stratum content, not a satisfaction clause.

**Adequacy (the initial face as a port).** The mesh square above keeps its time windows open — interior discipline; no statement mentions  $t = 0$ . The suite’s seventh square wires the initial condition as an *exposed port*: the initial face enters as a boundary trace along the window’s own filter — the field on each spatial

ray attains the port datum in the limit from inside the window, never as a row at an interior point — and three port atoms, one per arc, extend the six mesh atoms to nine. The ported square closes, still two-sided at the pointwise-record stratum. Its most important content is a deliberate absence: *feeding* the port — exhibiting, for a given datum, a global periodic solution attaining its trace — is a named definition, and the consistency discharge is conditional on a feed, because for general admissible data the feed *is* the heat semigroup on the circle [70]: enclosure-stratum content, cited and never mechanized — the heat analogue of the chain square’s Ionescu–Tulcea stratum, with the unconditional consistency theorem deliberately absent. The constructive floor is mechanized without any semigroup: equilibria feed constant data, and a single Fourier mode feeds a genuinely nonconstant datum — what the semigroup buys is only the limit over infinitely many modes for merely-continuous data. Compatibility conditions on the datum are *derived*, never assumed: the wrap wall plus the traces force the datum’s holonomy identity on the ring-closing collar — the delay line’s corner-condition lesson recurring one cover rung up — and periodicity of a global solution forces the same identity from the classical side. The face’s only new fee is the non-triviality of the window’s boundary filter — without it the traces are vacuous and the compatibility statements fail, so the fee is load-bearing — discharged for the concrete window. The sign of  $D$  remains nowhere, even at the port: backward heat feeds the same single mode, growing, so the record stratum still cannot see the direction of well-posedness — uniqueness of the fed solution, and its genuine failure backward, are enclosure-stratum business, cited with the semigroup. Mechanized, sorry-free, beside the mesh square (lean/su-functor/, module `SuFunctor.Adequacy.HeatTorusInit`): the invertible cell is `heat_init_adequacy`, the conditional consistency discharge is `heatInitAtoms_consistent_of_feed`, the derived wrap-compatibility pair is `initFace_wrap_compat` with its classical-side twin `IsHeatTorusSolInit.trace_wrap_compat`, the constructive floor is `portFeed_const` and `portFeed_mode` (with `modeDatum_ne_const`), and the filter fee’s discharge is `initFace_neBot`.

**Adequacy (solution sets on the mesh).** Where a single field glues, a *set* of fields need not: nonuniqueness enters the mesh as imprecision — each cell carrying a closed set of local solutions, its observable face the union of their graphs — and at that stratum the descent question splits cleanly in two. Descent of the closed sets themselves is exact: over any open cover of the window, closed subsets of the spacetime-value ambient correspond homeomorphically, in the Fell topology, to overlap-compatible families of closed subsets of the cells — a sheaf condition in equalizer form, demanding neither directedness nor finiteness of the cover — and the ring’s three arcs are its first working instance. What fails is citizenship transport. The witness is the integer orbit of the linear field — all period-translates at once, a closed set because the branch set is discrete. Its every clip is a union of cell-solution graphs; its clip family is overlap-compatible; it satisfies the set-level wrap wall (the orbit is invariant under the period shift, so the wall that refused the single linear field is blind to the full orbit); it is even a union of window-solution graphs. Yet it is the graph-union of no family of periodic solutions whatsoever: a global solution threading the tube is pinned to a single branch — continuity against a discrete branch set — and single branches do not close up. The gluing that one refusable wall purchased at the record stratum is purchasable by no pointwise wall at the record-set stratum: the missing datum is the pairing of branches across seams — which branch continues which around the ring, correspondence data subject to a cocycle condition — and forgetting exactly that labelling is the whole failure, since labelled families glue member by member through the record-stratum theorem. Consistency, by contrast, is free at this stratum: the top tube — every value at every point, maximal imprecision — is the graph-union of all constant equilibria and satisfies every wall, so the consistency obligation concentrates entirely on non-vacuous imprecision. And the failure remains detectable one stratum down, as consistency of a composed object rather than as any wall’s refusal: compiling the orbit tube to per-cell enclosure constraints and conjoining with the mesh’s own rows forces the empty model — the branch indices are frustrated around the ring by an integer cocycle, the seams forcing them equal and the wrap wall forcing a shift. All of it is mechanized, sorry-free, beside the heat module (lean/su-functor/, module `SuFunctor.Adequacy.SolutionSetMesh`): the descent equivalence is `FellClosedEqualizerHomeomorph` instantiated as `meshDescent`, the packaged witness is `setStratum_holonomy` with `orbitTube_not_torusTube`, the failure of local character is `not_isLocalLocusEq_torusTube`, the one-sidedness of the compiled square is `tubeAtoms_not_classical` against the forward cell `solset_forward_cell`, the free consistency witness is `isTorusTube_univ`, the labelled positive face is `labelled_ensembles_glue`, and the record-tier shadow is `orbit_enclosure_forces_zero`.

**Adequacy (the frame, at every mechanism).** The three squares above are instances — heat, with one operator hard-coded. The suite’s frame-level statement proves the mesh square once for *every* mechanism

of the formation rule, and in doing so isolates exactly what a solution concept owes the mesh: two lemmas.

**Problem 12.3** (the mesh square, at every mechanism). Let a mechanism assign to each of finitely many components a drift polynomial in the components' spatial jets through a declared order, and take as its pointwise rows: joint continuity of each component, spatial differentiability of each iterated derivative below the declared order in canonical-value form, and the drift equations — each component's time derivative equals its polynomial read on the jets. On the cyclic three-arc mesh, with the same six constraint families as the heat square, show: (a) the two *interface lemmas* hold uniformly in the mechanism — germ locality (agreement on an open rectangle transports the rows) and spatial homogeneity (translates of solutions solve at translated points); (b) for any solution concept satisfying the two lemmas, a mesh record satisfies all six constraints if and only if a single  $L$ -periodic global solution of the concept agrees with every cell field on its arc; (c) constants at roots of the drift's reaction structure satisfy the compiled conjunction, and  $1/2$  is not a root of Fisher–KPP's.

*Sketch.* (a) Locality is an induction on jet order — agreement on an open set transports every iterated derivative, pointwise on the set — and homogeneity an induction from the first-order translation identity, junk values included; neither consumes any property of the polynomial beyond its evaluation. (b) The forward direction is the heat square's paste-and-periodize argument verbatim, with the interface fields in place of the heat lemmas: the chain paste consumes locality alone, the ring-closing periodization additionally consumes homogeneity — locality buys the interval, homogeneity buys the holonomy leg. Backward is clipping. (c) The jet read of a constant is its value at order zero and zero above, so the drift equations reduce to the reaction structure's root condition; evaluating Fisher–KPP's reaction at  $1/2$  gives  $1/4 \neq 0$ .

The mechanized statement is parametric twice over. Over *mechanisms*: the heat square is re-derived as a specialization — the compiled atoms of the heat instance are the frame square's at the heat concept definitionally, and the heat rows are the jet rows of the declared second-order mechanism, so the instance's hard-coded operator and the formation rule's jet data present the same constraints — while viscous Burgers, Fisher–KPP, and a cross-coupled second-order pair receive their squares as one-line corollaries, with zero new inductions. Over *solution concepts*: a rung enters as data — its rows in germ form together with the named obligations the square consumes (locality, at weak rungs the partition-of-unity argument of the literature [70]; homogeneity; the strong-rung inclusion) — and every rung meeting them inherits the two-sided square, plus a one-directional cell from the pointwise rung's compiled mesh to its own, the solution-concept ladder's enclosure-shaped transfer. At the generality of the formation rule there is no canonical weak form — a fully nonlinear differential polynomial admits none, and weak formulations are per-structural-class — so the rung interface is the strongest true frame-level statement, with each class's community tie an instantiation obligation. Consistency is equilibrium-conditional at this level: heat and Burgers admit every constant, Fisher–KPP exactly two, and a mechanism without real equilibria defers non-vacuity to existence theory, the named separate stratum. The sign blindness of the heat square is now uniform: no statement consumes any property of the drift polynomial beyond evaluation. Mechanized, sorry-free, beside the heat module (`lean/su-functor/`, module `SuFunctor.Adequacy.Pde`): the concept-generic square is `LocalRows.mesh_adequacy`, the interface lemmas are `PdeAt.congr` and `PdeAt.comp_sub_const`, the frame square is `pde_adequacy`, the heat re-derivation is `heat_adequacy_of_mesh` with the jet-grain tie `heatAt_iff_pdeAt`, the instance corollaries are `burgers_adequacy`, `kpp_adequacy`, and `coupled_adequacy`, the rung ladder is `PdeRung` with `rung_adequacy` and `pde_rung_transfer`, the consistency witness is `pdeAtoms_consistent_of_equilibrium` with the non-equilibrium receipt `kpp_not_equilibrium_half`, and the semantic-universe wiring is `su_pde_adequacy`.

The suite also carries the *declared-data* face of the same statement (module `SuFunctor.Adequacy.PdeRows`): a row specification — state variables, jet declarations, sparse exponent-tuple drift rows over the positional index space `state ++ jets` — compiles into the formation rule's mechanism (`PdeRowSpec.toMechanism`), the compile provably reads no position beyond a row's declared width (`toMv_congr`, `toMechanism_congr_inhabited`), and the four instance mechanisms above are re-derived *from declared rows* as polynomial identities (`heatRowSpec_toMechanism` and `kin`, at the certified constants), so the squares and the semantic-universe wiring specialize to presented data with no new proof (`heat_rows_adequacy`, `su_pde_rows_adequacy`).

**Status.** Certification: four exact-rational artefacts — viscous Burgers energy dissipation at the sharp rate (every mutant of its battery refuses), a Fisher–KPP certificate that is the family's first to *need* a box constraint, and Kuramoto–Sivashinsky twice, checked by an independently implemented exact checker. To

our knowledge, exact-certified PDE dissipation bounds with replayable proof objects do not appear in the surveyed literature, whose own methods sections describe floating-point and semidefinite-programming foundations; this frame was also a not-covered cell in the 2023 SDCPN grading. The bridge lemmas, staged as named obligations when this catalogue was first graded, have since been largely discharged in mechanized form: axiom-clean Lean theorems now cover the classical licensing lemmas, the Fourier bridges, the energy laws at their sharp decay rates, uniqueness for viscous Burgers at the classical rung, and the weak-rung transfer of every certificate conclusion — the last consuming nothing beyond the solution-class definition, Parseval, and elementary iteration: a machine-checked confirmation that certificate soundness never rested on the hard existence theory. Representation and production: the toolkit constructs jet-grain presentations in both languages and derives the canonical differential-polynomial right-hand side from the declared rows (the producer face of the mechanized compile edge, shared with the stochastic sibling's kit, section 12.8); the viscous-Burgers mechanism at jet grain is a committed cross-language byte-identity vector, and the API block below quotes its receipted construction. Deliberately open: one comparison-principle lemma and the weak-rung energy-inequality variants of the fourth-order family, each small and well-scoped; classical existence itself — needed for non-vacuity, never for soundness; boundary-condition classes beyond the exercised torus and mean-zero settings; certificate artefacts for PDE mechanisms inside compositions (the mesh square above, Problem 12.2, carries the semantic square for the first such composition); and the ingestion gap.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema's own (`crates/mtk-catalogue/src/system/`), and the former's mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{J : \overrightarrow{\langle \text{base}, \alpha \rangle} \text{ jet variables} \quad \begin{array}{l} \vec{x} \text{ declared state variables} \\ D \text{ drift rows over the extended index space } (\vec{x}, J) \end{array}}{\text{PDE}(\vec{x}; J; D) : \text{System}} \text{ S-PDE}$$

Jet variables extend the polynomial index space (state variables first, jet variables after), each declaring its base field variable and spatial multi-index  $\alpha$ ; drift rows over that space are the differential-polynomial mechanism.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.pde`

signature — `ts/src/system.ts`

```
/** S-PDE: differential-polynomial mechanism over jet variables,
 * deterministic (the stochastic arm is `spde`, which adds the
 * noise-spectrum family to exactly this signature). */
export const pde = profileFactory("pde");

/** The derived PDE kit: the canonical differential-polynomial
 * right-hand side, one exact polynomial per field component at the
 * full index-space arity `state ++ jetVars` (rows targeting one
```

```

* component sum) – the producer face of the declared-data compile
* edge, mechanized in the toolkit's Lean tree. Derived, never
* trusted – the checker re-derives it. */
export interface PdeGenKit {
  readonly rhs: ReadonlyArray<PolyWitness>;
}

/** Derive the PDE kit from a mechanism declaration. The stratum
* fences (single implicit mode, no transcendental terms, drift rows
* targeting field components only – a jet variable has no
* independent evolution row) come back as refusals naming their
* geometry. */
export function pdeGenKit(init: MechanismInit): Result<PdeGenKit, Refusal> {
  const raw = native.sysPdeGenKit(mechanism(init).nativeHandle) as
    | { kit: PdeGenKit }
    | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}

```

construction — ts/tests/system-native.test.ts

```

// Viscous Burgers at jet grain – the deterministic sibling of the
// stochastic-heat fixture, at the reference constants: index space
// (u, u_x, u_xx), the whole equation as one differential
// polynomial d/dt u = (1/10) u_xx - u u_x.
return sys.unwrap(
  sys.pde({
    name: "viscous-burgers-jet",
    state: [{ id: 0, name: "u" }],
    jetVars: [
      { id: 1, base: 0, deriv: [1] },
      { id: 2, base: 0, deriv: [2] },
    ],
    drift: [
      {
        var: 0,
        poly: {
          vars: 3,
          terms: [
            { exps: [1, 1, 0], coeff: "-1" },
            { exps: [0, 0, 1], coeff: "1/10" },
          ],
        },
      },
    ],
  }),
);

```

**Rust** mtk\_catalogue::system::presets::Pde::manifest

signature — crates/mtk-catalogue/src/system/presets.rs

```

preset!(
  /// System Catalogue: PDE profile – differential-polynomial
  /// mechanism over jet variables, deterministic. The stochastic
  /// arm is the `Spde` preset's, which adds the noise-spectrum
  /// family to exactly this signature.
  Pde, "PDE",
  req = [JetVars, Drift],
  allowed = [],
  extra = ok
);

```

construction — crates/mtk-catalogue/tests/builders.rs

```

let pde = MechanismBuilder::new()
  .state_var(0, Some("u"))
  .jet_var(1, 0, vec![2])
  .drift(None, 0, mono(2, 1, 1, r(1, 1)))
  .finish();
Pde::manifest(pde).unwrap();

```



The derived kit's Rust face is `mtk_catalogue::system::pde_gen_kit` — the canonical differential-polynomial right-hand side per field component at the extended index space, the producer face of the mechanized declared-data compile edge; the quoted TS construction is the committed viscous-burgers-jet golden vector's own presentation.

## 12.6 Delay Differential Equations

**The frame.** Delay differential equations  $\dot{y}(t) = f(y(t), y(t - \tau))$  with delay  $\tau > 0$  and a continuous history  $\varphi$  on  $[-\tau, 0]$ , in the classical sense of Hale–Verduyn Lunel [107]: solutions are continuous on  $[-\tau, \infty)$ , agree with the history, and satisfy the equation forward in time from  $t = 0$ . The textbook state space is the *history segment*  $y|_{[t-\tau, t]}$ , an element of the infinite-dimensional space  $C([-\tau, 0])$  — the feature that makes delay equations look like a new semantic species.

**Profile: none — a wiring pattern.** This frame adds no formation rule, no new mechanism family, and no new semantic species. A delay equation is a composite of inventory the catalogue already carries: an ODE cell (section 12.4) wired to an *ideal delay line* — the transport equation  $\partial_t u + c \partial_x u = 0$  on a segment of length  $c\tau$ , the simplest instance of the PDE frame (section 12.5)<sup>1</sup> — through three interface wires: a *load* wire placing the history on the line at time zero, an *inflow* trace wire pinning the upstream face to the cell's state, and an *outflow* tap reading the downstream face into the cell's vector field. The wires are directional in exactly the way the transport operator's dependence structure dictates: characteristics enter at the inflow and initial faces, which are the faces the wiring pins; the downstream face is read, never pinned. The infinite-dimensional history state is then nothing but the field on the spatial cell, read along characteristics — a record of a cell the universe already has, not a new kind of state.

**Adequacy.** The reduction is not an analogy; it is a theorem, mechanized end to end. Stated self-contained, at the working scale of an exam problem:

**Problem 12.4** (the delay line is a composite). Fix  $c, \tau > 0$ , a vector field  $f : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ , and a continuous history  $\varphi : [-\tau, 0] \rightarrow \mathbb{R}^n$ . On the cylinder  $[0, c\tau] \times [0, \infty)$ , say a triple  $(y, z, u)$  — an  $\mathbb{R}^n$ -valued path, a tap signal, and a field — satisfies the compiled constraints when: (i) along every characteristic  $s \mapsto (c(s - \zeta), s)$  of the cylinder, the field's trace is continuous with vanishing derivative on its window (the transport rows); (ii)  $u(x, 0) = \varphi(-x/c)$  (the load); (iii)  $u(0, t) = y(t)$  (the inflow); (iv)  $z(t) = u(c\tau, t)$  (the tap); and (v)  $y(0) = \varphi(0)$ ,  $y$  is continuous, and  $\dot{y}(t) = f(y(t), z(t))$  forward in time (the ODE cell). Show that  $(y, z, u)$  satisfies the compiled constraints if and only if there is a classical solution  $Y$  of the delay equation with history  $\varphi$  such that, on the composite's domains,  $y = Y$ ,  $z(t) = Y(t - \tau)$ , and  $u(x, t) = Y(t - x/c)$ . In particular the tap carries the state delayed by exactly  $\tau$ , and the spatial slice  $x \mapsto u(x, t)$  carries exactly the history segment  $Y|_{[t-\tau, t]}$ .

*Sketch.* ( $\Leftarrow$ ) Along each characteristic the coordinate  $t - x/c$  is constant, so the reparameterized read of  $Y$  is constant on each trace and the transport rows hold; the wire clauses are the definitional reads of  $Y$ . ( $\Rightarrow$ ) The one-variable mean value theorem pins each trace to its entry value — the inflow face for characteristics launched at  $\zeta \geq 0$ , the initial face for  $\zeta \in [-\tau, 0)$  — so the field is the concatenation of history and state read along characteristics (the method of characteristics [70], in its entirety); the tap then reads that concatenation at  $t - \tau$ , the ODE row becomes the delay equation, and the corner compatibility where the inflow face meets the initial face is precisely the initial condition  $y(0) = \varphi(0)$ . No stepwise construction appears: the correspondence is uniform in  $t$ , and the method of steps is an existence technique the equivalence never consumes.

In the universe's terms: the five constraint families — two cells and three wires, the wiring diagram read clause for clause — form a loose-arrow diagram of the thin double category of per-element constraints, each clause read at the possibilistic stratum (every record the model deems possible satisfies it); the theorem says this diagram is isomorphic, by an invertible globular cell, to the support inclusion of the classical solution set. The inclusion is set-valued because the square presupposes no uniqueness theory — solution sets play the role that credal sets play for the stochastic frames — and the per-solution face recovers the pointwise deterministic inclusion: each classical solution's point mass satisfies the compiled constraints. Unlike the stochastic squares,

<sup>1</sup> And the idealization of a real machine: the mercury acoustic delay-line memories of the earliest stored-program computers held their bits as waves in transit down a tube, read at the far end and re-injected at the near one — a store whose state is a field on a spatial segment, as the wiring pattern reads the history.



no constraint forces total mass: the possibilistic clauses are closed under sub-mass and mixtures, and the demand that the system realize anything at all is judgment-level content, discharged constructively at equilibria of the closed loop. The statement and both directions are mechanized, sorry-free, in the toolkit's Lean tree (`lean/su-functor/`, module `SuFunctor.Adequacy.DelayLine`): the record-level equivalence is `composite_iff_dde`, the invertible cell is `delayline_adequacy`, the delay identity is `outflow_delay`, and the history-segment identification is `field_slice_is_history`. Where the PDE frame's semantic bridge is staged as named obligations, this instance's left edge is total: characteristics reduce it to elementary one-variable calculus, which is what makes the delay line the transport instance whose adequacy square closes two-sided today. Dynamics enter as constraint rows, never as a solution operator: the shift semigroup the functional-analytic literature takes as primitive is *derived* content of this left edge.

**Status.** The wiring-pattern theorem and the square are mechanized as above. Certification inherits the PDE frame's machinery in its exactly-solvable corner: for rational  $c$ ,  $\tau$ , and piecewise-polynomial data with rational breakpoints, solution-by-characteristics is exact rational arithmetic — the field's natural finite presentation is its Bernstein coefficient vector in the characteristic coordinate, on which transport acts as the identity (the read head moves; the tape stands still), and endpoint traces are endpoint coefficients. Multiple discrete delays are multiple transport cells in parallel, a corollary of the same pattern. Deliberately open: an ingestion profile for the wiring pattern (the chapter's recurring lesson applies here too); Lyapunov–Krasovskii functionals read as cell-wise energies with flux cancellation at the wires, which is designed, not built; and state-dependent or distributed delays, which leave the ideal-line instance and are out of scope.

No formation rule accompanies this section: that absence is the section's content.

## 12.7 Stochastic Differential Equations

**The frame.**  $dX = f(X)dt + \sigma(X)dW$ ; certified questions are reach-avoidance and invariance in probability, via supermartingale arguments on the generator. The one-dimensional Ornstein–Uhlenbeck process  $dX = -Xdt + \frac{1}{2}dW$  is the frame's harmonic oscillator: the construction quoted in the API block below, and the first shipped certificates' subject (see Status).

**Profile.** Drift plus diffusion. Rank-one noise populates a single diffusion column; correlated multi-driver noise populates one exact-rational  $\sigma$  column per independent Brownian driver, with the covariance  $a = \sum_k \sigma^k (\sigma^k)^\top$  *derived, never stored* — a representation choice that keeps the checker's inputs primitive. State-dependent  $\sigma(x)$  is representable (the columns are polynomials). Time: continuous.

**Semantics.** The extended-generator rung: certificates are polynomial inequalities on  $\mathcal{A}V = f \cdot \nabla V + \frac{1}{2} \sum_{ij} a_{ij} \partial_{ij} V$ , and the reading of an accepted sequent as a probability bound rests on staged Dynkin and optional-sampling hypotheses — named, textbook-standard for polynomial coefficients on compacts, and outside the kernel. The bound so read is certificate membership in the denotation's defining infimum, on the per-element route whose continuous-time scope section 9.5 prices. The Adequacy paragraph below discharges the staged hypotheses for solutions presented by their martingale problem; what stays staged is scoped to presentations that are not.

**Adequacy.** The square for this frame closes at the community's own solution concept — not the Itô integral, which the mechanized library deliberately does not assume, but the *martingale problem* [69, 227]:  $X$  solves the martingale problem for  $\mathcal{A}$  when, for every admissible test observable  $V$ , the compensated process  $V(X_t) - \int_0^t (\mathcal{A}V)(X_s) ds$  is a martingale. Weak Itô solutions and martingale-problem solutions coincide; that equivalence is the community's own theorem, cited here as the licence and deliberately not re-proved (it needs the stochastic integral, which is absent from the underlying mathematics library at the pinned version — a strong-solution left edge waits on upstream stochastic integration). For the compiled fragment the identification is definitional in the literature itself: polynomial diffusions are *defined* by the martingale problem with polynomial test functions [78]. Stated self-contained:

**Problem 12.5** (the martingale-problem square). Fix an SDE profile with polynomial coefficients — drift  $f$ , diffusion columns  $\sigma^k$ , jump rate  $\lambda$  with reset  $g$  — a closed working domain  $D$ , and a certificate polynomial  $V$  with exact rational coefficients. Suppose a process  $X$ , confined to  $D$ , solves the stopped martingale problem for the profile’s integro-differential generator  $\mathcal{A}$ , and the checker’s own compiled row holds: the exact-rational polynomial  $\mathcal{A}V$ , computed symbolically from the presentation, evaluates  $\leq 0$  everywhere on  $D$ , with  $V \geq 0$  on  $D$  and  $V \geq 1$  on a bad set  $B$ . Show that along every monotone sampling grid  $\tau$ ,  $\Pr(\exists k, X_{\tau_k} \in B) \leq \mathbb{E}[V(X_{\tau_0})]$ .

*Sketch.* Two statics identify the formal with the analytic: the Fréchet derivative of polynomial evaluation is the formal gradient — at order one, and, instantiated at each first partial, at order two — so the triple  $(V, \nabla V, \nabla^2 V)$  of exact-rational row data enters the martingale problem as a lawful  $C^2$  test datum; and evaluating the symbolically computed generator equals the semantic generator at that datum. The compensated certificate process is then a martingale whose compensator has nonpositive density along the confined path, so  $V(X_t)$  is a continuous-time supermartingale — Dynkin’s formula *derived* from the presentation, not assumed — and Ville’s inequality [234] along the sampled grid gives the bound.

The point of the square is what it retires: the staged Dynkin and optional-sampling hypotheses of the Semantics paragraph are, for martingale-problem-presented solutions, theorems, and the per-model residue is being a solution — the community’s own object — plus the per-datum integrability provisos, which are the textbook’s standing assumptions for unbounded test functions. The boundaries, stated as such: verdicts are on the sampled grid (the continuum-time supremum needs almost-sure path regularity, which stays with the constructor); well-posedness of the martingale problem is not claimed, exactly as the ODE square proved no Picard existence; and unlike this chapter’s discrete squares the transport here is one-directional — community semantics to certified verdict — with no invertible-cell claim at the path-law level. The statement and the discharge are mechanized, sorry-free, in the toolkit’s Lean tree (`lean/qtsl-aux/`, modules `QtslAux.SdeGenerator`, `QtslAux.SdeGeneratorPoly`, `QtslAux.SdeMartingaleProblem`): the solution concept is `MpSolution` (stopped/confined, generator-as-pairs form), the derived continuous-time supermartingale is `supermartingale_of_gen_nonpos`, the discharge of the Dynkin class field is `MpSolution.toSdeDynkinVille`, the compilation splice is `genPoly_eval_eq_generator`, the order-1/2 statics are `isC2Fn_poly`, and the compiled headline is `mp_avoid_row_of_genPoly` with its deterministic-start form `mp_avoid_init_of_genPoly`, exercised end-to-end by a zero-dynamics control (`mp_control`).

**Status.** Certification is in place at three scales: the one-dimensional Ornstein–Uhlenbeck certificates (the Hessian term present in the checked sequents); the same second-order term running inside jump-diffusion and hybrid composites (sections 12.10 and 12.16); and an encounter-scale correlated-noise family with the cross-term Hessian certified over two- and three-dimensional simplicial covers. The correlated family carries the frame’s signature results: a refusal witness for the decorrelated misreading of the same system, with the cross-term’s contribution in closed form, and a certified anisotropic bound pair — 0.242 against 0.828 at equal Euclidean distance — quantifying exactly what correlation is worth. Representation and production: the noise declarations are shipped theory rows, the correlated systems exist as instance rows, and the toolkit’s producer surface constructs presentations in both languages and derives the generator polynomial symbolically from the presentation — the exact computation whose identity with the semantic generator, at the pure-diffusion face, is the Adequacy paragraph’s mechanized splice; derived and never trusted, the checker re-derives it. The API block below quotes receipted constructions, and a correlated two-dimensional presentation — one shared driver loading both coordinates, so the cross term is live in the derived covariance — is a committed cross-language byte-identity vector whose generator the kit’s tests pin in closed form, against its decorrelated misreading. Open: state-dependent  $\sigma$  at  $n \geq 2$  (fully representable and derivable, but no shipped certificate exercises it yet); the imprecise denotation exercised as a certificate; the ingestion gap.

**SPDEs.** No longer an open flank: stochastic PDEs have their own entry (section 12.8). The road ran through the deterministic PDE machinery as promised — at the moment rungs, a stochastic certificate is the deterministic row stack plus a single Itô-correction row.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset

signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{\vec{x} \text{ declared state variables} \quad D \text{ drift rows} \quad N \text{ a noise presentation}}{\text{SDE}(\vec{x}; D; N) : \text{System}} \quad \text{S-SDE}$$

$$N ::= \text{diffusion}(\overline{\langle \text{var}, \text{poly} \rangle}) \mid \text{noise}(\sigma^1, \dots, \sigma^K)$$

Exactly one noise presentation, by the grammar rather than by a side condition: diffusion is the single-column form, noise the per-driver sigma-column form (one exact-rational polynomial column per independent Brownian driver); the covariance  $a = \sum_k \sigma^k (\sigma^k)^\top$  is derived, never stored.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named fx\_\* or f.\* are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript system.sde

signature — ts/src/system.ts

```
/** S-SDE: drift + diffusion (single-column) XOR noise (multi-driver
 * sigma columns; the covariance is derived, never stored). */
export const sde = profileFactory("sde");

/** The derived SDE generator kit: the exact generator polynomial
 *  $\mathcal{V} = \nabla V \cdot f + \frac{1}{2} \text{tr}(a \cdot \text{Hess } V)$ , the covariance  $a = \sum_k \sigma^k (\sigma^k)^\top$ 
 * derived from the presented columns, never stored. Derived, never
 * trusted — the checker re-derives it. */
export interface SdeGenKit {
  readonly gen: PolyWitness;
}

/** Derive the SDE generator kit from a mechanism declaration and a
 * certificate observable. Admission is the SDE profile's — jump or
 * killing rows refuse naming the stray family (those presentations
 * belong to `jdGenKit`) — and the computed polynomial's identity
 * with the semantic generator at the pure-diffusion face is
 * mechanized in the toolkit's Lean tree. The shared stratum fences
 * (single implicit mode, no transcendental terms) come back as
 * refusals naming their geometry. */
export function sdeGenKit(
  init: MechanismInit,
  cert: Polynomial,
): Result<SdeGenKit, Refusal> {
  const raw = native.sysSdeGenKit(
    mechanism(init).nativeHandle,
    polyToNative(cert),
  ) as { kit: SdeGenKit } | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}
```

construction — ts/tests/system-native.test.ts

```

return sys.unwrap(
  sys.sde({
    name: "ou-1d",
    state: [{ id: 0, name: "x" }],
    drift: [{ var: 0, poly: mono(1, 0, 1, "-1") }],
    diffusion: [{ var: 0, poly: constant(1, "1/2") }],
    init: { kind: "point", coords: [BIG] },
  }),
);

```

**Rust** `mtk_catalogue::system::presets::Sde::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```

preset!(
  /// System Catalogue: SDE profile – drift + diffusion
  /// (single-column) or noise (per-driver sigma columns; the
  /// covariance is derived, never stored).
  Sde, "SDE",
  req = [Drift],
  allowed = [Diffusion, Noise],
  extra = |m| diffusion_xor_noise("SDE", m)
);

```

construction — `crates/mtk-catalogue/tests/common/mod.rs`

```

let big = WitRat::new(
  "1234567890123456789012345678901234567890"
  .parse::<BigInt>()
  .unwrap(),
  BigInt::from(7u8),
);
let m = MechanismBuilder::new()
  .state_var(0, Some("x"))
  .drift(None, 0, mono(1, 0, 1, r(-1, 1)))
  .diffusion(None, 0, constant(1, r(1, 2)))
  .init(InitRegion::Point { coords: vec![big] })
  .finish();
presets::Sde::manifest(m)
  .expect("fixture conforms to the SDE profile")
  .named("ou-1d")

```

The multi-driver presentation (per-driver sigma columns) is the corpus's committed correlated-2d golden vector (fixture `sde_noise`, whose derived cross term the generator-kit tests pin in closed form); the generator kit's Rust face is `mtk_catalogue::system::sde_gen_kit`.

## 12.8 Stochastic Partial Differential Equations

**The frame.** Stochastic evolution equations  $du = F(u, \partial_x u, \partial_x^2 u, \dots) dt + dW_Q$ , with  $W_Q$  a  $Q$ -Wiener process over the declared spatial domain; the certified questions are expected-energy dissipation to a noise floor, absorbing balls, and probability bounds on energy excursions.

**Profile.** The deterministic jet mechanism plus a declared noise spectrum: finitely many explicit driver shapes — exact-rational combinations of the domain's orthonormal Fourier basis, one amplitude column per driver — and a required tail form in one of three shapes (zero, a rational trace bracket, or declared-divergent), so the covariance's trace is derived, never stored, and a spectrum without a declared tail is refused rather than read as zero. Amplitude columns are constants at this preview (additive noise), but their declared type is a polynomial over the jet space, so multiplicative noise is a fence-lift rather than a schema change. Time: continuous.

**Semantics.** The stochastic solution-concept rungs, weakest first: the expected-energy class (the inhomogeneous inequality with the trace of  $Q$  as its forcing constant), the compensated-supermartingale form (horizon-tagged), and the mode-law rung where explicit Ornstein–Uhlenbeck second-moment trajectories live. Every stochastic-analysis fact is a named obligation behind a class field with stated discharge routes — no theorem below the docstring mentions a stochastic integral — extending the staging discipline section 12.5 established.

**Status.** Certification: five exact-rational artefacts — stochastic Burgers and stochastic heat moment certificates (the deterministic rows carried verbatim, the Itô correction docked as one constant row; the heat floor  $\text{tr}(Q)/2\nu = 13/10$  exact, with a deliberately sharp width-zero instance) and two stochastic Kuramoto–Sivashinsky absorbing balls that embed and replay their deterministic siblings byte-identically before any stochastic row is added. All five carry probability rows: a horizon-tagged supremum bound of Kushner’s classical shape [147], Chebyshev on certified variances, and an exponential-compensation row strictly sharper at both registered points. Mutant batteries assert rejection messages, and witnesses realizing the noise are checker-required, so a green moment bound cannot be vacuously noise-silent. The jet-level declaration now has its proof-layer face: spectrum declarations with the required tail carry mechanized derived-covariance data (module `SuFuctor.Adequacy.PdeRows`: the explicit trace by Parseval with per-element grouping, the q-table diagonal with trace-equals-summed-diagonal, and the certified heat spectrum’s  $\text{tr}(Q) = 13/50$ ,  $q_1 = 1/5$ ,  $q_2 = 3/50$  recomputed by theorem), and the plain-trace consumer is total-or-refusing — a declared-divergent tail refuses structurally (`plainTrace_divergent_none`), never defaulting to zero. Representation and production: the toolkit’s producer surface now carries the derived-kit sibling of that proof layer, shared with section 12.5 — from a declared presentation the kit derives the canonical differential-polynomial right-hand side (the compile edge’s producer face) and, from the spectrum, the q-table diagonal, the trace as the summed diagonal, the plain-trace enclosure (structurally refused on a divergent tail, in both languages), and the dissipation floor  $\text{tr}(Q)/2\nu$  exactly where the presentation exhibits it — transport terms and mean-mode mass forfeit the floor rather than have it guessed; derived and never trusted, the checker re-derives every row. The kit’s tests recompute the certified constants from the committed golden vector’s own presentation, and the deterministic sibling (viscous Burgers at jet grain) is now itself a committed cross-language byte-identity vector. Deliberately open: trace-class tails (declaration vocabulary without a certified instance); the cylindrical mode-resolved variant; multiplicative noise; the Fourier mode-projection semantics (the compile-for-semantics leg, a named obligation at the Lean stratum); composition.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{J : \overrightarrow{\langle \text{base}, \alpha \rangle} \text{ jet variables} \quad \overrightarrow{\vec{x}} \text{ declared state variables} \quad D \text{ drift rows over } (\vec{x}, J) \quad N = (\overrightarrow{\langle \text{shape}, \text{amp} \rangle}; \tau) \text{ a noise spectrum}}{\text{SPDE}(\vec{x}; J; D; N) : \text{System}} \quad \text{S-SPDE}$$

$$\tau ::= \text{zero} \mid \text{traceBracket}(T_{\text{lo}}, T_{\text{hi}}) \mid \text{divergent}$$

The jet families are section 12.5’s, verbatim; the noise spectrum adds exactly one family. Each driver is a spatial shape (an exact-rational combination of the domain’s orthonormal Fourier basis) with an amplitude column; the tail form  $\tau$  is required by the grammar rather than by a side condition, so a spectrum without a declared tail is unrepresentable here and refused at preset admission in the toolkit. The covariance  $Q = \sum_j \text{shape}_j \text{shape}_j^*$  (plus the declared tail) and its trace are derived, never stored; a consumer needing the plain trace refuses a divergent tail.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s

shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

## TypeScript system.spde

signature — ts/src/system.ts

```
/** S-SPDE: the PDE families plus a noise spectrum – Fourier-shape
 * driver rows (`noiseDrivers`) and a REQUIRED tail form
 * (`noiseTail`: zero XOR traceBracket XOR divergent). A spectrum
 * whose tail was never declared refuses here – never read as zero;
 * the covariance and its trace are derived, never stored. */
export const spde = profileFactory("spde");

/** One entry of the derived covariance diagonal: the q-table value
 * at an orthonormal Fourier-basis element – amplitudes and grouped
 * coefficients squared and summed over the drivers. Zero entries
 * (cancelled groups; `sin` at mode 0, the zero function) are
 * omitted, never stored. */
export interface QTableEntry {
  readonly kind: "sin" | "cos";
  readonly mode: number;
  readonly q: string;
}

/** The derived SPDE kit: the PDE right-hand side plus covariance
 * data derived from the noise-spectrum declaration – the q-table
 * diagonal, the explicit trace (their sum, by the mechanized
 * finite-Fubini identity), the plain-trace enclosure under the
 * declared tail (`null` on a declared-divergent tail – the
 * structural refusal, never a zero default), and the dissipation
 * floor  $\text{tr}(Q)/2\nu$  exactly where the presentation exhibits it
 * (`null` otherwise – transport terms and mean-mode mass are
 * certificate business, not producer recognition). Derived, never
 * stored, never trusted. */
export interface SpdeGenKit {
  readonly rhs: ReadonlyArray<PolyWitness>;
  readonly qTable: ReadonlyArray<QTableEntry>;
  readonly traceExplicit: string;
  readonly plainTrace: readonly [string, string] | null;
  readonly heatFloor: readonly [string, string] | null;
}

/** Derive the SPDE kit from a mechanism declaration. Admission is
 * the SPDE profile's – a spectrum whose tail was never declared
 * refuses there, never read as zero – and the constant-amplitude
 * stratum's multiplicative fence comes back as a refusal naming its
 * driver. */
export function spdeGenKit(init: MechanismInit): Result<SpdeGenKit, Refusal> {
  const raw = native.sysSpdeGenKit(mechanism(init).nativeHandle) as
    | { kit: SpdeGenKit }
    | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}
```

construction — ts/tests/system-native.test.ts

```
// Stochastic heat on the torus, the reference constants: nu =
// 1/10; q_1 = 1/5 and q_2 = 3/50 realized as sum-of-rational-
// squares driver decompositions (sigma_k = sqrt(q_k) is
// irrational); derived trace 13/50, floor  $\text{tr}(Q)/2\nu = 13/10$  exact.
// The tail form is REQUIRED – zero is said, never implied.
const amp = constant(2, "1");
return sys.unwrap(
  sys.spde({
    name: "stochastic-heat-torus",
    state: [{ id: 0, name: "u" }],
    jetVars: [{ id: 1, base: 0, deriv: [2] }],
    drift: [{ var: 0, poly: mono(2, 1, 1, "1/10") }],
    noiseDrivers: [
```



```

        { shape: [{ kind: "sin", mode: 1, coeff: "1/5" }], amp },
        { shape: [{ kind: "sin", mode: 1, coeff: "2/5" }], amp },
        { shape: [{ kind: "sin", mode: 2, coeff: "1/10" }], amp },
        { shape: [{ kind: "sin", mode: 2, coeff: "1/10" }], amp },
        { shape: [{ kind: "sin", mode: 2, coeff: "1/5" }], amp },
    ],
    noiseTail: { kind: "zero" },
  })),
);

```

## Rust mtk\_catalogue::system::presets::Spde::manifest

signature — crates/mtk-catalogue/src/system/presets.rs

```

preset!(
  /// System Catalogue: SPDE profile – the PDE profile's families
  /// (jet variables + differential-polynomial drift) plus a noise
  /// spectrum: finitely many explicit Fourier-shape driver rows
  /// and a REQUIRED tail form in exactly one of three shapes
  /// (zero / trace-bracket / declared-divergent). A spectrum whose
  /// tail was never declared is refused here – never read as zero
  /// – and the covariance (with its trace) is derived, never
  /// stored.
  Spde, "SPDE",
  req = [JetVars, Drift, NoiseSpectrum],
  allowed = [],
  extra = |m| {
    let spectrum = m.noise_spectrum.as_ref().expect("required by signature");
    if spectrum.tail.is_none() {
      return Err(Refusal::MissingTailForm { profile: "SPDE" });
    }
    Ok(())
  }
);

```

construction — crates/mtk-catalogue/tests/builders.rs

```

let spde = MechanismBuilder::new()
  .state_var(0, Some("u"))
  .jet_var(1, 0, vec![2])
  .drift(None, 0, mono(2, 1, 1, r(1, 10)))
  .noise_driver(
    vec![FourierTerm {
      kind: FourierKind::Sin,
      mode: 1,
      coeff: r(1, 5),
    }],
    constant(2, r(1, 1)),
  )
  .noise_driver(
    vec![FourierTerm {
      kind: FourierKind::Sin,
      mode: 1,
      coeff: r(2, 5),
    }],
    constant(2, r(1, 1)),
  )
  .noise_tail(TailForm::Zero)
  .finish();
Spde::manifest(spde).unwrap();

```

The tail form is the spectrum's required member: the corpus's `spde_missing_tail_refused_never_read_as_zero` sibling exercises the admission refusal, and the divergent-tail plain-trace refusal is witnessed in `plain_trace_derived_never_stored`. The derived kit's Rust face is `mtk_catalogue::system::spde_gen_kit`: the  $q$ -table diagonal, the explicit trace as the summed diagonal, the plain-trace enclosure (structurally refused on a divergent tail), and the dissipation floor  $\text{tr}(Q)/2\nu$  exactly where the presentation exhibits it — the kit's tests recompute the reference constants ( $q_1 = 1/5$ ,  $q_2 = 3/50$ , trace  $13/50$ , floor  $13/10$ ) from the committed golden vector's own presentation.

## 12.9 Random Differential Equations

**The frame.** An ODE or PDE whose parameters  $\theta$  are drawn from a declared uncertainty set; the certified question is a property *uniform* over the set.

**Profile.** Drift plus a boxed parameter axis: the parameter is one more coordinate of the region, constant along trajectories. Disturbance semantics: uniform over the box — the  $\forall\theta$  leg is carried entirely by the region, not by any new machinery. Time: continuous.

**Semantics.** Per- $\theta$  trajectory semantics — each theorem a one-line specialization of the ODE semantics seam, and that one-line-ness is the measured content of the claim that the kernel’s cone treats parameter boxes natively. Uniformity over the box is, in the universe’s terms, a credal body — the possibilistic face over the box-indexed trajectory family (Remark 9.22) — and the  $\forall\theta$  verdict is its lower prevision (Definition 9.21).

**Status.** Certification: a damped oscillator  $\dot{x} = v$ ,  $\dot{v} = -x - \theta v$  with damping boxed  $\theta \in [1/4, 1]$  carries a barrier decrease  $\dot{V} \leq -\frac{1}{8}(x^2 + v^2)$  on an annulus, uniformly over the box, as one polynomial row family — checked, round-tripped through the wire format, with refusal probes on both producer and checker sides, and a mechanized per- $\theta$  semantics layer. This frame was priced as nearly free before it was built, and building it confirmed the price: the thesis’s own  $\forall\theta$  property form is RODE-shaped. Production: the toolkit’s producer surface constructs the boxed-parameter presentation in both languages, and the damped oscillator itself is a committed cross-language byte-identity vector, the parameter family’s first. The kit derives the state  $\times$  parameter generator row symbolically at the product index space, with derivatives over state axes only ( $\dot{\theta} = 0$ : a  $\theta$ -only observable is conserved, by test). Its tests pin  $\mathcal{A}(x^2 + v^2) = -2\theta v^2$  in closed form in both languages — one polynomial over the product space is the whole  $\forall\theta$  row family — and a collapse differential checks that a  $\theta$ -free drift derives exactly the ODE kit’s polynomial. Open: the mechanism as ingested rows (the ingestion gap); a fixture nonlinear in  $\theta$ ; and the RPDE side, which waits on section 12.5’s bridges.

**Adequacy.** The square for this frame is deliberately slight, and the slightness is itself the certified content: the community’s solution concept for random ODEs is *pathwise* — for each realization  $\theta$ , a classical solution of the  $\theta$ -instance, analyzed with deterministic calculus [109] — so the per-draw connector, square, and uniqueness theorems are the ODE square of section 12.4 at the drawn drift  $f_\theta$ , consumed verbatim rather than re-proved. What remains is the box. Stated self-contained:

**Problem 12.6** (the boxed-parameter square). Fix a drift family  $f : \Theta \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ , continuous in the state at each draw in a parameter box  $P \subseteq \Theta$ , and  $x_0 \in \mathbb{R}^n$ . Records are paths  $y : \mathbb{R} \rightarrow \mathbb{R}^n$ ; the compiled constraints are the initial row and the *boxed cell* — some draw  $\theta \in P$  drives  $y$ ’s forward rows. Show the compiled records are exactly  $\bigcup_{\theta \in P} S_\theta$ , where  $S_\theta$  is the classical solution set of the  $\theta$ -instance; show the same for the rectangle with both boxes open — the initial row replaced by an initial box  $B \subseteq \mathbb{R}^n$  — denoting  $\bigcup_{\theta \in P} \bigcup_{x \in B} S_{\theta,x}$ , with  $S_{\theta,x}$  the  $\theta$ -instance’s solution set from  $x$ ; and show the polarity of the box matters: over the empty box the compiled body is empty, while a uniform-verdict reading — intersection over draws — would admit every record.

*Sketch.* Intersecting with the initial row distributes into the parameter union, and each summand is the ODE square at  $f_\theta$ ; the rectangle composes the parameter union with the ODE square’s initial-box identity, one draw at a time; the empty union is empty, the empty intersection is everything.

The polarity is easy to get backwards. The compiled *body* over the box is a union — existential in  $\theta$ , the same demonic convention as the initial box — while the certified uniform-over- $\theta$  property of the Semantics paragraph above is the opposite polarity: a verdict over the whole box-indexed family, the lower prevision of Remark 9.22, living at the judgment, never as an intersection body. Uniqueness enters per draw with a certificate-shaped hypothesis — a single Lipschitz constant uniform over the box pins every draw in it, the semantics shadow of a certificate stated as one row family over state and parameter. The statement and both directions are mechanized, sorry-free, in the toolkit’s Lean tree (lean/su-functor/, module SuFunctor.Adequacy.Rode): the per-draw connector and square are `isRodeSol_iff_integralCurve` and `rode_draw_adequacy` (one-line specializations, stated as such), the union face is `rode_adequacy` with the

rectangle `rode_box_adequacy`, the uniform-constant pinning is `rodeSols_eq0n_of_lipschitz0nWith`, and the wiring into the semantic universe's judgment (`su_rode_adequacy`) runs through the same transport layer as the sibling squares, with the zero-law contrast holding at every box including the empty one — the infeasible frame denotes a bottom body, never an absence of models. The completeness hypothesis enters exactly where the ODE connector is consumed, nowhere else.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema's own (`crates/mtk-catalogue/src/system/`), and the former's mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ ,  $\text{pop}$ , and profile conformance are defined in section 12.1.

$$\frac{\Theta : \text{ParamDecl} \quad \exists p \in \Theta. p.\text{bounds} \neq \perp \quad D \text{ drift rows, each } \text{poly} \in \mathbb{Q}[\vec{x}, \Theta]}{\text{RODE}(\vec{x}; \Theta; D) : \text{System}} \text{ S-RODE}$$

$\vec{x}$  declared state variables

A boxed parameter (`bounds = [lo, hi]`, `exact`) is the  $\forall\theta$  axis; an unboxed parameter is a plain named constant, which is why the existential premise demands at least one box.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.rode`

signature — `ts/src/system.ts`

```
/** S-RODE: drift + a boxed parameter axis (disturbance: parameter
 * box, uniform-over- $\theta$ ). */
export const rode = profileFactory("rode");

/** The derived RODE generator kit: the exact Lie-derivative
 * polynomial  $\dot{V} = \sum_i \partial_{iV} \cdot f_i$  over the product index space
 * `state ++ params` (parameters in declaration order after the state
 * roster), state axes only — parameters are constant along
 * realizations, so no parameter-derivative term exists, and one
 * polynomial in state  $\times$  parameter is the whole  $\forall\theta$  row family.
 * Derived, never trusted — the checker re-derives it. */
export interface RodeGenKit {
  readonly gen: PolyWitness;
}

/** Derive the RODE generator kit from a mechanism declaration and a
 * certificate observable (read at the product arity — a certificate
 * may itself name the parameter axes, the family-certificate face).
 * Admission is the RODE profile's — an unboxed-only roster refuses
 * as `noBoxedParam`, and a drift row addressed past the state roster
 * refuses naming the row (a parameter has no evolution row). */
export function rodeGenKit(
  init: MechanismInit,
  cert: Polynomial,
): Result<RodeGenKit, Refusal> {
  const raw = native.sysRodeGenKit(
```

```

        mechanism(init).nativeHandle,
        polyToNative(cert),
    ) as { kit: RodeGenKit } | { refusal: unknown };
    return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}

```

construction — ts/tests/system-native.test.ts

```

return sys.unwrap(
  sys.rode({
    name: "damped-oscillator",
    state: [
      { id: 0, name: "x" },
      { id: 1, name: "v" },
    ],
    params: [{ id: 0, name: "theta", bounds: ["1/4", "1"] }],
    drift: [
      { var: 0, poly: mono(3, 1, 1, "1") },
      {
        var: 1,
        poly: {
          vars: 3,
          terms: [
            { exps: [1, 0, 0], coeff: "-1" },
            { exps: [0, 1, 1], coeff: "-1" },
          ],
        },
      },
    ],
  }),
);

```

## Rust mtk\_catalogue::system::presets::Rode::manifest

signature — crates/mtk-catalogue/src/system/presets.rs

```

preset!(
  /// System Catalogue: RODE profile – drift + a boxed parameter
  /// axis (disturbance: parameter box, uniform-over-θ).
  Rode, "RODE",
  req = [Drift, Params],
  allowed = [],
  extra = |m| {
    if m.params.iter().any(|p| p.bounds.is_some()) {
      Ok(())
    } else {
      Err(Refusal::NoBoxedParam { profile: "RODE" })
    }
  }
);

```

construction — crates/mtk-catalogue/tests/common/mod.rs

```

let vdot = Poly {
  vars: 3,
  terms: vec![
    PolyTerm {
      exps: vec![1, 0, 0],
      coeff: r(-1, 1),
    },
    PolyTerm {
      exps: vec![0, 1, 1],
      coeff: r(-1, 1),
    },
  ],
};
let m = MechanismBuilder::new()
  .state_var(0, Some("x"))
  .state_var(1, Some("v"))
  .param(0, Some("theta"), Some([r(1, 4), r(1, 1)]))

```

```

    .drift(None, 0, mono(3, 1, 1, r(1, 1)))
    .drift(None, 1, vdot)
    .finish();
presets::Rode::manifest(m)
    .expect("fixture conforms to the RODE profile")
    .named("damped-oscillator")

```

The construction is the committed damped-oscillator golden vector (fixture rode — the parameter family’s first golden bytes: drift over the product index space  $state ++ params$ , damping boxed  $\theta \in [1/4, 1]$ ), and it doubles as the generator kit’s closed-form control ( $\mathcal{A}(x^2 + v^2) = -2\theta v^2$  — one polynomial in state  $\times$  parameter is the whole  $\forall \theta$  row family); the kit’s Rust face is `mtk_catalogue::system::rode_gen_kit`.

## 12.10 Jump-Diffusion

**The frame.** Continuous drift and diffusion punctuated by Poisson-rate jumps; certified questions are ruin and reach-avoidance bounds through the extended generator with its jump term.

**Profile.** All three field families at once — drift, diffusion, and jump legs — on one mechanism, plus an optional killing family. Each jump row is one leg: a polynomial rate paired with a reset kernel, either a single polynomial reset (the Dirac form) or a finite mixture of weighted polynomial resets, with an optional reset-image box. Mixture weights may total *below* one: a sub-stochastic kernel is lawful by design, the mass deficit being killing (the standard convention for jump kernels of total mass below one [60]); a row presenting both kernel forms at once is refused, naming the row. Disturbance: Brownian drivers plus finite-activity jumps. Time: continuous, with jumps as events in real time; per-execution jump counters are bookkeeping, never a composition base.

**Semantics.** The extended generator  $\mathcal{A}V = fV' + \frac{\sigma^2}{2}V'' + \lambda(g \circ V - V)$ , composed from the same shipped summands the pure-diffusion and pure-jump frames use, with no new checker machinery — each degenerate case asserts its reduction to the shipped frame. Finite-activity hypotheses are named; the bridge set is the diffusion frame’s plus an Itô–Lévy step.

**Adequacy (generator face).** The Semantics paragraph’s composition claim — the generator assembled from the shipped summands, each degenerate case asserting its reduction — is mechanized at the frame’s full profile generality, and the identity holds with *no side conditions*. Stated self-contained:

**Problem 12.7** (the generator splice, at full profile generality). Fix a polynomial jump-diffusion presentation: drift  $f$ , diffusion columns  $\sigma^k$ , a killing rate  $c$ , and finitely many jump legs, each a polynomial rate  $\lambda_\ell$  paired with a finite sub-stochastic mixture of polynomial resets  $(w_{\ell i}, g_{\ell i})$ . Write  $T_{\mu_\ell}V = \sum_i w_{\ell i} (g_{\ell i} \circ V)$  for the transfer action of the  $\ell$ -th mixture, and let  $\mathcal{A}V$  be the exact-rational polynomial computed symbolically from the presentation: killing multiplies, drift pairs the formal gradient, diffusion traces the formal Hessian against the derived covariance, each jump leg contributes  $\lambda_\ell (T_{\mu_\ell}V - V)$ . Show that evaluating  $\mathcal{A}V$  anywhere on  $\mathbb{R}^n$  equals the semantic integro-differential generator

$$-cV + \nabla V \cdot f + \frac{1}{2} \sum_{ij} a_{ij} \partial_{ij} V + \sum_{\ell} \lambda_{\ell} (T_{\mu_{\ell}}V - V)$$

of the evaluated profile, with the observable’s derivative data instantiated at the formal derivatives — for *every* presentation, with no conservativity or leg-shape hypothesis — and that at one Dirac leg with zero killing the semantic generator is the SDE frame’s operator (section 12.7) on the nose.

*Sketch.* Evaluation is an algebra homomorphism, so each field family passes through separately; the transfer of a finite mixture is a finite sum, exact at the polynomial stratum, and a list induction carries the leg sum. A sub-stochastic mixture adds no new semantics: its jump increment decomposes as Dirac-leg increments minus the mass deficit spent as killing — the standard convention for jump kernels of total mass below one [60]. The one-Dirac-leg tie is a computation collapsing the leg sum.

The identity of operators carries the verdict route with it: the martingale problem consumes its generator opaquely — the compensator argument never inspects the drift, diffusion, killing, or jump structure — so the SDE section’s solution concept, its derived continuous-time supermartingale, and its verdict headlines transplant verbatim to the multi-leg generator. Because this section’s splice holds with no side conditions,

the transplanted headlines hold for *every* presentation — killing, finitely many legs, sub-stochastic mixtures — with no conservativity or leg-shape hypothesis, where the SDE section’s headlines carried both. The single-Dirac-leg square then *factors* through the multi-leg one: profiles embed, the operators agree on the nose, and solutions transport along the identity, verdicts included. The boundaries that remain are the SDE square’s own: verdicts on the sampled grid, well-posedness not claimed, transport one-directional. All of it is mechanized, sorry-free, in the toolkit’s Lean tree (`lean/qlAux/`, modules `QtslAux.JumpDiffusion` and `QtslAux.JumpDiffusionMartingaleProblem`): the profile is `JdProfile`, the hypothesis-free splice is `genPoly_eval_eq_generator`, the degeneracy ties are `generator_one_dirac` and `toJd_generator_one_dirac`, the mixture-to-killing decomposition is `transferSum_sub_self`, and a two-leg control with every field family live pins the closed form (`two_leg_control`); on the verdict side, the solution concept is `MpSolution` at the multi-leg generator, the derived supermartingale is `supermartingale_of_gen_nonpos`, the sampled Dynkin class is `JdDynkinVil`, the headlines are `mp_avoid_row_of_genPoly` and `mp_avoid_init_of_genPoly`, the factoring is `generator_toJd` with `MpSolution.toJd`, and the end-to-end control’s presentation carries live rates on two legs with a genuine mixture (`mp_control`).

**Status.** Certification: two hand-derived systems check through the unmodified stack — a Brownian-perturbed risk process of the classical ruin-theory shape (drift helps, diffusion and claim jumps hurt; certified ruin bound 49/121 from its stated start), and a load-shedding system in which the state-dependent jump term *alone* carries the certificate against both unstable drift and diffusion. At  $n = 2$  the encounter composite carries all three families with full-rank correlated noise. Representation and production: the toolkit’s producer surface constructs presentations at the frame’s full profile generality — killing rows and multi-leg sub-stochastic mixture kernels included — in both languages, and derives the generator polynomial together with its per-leg rate rows symbolically from the presentation (the exact computation whose identity with the semantic generator is the Adequacy paragraph’s mechanized splice; derived and never trusted — the checker re-derives both). The API block below quotes receipted constructions, and the reference presentation exercising every field family at once — a Dirac leg beside a genuinely sub-stochastic mixture leg, with killing — is a committed cross-language byte-identity vector. Open: certification exercise of multi-leg jump kernels and state-dependent  $\sigma$  (fully representable and derivable as above, but no shipped certificate exercises them yet); time-bounded and discounted variants; closure.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ ,  $\text{pop}$ , and profile conformance are defined in section 12.1.

$$\frac{\begin{array}{c} \vec{x} \text{ declared state variables} \quad D \text{ drift rows} \quad N \text{ a noise presentation (section 12.7)} \\ J : \langle \text{mode?}, \text{rate}, \text{reset} \mid \text{mixture}, \text{reset\_image?} \rangle \text{ jump rows} \quad K \text{ killing rows (optional)} \end{array}}{\text{JumpDiffusion}(\vec{x}; D; N; J; K) : \text{System}} \quad \text{S-JUMPDIFF}$$

Jump rows carry rate a polynomial intensity and exactly one kernel form: reset a polynomial substitution, or mixture a finite weighted list of polynomial substitutions (sub-stochastic totals lawful — the deficit is killing); reset image an exact box. Killing rows carry a polynomial rate; an absent family is the conservative declaration.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s



shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

## TypeScript `system.jumpDiffusion`

signature — `ts/src/system.ts`

```
/** S-JUMPDIFF: drift + jumps (+ diffusion/noise, + killing). Jump
 * kernels at full generality: each row is a leg whose reset is a
 * Dirac reset or a finite sub-stochastic mixture of resets (the
 * deficit is killing). */
export const jumpDiffusion = profileFactory("jumpDiffusion");

/** The derived jump-diffusion generator kit: the exact generator
 * polynomial  $\mathcal{A}V = -c \cdot V + \nabla V \cdot f + \frac{1}{2} \text{tr}(a \cdot \text{Hess } V) + \sum \lambda_{\ell} (T_{\ell} V - V)$ 
 * and, per jump leg in row order, the rate-validity row  $-\lambda_{\ell}$  (its
 * nonpositivity on the working region is the leg's model-validity
 * row). Derived, never trusted — the checker re-derives both. */
export interface JdGenKit {
  readonly gen: PolyWitness;
  readonly rateRows: ReadonlyArray<PolyWitness>;
}

/** Derive the jump-diffusion generator kit from a mechanism
 * declaration and a certificate observable. The computed polynomial's
 * identity with the semantic integro-differential generator — for
 * every presentation: killing, multi-leg, sub-stochastic mixtures —
 * is mechanized with no hypotheses in the toolkit's Lean tree. The
 * stratum fences (single implicit mode, no transcendental terms, no
 * jet variables) come back as refusals naming their geometry. */
export function jdGenKit(
  init: MechanismInit,
  cert: Polynomial,
): Result<JdGenKit, Refusal> {
  const raw = native.sysJdGenKit(
    mechanism(init).nativeHandle,
    polyToNative(cert),
  ) as { kit: JdGenKit } | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}
```

construction — `ts/tests/system-native.test.ts`

```
return sys.unwrap(
  sys.jumpDiffusion({
    name: "two-leg-jd",
    state: [{ id: 0, name: "x" }],
    drift: [{ var: 0, poly: mono(1, 0, 1, "1") }],
    diffusion: [{ var: 0, poly: constant(1, "1") }],
    kill: [{ poly: constant(1, "1") }],
    jumps: [
      {
        rate: constant(1, "1"),
        reset: [{ var: 0, poly: mono(1, 0, 1, "1/2") }],
      },
      {
        rate: mono(1, 0, 2, "1"),
        mixture: [
          { weight: "1/2", reset: [{ var: 0, poly: constant(1, "0") }] },
          { weight: "1/4" },
        ],
      },
    ],
  }),
);
```

## Rust `mtk_catalogue::system::presets::JumpDiffusion::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```

preset!(
  /// System Catalogue: jump-diffusion profile – drift + diffusion +
  /// jumps at full kernel generality (each row a leg: rate + Dirac
  /// reset or finite sub-stochastic mixture of resets, reset-image
  /// optional), plus an optional killing family. Sub-stochastic
  /// mixture totals are lawful (the deficit is killing); a row
  /// carrying both reset forms is refused, naming the row.
  JumpDiffusion, "jump-diffusion",
  req = [Drift, Jumps],
  allowed = [Diffusion, Noise, Kill],
  extra = |m| {
    diffusion_xor_noise("jump-diffusion", m)?;
    jump_reset_forms("jump-diffusion", m)
  }
);

```

construction — crates/mtk-catalogue/tests/builders.rs

```

// book: api begin jd_full_construction
// Full kernel generality on one mechanism: drift, diffusion,
// killing, a Dirac leg, and a sub-stochastic two-branch mixture
// leg (total mass 3/4; the deficit is killing).
let m = MechanismBuilder::new()
  .state_var(0, None)
  .drift(None, 0, mono(1, 0, 1, r(-1, 1)))
  .diffusion(None, 0, constant(1, r(1, 2)))
  .kill(None, constant(1, r(1, 20)))
  .jump(JumpRow {
    mode: None,
    rate: constant(1, r(1, 10)),
    reset: vec![],
    mixture: vec![],
    reset_image: None,
  })
  .jump(JumpRow {
    mode: None,
    rate: mono(1, 0, 2, r(1, 1)),
    reset: vec![],
    mixture: vec![
      MixtureBranch {
        weight: r(1, 2),
        reset: vec![ResetRow {
          var: 0,
          poly: constant(1, r(0, 1)),
        }],
      },
      MixtureBranch {
        weight: r(1, 4),
        reset: vec![],
      },
    ],
    reset_image: None,
  })
  .finish();
JumpDiffusion::manifest(m).unwrap();
// book: api end jd_full_construction

```

## 12.11 Discrete-Time Markov Chains

**The frame.** Finite-state stochastic transition systems; the bread-and-butter question is certified reachability, with reward and hitting-time bounds beside it. A two-state weather chain makes the section's objects concrete: sunny stays sunny with probability  $\frac{3}{4}$ , rain persists with probability  $\frac{1}{2}$ , and a sunny start yields the two-step record (sunny, sunny, rain) with probability  $\frac{3}{4} \cdot \frac{1}{4} = \frac{3}{16}$  — one stored row spent per step, which is the whole compilation story below in miniature.

**Profile.** Outcome rows (*src*, *dst*, *reward*, *prob*) in exact rational arithmetic, rows summing to one per source, plus initial-distribution rows. Carrier: the declared finite state space. Time: counter.

**Counter time and the tick.** Composition into real time passes through the tick declaration of section 12.2 — part of the model type, not a checker convention, with the price of a jitter bracket over an exact tick certified rather than assumed (the factor-2.66 pair quoted there).

**Semantics.** The finite-state path measure, exact; backward-induction table soundness carried in checker-adjacent prose with its mechanized statement named. In the universe’s terms: the precise face (Remark 9.22) over discrete records, glued by the kernel-presented route of section 9.5, on which the natural extension reproduces local data exactly.

**Adequacy.** The compilation above is not merely well-typed; it is *adequate*: compiling and then denoting in the semantic universe agrees with interpreting classically and then including the classical object into the universe’s precise face. Stated self-contained, at the working scale of an exam problem:

**Problem 12.8** (adequacy of the chain compilation). Let  $S$  be a finite set,  $P \in \mathbb{Q}^{S \times S}$  a row-stochastic matrix,  $\iota \in \mathbb{Q}^S$  a distribution, and fix a horizon  $k$ . On the record space  $\Omega = S^{\{0, \dots, k\}}$  write  $\mu_{P, \iota}$  for the horizon- $k$  finite-dimensional distribution of the chain,

$$\mu_{P, \iota}(\omega) = \iota(\omega_0) \prod_{t < k} P(\omega_t, \omega_{t+1}),$$

the product form of Kemeny–Snell [135] and the horizon- $k$  generator of the cylinder-set path measure of Baier–Katoen [14]. Say a subprobability mass function  $\mu$  on  $\Omega$  *satisfies the compiled rows* when (i) for every  $\omega$ , the mass of  $\{\omega' : \omega'_0 = \omega_0\}$  equals the stored initial row  $\iota(\omega_0)$ , an absent row read as 0; and (ii) for every  $t < k$  and every  $\omega$ , writing  $C_t(\omega)$  for the event that a record agrees with  $\omega$  at all times  $\leq t$ , refinement by one step spends exactly one outcome row:  $\mu(C_{t+1}(\omega)) = \mu(C_t(\omega)) \cdot P(\omega_t, \omega_{t+1})$ , an absent row again read as 0. Show that  $\mu$  satisfies the compiled rows if and only if  $\mu = \mu_{P, \iota}$  — in particular the constraints force total mass one, so the ambient subprobability slack is never taken up.

*Sketch.* ( $\Leftarrow$ ) Downward induction from  $t = k$  gives the closed form  $\mu_{P, \iota}(C_t(\omega)) = \iota(\omega_0) \prod_{i < t} P(\omega_i, \omega_{i+1})$ : partition  $C_t(\omega)$  by the value at time  $t + 1$  and spend one row sum  $\sum_s P(\omega_t, s) = 1$  per step; both constraint families read off the closed form. ( $\Rightarrow$ ) Upward induction on  $t$  pins  $\mu(C_t(\omega))$  to the same prefix products; at  $t = k$  the prefix events are the atoms  $\{\omega\}$ , so all weights agree, and summing the atoms puts the total mass at one.

In the universe’s terms: the two constraint families, read as a formal conjunction, are the compiled system’s denotation at the per-element stratum of the credal stack, and the theorem says this loose arrow is isomorphic — by an invertible globular cell of the thin double category of per-element constraints — to the singleton inclusion of  $\mu_{P, \iota}$ : the precise face of Remark 9.22, whose ray form agrees with the singleton on the radially maximal slice that the verdict operators read. The statement and both directions are mechanized, sorry-free, in the toolkit’s Lean tree (Lean/su-functor/, module SuFunctor.Adequacy.Dtmc): the pointwise equivalence is `satAtoms_iff`, the invertible cell is `dtmc_adequacy`, and the mass identity is `pathLaw_mass`. The same module ties the exact-rational law to the standard monadic construction of the chain law in the mechanized probability library — iterated `PMF.bind` against the kernel — as `trajPMF_eq`, so the classical side of the square is the community’s mechanized object rather than a private restatement; the inclusion vocabulary shared by the sibling frames of this catalogue lives beside it (SuFunctor.Adequacy.Inclusions). The infinite-horizon path measure is deliberately a separate stratum: the laws above are its finite-dimensional distributions, and the  $\sigma$ -additive extension (Ionescu–Tulcea) belongs to the measure-theoretic tier.

**Status.** Certification is the most heavily exercised leg in the toolkit: the published-table campaign of section 2.3 (all seventeen rows of a peer-reviewed neural-certificate table [4] reproduced or refuted with exact arithmetic) and the tabular-tier campaign of section 2.4 (dozens of hitting and reward certificates) are both, at bottom, this frame’s machinery. Representation and production: the toolkit’s producer surface constructs the row grain in both languages and derives the one-step transfer and the generator values exactly from the stored rows — the compiled face of the Adequacy paragraph’s square and its generator kernel. Admission is the row grain’s own well-formedness: every source fibre totals exactly one — stochasticity is *declared by* the stored rows here, where the continuous-time sibling *derives* it from unconstrained rates, the same checker posture with the opposite direction of creation — so a sub-stochastic fibre refuses with its total (a deficit would be killing, which is the jump-diffusion frame’s vocabulary, never the chain’s).

The discrete segment-Dynkin identity is exercised in the test suite as an exact differential over a spread initial distribution, and the chain-3 mechanism is a committed cross-language byte-identity vector whose closed-form control includes the absorbing self-loop's computed zero (an absorbing state is its explicit self-loop row, the compiled grain's own representation — no flag). Open — and stated in full, since it is the list's recurring lesson: the frame has been exercised *beside* the modelling language, through producer pipelines, faster than it has been represented *in* it; native outcome rows through the ingestion path are the gap. The temporal-logic closure beyond reach and reach-avoidance (nested operators in the style of PCTL\*) is designed, and its first two strata are already in the language: state-level nesting is the tower discipline of the probability former — certificate-mediated: a child certificate's sublevel set enters as a parent claim's ordinary atom, cited through the judgment stratum (section 14.6.1) — and path-level recurrence shapes are definitional Streett instances (section 14.5.5). The genuine surplus — temporal operators nested with no intervening threshold — compiles by deterministic-monitor product: a finite deterministic monitor composed with the chain, acceptance read as Streett acceptance on the product, satisfaction transported back along the projection, with the quantitative case a reach stratum into the almost-sure accepting core. The next-step operator and the unbounded-until fence (section 14.9.4) are the named gaps; a certified depth-two artefact is the open exercise.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema's own (`crates/mtk-catalogue/src/system/`), and the former's mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{O : \langle \overline{src}, \overline{dst}, \overline{prob}, \overline{reward?} \rangle \text{ outcome rows in exact } \mathbb{Q}, \text{ unkeyed} \quad \iota? : \langle \overline{state}, \overline{prob} \rangle \text{ initial-distribution rows} \quad \tau? : \text{TickDecl}}{\text{DTMC}(O) : \text{System}} \quad \text{S-DTMC}$$

Unkeyed means no action anywhere ( $o.\text{action} = \perp$ , row by row) — the discipline that separates this former from the decision former sharing its rows.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.dtmc`

signature — `ts/src/system.ts`

```
/** S-DTMC: unkeyed outcome rows (+ init distribution, tick). */
export const dtmc = profileFactory("dtmc");

/** The derived DTMC generator kit: the one-step transfer
 * (P V)(s) =  $\sum_d p(s,d) \cdot V(d)$  and the generator values
 *  $\mathcal{N}V(s) = (P V)(s) - V(s)$  — the mechanized transferRow/genRow
 * contractions, whose recomputed rows are theorem-grade drift
 * premises. Derived, never trusted — the checker re-derives both. */
export interface DtmcGenKit {
  readonly gen: readonly StateVal[];
  readonly transfer: readonly StateVal[];
}
```

```

/** Derive the DTMC generator kit from a mechanism declaration and a
 * per-state certificate observable. Admission mirrors the WF
 * hypotheses under which the rows denote a chain: every roster
 * state's outcome fibre totals one (a sub-stochastic fibre refuses
 * with its total – deficit-as-killing belongs to the jump-diffusion
 * former), an action-keyed row refuses (the discipline separating
 * this former from the decision former sharing its rows), a stored
 * zero refuses (absence is the representation). Witness-shaped
 * refusals, never exceptions. */
export function dtmcGenKit(
  init: MechanismInit,
  cert: readonly StateVal[],
): Result<DtmcGenKit, Refusal> {
  const raw = native.sysDtmcGenKit(mechanism(init).nativeHandle, cert) as
    | { kit: DtmcGenKit }
    | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}

```

construction — ts/tests/system-native.test.ts

```

return sys.unwrap(
  sys.dtmc({
    name: "chain-3",
    outcomes: [
      { src: 0, dst: 0, prob: "1/3", reward: "0" },
      { src: 0, dst: 1, prob: "2/3", reward: "1" },
      { src: 1, dst: 2, prob: "1", reward: "5" },
      { src: 2, dst: 2, prob: "1" },
    ],
    initDist: [{ state: 0, prob: "1" }],
    tick: { kind: "exact", period: "1/50" },
  }),
);

```

The quoted construction is the committed cross-language byte-identity vector's own (the chain-3 mechanism), and the generator kit's closed-form control — transfer and generator on that same presentation, the absorbing self-loop's computed zero included — rides the test suite beside it. The kit's Rust face is `mtk_catalogue::system::dtmc_gen_kit`.

## Rust `mtk_catalogue::system::presets::Dtmc::manifest`

signature — crates/mtk-catalogue/src/system/presets.rs

```

preset!(
  /// System Catalogue: DTMC profile – outcome rows (exact-Q,
  /// unkeyed) + init distribution rows.
  Dtmc, "DTMC",
  req = [Outcomes],
  allowed = [InitDist, Tick],
  extra = |m| outcomes_keyed("DTMC", m, false)
);

```

construction — crates/mtk-catalogue/tests/common/mod.rs

```

let m = MechanismBuilder::new()
  .outcome(OutcomeRow {
    src: 0,
    action: None,
    dst: 0,
    prob: r(1, 3),
    reward: Some(r(0, 1)),
  })
  .outcome(OutcomeRow {
    src: 0,
    action: None,
    dst: 1,
    prob: r(2, 3),
    reward: Some(r(1, 1)),
  })

```

```

.outcome(OutcomeRow {
  src: 1,
  action: None,
  dst: 2,
  prob: r(1, 1),
  reward: Some(r(5, 1)),
})
.outcome(OutcomeRow {
  src: 2,
  action: None,
  dst: 2,
  prob: r(1, 1),
  reward: None,
})
.init_dist(0, r(1, 1))
.tick(TickDecl::Exact { period: r(1, 50) })
.finish();
presets::Dtmc::manifest(m)
.expect("fixture conforms to the DTMC profile")
.named("chain-3")

```

## 12.12 Continuous-Time Markov Chains

**The frame.** Exponential races over a finite state space; certified questions are time-bounded and unbounded reachability. A two-state repair chain is the toy to hold: a machine fails at rate  $\lambda$  and is repaired at rate  $\mu$ , so its jump chain alternates deterministically between up and down while the rates set only the clocks — the jump-chain/holding-time split at which the adequacy square below lives.

**Profile.** A shipped theory of its own: states, plus off-diagonal rate entries as a partial function — zero entries absent by construction, diagonal determined, so malformed rate matrices are unrepresentable rather than linted — plus generator-certificate rows. Time: continuous.

**Semantics.** The finite-state generator; the path measure is unique (no explosion at finite state), and the Dynkin and Ville bridges are named obligations, unconditional in this setting but still deliberately outside the kernel. At finite state the classical path-measure construction discharges, for this frame, the continuous-time extension debt section 9.5 names in general.

**Adequacy (jump-chain face).** The compilation above is adequate at its *time-abstract* stratum: the embedded jump chain — which states the process visits, in which order — is exact rational data, and compiling the rate rows then denoting agrees with interpreting the jump chain classically and including into the universe's precise face. Stated self-contained, at the working scale of an exam problem:

**Problem 12.9** (the jump-chain square). Let  $S$  be a finite set,  $q(s, s') \in \mathbb{Q}_{\geq 0}$  a transition rate for each ordered pair of distinct states, and  $\iota \in \mathbb{Q}^S$  a distribution. Write  $q(s) = \sum_{s' \neq s} q(s, s')$  for the exit rates, and let  $\Pi$  be the jump matrix of Norris [190]: when  $q(s) \neq 0$ ,  $\Pi(s, s') = q(s, s')/q(s)$  for  $s' \neq s$  and  $\Pi(s, s) = 0$ ; when  $q(s) = 0$ , the absorbing convention  $\Pi(s, s) = 1$  and  $\Pi(s, s') = 0$  otherwise. Fix a jump horizon  $k$  and write  $\mu_{q, \iota}$  for the horizon- $k$  path law of the chain  $(\Pi, \iota)$  on  $\Omega = S^{\{0, \dots, k\}}$  — the record of the first  $k$  jumps, not of clock time. The compiled presentation stores the off-diagonal rate rows  $(s, s') \mapsto q(s, s')$ , an absent row read as 0 and a diagonal row unrepresentable; the jump closure is the *computed* row set  $\Pi$ , and a subprobability mass function  $\mu$  on  $\Omega$  satisfies the closure exactly when it satisfies the chain constraints of section 12.11 at the rows  $(\Pi, \iota)$ . Show: (i) the closed rows are row-stochastic, although the rate rows carry no row-sum constraint; (ii)  $\mu$  satisfies the closure's constraints if and only if  $\mu = \mu_{q, \iota}$  — in particular the constraints force total mass one; and (iii)  $\Pi(s, s') q(s) = q(s, s')$  for all  $s \neq s'$ , absorbing states included.

*Sketch.* (i) is one division: at a non-absorbing state the row sums to  $q(s)/q(s)$ , at an absorbing state to a point mass — the stochasticity of the jump chain is derived, never assumed. The pair  $(\Pi, \iota)$  is then a finite chain, and both directions of (ii) are the chain problem verbatim at these rows: the downward induction reads off the closed form on prefix events, the upward induction pins any satisfying  $\mu$ , and mass one follows by summing the atoms. (iii) is the recovery direction of the jump-chain/holding-times decomposition: at



an absorbing state both sides vanish, since nonnegative rates summing to zero vanish individually. No further stochastic content is specific to the continuous-time former — closing the rate rows before or after compilation stores the same rows, which is the whole remaining obligation.

In the universe’s terms: the jump closure denotes as a loose-arrow diagram at the per-element stratum, and the theorem exhibits an invertible globular cell to the singleton inclusion of the jump-chain law — the chain square instantiated at the jump chain. The statement and both directions are mechanized, sorry-free, in the toolkit’s Lean tree (`lean/su-functor/`, module `SuFunctor.Adequacy.CtmcJumpChain`): the invertible cell is `ctmc_jump_adequacy`; the compile-commutes triangle — definitional, and the one obligation well-formedness alone could not carry, since any rate table closes to *some* stochastic rows — is `compile_jump_comm`; the derived stochasticity is `jumpRows_wf`; the rate factorization is `jumpChain_step_mul_exit`; and the profile’s diagonal-determined convention is carried as theorems rather than prose (`compileCtmc_diag_blind`, `jumpLaw_diag_blind`: the compiled key type cannot express a diagonal read). The chain connector transports unchanged (`jumpTrajPMF_eq`), which is the exact extent of the community tie: the mechanized probability library has no continuous-time Markov machinery, so the jump chain is tied as the discrete-time chain it is. Deliberately separate strata: the *time-law* stratum — exponential holding times with parameter  $q(s)$ , whose transients are irrational and hence outside this exact-rational grain — carries the rest of the continuous-time path semantics named in the Semantics paragraph above; *uniformization* is a different discrete reduction (self-loops against a common rate) and not this one; Markov automata under a declared scheduler consume this square together with the decision-former machinery (section 12.15); the infinite-horizon extension is separate exactly as for chains.

**Status.** Certification: three certified bounds over four kernel-checked fixtures — time-bounded reach for an explicit repair chain ( $P(\text{down by } T) \leq T/200$ , with the equalized certificate’s constant meeting the expected hitting time exactly, no matrix exponential anywhere), a catastrophe-entry bound exercising arbitrary rate matrices, and an unbounded-reach bound for a nine-state epidemic chain (a Ville argument spending two fixtures: the generator sequent and its nonnegativity side condition), against an exact ground-truth ruin probability of 35/274391. Representation and production: theory and instance rows are in place and compile through the reference lane, and the toolkit’s producer surface constructs the rate grain in both languages and derives the generator values, the exit rates, and the jump-closure rows exactly from the stored entries — the compiled face of the Adequacy paragraph’s square: the closure’s row-stochasticity, a theorem there, is re-run as arithmetic here, the rate factorization is exercised in the test suite as an exact differential, and the machine-repair chain is a committed cross-language byte-identity vector whose closed-form control is its own certified values ( $AV = 3/755$  at every non-bad state, the equalized certificate’s constant, derived and never trusted). Open: nested temporal operators; the ingestion gap; closure.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ ,  $\text{pop}$ , and profile conformance are defined in section 12.1.

$$\frac{R : \overrightarrow{\langle \text{src}, \text{dst}, \text{rate} \rangle} \text{ off-diagonal rate entries } (\text{src} \neq \text{dst}) \quad \iota? \text{ initial-distribution rows}}{\text{CTMC}(R) : \text{System}} \text{ S-CTMC}$$

Rate entries are an off-diagonal partial function: zero entries are absent by construction and the diagonal is determined, so a malformed rate matrix is unrepresentable rather than linted.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a

native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

## TypeScript `system.ctmc`

signature — `ts/src/system.ts`

```
/** S-CTMC: off-diagonal rate entries as a partial function. */
export const ctmc = profileFactory("ctmc");

/** One per-state exact-rational value: the discrete-former
 * certificate observable's grain (and the kit outputs' row shape). */
export interface StateVal {
  readonly state: number;
  readonly val: string;
}

/** The derived CTMC generator kit: the generator values
 *  $\dot{V}(s) = \sum_d q(s,d) \cdot (V(d) - V(s))$ , the exit rates, and the
 * jump-closure rows (Norris's convention: proportional split at a
 * non-absorbing state, the self-loop point mass at an absorbing
 * one) — row-stochastic by derivation, never by declaration.
 * Derived, never trusted — the checker re-derives all three. */
export interface CtmcGenKit {
  readonly gen: readonly StateVal[];
  readonly exitRates: readonly StateVal[];
  readonly jumpRows: readonly {
    readonly src: number;
    readonly dst: number;
    readonly prob: string;
  }[];
}

/** Derive the CTMC generator kit from a mechanism declaration and a
 * per-state certificate observable. The mechanized contract is the
 * CTMC adequacy square's, jump-chain face: the closure's
 * row-stochasticity is a theorem there (derived stochasticity), and
 * the rate factorization  $\Pi(s,d) \cdot q(s) = q(s,d)$  ties the outputs. The
 * grain fences (diagonal, parallel, or nonpositive rate entries;
 * missing or duplicate certificate values) come back as
 * witness-shaped refusals, never exceptions. */
export function ctmcGenKit(
  init: MechanismInit,
  cert: readonly StateVal[],
): Result<CtmcGenKit, Refusal> {
  const raw = native.sysCtmcGenKit(mechanism(init).nativeHandle, cert) as
    | { kit: CtmcGenKit }
    | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}
```

construction — `ts/tests/system-native.test.ts`

```
// CTMC diagonal rate entry: index and state.
const diag = sys.ctmc({
  rates: [
    { src: 0, dst: 1, rate: "1" },
    { src: 1, dst: 1, rate: "2" },
  ],
});
assert.ok(!diag.ok);
```

The TS corpus exercises this factory through its refusal geometry (the corpus's discipline that failure geometry is routing advice); the quoted construction is that witness, and the happy-path byte authority is the Rust corpus opposite — the machine-repair mechanism is a committed cross-language byte-identity vector, and the generator kit's closed-form control is its own certified values. The kit's

Rust face is `mtk_catalogue::system::ctmc_gen_kit`.

**Rust** `mtk_catalogue::system::presets::Ctmc::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```

    preset!(
        /// System Catalogue: CTMC profile – off-diagonal rate entries as
        /// a partial function (zero entries absent; diagonal determined,
        /// never stored).
        Ctmc, "CTMC",
        req = [Rates],
        allowed = [InitDist],
        extra = no_diagonal_rates
    );

```

construction — `crates/mtk-catalogue/tests/builders.rs`

```

    let ctmc = MechanismBuilder::new().rate(0, 1, r(3, 2)).finish();
    Ctmc::manifest(ctmc).unwrap();

```

## 12.13 Markov Decision Processes

**The frame.** Kernels keyed by action; the frame’s whole point is quantification over policies. The toy to hold: a machine that can be repaired now (a certain cost) or run on (a risk of failure) — two actions at every state, and the question that makes the frame is a guarantee holding whichever way the operator chooses, forever.

**Profile.** Outcome rows keyed by action; a policy is a separate mechanism, composed in — the closed loop is a composite, not a monolith. Time: counter.

**Semantics.** Per-policy path semantics as for chains; the quantification over policies lives in the credal (imprecise-measure) layer of the System semantics (Definition 9.15), where a specification reads the denotation at a stated degree of pessimism.

**Adequacy (fixed-policy face).** The closed-loop composite is adequate in the same sense as the chain compilation of section 12.11 — compiling both mechanisms, closing the loop at the row level, and denoting agrees with interpreting classically and including into the universe’s precise face — and the proof is the chain square, reused wholesale. Stated self-contained, at the working scale of an exam problem:

**Problem 12.10** (the fixed-policy square). Let  $S$  and  $A$  be finite sets,  $p \in \mathbb{Q}^{S \times A \times S}$  a family of kernels with  $\sum_{s'} p(s' | s, a) = 1$  for every state–action pair,  $\iota \in \mathbb{Q}^S$  a distribution, and  $\pi$  a memoryless randomized policy: at each state, a distribution over actions — the stationary randomized Markov class of Puterman [203], a deterministic policy being the unit-mass special case. Fix a horizon  $k$  and write, on  $\Omega = S^{\{0, \dots, k\}}$ ,

$$\mu_{p, \iota, \pi}(\omega) = \iota(\omega_0) \prod_{t < k} \sum_a \pi(a | \omega_t) p(\omega_{t+1} | \omega_t, a),$$

the fixed-policy trajectory law. The compiled presentation keeps the two mechanisms separate: action-keyed outcome rows  $(s, a, s') \mapsto p(s' | s, a)$  and policy rows  $(s, a) \mapsto \pi(a | s)$ , an absent row in either read as 0; the closed loop is the *computed* row set  $P_\pi(s, s') = \sum_a \pi(a | s) p(s' | s, a)$ , and a subprobability mass function on  $\Omega$  satisfies the composite exactly when it satisfies the chain constraints of section 12.11 at the rows  $(P_\pi, \iota)$ . Show that  $\mu$  satisfies the composite’s constraints if and only if  $\mu = \mu_{p, \iota, \pi}$  — in particular the constraints force total mass one.

*Sketch.* One exchange of finite sums shows the mixed matrix  $P_\pi$  is row-stochastic —  $\sum_{s'} P_\pi(s, s') = \sum_a \pi(a | s) \sum_{s'} p(s' | s, a) = 1$  — spending each policy row’s total exactly once against the per-action row sums. The pair  $(P_\pi, \iota)$  is therefore a finite chain, and  $\mu_{p, \iota, \pi}$  is by construction its product-form path law. Both directions are then the chain problem verbatim at these rows: the downward induction reads off the closed form on prefix events, the upward induction pins any satisfying  $\mu$ , and mass one follows by

summing the atoms. No further stochastic content is specific to the decision former — closing the loop before or after compilation stores the same rows, which is the whole remaining obligation.

In the universe’s terms: the composite denotes as a loose-arrow diagram at the per-element stratum, and the theorem exhibits an invertible globular cell to the singleton inclusion of the fixed-policy law — the same form as the chain square, because it *is* the chain square instantiated at the closed loop. The statement and both directions are mechanized, sorry-free, in the toolkit’s Lean tree (lean/su-functor/, module SuFunctor.Adequacy.Mdp): the invertible cell is `mdp_adequacy`, the compile-commutes triangle — definitional, and the one obligation well-formedness alone could not carry, since any single action’s rows are also row-stochastic — is `compile_close_comm`, and the mass identity is `pathLaw_mass`. The module also ties policy selection to the mechanized probability library’s own vocabulary: the closed-loop kernel is the monadic composition of the policy’s action distribution with the action-indexed kernel (`closedLoop_stepPMF`, one `PMF.bind`), and the trajectory-law connector of the chain section transports unchanged (`trajPMF_eq`). Deliberately separate strata: the *open-policy* face — the claim uniform over all policies, whose inclusion lands a *set* of laws in the credal layer rather than a singleton — is the imprecise reading of the Semantics paragraph above and is not this square (it is the next); *time-dependent* policies close to a time-inhomogeneous chain, outside the stationary grain stated here; Markov automata under a declared scheduler share this square’s mixing machinery but carry exponential holding times, so their square is *continuous-time* material (section 12.15); the infinite-horizon extension is separate exactly as for chains.

**Adequacy (demonic face).** The open-policy face is the first *genuinely imprecise* adequacy statement: what compilation must preserve is no longer a single law but the *set* of laws the decision process can exhibit, and the square says the compiled, policy-free presentation determines that set exactly. Stated self-contained, at exam scale:

**Problem 12.11** (the demonic square). Data as in the fixed-policy problem above — finite  $S$  and  $A$ , action-keyed kernels  $p$ , initial distribution  $\iota$  — but with *no* policy declared. The demonic semantics of the decision process at horizon  $k$  is the set of its closed-loop trajectory laws,

$$D = \{ \mu_{p,\iota,\pi} : \pi \text{ a memoryless randomized policy} \},$$

the adversary ranging over the stationary randomized class of Puterman [203] — the scheduler-quantified semantics of probabilistic model checking [14], and demonic choice in the sense of McIver and Morgan [176]. On the compiled side, call a subprobability mass function  $\mu$  on  $\Omega$  *admitted* by the action-keyed rows when *some* well-formed policy-row set closes the loop onto chain constraints that  $\mu$  satisfies — the rows carry no policy family; the demon’s choice is the separate mechanism, existentially quantified. Show: (i)  $\mu$  is admitted if and only if  $\mu \in D$ ; (ii) every member of  $D$  has total mass one, and consequently the downward-closed credal set generated by  $D$  — the union of the segments  $[0, 1] \cdot \mu$  over  $\mu \in D$  — has exactly  $D$  as its set of radially maximal elements; (iii) for an event  $E$  and threshold  $q$ , the credal set’s lower verdict for  $P(E) \geq q$ , and its upper verdict for the downward-closed constraint  $P(E) \leq q$ , each hold if and only if the corresponding bound holds under *every* policy.

*Sketch.* Claim (i) is the fixed-policy problem, quantified: a witnessing scheduler’s rows assemble a memoryless policy whose closed loop the chain constraints pin, and conversely every policy compiles to a witnessing scheduler — no new stochastic content. In (ii), mass one is the chain mass identity at each closed loop; for the slice, a strictly interior point of a segment escapes upward to its own base, while an upward scaling of one member landing on *another* member’s segment would have to scale mass past the common value — equal mass is exactly what forbids cross-member domination, and it is not decorative: in the unequal-mass pair  $\{\mu, \frac{1}{2}\mu\}$  the half element is dominated across segments and drops out of the slice. In (iii), the lower verdict reads the radially maximal slice (Remark 9.16), which by (ii) is  $D$  itself; the upper verdict reads the whole set, and downward closure of  $P(E) \leq q$  makes it blind to the segments’ interiors. Reading a lower bound against the whole set instead would refuse every positive threshold — the zero measure lies in every segment — which is the slice discipline’s whole content, surviving the passage to sets.

In the universe’s terms: the demonic reading of the compiled former and the union-of-singletons inclusion of  $D$  are isomorphic by an invertible globular cell at the per-element stratum — the first adequacy square whose bottom edge is set-valued, one singleton inclusion per policy (`unionSingletons`) — and the credal presentation, the union of the members’ rays (`FinMDP.demonicCredal`), reconciles with  $D$  through the

radially maximal slice, which is what the verdict operators read. The presentation is radial, and in general *not* convex: a mixture of two deterministic-policy path laws is realized by no memoryless policy, only by an ex-ante correlated scheduler, so the closed convex hull is the coarser convex reading — and the slice reconciliation is a statement the hull would coarsen. The mechanized results consume radially and equal mass only; closedness of the policy-to-law image, which would make the union a credal body outright, is not among them. The statement and both directions are mechanized, sorry-free, in the toolkit’s Lean tree (`lean/su-functor/`, modules `SuFunctor.Adequacy.EnvelopeUnion` and `SuFunctor.Adequacy.MdpDemonic`): the invertible cell is `mdp_demonic_adequacy`, with admission predicate `demonicC` and feeding lemma `demonicC_iff`; the equal-mass slice reconciliation is `maxSlice_unionRays`, its family-bound direction hypothesis-free (`maxSlice_unionRays_subset`), instantiated at mass one by `demonic_maxSlice`; the uniform-over-policies verdict theorems are `satCredal_lower_demonicCredal` and `satCredal_upper_demonicCredal`; and the degeneracies are mechanized rather than smoothed: the whole-set reading of a lower bound refuses every positive threshold (`not_satCredal_upper_demonicCredal_gec`); an action-free process is well-formed yet admits no policy at all (`isEmpty_policy_iff`), so both verdict kinds pass vacuously (the  $\forall$ -over-no-adversaries convention, the empty-policy union staying below bottom with no body); and the unequal-mass pair witnesses the slice lemma’s hypothesis (`not_maxSlice_unionRays_halfPair`). Deliberately separate strata, unchanged from the fixed-policy face: memoryless policies only; the *angelic* reading — existential over policies — is a different operator from the universally quantified verdict pair and is not built here; the infinite-horizon extension is separate exactly as for chains.

**Status.** An asymmetric grade, stated as such. *Fixed policy*: shipped — the controller-checkpoint certificates of the experiments chapters are policy-composed MDP verification, and the exact value-iteration route [112, 204] exports value functions as certificate input. The toolkit’s producer surface now constructs the closed loop itself in both languages: the policy arrives as the scheduler grain — the compiled face of the composed policy mechanism, never a family of the decision former, exactly the factoring the formation rule below states — and the kit derives the closed chain (self-loops kept, initial rows verbatim) and its generator by handing the closure to the chain kit, the square’s engine reuse run producer-side. The mixing exchange — the compile-commutes triangle — is re-derived as an exact differential in the test suite through a read that never touches the closure, with the deterministic read-through as its control; policy support discipline is the automaton’s (support within the enabled pairs, mass one at every state, and a state with no enabled actions refuses — nothing sound closes there, the empty-action degeneracy’s producer face); and the patrol mechanism is a committed cross-language byte-identity vector. *Uniform over policies*: the open-nondeterminism question is designed at the semantics layer and exercised by workshop fixtures with checker acceptance — not yet by a published reproduction target. Until that leg ships as a reproduction target, this frame is graded on its closed-loop half alone — which is precisely the half that does not distinguish it from section 12.11. Open: the quantified leg end to end; the ingestion gap; closure.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{O : \langle \text{src, action, dst, prob, reward?} \rangle \text{ outcome rows, every row action-keyed} \quad \iota? \text{ initial-distribution rows} \quad \tau? : \text{TickDecl}}{\text{MDP}(O) : \text{System}} \quad \text{S-MDP}$$

The same outcome rows as the chain former, all action-keyed; the policy is deliberately *not* a premise — it is a separate mechanism, composed in, so the closed loop is a composite.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

## TypeScript `system.mdp`

signature — `ts/src/system.ts`

```
/** S-MDP: action-keyed outcome rows; the policy is a separate
 * mechanism composed in (closed-loop composite), never a family. */
export const mdp = profileFactory("mdp");

/** The derived MDP fixed-policy kit: the closed discrete-time chain
 * as ordinary DTMC rows (self-loops kept, initial rows verbatim) and
 * the closed chain's generator kit — the mechanized square's engine
 * reuse (mdp_adequacy := dtmc_adequacy at the closed loop), run
 * producer-side. Derived, never trusted — the checker re-runs the
 * closure and the contraction. */
export interface MdpGenKit {
  readonly closedRows: readonly {
    readonly src: number;
    readonly dst: number;
    readonly prob: string;
  }[];
  readonly closedInit: readonly {
    readonly state: number;
    readonly prob: string;
  }[];
  readonly closed: DtmcGenKit;
}

/** Derive the MDP fixed-policy kit from a mechanism declaration, a
 * policy, and a per-state certificate observable. The policy arrives
 * as the scheduler grain — the compiled face of the COMPOSED POLICY
 * MECHANISM (the decision former deliberately has no policy family;
 * an absent policy is the demonic face, which quantifies
 * certificate-side and needs nothing from this surface). Policy
 * support discipline as at the automaton: support within the enabled
 * pairs, mass one at every roster state, and a state with no enabled
 * actions refuses (nothing sound closes there). Witness-shaped
 * refusals, never exceptions. */
export function mdpGenKit(
  init: MechanismInit,
  policy: SchedulerDecl,
  cert: readonly StateVal[],
): Result<MdpGenKit, Refusal> {
  const raw = native.sysMdpGenKit(
    mechanism(init).nativeHandle,
    {
      rows: policy.rows.map((r) => ({
        state: r.state,
        action: r.action,
        prob: qText(r.prob),
      })),
      note: policy.note,
    },
    cert,
  ) as { kit: MdpGenKit } | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}
```

construction — `ts/tests/mdpgen-native.test.ts`

```
/** The `patrol-2s2a` presentation (the committed golden fixture's):
 * at state 0, `stay` self-loops and `go` branches 1/4 / 3/4; at
```



```

* state 1, `go` branches evenly. The policy is passed SEPARATELY –
* the decision former has no policy family, and the scheduler grain
* here is the compiled face of the composed policy mechanism. */
function patrolInit() {
  return {
    outcomes: [
      { src: 0, action: 0, dst: 0, prob: "1" },
      { src: 0, action: 1, dst: 0, prob: "1/4" },
      { src: 0, action: 1, dst: 1, prob: "3/4" },
      { src: 1, action: 1, dst: 0, prob: "1/2" },
      { src: 1, action: 1, dst: 1, prob: "1/2" },
    ],
    initDist: [{ state: 0, prob: "1" }],
  } as const;
}

const patrolPolicy = {
  rows: [
    { state: 0, action: 0, prob: "1/3" },
    { state: 0, action: 1, prob: "2/3" },
    { state: 1, action: 1, prob: "1" },
  ],
} as const;

const patrolCert = [
  { state: 0, val: "0" },
  { state: 1, val: "1" },
];

test("the patrol closes to the all-halves chain under its policy", { skip }, async () => {
  const sys = await ns();
  const kit = sys.unwrap(
    sys.mdpGenKit(patrolInit(), patrolPolicy, patrolCert),
  );
  const closed = new Map(
    kit.closedRows.map((row) => [`${row.src}->${row.dst}`, row.prob]),
  );
  assert.equal(closed.get("0->0"), "1/2"); // self-loop KEPT
  assert.equal(closed.get("0->1"), "1/2");
  assert.equal(closed.get("1->0"), "1/2");
  assert.equal(closed.get("1->1"), "1/2");
  assert.deepEqual(kit.closedInit, [{ state: 0, prob: "1" }]);
  const gen = new Map(kit.closed.gen.map((v) => [v.state, v.val]));
  assert.equal(gen.get(0), "1/2");
  assert.equal(gen.get(1), "-1/2");
});

```

The quoted construction is the committed cross-language byte-identity vector's own (the patrol mechanism), with the policy passed separately — the decision former has no policy family, and the scheduler grain at the kit call is the compiled face of the composed policy mechanism. The kit's Rust face is `mtk_catalogue::system::mdp_gen_kit`; the refusal-geometry witness remains in the TS corpus beside it.

## Rust `mtk_catalogue::system::presets::Mdp::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```

preset!(
  /// System Catalogue: MDP profile – outcome rows keyed by action;
  /// the policy is a separate mechanism composed in (closed-loop
  /// composite), never a family here.
  Mdp, "MDP",
  req = [Outcomes],
  allowed = [InitDist, Tick],
  extra = |m| outcomes_keyed("MDP", m, true)
);

```

construction — `crates/mtk-catalogue/tests/builders.rs`

```

let mdp = MechanismBuilder::new()
  .outcome(OutcomeRow {
    src: 0,

```

```

        action: Some(0),
        dst: 0,
        prob: r(1, 1),
        reward: None,
    })
    .finish();
Mdp::manifest(mdp).unwrap();

```

## 12.14 Partially Observable Markov Decision Processes

**The frame.** Kernels keyed by action, as for the decision former — plus an observation channel, and the frame’s whole point is that the policy sees observations, not states. The classical formulation is Åström’s [10]: optimal control of a Markov process with incomplete state information. The toy to hold is the classic two-door problem — the *tiger* problem of the planning literature [129]: a prize behind one of two doors, a hazard behind the other, and a listening action whose report is right only most of the time. The question that makes the frame is a guarantee on achieved value for a controller that can never look behind the doors — only accumulate evidence about them.

**Profile: none — a composition pattern.** This frame adds no formation rule, no new mechanism family, and no new semantic species; it stands to the decision former of section 12.13 as the delay equation of section 12.6 stands to the ODE cell — a composite of inventory the catalogue already carries, presented through a wiring. The pieces: the *environment* is a decision-former mechanism, action-keyed outcome rows over a state the composite will hide; the *observation channel* is an ordinary mechanism whose rows carry kernels from state to observation symbols (an action-dependent channel folds in by state augmentation); the *controller* is a separate mechanism whose inputs are the channel’s outputs, never the environment’s state; and the loop closes through a *delayed trace* — the feedback row of the composition-patterns chapter’s delay inventory (section 13.10.5), which names this very loop as its flagship construction, and whose one-step degenerate case, the tick register, is the medium a counter-time closed loop needs. Where the delay-equation composite ran its feedback through a continuous transport medium, this one runs it through the tick register: the same pattern family, at the counter-time profile.

The wiring is not packaging; it is the frame’s semantic content. What distinguishes a partially observable decision process from the decision former is *which wires reach the policy*, and the composite carries that distinction as typing: the controller’s inputs are, by the shape of the pattern, the observation channel’s outputs together with its own state, so there is no way to write an instance of the pattern whose policy reads hidden state. Partial observability is a composition pattern, not a new model class.

**Semantics.** Per-policy path semantics on the closed loop’s trajectory law, exactly as for the decision former, with the quantification over policies again living in the credal layer (Definition 9.15) — and the one genuinely new ingredient relative to section 12.13 is the class quantified over: *observation-measurable* policies, policies factoring through the history of observations and of their own past actions, never the history of states. Stated classically, the policy is adapted to the observation filtration; stated compositionally, the restriction is carried by the wiring itself, which is what the paragraph above bought — the measurable-selection bookkeeping that the frame’s textbook development devotes to this restriction attaches here to compile-level metatheory, part of the square owed below, not of the presentation. The belief-state reformulation enters the same way, as presentation rather than semantics: Åström’s re-reading of the loop as a fully observed decision process on the belief simplex [10] is, in this inventory, a particular choice of controller — a belief-tracking mechanism whose internal state is a distribution over environment states, updated observation by observation, composed in like any other policy. (In unnormalized coordinates the exact Bayes update is polynomial — linear in the belief and in each mechanism’s rows, with no division — hence ordinary mechanism content; the normalized update, a ratio, is not, which is why the unnormalized form is the one the inventory stages.) The simplex is controller state, not a new semantic carrier.

**Adequacy.** Owed, and stated as such: no square is mechanized for this frame yet. What the fixed-policy face must say is engine reuse once more — at a declared observation-measurable policy, the environment, channel, and controller compile to rows, the loop closes at the row level, and the composite is a chain on finitely many reachable states at any fixed horizon (one belief per action–observation history), so

the obligation has the shape the decision former discharged in section 12.13, with the controller's belief coordinate folded into the composite state. The quantified face inherits the demonic square's set-valued form, with the adversary now ranging over the observation-measurable class — policies reading strictly less than the state, though carrying memory the stationary class of section 12.13 deliberately excluded — and the delayed trace is what makes the restriction structural rather than side-conditional. The intended first certificate is a lower bound on achieved value for the two-door problem, living on the belief simplex: a finite family of affine functionals on the simplex — the  $\alpha$ -vector presentation of Smallwood and Sondik [221] — each dominated by its witness action's Bellman backup under the discounted criterion, certifying at once a value floor and the policy that achieves it.

**Status.** Design of record, with the certificate route already exercised. The wiring — environment, observation channel, belief-tracking controller, delayed trace — is the composition-patterns chapter's own flagship construction (section 13.10.5), the producer surface for the delayed trace is landed there, and the decision former's closed-loop producer machinery is the engine the fixed-policy face will reuse. The two-door lower bound has been produced and checked in the workshop by the route the adequacy paragraph names — an exact-rational  $\alpha$ -vector certificate with per-state Bellman-domination witnesses, re-verified by a small exact checker — not yet as a published reproduction target, so the frame is graded here on design and workshop evidence alone. Open: the fixed-policy square; the quantified face end to end; promotion of the two-door certificate to a reproduction target; and the ingestion gap, as everywhere.

No formation rule accompanies this section: that absence is the section's content, for the same reason as in section 12.6 — the frame *is* a wiring of formers the catalogue already carries, and a dedicated former would store as data what the composite derives by shape.

## 12.15 Markov Automata

**The frame.** Exponential races, probabilistic branching, and nondeterministic action under maximal progress — the frame that composes sections 12.12 and 12.13.

**Profile.** Assembled by composition, not by a new theory: the chain frame's rate idiom, the decision frame's action-kernel idiom aimed at immediate transitions, and exactly one new atom — a scheduler declaration — with a closure bridge delivering the closed object as an ordinary continuous-time chain, certificate and all. Maximal progress is stated as structure: timed rows at instantaneous states are unrepresentable.

**Semantics.** The interleaving semantics via a fixed-scheduler reduction note (closure formula, self-loop thinning, Zeno-freedom as the closure's checked assertion), with path-measure bridges inherited unchanged from the chain frame.

**Adequacy (fixed-scheduler face).** The closure bridge above is adequate at the automaton's *time-abstract* stratum: under a declared scheduler, closing the compiled rows to a chain and then reading its jump skeleton agrees with interpreting the automaton classically and including into the universe's precise face. Stated self-contained, at the working scale of an exam problem:

**Problem 12.12** (the fixed-scheduler square). Let  $S$  and  $J$  be finite sets (the Markovian and the instantaneous states of a closed Markov automaton in the sense of Eisentraut, Hermanns, and Zhang [67]),  $A$  a finite action set. The data are: rates  $q(s, x) \in \mathbb{Q}_{\geq 0}$  for  $s \in S$ ,  $x \in S \sqcup J$  with  $x \neq s$  — a timed row at an instantaneous state and a diagonal entry are both *unrepresentable*, which is the maximal-progress assumption and the determined diagonal stated as structure — immediate kernels  $b(j, a, \cdot)$ , each a distribution on  $S \sqcup J$ ; an initial distribution  $\iota$  on  $S \sqcup J$ ; a scheduler  $\varsigma$  assigning each  $j \in J$  a distribution over  $A$ ; and a rank  $\rho: J \rightarrow \mathbb{N}$  such that the mixed immediate kernel  $f(j, x) = \sum_a \varsigma(j)(a) b(j, a, x)$  satisfies  $\rho(j') < \rho(j)$  whenever  $f(j, \text{in}(j')) \neq 0$  — the Zeno-freedom assertion, carried as a ranking witness. Define the absorption kernel  $P$  by iterating  $f$  (fuel one past the largest rank), a point mass at each Markovian state, and the closed chain by  $q^\varsigma(s, s') = \sum_x q(s, x) P(x)(s')$  for  $s' \neq s$ , with initial distribution  $\iota$  pushed through  $P$ . Show: (i)  $P$  is a distribution over  $S$  from *every* state, although neither the rate rows nor the iteration carried a normalization constraint, and it is the *unique* solution of the absorbing-system equations; (ii) the closed exit rate is the original exit rate minus the mass  $P$  returns to the source (self-loop thinning), so a scheduler can only remove

activity; (iii) writing  $q(s)$  for the exit rates and  $\Pi$  for the jump matrix of the automaton's timed part on  $S \sqcup J$  (instantaneous states rate-absorbing), the closure also reads  $q^\epsilon(s, s') = q(s) \sum_x \Pi(\text{in}(s), x) P(x)(s')$  — the race-then-jump-then-absorb spelling agrees with the rate spelling; and (iv) a subprobability mass function  $\mu$  on the jump-horizon- $k$  records of  $S$  satisfies the jump closure of the scheduler closure of the compiled rows if and only if  $\mu$  is the closed chain's jump-chain path law.

*Sketch.* (i) is an induction along the rank: the mixed kernel is row-stochastic by the policy mixing exchange of section 12.13, and past the rank the iteration has stabilized — Zeno-freedom is spent exactly here, since an immediate self-loop would never absorb; uniqueness is the same induction applied to the difference of two solutions. (ii) is the mass computation of (i) summed against the rate rows. (iii) is the rate-factorization identity of section 12.12 —  $\Pi(s, x) q(s) = q(s, x)$ , absorbing states included — applied under the absorption sum. (iv) is the chain problem verbatim at the closed rows: the one genuinely automaton-shaped obligation is that scheduler-closing the compiled rows is the compilation of the classically closed chain, which holds definitionally when both routes read the same rows through the same guarded reads and the same absorbing iteration — and it is what pins the closure to the declared scheduler and the declared rates, since well-formedness alone can see neither.

In the universe's terms: the scheduler closure followed by the jump closure denotes as a loose-arrow diagram at the per-element stratum, and the theorem exhibits an invertible globular cell to the singleton inclusion of the closed jump law — the chain square instantiated twice, three storeys of engine reuse with no new induction. The statement and both directions are mechanized, sorry-free, in the toolkit's Lean tree (lean/su-functor/, module SuFunctor.Adequacy.MarkovAutomaton): the invertible cell is `ma_adequacy`; the rate-closure triangle — definitional — is `compile_closeMa_comm`; the acyclic-absorption package is `absorbIter_mass`, `absorbIter_unique`, and `absorbIter_stab`; the two closure spellings are `close_offRate_factor`, consuming the chain face's `jumpChain_step_mul_exit`; the thinning identity and its inequality are `close_exitRate_thinning` and `close_exitRate_le`; and the profile's structural conventions are carried as theorems (`compileMa_diag_blind`: the compiled key type can express neither a self-rate nor a timed row at an instantaneous state). The scheduler grain is the decision former's policy grain reused verbatim, and the community tie is the chain connector transported through both closures (`FinMA.trajPMF_eq`) — the mechanized probability library has no Markov-automaton machinery, and the connector says so rather than dressing the transport as more. Deliberately separate strata: the exponential *time-law* stratum, inherited from section 12.12; *scheduler quantification*, the envelope stratum, addressed by the demonic face in the next paragraph; *parallel composition and visible actions*, the open-automaton face (this square stages one closed component); history-dependent schedulers; and non-Zeno-free automata, outside the square by hypothesis rather than silently discharged.

**Adequacy (demonic face).** The scheduler-quantification face, and the set-valued square at continuous time — the automaton sibling of section 12.13's demonic face, with one genuinely automaton-shaped complication: each scheduler's closure consumes a Zeno-freedom certificate. Stated self-contained, at exam scale:

**Problem 12.13** (the demonic square, at continuous time). Data as in the fixed-scheduler problem above — finite  $S, J, A$ ; rates  $q$ ; immediate kernels  $b$ ; initial distribution  $\iota$  — but with *no* scheduler declared. Instead, a single rank  $\rho: J \rightarrow \mathbb{N}$  is given that strictly decreases along every immediate *branch* row:  $\rho(j') < \rho(j)$  whenever  $b(j, a, \text{in}(j')) \neq 0$ . (i) Show one rank serves every adversary: for each scheduler  $\varsigma$ , the mixed kernel  $f_\varsigma$  satisfies the fixed-scheduler problem's Zeno-freedom assertion with the same  $\rho$ , so the closed chain and its jump law  $\mu_\varsigma$  are defined for all  $\varsigma$  simultaneously. (ii) Show the object outranks its certificate: if  $\rho'$  is any other rank certifying some scheduler's mixed kernel, the closure at  $\rho'$  yields the same law — so the demonic semantics at horizon  $k$ ,

$$D = \{ \mu_\varsigma : \varsigma \text{ a memoryless randomized scheduler} \},$$

depends on the automaton and the horizon alone. (iii) On the compiled side, call  $\mu$  *admitted* when *some* well-formed scheduler-row set closes the compiled automaton rows, at the rank's working fuel, onto chain constraints that  $\mu$  satisfies; show  $\mu$  is admitted if and only if  $\mu \in D$ , and that every member of  $D$  has mass one, so the downward-closed credal set generated by  $D$  has exactly  $D$  as its radially maximal slice and the verdict pair reads as bounds uniform over *every* scheduler. (iv) Show the certificate forces somewhere to absorb into: if  $J$  and  $A$  are inhabited, a branch-certified automaton must have a Markovian state.

*Sketch.* (i) is support bookkeeping: a mixture's support lies in the union of the branch supports, so a rank uniform in the branches is uniform in the adversary — the quantifier commutes through the certificate. (ii) both certificates are spent at a common fuel past both ranks, where the absorbing iteration has stabilized twice over; since the two closures then agree on everything the jump law reads, the laws coincide. (iii) is the demonic decision problem of section 12.13 run through two closures: stored scheduler rows assemble a scheduler, the rate-closure and jump triangles collapse the pipeline to a compiled chain, and the chain constraints pin the law — four storeys of engine reuse, no new induction; the mass, slice, and verdict claims are the equal-mass reconciliation verbatim, the working fuel being exactly where the branch certificate is spent on the compiled side. (iv) the minimal-rank instantaneous state's branch row is a distribution, so it is supported somewhere, and every instantaneous destination would have to descend below the minimum.

In the universe's terms: the demonic reading of the compiled automaton rows and the union-of-singletons inclusion of  $D$  are isomorphic by an invertible globular cell at the per-element stratum, and the credal presentation reconciles with  $D$  through the radially maximal slice, exactly as at the decision former — the presentation radial, not convex, for the same reason. The statement and both directions are mechanized, sorry-free, in the toolkit's Lean tree (lean/su-functor/, module SuFunctor.Adequacy.MaDemonic): the invertible cell is `ma_demonic_adequacy`, with admission predicate `maDemonicC` and feeding lemma `maDemonicC_iff`; the one-rank-serves-every-adversary certificate is `FinMA.BranchZenoFree` with `zenoFree_of_branch`; the canonicity receipts are `FinMA.pathLaw_rank_irrel` and `FinMA.demonicLaws_rank_irrel`; the slice reconciliation is `ma_demonic_maxSlice`; the uniform-over-scheduler verdict theorems are `satCredal_lower_maDemonicCredal` and `satCredal_upper_maDemonicCredal`; and the degeneracies are mechanized rather than smoothed — the scheduler space has two nonemptiness routes (with no instantaneous states, the closed-chain boundary, `Policy.nonempty_of_isEmpty`; with them, schedulers vanish exactly when actions do, `Policy.isEmpty_iff`), the whole-set reading of a lower bound refuses every positive threshold (`not_satCredal_upper_maDemonicCredal_geC`), the empty scheduler space validates both verdict kinds vacuously, and the Markovian floor of task (iv) is `FinMA.nonempty_markovian`. The general object for automata where only *some* schedulers admit a rank — the union over Zeno-certified scheduler resolutions, each carrying its own rank (`FinMA.CertScheduler`, `FinMA.demonicLawsCert`) — is built at the classical level and collapses to  $D$  under a branch-level certificate (`FinMA.demonicLawsCert_iff`, spending rank-irrelevance); a compiled square for that partial-Zeno stratum would carry rank witnesses inside the admission predicate, and is deliberately separate. Also separate, unchanged from the fixed-scheduler face: the time-law stratum; history-dependent schedulers; parallel composition; the angelic (existential-over-scheduler) reading, a different operator from the universally quantified verdict pair.

**Status.** Certified at the fixed-scheduler grade, with the scheduler *declared as part of the claim*: a two-station polling automaton with an unreliable server — the classic Markov-automata polling benchmark [102], shrunk to capacity one — closed under a stated preference scheduler to a ten-state chain, carries  $P(\text{job lost by } T \mid \text{empty}) \leq T/5$  through the unmodified chain machinery. The mutant battery has teeth in the direction that matters: tampering with the declared scheduler makes the same certificate refuse with direct witnesses — the declaration does real work in the claim. Representation and production: the toolkit's producer surface constructs the composed profile in both languages and derives the entire closure bridge exactly from the stored rows — the sort partition, the Zeno rank certificate (with the branch-level rank that serves every adversary, where one exists — the demonic face's object — and absent, reported as such, otherwise), the absorption kernel at the working fuel, the closed chain as ordinary continuous-time rows with the self-loop mass thinned by the grain itself, and the closed generator by handing the closure to the chain kit: the square's engine reuse, run producer-side, with the race-then-jump-then-absorb spelling re-derived as an exact differential in the test suite. The relay mechanism is a committed cross-language byte-identity vector, and the API block below quotes its receipted construction. Scheduler quantification is mechanized at the semantics layer (the demonic face above); as for the decision frame, the uniform-over-schedulers story is not yet exercised by a published reproduction target — in workshop fixtures with checker acceptance, the bound above survives quantification verbatim, uniform over an adapted scheduler class, through the same machinery as section 12.13's open leg (one design discharges both), and this entry will say more only when that leg ships — and the certificate campaigns remain fixed-scheduler. Open: that reproduction leg; parallel composition of automata, the frame's own design point; logic closure.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema's own (`crates/mtk-catalogue/src/system/`), and the former's mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{\begin{array}{c} R \text{ off-diagonal rate entries, as for CTMC} \quad O \text{ action-keyed outcome rows, as for MDP} \\ \hline \varsigma : \langle \text{state, action, prob} \rangle \text{ scheduler rows} \quad \iota? \text{ initial-distribution rows} \end{array}}{\text{MA}(R; O; \varsigma) : \text{System}} \quad \text{S-MA}$$

Both parents' row disciplines apply verbatim: no diagonal rate entry, every outcome row action-keyed (immediate transitions use the decision idiom). The scheduler is declared data in this version; quantification over schedulers is mechanized at the semantics layer (the demonic face's verdict pair), with the published reproduction target the named open leg.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.markovAutomaton`

signature — `ts/src/system.ts`

```
/** S-MA: Markov automaton – rates + action-keyed immediate rows + a
 * scheduler declaration (declared data in v1). */
export const markovAutomaton = profileFactory("markovAutomaton");

/** The derived Markov-automaton fixed-scheduler kit: the sort
 * partition, the Zeno rank certificate (mixed-kernel face, plus the
 * scheduler-free branch-level rank when one exists – the demonic
 * face's one-certificate-for-all object), the absorption kernel at
 * the working fuel, the closed continuous-time chain as ordinary
 * CTMC rows, and the closed chain's generator kit – the mechanized
 * square's engine reuse, run producer-side. Derived, never trusted –
 * the checker re-runs the closure's three storeys. */
export interface MaGenKit {
  readonly markovian: readonly number[];
  readonly instantaneous: readonly number[];
  readonly rank: readonly { readonly state: number; readonly rank: number }[];
  readonly branchRank:
    | readonly { readonly state: number; readonly rank: number }[]
    | null;
  readonly absorb: readonly {
    readonly from: number;
    readonly to: number;
    readonly mass: string;
  }[];
  readonly closedRates: readonly {
    readonly src: number;
    readonly dst: number;
    readonly rate: string;
  }[];
  readonly closedInit: readonly {
    readonly state: number;
    readonly prob: string;
  }[];
  readonly closed: CtmcGenKit;
```



```

}

/** Derive the Markov-automaton fixed-scheduler kit from a mechanism
 * declaration and a per-state certificate observable on the
 * Markovian states. Maximal progress (a state with both timed and
 * immediate rows), scheduler discipline (coverage, mass one, support
 * within the enabled pairs), and the immediate-cycle Zeno door all
 * come back as witness-shaped refusals naming their geometry. */
export function maGenKit(
  init: MechanismInit,
  cert: readonly StateVal[],
): Result<MaGenKit, Refusal> {
  const raw = native.sysMaGenKit(mechanism(init).nativeHandle, cert) as
    | { kit: MaGenKit }
    | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}

```

construction — ts/tests/magen-native.test.ts

```

/** The `relay-2s1i` presentation (the committed golden fixture's):
 * Markovian 0 and 1, instantaneous relay 2; state 0 races at rate 2
 * into the relay, which returns a third of the mass under the
 * declared scheduler. */
function relayInit() {
  return {
    rates: [
      { src: 0, dst: 2, rate: "2" },
      { src: 1, dst: 0, rate: "1" },
    ],
    outcomes: [
      { src: 2, action: 0, dst: 0, prob: "1" },
      { src: 2, action: 1, dst: 1, prob: "1" },
    ],
    scheduler: {
      rows: [
        { state: 2, action: 0, prob: "1/3" },
        { state: 2, action: 1, prob: "2/3" },
      ],
    },
    initDist: [{ state: 0, prob: "1" }],
  } as const;
}

```

The quoted construction is the committed relay-2s1i cross-language byte-identity vector's presentation, whose thinned closure the kit tests pin in closed form on both surfaces; the kit's Rust face is `mtk_catalogue::system::ma_gen_kit`.

## Rust `mtk_catalogue::system::presets::MarkovAutomaton::manifest`

signature — crates/mtk-catalogue/src/system/presets.rs

```

preset!(
  /// System Catalogue: Markov automaton profile – the CTMC rate
  /// idiom + the MDP kernel idiom aimed at immediate transitions +
  /// a scheduler declaration (declared data in v1).
  MarkovAutomaton, "Markov automaton",
  req = [Rates, Outcomes, Scheduler],
  allowed = [InitDist],
  extra = |m| {
    no_diagonal_rates(m)?;
    outcomes_keyed("Markov automaton", m, true)
  }
);

```

construction — crates/mtk-catalogue/tests/builders.rs

```

let ma = MechanismBuilder::new()
  .rate(0, 1, r(1, 1))
  .outcome(OutcomeRow {
    src: 1,

```

```

        action: Some(0),
        dst: 0,
        prob: r(1, 1),
        reward: None,
    })
    .scheduler(SchedulerDecl {
        rows: vec![SchedulerRow {
            state: 1,
            action: 0,
            prob: r(1, 1),
        }],
        note: None,
    })
    .finish();
MarkovAutomaton::manifest(ma).unwrap();

```

## 12.16 Generalized Stochastic Hybrid Systems

**The frame.** Modes with continuous stochastic dynamics, guarded and forced transitions with resets, spontaneous jumps — the frame of Bujorianu and Lygeros [32], and the frame the System notion was designed around, because a modelling language that cannot host it cannot host the systems this toolkit exists for.

**Profile.** The full manifest complement: drift, diffusion, jump fields, plus guarded transitions (guard slabs, resets, forced marks) across modes. Boundary behaviour is first-class and the uniformity break is recorded rather than smoothed: forced transitions are *domain conditions*, not generator terms. Time: hybrid — jumps are events in real time.

**Semantics.** Extended-generator semantics on segments, with the Dynkin argument staged segmentwise. Switched trajectories make per-point differentiability semantically *false*, so decrease obligations for hybrid rows discharge at an upper Dini derivative — corner- and Zeno-immune, with mechanized rungs for switched flows, an integrated almost-everywhere stratum, and Filippov sliding reduced to per-mode vertex rows, at no new row shape in the checker. Hybrid records are native to the universe’s record structure — resets are zero-duration stage pairs (Definition 9.6) — which is why segmentwise semantics needs no second time axis.

**Status.** Certification: the frame’s namesake example is shipped — a three-mode kiln with three forced guard transitions and resets (one nontrivial and load-bearing: the identity-reset probe refuses), spontaneous spikes, and diffusion in both live modes. Nine sequents, every one an ordinary polynomial inequality over a region through the unmodified stack, yield  $P(\text{overheat}) \leq 671/2048$  from the hot start; the battery includes a diffusion-ablation witness showing the second-order term alone carries the hot mode, and the non-Zeno bridge is discharged by inspection of an acyclic transition structure. Representation: the first instance rows to exercise transitions, resets, and forced marks at all. The composite SDCPN formalism of Everdij and Blom — the 2023 candidate — now carries its own entry (section 12.18): places are Bags of coloured-token Systems, and the per-place colour dynamics are this profile’s. Open: a certified example with guards and resets at  $n \geq 2$  (the kiln is one-dimensional and the encounter systems are single-mode); stochastic reset kernels (deterministic polynomial resets are exercised; distributions enter as finitely many reset legs); curved or interior guard geometry beyond boundary slabs; the ingestion gap; closure.

### Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ ,  $\text{pop}$ , and profile conformance are defined in section 12.1.

$$\frac{\begin{array}{l} \vec{x} \text{ declared state variables} \quad M : \overline{\langle \text{id}, \text{name?}, \text{domain} \rangle} \text{ modes with domain constraints} \\ T : \overline{\langle \text{from\_mode}, \text{to\_mode}, \text{guard}, \text{reset}, \text{forced} \rangle} \text{ guarded transitions} \\ D?, N?, J? \text{ mode-indexed drift, noise-presentation, and jump families} \end{array}}{\text{GSHS}(\vec{x}; M; T) : \text{System}} \quad \text{S-GSHS}$$

Forced transitions are domain conditions rather than generator terms — the uniformity break, recorded as structure.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.gshs`

signature — `ts/src/system.ts`

```
/** S-GSHS: modes with domains + guarded transitions (+ the full
 * continuous complement). */
export const gshs = profileFactory("gshs");
```

construction — `ts/tests/system-native.test.ts`

```
return sys.unwrap(
  sys.gshs({
    name: "kiln-2mode",
    state: [{ id: 0, name: "temp" }],
    modes: [
      {
        id: 0,
        name: "heat",
        domain: [{ poly: mono(1, 0, 1, "1"), op: "le", rhs: "100" }],
      },
      {
        id: 1,
        name: "cool",
        domain: [{ poly: mono(1, 0, 1, "1"), op: "ge", rhs: "20" }],
      },
    ],
    drift: [
      { mode: 0, var: 0, poly: constant(1, "3") },
      { mode: 1, var: 0, poly: mono(1, 0, 1, "-1/10") },
    ],
    diffusion: [
      { mode: 0, var: 0, poly: constant(1, "1/4") },
      { mode: 1, var: 0, poly: constant(1, "1/4") },
    ],
    jumps: [
      {
        mode: 0,
        rate: constant(1, "1/100"),
        reset: [{ var: 0, poly: mono(1, 0, 1, "9/10") }],
        resetImage: { axes: [{"18", "90"}] },
      },
    ],
    transitions: [
      {
        fromMode: 0,
        toMode: 1,
```

```

        guard: [{ poly: mono(1, 0, 1, "1"), op: "ge", rhs: "100" }],
        forced: true,
    },
    1,
  }},
);

```

**Rust** mtk\_catalogue::system::presets::Gshs::manifest

signature — crates/mtk-catalogue/src/system/presets.rs

```

preset!(
    /// System Catalogue: GSHS profile – the full complement (drift,
    /// diffusion/noise, jumps) + modes with domains and guarded
    /// transitions (guard slabs, resets, forced marks).
    Gshs, "GSHS",
    req = [Modes, Transitions],
    allowed = [Drift, Diffusion, Noise, Jumps],
    extra = ok
);

```

construction — crates/mtk-catalogue/tests/common/mod.rs

```

let below = DomainConstraint {
    poly: mono(1, 0, 1, r(1, 1)),
    op: "le".to_owned(),
    rhs: r(100, 1),
};
let above = DomainConstraint {
    poly: mono(1, 0, 1, r(1, 1)),
    op: "ge".to_owned(),
    rhs: r(20, 1),
};
let m = MechanismBuilder::new()
    .state_var(0, Some("temp"))
    .mode(Mode {
        id: 0,
        name: Some("heat".to_owned()),
        domain: vec![below],
    })
    .mode(Mode {
        id: 1,
        name: Some("cool".to_owned()),
        domain: vec![above],
    })
    .drift(Some(0), 0, constant(1, r(3, 1)))
    .drift(Some(1), 0, mono(1, 0, 1, r(-1, 10)))
    .diffusion(Some(0), 0, constant(1, r(1, 4)))
    .diffusion(Some(1), 0, constant(1, r(1, 4)))
    .jump(JumpRow {
        mode: Some(0),
        rate: constant(1, r(1, 100)),
        reset: vec![ResetRow {
            var: 0,
            poly: mono(1, 0, 1, r(9, 10)),
        }],
        mixture: vec![],
        reset_image: Some(BoxRegion {
            axes: vec![[r(18, 1), r(90, 1)]],
        }),
    })
    .transition(GuardedTransition {
        from_mode: 0,
        to_mode: 1,
        guard: vec![DomainConstraint {
            poly: mono(1, 0, 1, r(1, 1)),
            op: "ge".to_owned(),
            rhs: r(100, 1),
        }],
        reset: vec![],
        forced: true,
    })
);

```

```

    .finish();
    presets::Gshs::manifest(m)
    .expect("fixture conforms to the GSHS profile")
    .named("kiln-2mode")

```

## 12.17 The Bag Former

**The frame.** Not a frame at all — a primitive *System-former*:  $\text{Bag}(S)$  is the System of a varying finite multiset of tokens, each token an instance of the argument System  $S$ . It exists as its own entry because structured-population types (the coloured nets of section 12.18 first among them) want “a place holds finitely many tokens with per-token dynamics” as one reusable construction rather than a per-frame encoding trick.

**Profile.** A countably-indexed switched-mode carrier: mode  $n$  is the  $n$ -fold juxtaposition power of  $S$ ’s state, quotiented by exchange — the canonical-multiset representation at the presentation stratum, the Polish symmetric-power quotient at the semantics stratum. Token creation and destruction are the former’s own variant-crossing switch edges (self-loops between adjacent powers); noise is per-token independent. Monad structure (singleton unit, flatten multiplication) sits at the admissible tier rather than in the rows.

**Semantics.** The former denotes

$$\text{Bag}(S) = \coprod_{n \in \mathbb{N}} S^{\boxplus n} / S_n,$$

and the display is a colimit formula in the semantic universe, not a set-theoretic sum; every operation in it is licensed by the standing structure of Definition 9.15. The power  $S^{\boxplus n}$  is the  $n$ -fold juxtaposition composite of the composition calculus, written with the belief laxator’s symbol because its decoration is the  $n$ -fold  $\boxplus$  of  $S$ ’s belief content (its level-of-inconsistency functional) with itself (Definition 9.44); its state space is the  $n$ -fold product  $X^n$  of the inner System’s state space  $X$  (Definition 9.41), and the empty power is the unit System on the one-point space. The symmetric group acts through the juxtaposition tensor’s symmetries — permutations of the  $n$  interface copies, injective tight isomorphisms of the wiring base, acting on states by coordinate permutation — and  $X^n / S_n$  is the coequalizer of those  $n!$  homeomorphisms in the category of Polish spaces and continuous maps, Polish under the matching metric; the disjoint union over  $n$  is the countable coproduct in the same category, its mass sectors clopen (both licences: the bag widget of Definition 9.15). Equivalently the carrier is the space of integer-valued finite counting measures on  $X$  under the weak topology — portmanteau on small balls around the atoms identifies each mass sector with the corresponding symmetric power, and the matching metric on the mass- $n$  sector is the optimal-matching ( $\infty$ -transport) distance between empirical measures — the presentation through which the creation and destruction rate species acquire their point-process reading. So Bag preserves Polishness of the carrier, and bag-valued places stay inside the record-space semantics the rest of the catalogue assumes, with token creation and destruction as zero-duration variant crossings, exactly as hybrid records carry resets (Definition 9.6).

The imprecise denotation passes through the quotient rather than around it. Bounded continuous observables on  $X^n / S_n$  are exactly the  $S_n$ -invariant ones on  $X^n$  (the quotient topology’s universal property), so the weak topology over the quotient is generated by symmetrized test functions, and pushforward along the quotient map is an affine homeomorphism from the exchangeable (permutation-invariant) subprobabilities on  $X^n$  onto  $\Delta_{\leq 1}(X^n / S_n)$  — injective by symmetrizing test functions, surjective by symmetrizing the lift along the canonical-multiset section. A credal body on the bag carrier is therefore the same thing as an exchangeable credal body on the ordered representation — exchangeability is well-definedness, not a preference — and the profile’s per-token-uniform dynamics and beliefs descend because they are equivariant by construction; invariant certificate observables are the quotient’s observables, which is what the canonicalization discipline of the Status leg realizes at the presentation stratum. Two boundaries, kept sharp: bags are finite — infinite multisets break both presentations, and token counts stay finite along every record by construction — and the colimits here are colimits of *carriers*: the formula lifts to Systems through the former, whose monad structure (singleton unit, flatten multiplication) is stated on the fibre categories of the composition construction at the admissible tier, not as a colimit in them.

**Status.** Representation: shipped — the declaration wraps an inner System, an optional capacity, typed creation/destruction edges, and a stored marking under the canonical-multiset discipline (tokens sorted in the exact coordinatewise-lexicographic order, duplicates adjacent, the count derived as the list’s length; the conformance gates refuse a non-canonical stored value, so preset-accepted declarations carry canonical bytes). Semantics and certification: the rate species of the creation/destruction edges is landed as a typed vocabulary — a creation edge reads aggregate observables (the token count), a destruction edge races per unordered token selection with slot-exchange symmetry checked exactly, and a synced edge declares a fused capability whose one firing law lives on the event-port fusion row — and the first certificates exercise a bag carrier end to end: a marked bag of ordinary-differential-equation tokens compiles by the landed view pair into its single-place zero-arc coloured net (the two constructions are one object, checked to the byte on the shared locus), projects onto the Petri incidence presentation, and carries a certified population invariant through two independent checker lanes, with mutants at the designed loci — a non-canonical stored value refuses, an asymmetric selection rate refuses, and a desynchronized transfer flips both checkers. The billiard table as  $\text{Bag}(\mathbb{R}^4)$  remains the worked example on the road for the colour stratum.

**Adequacy.** The square’s measure-theoretic cell — the ordered representation and the collective marking process present one process — is mechanized, sorry-free, in the toolkit’s Lean tree (`lean/su-functor/`, module `SuFunctor.Adequacy.Bag`). The sorted-representative section of the exchange quotient is measurable under the pushforward  $\sigma$ -algebra from nothing but a linear order on colours with a measurable order graph: each sorted coordinate is an order statistic, a finite minimum of maxima of the unsorted coordinates (`sortTuple_apply_eq_inf_sup`, `measurable_symSec`), so no measurable-selection theorem enters, and ties on the diagonal never demand a choice — monotone presentations of one multiset are unique (`monotone_eq_of_toSym_eq`). On that section, the quotient kernel of an exchange-invariant chain closes the commuting square at every finite horizon by kernel-category algebra alone (`quotKernel_comp_eq`, `iter_intertwine`); the construction is independent of the section chosen (`comap_section_congr`); and the transport assembles on the full shape-indexed carrier, token creation and destruction included, with per-permutation exchange-invariance and invariance under canonicalization interderivable (`bag_transport`, `invariance_iff_perm`). The order-graph hypothesis is discharged for the coordinatewise-lexicographic order on real colour vectors that the stored representation sorts by (`measurableSet_le_lex`) — the tie corner lives exactly there, and it is an instance, not an assumption. What remains at its named home: dynamics preservation for the monad’s unit and multiplication (a process-stratum statement), and the per-shape generator agreement that belongs to the coloured-net stratum of the next section.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ ,  $\text{pop}$ , and profile conformance are defined in section 12.1.

$$\frac{S : \text{System} \quad \text{capacity? a declared population cap} \quad M? \text{ a stored marking}}{L : \langle \text{id, kind, } \Delta, \text{law} \rangle \text{ self-loops, kind} \in \{\text{create, destroy}\} \quad \text{Bag}(S) : \text{System}} \quad \text{S-BAG}$$

$m.\text{bag} = \langle \text{inner, capacity?, self\_loops, marking?} \rangle$ ; each self-loop declares its id (the fusion-reference key), kind (create/destroy), per-firing delta, and firing law — delay with a typed rate basis (per-selection or aggregate), guard, or synced; a stored marking is the canonical sorted token list, its count the list’s length.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted



certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

## TypeScript `system.bag`

signature — `ts/src/system.ts`

```
/** S-BAG: the Bag former (primitive System-former over an inner
 * System). */
export const bag = profileFactory("bag");
```

The TS conformance corpus holds no construction through this factory at this preview — the cross-language byte-identity fixtures exercise six profiles, and this one's byte authority is the Rust corpus opposite; the factory takes the same init-object vocabulary as every other profile.

## Rust `mtk_catalogue::system::presets::Bag::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```
preset!(
    /// System Catalogue: Bag former – primitive System-former (the
    /// canonical-multiset carrier over the argument System), not a
    /// frame. The extra gates enforce the former's semantics-bearing
    /// discipline: typed edges (unique ids;  $\delta \geq 1$ ; create reads
    /// aggregates, destroy races per selection, guards select the
    /// hitting token; W1 slot-exchange symmetry on per-selection
    /// rates) and canonical stored markings (arity, canonical
    /// rationals, the unique sorted multiset representative, the
    /// capacity cap).
    Bag, "Bag",
    req = [Bag],
    allowed = [],
    extra = |m| {
        let b = m.bag.as_ref().expect("required by signature");
        let d = inner_state_arity(&b.inner, "bag.inner"?);
        bag_edge_gates(&b.self_loops, d)?;
        if let Some(tokens) = &b.marking {
            bag_marking_gates(tokens, d, b.capacity.as_ref(), "bag.marking"?);
        }
        Ok(())
    }
);
```

construction — `crates/mtk-catalogue/tests/builders.rs`

```
let bag = MechanismBuilder::new()
    .bag(BagDecl {
        inner: Box::new(common::sde()),
        capacity: None,
        self_loops: vec![
            // Arrivals: aggregate basis (var 0 = the token count).
            BagEdge {
                id: 0,
                kind: BagEdgeKind::Create,
                delta: WitInt::from(1),
                law: BagEdgeLaw::Delay {
                    rate: BagRate::Aggregate {
                        rate: constant(1, r(1, 2)),
                    },
                },
            },
            // Departures: per-selection basis (the selected
            // token's colour coordinates).
            BagEdge {
                id: 1,
                kind: BagEdgeKind::Destroy,
                delta: WitInt::from(1),
                law: BagEdgeLaw::Delay {
                    rate: BagRate::PerSelection {
                        rate: mono(1, 0, 2, r(1, 10)),
                    },
                },
            },
        ],
    })
```

```

    },
    },
    },
    1,
    // Stored multiset value, canonical order (count = length).
    marking: Some(vec![vec![r(-1, 2)], vec![r(0, 1)], vec![r(0, 1)]]),
  })
  .finish();
Bag::manifest(bag).unwrap();

```

## 12.18 Stochastically and Dynamically Coloured Petri Nets

**The frame.** The SDCPN formalism of Everdij and Blom — the candidate unifier of the 2023 grading — as a first-class entry: Petri-net structure whose tokens carry continuous stochastic colour dynamics. The reading this catalogue commits to is compositional rather than monolithic: *places are Bags, arcs are patterns*. The stance has good company in the categorical net literature — nets opened along place boundaries and composed as cospans [12], coloured nets read as monoidal double functors [172], and individuals-based readings of the token game [140] — though the construction here factors through the catalogue’s own composition patterns rather than through a dedicated category of nets.

**Profile.** Places are Bags of a coloured-token System (section 12.17), each with an optional declared capacity; transitions are per-tuple edge families in three species — delay (a rate law), guard (forced), immediate (cascade glue); ordinary arcs are variant-crossing consume/produce resets with multiplicities, and enabling arcs are per-tuple read-outs. Every arc kind is an instance of an existing composition pattern — with one deliberate exclusion: **inhibitor arcs are not a primitive**. Absence-testing breaks enabledness monotonicity, the free commutative-monoid factoring, and the negation-free character of the chase all at once — a unique upper-bound gate in a lower-bound formalism. What survives is the bounded-locus converse: on a place with declared capacity  $k$ , an inhibitor arc is admissible as a *view* — the complement-place compile, whose elaborator is a named successor: a complement place with mirrored arcs and a weight- $k$  enabling arc under a certified linear place-invariant  $m(P) + m(P') = k$  — and the compile provably cannot extend to an unbounded place, which is why no unbounded inhibitor constructor exists at all.

The vocabulary, enumerated. Places being Bag mechanisms, each arc is a composition shape wiring Bag populations: a transition compiles to a *rendezvous* — one joint jump in the fusion discipline of section 13.10, timed by its active party, the transition species — whose other parties are the incident places’ bag edges, themselves the mode-switching machinery of section 13.3. Of the eleven elements of the founding formalism — the arc kinds, the transition species, and the net-level data — ten map onto the pattern inventory, nothing is added to it, and the eleventh is the inhibitor exclusion above. The tenses matter: the per-Bag edge vocabulary, the read-out wires, and the fusion rows themselves are shipped declaration rows — the fusion primitive now carries a producer surface, and the coloured-net decomposition emits its rows (section 13.10) — while the pattern vocabulary downstream of the primitive still speaks in design tense there, so the compiled artefacts this table calls for are graded in the Status leg below.

| element                   | semantics                                                                                                         | compilation target                                                                                                    |
|---------------------------|-------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| ordinary arc, input face  | one token read into the firing data and removed: variant-crossing reset $n \rightarrow n - 1$ at the source place | a rendezvous party: the source Bag's destroy edge, its reset carried by the joint jump                                |
| ordinary arc, output face | a token produced at the target place, $n \rightarrow n + 1$ , its colour drawn from the firing measure            | a rendezvous party: the target Bag's create edge, spawn direction                                                     |
| enabling arc              | the selected token's colour read by rate, guard, or firing measure; no removal                                    | a passive party: a per-tuple parameter-setting wire (section 13.1) into the timing owner                              |
| enabledness count         | pre-enabling threshold: one token per ordinary and enabling input arc                                             | scope rows on the rendezvous's edge family — a monotone, negation-free count guard                                    |
| inhibitor arc             | pre-enabling gate on absence                                                                                      | not compiled — excluded as a primitive; the bounded-locus view above enters through the complement place's rendezvous |
| delay transition          | rate-driven firing: a per-tuple survivor law raced across enabled tuples                                          | the active party: a rate law timing the joint jump                                                                    |
| guard transition          | forced firing on the guard boundary                                                                               | the active party, forced species                                                                                      |
| immediate transition      | fires as soon as enabled; vanishing markings leave the state space                                                | the active party, immediate species: cascade glue under the no-Zeno side condition                                    |
| conflict resolution       | priority and randomization among competing enabled tuples                                                         | certificate-stratum dominance; no composition pattern consumed                                                        |
| firing measure            | the joint law of the output vector and the produced colours                                                       | reset-leg moment data on the produce edges                                                                            |
| initial marking           | the initial distribution over variants and canonical colours                                                      | root init rows                                                                                                        |

**Semantics.** Marking spaces are Polish by construction: finitely many places, each a Bag carrier — a finite product of the previous section's colimits, inside the standing structure of Definition 9.15. The correspondence with section 12.16 (SDCPN executions as generalized stochastic hybrid processes, the founding papers' own bridge) enters as a translation between *presentations* — never as the definition of the net's semantics, whose compiled reading remains structural in the net while the flattened object is combinatorially large — and its jump-skeleton stratum is mechanized at the uncoloured grain *and* at the coloured deterministic-firing slice: definedness-premise rows, the commuting-triangle cell, the constructed flattening with its agreement derived, and the composed community-to-community corollary included at both grains (the Adequacy leg below), with the remaining coloured obligations — the hybrid-grain carrier identification and the firing-measure kernels — enumerated by name. The compilation direction is Bag-primary: arcs compile into composition patterns — rendezvous of creation/destruction capabilities — over the places' Bag mechanisms, the zero-arc case being the identity.

**Adequacy (statics, the delay-race face, the immediate/cascade face at all three scheduler readings, and the flattening bridge).** The compositional reading — *places are Bags, arcs are patterns* — is not a slogan: the adequacy square for this frame factors through the composition patterns' own mechanized semantics, and it closes at two strata as of this preview, with the rest staged by name. At the statics stratum the square is whole. Stated self-contained:

**Problem 12.14** (the token-game statics square). A marking assigns each place a finite multiset of token colours. A transition of the inhibitor-free fragment reserves, per place, one token for each ordinary and each enabling input arc (distinct tokens), removes the ordinary ones, keeps the enabling ones, adds fresh tokens per output arc, and feeds every reserved colour into one firing datum along its arc's readout [71]. Compile instead: each place a bag composite (one count wire, a token power per mode, states read modulo exchange), consumption a token-split rename, each colour readout an identification-pattern row. Show the two readings agree: markings of each shape are exactly the compiled states modulo per-place exchange renames, and the token game holds between two markings if and only if compiled states realizing them are

connected by split renames and pattern apex states. Show also that the fragment's token game is upward closed — adding tokens never disables a step — and exhibit the gate that would break it.

*Sketch.* The multiset readout of a compiled state is surjective onto markings of its shape and a complete exchange invariant — two tuples enumerate one multiset exactly when a permutation relates them — so the community's configuration space is the compiled state space modulo exchange, shape by shape. For the firing correspondence, forward: enumerate the reserved and untouched tokens through a chosen index split, and the required states are defined directly through the split; backward: the split decomposes the bag as selected-block plus kept-block (the many-token form of the cons identity), and the pattern apex condition is precisely colour-readout agreement. Upward closure is one line per field — the surplus sits in the untouched remainder — and the gate that breaks it is the zero-test: an inhibitor arc is the unique arc kind whose pre-condition is not upward closed, which is why the fragment's transition type has no constructor for it. The exclusion is a formation restriction with the monotonicity theorem as its warrant, not a lint.

At the finite time-abstract stratum the delay species closes as a square. The community execution races every pre-enabled *ordered* assignment of tokens to input arcs on its own exponential clock [71], so from a bounded uncoloured marking the total intensity to a target is the ordered census — per place, a falling factorial of supply over demand — times the rate, summed over transitions; the compiled rows carry the *unordered* convention, one row per token set, with the arc-multiplicity dictionary  $\prod_p (\text{demand}_p)!$  owned by the compiler. The compile-commutes triangle is therefore not definitional, deliberately: it is the dictionary theorem — falling factorial equals factorial times binomial, per place — and through it the square is the continuous-time jump-chain square instantiated at the race chain, engine reuse with zero new inductions; the initial marking enters through the same square's init rows, and firings whose outcome equals their source are thinned by the off-diagonal row grain, the standard jump-chain convention stated as a receipt. Mechanized end to end (`lean/su-functor/`, modules `SuFunctor.TokenGame` and `SuFunctor.Adequacy.StochasticPetri`): the vertical identification, the firing correspondence, the monotonicity warrant, the census-to-embeddings tie, the dictionary triangle, the reused square, and the thinning receipt.

The immediate species closes as a square at the same stratum, by the Markov-automaton closure consumed as built. A marking that pre-enables an immediate transition is *vanishing* — the community excludes it from the state space of the observed process [71] — and here the exclusion is carried by the type: markings split into tangible and vanishing subtypes, the net compiles to a two-sorted Markov automaton whose Markovian sort is the tangible markings, and a delay row at a vanishing marking is thereby unrepresentable — the maximal-progress priority as the domain type of the rate field. Immediate firings are the automaton's action-keyed branches (point masses at this uncoloured stratum); a declared memoryless scheduler resolves conflicts, so the square is scheduler-generic and dominates the community's randomized choice rule, with the uniform-over-enabled scheduler as the shipped instance; cascade termination is a rank certificate — a rank on vanishing markings that every enabled immediate firing strictly decreases — and the fixed-scheduler closure delivers the reduced tangible chain, with vanishing initial mass pushed through the cascade rather than excluded by premise (when the community's initial-tangibility premise does hold, the initial distribution passes through untouched, as a receipt). Timed rows compile in the same unordered convention as the delay face, so the compile triangle is again the dictionary theorem, one storey up, and the square is the automaton square at the compiled net — five-storey engine reuse, zero new inductions. Mechanized in `SuFunctor.Adequacy.StochasticPetriCascade`: the classified state space, the automaton reading, the enabling-respect discipline and rank certificate, the tangible chain with the initial-tangibility receipt, the compile triangle, the reused square, and controls pinning the uniform conflict split, the thinning of returned cascade mass, and a genuine two-step cascade.

The scheduler-quantified sibling closes beside it — the demonic face of the cascade square, and the point where the automaton donor's certificate does *not* transport. The branch-level Zeno certificate that serves every adversary at the automaton (`FinMA.BranchZenoFree`, section 12.15) is refutable at nets: a *disabled* immediate transition at a vanishing marking compiles to the zero-time no-op it is — a self-loop branch row that no rank can strictly decrease (`not_branchZenoFree_of_disabled`, discharged against the species' shipped control net for every rank at once) — so the demon is instead quantified over the subclass the cascade rank actually certifies: *enabling-respecting* schedulers (`RespScheduler`), the class the community's conflict randomizations inhabit, with the uniform scheduler as its shipped member, a discard scheduler witnessing that the arena is genuinely plural (`arena_plural`), and the compiled admission strengthened to

carry the side condition in the statement (`RespectingRows`) rather than smuggling it through a definitional coincidence. Over that arena the set-valued square closes, as at the decision and automaton frames (sections 12.13 and 12.15): the demonic reading of the compiled, scheduler-free net rows is isomorphic, by an invertible globular cell, to the inclusion of the net's demonic tangible law-set (`ispn_demonic_adequacy`, feeding lemma `ispnDemonicC_iff`), the law-set and its credal presentation are independent of which rank certificate is spent (`demonicTangibleLaws_rank_irrel` and `demonicTangibleCredal_rank_irrel`), the equal-mass slice reconciliation and the uniform-over-adversary verdict pair transport (`ispn_demonic_maxSlice`, `satCredal_lower_ispnDemonicCredal` and its upper twin), and the wiring into the semantic universe's judgment lands as the module's own quartet (`su_ispn_demonic_adequacy` with its mem, bottom, and slice forms). Six storeys of engine reuse, zero new inductions. Mechanized, sorry-free, beside the cascade module (`lean/su-functor/`, module `SuFunctor.Adequacy.StochasticPetriDemonic`); history-dependent adversaries stay outside the arena by definition, exactly as at the automaton.

The set-firing face closes beside them, at the marking stratum. The community fires *non-conflicting simultaneous enablings* together — one transition enabled by several token sets, or several transitions enabled at once, no firing disabling the rest [71] — and the face is driven by one value identity rather than any combinatorial induction: a zero-time chain of single firings whose every step is enabled is *exact* (no truncation fires), so per place its result satisfies “result plus ordinary-input total equals source plus ordinary-output total,” and the right-hand data depend on the firing multiset alone (`chainFire_val`). Order-invariance is then value uniqueness — two enabled enumerations of one multiset land the same marking, with no diamond or permutation induction anywhere (`chainFire_eq_of_enabled`; the pairwise diamond is the two-element corollary) — and on the non-conflicting locus the community's randomized choice rule has nothing left to choose between. The simultaneous composite — remove every ordinary-input token at once, then produce, the community's own firing order — equals the chain in every enabled order under *joint supply* (`chainFire_eq_compositeFire`), and the supply hypothesis is load-bearing in the statement, not merely the proof: two swap-token firings pass one token through a chain and land it unchanged, while the composite would read double (the module's refutation control; the community classifies that pair as conflicting — both sets reserve one token — so the hypothesis carves the rule's own boundary). The reservation precondition renders the community's side condition at the counting stratum — members enabled, consumption jointly supplied, each member's enabling-arc reads fitting the pool no sibling consumes, outcome within capacity (`JointReservation`) — and the scope condition the compositional reduction genuinely demands is explicit: intermediate markings need not interpolate between endpoints, and in the capacity fragment the cheap sufficient condition is headroom — source plus output total within capacity (`setNonConflicting_of_headroom`; a reservation-respecting three-element composite whose producer-first order blocks mid-chain is the control that keeps the hypothesis honest). Under reservation and headroom every enumeration is an enabled chain, all agree, and each equals the composite (`r1r3_composite_adequacy`); multiplicity is native — one transition with  $k$  token sets is a replicated multiset, so the one-transition and several-transition rules are one vocabulary. An enabled immediate's branch row in the compiled automaton is a point mass at its firing, so an enabled chain is an absorption path of the cascade closure as already built: the observed tangible process cannot distinguish the community's one-instant composite from any scheduler's zero-time chain. Mechanized, sorry-free, beside the cascade module (`lean/su-functor/`, module `SuFunctor.Adequacy.StochasticPetriSetFiring`).

The flattening toward the hybrid-process reading (section 12.16) enters as a bridge between presentations, not as a definition, and its jump-skeleton stratum is mechanized — at the uncoloured grain in full, and one colour grain up at the deterministic-firing slice (the coloured paragraph below). The translation's definedness premises are certificate-shaped rows: initial tangibility (no immediate transition enabled at the initial support — decidable, with the closed chain's initial row passing through untouched as a receipt); the invariant token bound in the witness-vector idiom — a positive place weighting that no transition of either species increases, with step soundness for both species, closure under reachability, and an intrinsic per-place bound independent of the declared capacity, which is the clause a flattening at the unbounded grain consumes for the finiteness of its mode roster (a token-creating control net admits *no* witness vector, so the row has teeth); and cascade termination, consumed from the landed rank certificate. The triangle itself is packaged from atom-transport agreement (`flatten_triangle_of_agree`): a flat presentation on an abstract mode carrier that agrees with the scheduler-closed net off-diagonally and initially denotes a constraint isomorphic to the net's compiled reading — an invertible globular cell between two house compiles, with no community semantics and no process construction inside the cell. The agreement is no longer a hypothesis: the flattening is *constructed* for every net and every declared

scheduler at this grain — the community’s own construction, one flattened transition per delay transition, its jump intensity the ordered race census times the rate, its reset the cascade-closed firing kernel — and the agreement law is derived from the construction (rate rows by Fubini regrouping and collapse of the fired-marking indicator; initial rows from the tangibility receipt, consumed not re-proved), so every `_of_agree` consumer downstream is discharged outright and the composed corollary states its conclusion with no agreement hypothesis. The drain-net instance survives as a check on the general definition: the constructed flattening reproduces its hand-written flattening data exactly, off-diagonally and initially (the flattened drain edge carrying the arc-multiplicity dictionary through a two-step cascade-closed reset). The *composed corollary* pastes this cell with the two adequacy squares it bridges — the net square’s cascade face on one side and, on the other, the flattened object’s own square, which at this stratum is the continuous-time chain square, a flattened presentation being a finite chain — so the token-game community law and the flattened execution’s community law are one invertible globular cell apart at every jump horizon, and the same pasting read at the judgment stratum says a law models the flattened compiled presentation exactly when it models the net’s. The demonic and angelic faces transfer verdicts through the same pasting as directed cells at the community grain (sound directions only; no envelope equality is stated), and on the drain-net instance an actual flattened-side computation — the three-mode chain takes its drain edge with probability  $3/4$  — is consumed as net-side verdicts by transport, never recomputation. Graded exactly: this is a finite-jump-skeleton analogue, free of the finite-expected-jumps hypothesis, of Everdij and Blom’s process-level bisimulation — different conclusion, different observational grain, neither result implying the other. The holding-time *data* (the exit-rate family) agrees on both sides; and the time-indexed comparison through transient distributions is now mechanized on the killed trajectory-measure side — the killed construction itself, a prefix-marginal factorization (timed prefix marginals as skeleton prefix law, holding attachment, and clock pairing), and the lift from skeleton-law agreement to equal killed transient reads at every horizon, with mutants refused at four preregistered loci — its grain bridge for consuming the published non-Zeno hypothesis verbatim remaining the named residue. The triangle, the derived agreement, the composed corollary, and the killed time-indexed lift are mechanized in `SuFunctor.Adequacy.StochasticPetriFlatten`, `SuFunctor.Adequacy.StochasticPetriFlattenAgreement`, `SuFunctor.Adequacy.StochasticPetriComposed`, and `QtslAux.KilledTimeIndexed`.

The same story closes one colour grain up, at the deterministic-firing slice. The premise rows carry over with their colour reading — initial tangibility unchanged; the invariant token bound with *colour-indexed* weights, its step soundness reading the consumed and produced colour counts of each transition at each colour vector; cascade termination from the coloured rank certificate — and the one premise the uncoloured development could only name, per-place colour-box discharge, is carried by the formation itself at this fragment: the colour boxes are finite types by construction. The triangle cell is packaged against a flattened presentation on an abstract mode carrier exactly as before (the socket a hybrid-grain carrier identification plugs into), and the flattening is again *constructed*: one flattened transition per delay transition *per colour vector*, its jump intensity the coloured ordered census times the colour-dependent rate, its reset the cascade-closed firing kernel through the coloured cascade, with the agreement derived and the composed corollary stated with no agreement hypothesis — the token-game community law of a coloured net and its flattened execution’s community law, one invertible globular cell apart at every jump horizon, at the deterministic slice. On the control net the certificate is the colour dependence itself: from a one-of-each marking the flattened jump chain drains with probability  $2/5$  and flips back with  $3/5$ , the two branches differing *only* in the colour their selections read — a rate map reading colours, flowing through the flattening into the community law, consumed as a token-game verdict by transport. Mechanized, sorry-free (`SuFunctor.Adequacy.StochasticPetriColouredComposed`); the hybrid-grain residue stays staged, named below.

The flattened object also admits its hybrid-systems reading, and the corollary composes with it. A finite chain is a general stochastic hybrid system whose modes are its states and whose continuous dynamics are trivial, and the embedding is faithful at the data level: the profile’s total alarm intensity at any flattened point is the chain’s exit rate — through the agreement law, the *coloured net’s own* tangible-chain exit rate — and the GSHS extended generator row of a chart-constant certificate degenerates to the chain’s rate row, for every choice of derivative data. So the martingale-problem square of the hybrid layer is consumed unchanged one more step out: a certificate whose Foster-Lyapunov drift rows hold at the scheduler-closed token-game chain yields the square’s safety and deadline verdicts on the flattened execution — token-game rows in, hybrid-skeleton verdicts out, with the square’s segment-model presentations remaining exactly its own hypothesis shape. On the control net the certificate that is 1 exactly on the two capacity-



truncation modes has nonpositive rows — the census-weighted colour-dependent rates and the truncation edge enter the arithmetic — and the verdict pins the chance of reaching the mode that only truncation produces at zero, with every hypothesis discharged on the halt-control realization. Mechanized, sorry-free (`SuFunctor.Adequacy.StochasticPetriColouredGshsComposed`); the hybrid-grain reading with genuine continuous colour coordinates stays with the staged residue below.

The angelic face closes the scheduler-reading triple. Liveness-style claims quantify *existentially* — some resolution of the conflicts achieves the claim — and the honest existential reads the radially maximal slice of the credal set, for the same reason the universal lower kind does: the credal presentation is downward closed, so the whole set contains the zero model, and an un-sliced existential would be witnessed by zero for every constraint the zero model satisfies (mechanized as a degeneracy lemma, with the zero-threshold upper-bound event atoms as the concrete family). On an equal-mass family the slice is the family, so the angelic verdict is existential quantification over the enabling-respecting arena (`SatAngelic`, `satAngelic_ispnDemonicCredal`; the operator is generic over unions of rays, so the decision and automaton frames inherit it unchanged whenever wanted). One membership fact yields both directed transfers, with the sense-opposition living at the verdict stratum rather than in any reversed cell: universally certified claims restrict to every arena member, and claims established on a member — the transported flattening’s law included — are angelic witnesses (`ispn_demonic_verdict_to_member`, `flatten_angelic_of_agree`, with the packaging as a directed cell into the slice object; sound directions only, no envelope equality). And the two verdicts genuinely differ on the shipped conflict net: the demonic face’s arena-plurality control is upgraded here to the *law* level — the uniform and always-discard members’ closed path laws differ at jump horizon one, the uniform member jump-stepping to the moved token with probability one while the always-discard member’s closed chain absorbs at the source — so the claim “the drain trajectory carries full weight” is angelically satisfied and demonically refused. Mechanized, sorry-free, closing the module family (`SuFunctor.Adequacy.StochasticPetriAngelic`).

The coloured generator faces are landed at their deterministic (Dirac) slice: per-place finite colour boxes, colour-dependent delay rates, guard predicates read as per-vector cylinders on both species, and deterministic firing measures, with the delay-race and cascade squares closed by the same engine reuse through a per-colour-class multiplicity dictionary, conflict resolution over (transition, colour-vector) pairs, and a degeneracy theorem recovering the uncoloured fragment on the nose at the one-colour instance — mechanized, sorry-free (`SuFunctor.Adequacy.StochasticPetriColoured`, `SuFunctor.Adequacy.StochasticPetriColouredCascade`). The scheduler-quantification faces ride the same coloured arena: the demonic reading quantifies enabling-respecting adversaries over (transition, colour-vector) pairs — at colour the demon selects *tokens*, not merely transitions, and a single transition’s blocked selections, whether by supply or by guard cylinder, already refute the branch-level Zeno certificate that would have transported the automaton square on the nose — with the set-valued square, the angelic dual, and both directed verdict transfers flowing from one membership lemma; a recolouring control net admitting no cascade rank at all pins the rank hypothesis as substantive rather than decorative. Mechanized, sorry-free (`SuFunctor.Adequacy.StochasticPetriColouredDemonic`). The firing-measure *kernels* beyond the deterministic slice are mechanized at the delay face, on both axes the formation admits. The uncoloured lift takes the firing outcome to a sub-stochastic kernel on the marking space: the census still kills under-supply, the kernel mass kills blocked and unreachable outcomes — so the race intensity needs no enabling gate at all — and mass deficit is the thinned-firing reading, proven semantically inert by a completion receipt (the deficit moves onto an explicit diagonal self-loop, honestly Markov, and the jump-chain path law does not move). The coloured lift is the founding formalism’s firing measure itself: a mass-one kernel over produced-colour assignments, with supply, capacity, and guard gating each sampled outcome — an infeasible draw is a no-op, thinned by the off-diagonal grain. Both delay-race squares close by the same engine reuse through their multiplicity dictionaries, and the deterministic slice embeds *on the nose*: race chains and compiled rows are equal as structures under the indicator-kernel embedding, and the landed Dirac squares re-derive from the kernel squares through the degeneracy ties alone — mechanized, sorry-free (`SuFunctor.Adequacy.StochasticPetriKernel`, `SuFunctor.Adequacy.StochasticPetriColouredKernel`). The immediate species follows at the same grain: an enabled immediate firing samples its successor marking from a kernel of conditional mass one — full mass exactly where the transition is enabled, because a sub-stochastic immediate draw’s zero-time no-op would enter the Zeno certificate’s support on an enabled action, where enabling-respect cannot gate it; the rank certificate prices sure termination, not merely almost-sure. The cascade square closes by the same automaton engine, whose branches were distribution-valued all along; the termination certificate widens to the kernel’s whole support, and the

sure-versus-almost-sure boundary is itself machine-checked — a kernel with mass at its own source admits no cascade rank at all, by theorem, and a control net whose cascade terminates almost surely (a geometric stay/leave coin) is formation-expressible and certifiably rank-free. The deterministic slice embeds one storey more strongly than at the delay face: the indicator embedding's Markov automaton is *equal as a structure* to the landed one, so the landed cascade square re-derives by rewriting alone — mechanized, sorry-free (`SuFunctor.Adequacy.StochasticPetriCascadeKernel`). The produced-colour axis closes the kernel-grain lattice: coloured immediate firing measures — the founding formalism's own — sample the colours an enabled immediate firing produces, from a kernel that reads the reserved colour vector *and the source marking*, with mass one on capacity-feasible outcomes exactly where the pair is enabled. The marking-dependence is forced, not chosen: a marking-independent mixing kernel always meets a jam marking (the mixed colour class full at a produced place, the declared one within capacity) where every semantics breaks — a sub-stochastic row is a Zeno-visible zero-time deficit, a stay-completion is certificate-refused mass at the source, and truncation fabricates successor markings — so the kernel renormalizes where capacity bites, and each sampled outcome rides the same per-outcome supply-and-capacity gate as at the delay face, infeasible cells thinned by the gate as dead data. The termination certificate widens to the kernel's feasible support, the stay/leave coin is refused at colour too, and the deterministic slice again embeds at automaton-structure level, the landed coloured cascade square re-deriving by rewriting alone — mechanized, sorry-free (`SuFunctor.Adequacy.StochasticPetriColouredCascadeKernel`). The flattening seam then closes the composition: both firing-measure axes ride one formation — the finite discrete skeleton of the stochastic-hybrid net class at kernel grain, delay draws and immediate draws each honestly random — and a delay firing's produced-colour draw, pushed through the token game and closed through the immediate cascade, is a *cascade-closed firing kernel* on tangible markings: the hybrid story's reset object, per transition and colour binding. Its row mass is exactly the capacity-gated delay mass — one wherever capacity does not bite, which is everywhere on the founding formalism's own capacity-free domain — the closed chain's rate rows are precisely the race-weighted sums of these reset rows, so the pure-jump hybrid reading consumes the kernel-grain net through its tangible chain unchanged, and at point-mass kernels on both axes the automaton reproduces the deterministic fragment as a structure, the reset row collapsing to the declared outcome's cascade closure — the deterministic-reset restriction of the flattening recovered as a theorem-level face, not a fence — mechanized, sorry-free (`SuFunctor.Adequacy.StochasticPetriColouredFlattenKernel`). What stays staged, by name: the kernel-valued transition schema on the hybrid side — the profile face that would store these reset kernels per transition, and with it the flattening cell at kernel grain; the guard species (boundary forcing); the coloured flattening's hybrid-grain residue — the colour-vector carrier identification with its padding obligations (the flattening itself, its premise rows, the composed corollary at the deterministic slice, and the pure-jump hybrid-systems reading with its martingale-problem verdicts having landed above); and the exponential time-law stratum, exactly as for the plain continuous-time frame.

**Variadic certificates: one row per Petri transition.** A certificate for a net whose token count is unbounded across runs faces two apparent infinities: unboundedly many token configurations, and — since a transition fires once per eligible token selection — a countably infinite family of transition *instances*, each seemingly demanding its own certificate row. Neither infinity survives contact with the syntax, and the two mechanisms are worth stating separately, because they fail separately when abused.

The state side is the plate discipline of section 10.4: certificates are polynomials in pooled coordinates — token counts  $\vec{k}$  and per-place feature pools  $\vec{s}$  — and obligations are polynomial sequents over  $(\vec{k}, \vec{s})$  with coupling, count, and rounding hypothesis rows, their number and dimension set by the net, never by a marking bound. The transition side rides the same grammar fact (no expression names “the  $i$ -th token,” so transition data depends on selected tokens only through their colours): the per-instance obligations of one Petri transition collapse to a single *footprint sequent* in distinguished-token variables — the selected tuple's colours, the pooled coordinates, the counts — with dimension set by the transition's arity and the feature dictionary. One symbolic row covers every instance at every marking. The countable family was never infinite.

What the collapsed row can *claim* depends on the aggregator, and the dependence is sharp. Write a firing's footprint as (consumed multiset  $X$ , produced multiset  $Y$ ), and consider a certificate value that is a *commutative-monoid fold* of per-token features — the fold structure is itself load-bearing: an arity-normalized aggregate such as a mean is not one (a birth changes its denominator under every untouched token, so a footprint row can hold with equality while the ambient value rises), and for such aggregates

only the direct polytope form above applies, never a footprint row. For *safety* rows (the aggregate does not increase), monotonicity of the fold's translations suffices, at every token count: the footprint row “the produced contribution is bounded by the consumed contribution” transfers to the whole marking by one monotone translation, births and deaths included (for signed features the death row is no longer automatic — a death whose consumed contribution is negative is *refused* unless a guard excludes it: a refusal, not an unsoundness). For *progress* rows the conditions are strictly stronger, and each is necessary — drop one and a finite countermodel follows:

- the per-firing decrease must carry a fixed rational gap  $\delta > 0$ , stated in the row — strict decrease alone certifies nothing (halve one token's value forever: every step strictly drops, the run is infinite, the value never reaches its floor);
- the gap must be stated in a scale where the fold is a *translated sum* — for sum-pools the footprint gap row transfers verbatim, uniformly in the marking; a mean fails twice over (no locality, as above, and a per-argument modulus of  $\delta/n$  at  $n$  tokens, so no uniform gap survives an unbounded token count);
- a floor row (the aggregate bounded below on reachable markings) must be discharged — with signed features and token births, every footprint row can hold while progress-rowed firings continue forever down an unbounded descent;
- and *idempotent folds are progress-incapable in every scale*: for max-aggregated certificates the footprint gap fails to move the aggregate whenever an untouched token dominates (two tokens of equal value; fire on one), a regenerating two-transition net keeps the max constant through infinitely many gap-verified firings while the corresponding sum-row is falsified — the sum correctly refuses a certificate the max would wave through — and no strictly monotone change of scale repairs this, since max is a fixed point of conjugation. Max-aggregated certificates certify safety but support no footprint-only progress row against arbitrary ambient tokens (structural anti-masking conditions — a singleton place, a guard consuming every maximal token — are a different and dearer certificate species, not this row); a footprint gap row on an idempotent fold is a formation error, not a modelling choice.

The positive family is correspondingly wide: for any strictly increasing  $\varphi$  on the feature range, the  $\varphi$ -conjugated sum  $\varphi^{-1}(\sum_t \varphi(v_t))$  — the unnormalized  $\ell_p$  folds at  $\varphi(x) = x^p$  over nonnegative features, interpolating from sum ( $p = 1$ ) toward max ( $p \rightarrow \infty$ ); log-sum-exp for  $\varphi(x) = e^{\beta x}$  (with the empty marking sent to  $-\infty$ ) — is strictly monotone in every argument at every finite parameter, which is exactly the property the max endpoint lacks. (The count-normalized *power mean* is deliberately not in the family: normalization re-imports the mean's locality failure.) Progress rows for the family are stated on the  $\varphi$ -space sum (an ordinary pool over the transformed features, with a pinned  $\varphi$ -space gap), where the transfer is uniform in the marking; verdicts transport monotonely for thresholds in  $\varphi$ 's domain (compare  $\sum v_t^p$  against  $t^p$  at  $t \geq 0$  — the checker never computes a root), while the drop measured in aggregate units necessarily shrinks as the *transformed mass*  $\sum_t \varphi(v_t)$  grows, for every superlinear  $\varphi$  — with an explicit rational floor on bounded ranges (for integer  $p$  and aggregate bound  $B$ : at least  $\delta_\varphi/(p B^{p-1})$ ) — which is the reason the  $\varphi$ -space statement is the normative one. For integer  $p$  the transformed features are polynomials and every obligation is an ordinary exact-rational sequent today; non-integer  $p$  and the log-sum-exp family ride the certified enclosure register for transcendental symbols (section 10.5).

Grading, in the book's standard senses: the grammar-equivariance layer is mechanized (the plate modules of section 10.4, including descent to marking-indexed denotations); the pooled compile and the footprint-sequent shape are the shipped row vocabulary of the certificate stratum; the fold-transfer statements above are elementary, each dropped side condition refuted by a finite exact-rational instance (the countermodels of the progress list above), with their formal lemma pack a named, priced successor rather than a landed artefact; the row vocabulary's first exercising campaign at industrial shape is the worked example of chapter 17 (section 17.5). Where the event *count* is itself the claim — expected crossings, Zeno exclusion, deadline tails — the certificate is the crossing budget of the hybrid-process correspondence (section 12.16), not a ranking row; the two prices are different and deliberately not interchangeable.

**Status.** Representation and production: shipped — the declaration vocabulary above is in the conformance corpus, whose worked fixture is a two-place queue/server net with a delay and an immediate transition, ordinary and enabling arcs, and one inhibitor view on a capacity-two place; the view-on-unbounded-place refusal is exercised from both language surfaces. At the uncoloured bounded fragment the squares

certify, the toolkit’s producer surface derives the entire marking-space bridge exactly from the stored net: the bounded marking space under a declared mixed-radix convention (shipped as a decode table, so the index correspondence is checkable), the compiled race rows in the unordered convention with the arc-multiplicity dictionary the compiler owns — the ordered census recomputed as an exact differential in the test suites, the dictionary theorem as arithmetic — the tangible/vanishing sort split with timed rows at tangible sources only (maximal progress as construction), point-mass immediate branches, the declared or uniform-over-enabled scheduler, and the closed observed chain by handing the compiled net to the automaton kit (section 12.15: cascade rank, absorption at the working fuel, vanishing initial mass pushed through the cascade, the initial-tangibility pass-through re-derived as a receipt) or, for the pure-delay net, to the chain kit: the squares’ five-storey engine reuse, run producer-side. The relay-cascade net — whose every delay firing returns through a two-step cascade, thinning the closed chain to nothing — is a committed cross-language byte-identity vector, and the API block below quotes the kit’s surface. The inhibitor view stays outside the compiled fragment by formation (its complement-place elaborator remains the named successor), and per-place capacities, state-dependent rates, and coloured tokens refuse at the kit, each naming its stratum. Semantics and certification: the count stratum has its first certified artefact — a population invariant carried across a transfer rendezvous through two independent checker lanes over a Bag carrier (section 12.17); the token game’s statics square, the delay-race face, the immediate/cascade face — at the declared scheduler and under scheduler quantification alike — and the set-firing face (simultaneous composite firings of non-conflicting immediates) are mechanized (the Adequacy paragraph above), with the coloured generator faces landed at their deterministic slice and the coloured scheduler-quantification faces alongside them, while the guard species (boundary forcing) and the firing-measure kernels beyond the deterministic slice remain the named next legs; net-of-nets composition runs through the token-colouring closure of the composition inventory (section 13.13), with only its shared-place face reserved to the state-sharing axis.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\begin{array}{c}
 \overline{P : \langle \text{id}, \text{name?}, \text{token\_system} : \text{System}, \text{capacity?} \rangle \text{ places}} \\
 T : \langle \text{id}, \text{name?}, \text{species} \rangle \text{ transitions} \quad A : \langle \text{transition}, \text{place}, \text{kind} \rangle \text{ arcs, closed over } P \text{ and } T \\
 M : \langle \text{place}, \text{tokens} \rangle \text{ a stored marking, canonical per place} \\
 \text{every inhibitorView arc's place declares a capacity} \\
 \hline
 \text{SDCPN}(P; T; A; M) : \text{System} \quad \text{S-SDCPN}
 \end{array}$$

`species ::= delay(rate) | immediate | guard`  
`kind ::= consume(k) | produce(k) | enabling(k) | inhibitorView(k)`

The capacity premise is the bounded locus as licence: an inhibitor view on an unbounded place is not a refused edge case but a missing constructor.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx*` or `f.*` are that corpus’s

shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

## TypeScript system.sdcpn

signature — ts/src/system.ts

```
/** S-SDCPN: places are Bags, arcs are patterns (the inhibitor VIEW is
 * admissible only on a place with declared capacity). */
export const sdcpn = profileFactory("sdcpn");

/** One marking-index decode row of the SDCPN kit: `index` in the
 * mixed-radix convention (places in declaration order, first place
 * most significant), `counts` the per-place token counts. */
export interface SdcpnMarkingVal {
  readonly index: number;
  readonly counts: readonly number[];
}

/** The derived SDCPN marking-space kit at the uncoloured bounded
 * fragment: the marking decode table, the compiled race rows in the
 * unordered convention (the compiler-owned arc-multiplicity
 * dictionary `[p (demand_p)]` times the per-token-set selection
 * count times the rate — the community's ordered census divided by
 * exactly the mechanized dictionary theorem), the tangible/vanishing
 * sort split (timed rows at tangible sources only — maximal progress
 * as construction), action-keyed point-mass immediate branches, the
 * resolved scheduler (declared, or the uniform-over-enabled instance
 * the mechanized square ships), point-mass initial rows, and the
 * closed observed process by ENGINE HANDOFF — the automaton kit
 * where immediate transitions exist (cascade rank, absorption, the
 * reduced tangible chain, vanishing initial mass pushed through the
 * cascade), the chain kit otherwise. Derived, never trusted — the
 * checker re-derives the marking space, the census, and the
 * closure's own storeys. */
export interface SdcpnGenKit {
  readonly capacity: number;
  readonly placeOrder: readonly number[];
  readonly markings: readonly SdcpnMarkingVal[];
  readonly raceRows: readonly {
    readonly src: number;
    readonly dst: number;
    readonly rate: string;
  }[];
  readonly immediateRows: readonly {
    readonly src: number;
    readonly action: number | null;
    readonly dst: number;
    readonly prob: string;
  }[];
  readonly schedRows: readonly {
    readonly state: number;
    readonly action: number;
    readonly prob: string;
  }[];
  readonly initRows: readonly {
    readonly state: number;
    readonly prob: string;
  }[];
  readonly closure:
    | { readonly kind: "chain"; readonly chain: CtmcGenKit }
    | { readonly kind: "automaton"; readonly ma: MaGenKit };
}

/** Derive the SDCPN marking-space kit from a mechanism declaration
 * (the `sdcpn` family; an optional scheduler rides the mechanism's
 * own scheduler family — absent is the uniform instance) and a
 * per-state certificate observable on the tangible markings, by
 * marking index. Admission fences — uniform capacity, unit colour,
 * constant nonnegative rates, no guard species, no inhibitor views,
 * no duplicate arcs, all-or-none marking, the engine cap on the
```

```

* marking space – and the closure's own discipline (scheduler
* support within the enabled immediates, the immediate-cycle Zeno
* door) all come back as witness-shaped refusals naming their
* geometry. */
export function sdcpnGenKit(
  init: MechanismInit,
  cert: readonly StateVal[],
): Result<SdcpnGenKit, Refusal> {
  const raw = native.sysSdcpnGenKit(mechanism(init).nativeHandle, cert) as
    | { kit: SdcpnGenKit }
    | { refusal: unknown };
  return "refusal" in raw
    ? refused(narrowRefusal<Refusal>(raw.refusal))
    : ok(raw.kit);
}

```

construction — ts/tests/system-native.test.ts

```

const token = sys.unwrap(
  sys.ode({
    name: "token",
    state: [{ id: 0, name: "energy" }],
    drift: [{ var: 0, poly: mono(1, 0, 1, "-1/100") }],
  }),
);
return sys.unwrap(
  sys.sdcpn({
    name: "queue-server",
    sdcpn: {
      places: [
        { id: 0, name: "queue", tokenSystem: token },
        { id: 1, name: "server", tokenSystem: token, capacity: 2 },
      ],
      transitions: [
        { id: 0, name: "arrive", species: { kind: "delay", rate: constant(1, "3/2") } },
        { id: 1, name: "admit", species: { kind: "immediate" } },
      ],
      arcs: [
        { transition: 0, place: 0, kind: { kind: "produce", mult: 1 } },
        { transition: 1, place: 0, kind: { kind: "consume", mult: 1 } },
        { transition: 1, place: 1, kind: { kind: "enabling", mult: 1 } },
        {
          transition: 1,
          place: 1,
          kind: {
            kind: "inhibitorView",
            threshold: 2n,
            invariant: "pinv:server-complement",
          },
        },
      ],
    },
  }),
);

```

The marking-space kit compiles the uncoloured bounded fragment the mechanized square certifies: race rows in the unordered convention (the compiler-owned arc-multiplicity dictionary — the ordered census divided by exactly the dictionary theorem), the tangible/vanishing sort split, point-mass immediate branches, and the closed observed process by engine reuse — the automaton kit where immediate transitions exist, the chain kit otherwise. The kit's Rust face is `mtk_catalogue::system::sdcpn_gen_kit`; its closed-form controls (the census through the dictionary, the two-step cascade with vanishing initial mass pushed through absorption) ride the test suites on both language surfaces.

## Rust `mtk_catalogue::system::presets::Sdcpn::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```

preset!(
  /// System Catalogue: SDCPN – places are Bags, arcs are patterns.
  /// The inhibitor VIEW is admissible only on a place with declared
  /// capacity (the bounded-locus converse); no unbounded-inhibitor
  /// constructor exists at all.

```



```

Sdcpn, "SDCPN",
req = [Sdcpn],
allowed = [],
extra = |m| {
  let s = m.sdcpn.as_ref().expect("required by signature");
  // Roster ids key arcs, markings, and fusion sketches: gate
  // uniqueness first (a duplicate transition id would let one
  // event id denote several species – review round 1, MF5).
  for (second, p) in s.places.iter().enumerate() {
    if let Some(first) = s.places[..second].iter().position(|q| q.id == p.id) {
      return Err(Refusal::DuplicateRosterId {
        kind: "place",
        first,
        second,
        id: p.id,
      });
    }
  }
  for (second, t) in s.transitions.iter().enumerate() {
    if let Some(first) = s.transitions[..second].iter().position(|u| u.id == t.id) {
      return Err(Refusal::DuplicateRosterId {
        kind: "transition",
        first,
        second,
        id: t.id,
      });
    }
  }
  for (arc_index, arc) in s.arcs.iter().enumerate() {
    let place = s.places.iter().find(|p| p.id == arc.place);
    let trans_ok = s.transitions.iter().any(|t| t.id == arc.transition);
    let Some(place) = place else {
      return Err(Refusal::DanglingEndpoint {
        profile: "SDCPN",
        index: arc_index,
        place: Some(arc.place),
        transition: (!trans_ok).then_some(arc.transition),
      });
    };
    if !trans_ok {
      return Err(Refusal::DanglingEndpoint {
        profile: "SDCPN",
        index: arc_index,
        place: None,
        transition: Some(arc.transition),
      });
    }
  }
  // The token game's lower-bound reasoning is defined over
  // positive counts (review round 1, MF5/MF10).
  let mult = match &arc.kind {
    ArcKind::Consume { mult }
  | ArcKind::Produce { mult }
  | ArcKind::Enabling { mult } => Some(mult),
    ArcKind::InhibitorView { threshold, .. } => Some(threshold),
  };
  if let Some(mult) = mult {
    if mult.0 < num_bigint::BigInt::from(1u8) {
      return Err(Refusal::NonPositiveArcMult {
        arc_index,
        transition: arc.transition,
        place: arc.place,
        mult: mult.0.to_string(),
      });
    }
  }
  if matches!(arc.kind, ArcKind::InhibitorView { .. }) && place.capacity.is_none() {
    return Err(Refusal::InhibitorOnUnboundedPlace {
      arc_index,
      place: arc.place,
    });
  }
}
}

```

```

    for (entry, mk) in s.marking.iter().enumerate() {
        if let Some(first) = s.marking[..entry].iter().position(|m2| m2.place == mk.place) {
            return Err(Refusal::DuplicateMarkingEntry {
                first,
                second: entry,
                place: mk.place,
            });
        }
        let Some(place) = s.places.iter().find(|p| p.id == mk.place) else {
            return Err(Refusal::DanglingMarkingPlace {
                entry,
                place: mk.place,
            });
        };
        let site = format!("sdcpn.marking[{entry}] (place {})", mk.place);
        let d = inner_state_arity(&place.token_system, &site)?;
        bag_marking_gates(&mk.tokens, d, place.capacity.as_ref(), &site)?;
    }
    // NAMED GAP, deferred with mechanism: per-tuple delay-rate
    // arity and slot symmetry at the NET stratum need the
    // selection roster across places – the staged
    // intensional-family tranche; the Bag preset checks both on
    // the single-place locus today.
    Ok(())
}
);

```

construction — crates/mtk-catalogue/tests/common/mod.rs

```

let token = {
    let m = MechanismBuilder::new()
        .state_var(0, Some("energy"))
        .drift(None, 0, mono(1, 0, 1, r(-1, 100)))
        .finish();
    presets::0de::manifest(m)
        .expect("token System conforms to the ODE profile")
        .named("token")
};
let decl = SdcpnDecl {
    places: vec![
        SdcpnPlace {
            id: 0,
            name: Some("queue".to_owned()),
            token_system: Box::new(token.clone()),
            capacity: None,
        },
        SdcpnPlace {
            id: 1,
            name: Some("server".to_owned()),
            token_system: Box::new(token),
            capacity: Some(maitria_witness_types::WitInt::from(2)),
        },
    ],
    transitions: vec![
        SdcpnTransition {
            id: 0,
            name: Some("arrive".to_owned()),
            species: TransitionSpecies::Delay {
                rate: constant(1, r(3, 2)),
            },
        },
        SdcpnTransition {
            id: 1,
            name: Some("admit".to_owned()),
            species: TransitionSpecies::Immediate,
        },
    ],
    arcs: vec![
        ArcRow {
            transition: 0,
            place: 0,
            kind: ArcKind::Produce {

```

```

        mult: maitria_witness_types::WitInt::from(1),
    },
},
ArcRow {
    transition: 1,
    place: 0,
    kind: ArcKind::Consume {
        mult: maitria_witness_types::WitInt::from(1),
    },
},
ArcRow {
    transition: 1,
    place: 1,
    kind: ArcKind::Enabling {
        mult: maitria_witness_types::WitInt::from(1),
    },
},
ArcRow {
    transition: 1,
    place: 1,
    kind: ArcKind::InhibitorView {
        threshold: maitria_witness_types::WitInt::from(2),
        invariant: Some("pinv:server-complement".to_owned()),
    },
},
],
// No declared marking: absent on the wire, so this fixture's
// committed bytes predate and survive the marking family.
marking: vec![],
};
let m = MechanismBuilder::new().sdcpn(decl).finish();
presets::Sdcpn::manifest(m)
    .expect("fixture conforms to the SDCPN profile")
    .named("queue-server")

```

## 12.19 Shallowly-Embedded Petri Nets

**The frame.** Places, transitions, tokens; the bread-and-butter questions are reachability and coverability of markings.

**Profile.** A shipped theory of its own: places, transitions, and total flow matrices in the incidence presentation (arc multiplicities native, which discharges the “generalized” qualifier of early programme drafts for the arc-weight reading), with markings as first-class elements. Time: none intrinsically — the token game is untimed at this rung; the stochastic and timed wings belong to section 12.16.

**Semantics.** The distinctive entry, and the shallow embedding the section title names: the token game runs *as chase derivation, inside the substrate* — enabledness by negation-free spine walk, firing by successor invention, and marking identity by equality generation, with no oracle in the loop. This is the one frame at this preview whose representation and semantics both run natively in the modelling substrate, rather than beside it.

**Adequacy (the interleaving square, through the compiled route).** The shallow embedding above runs the token game natively; the same untimed frame also enters the catalogue’s *compiled* route, and the adequacy square closes there whole. The compile, in one breath: places become bags of unit-colour tokens — and a bag of unit-colour tokens *is* its count, so markings are count vectors — while each transition becomes one row under the demonic wing of the transition-clock vocabulary (chapter 11) at a strictly positive dwell floor, with a Dirac outcome law, the atomic reset  $m \mapsto m - \text{pre} + \text{post}$ , and enabledness as the arc-cover gate. The compiled semantics is therefore a scheduler-seat family of deterministic timed runs, and its classical corner is not a path law but a relation: the square states that the possibilistic visited-marking face of the family *equals* classical interleaving reachability, for every strictly positive dwell assignment (*face\_eq\_reach*) — the soundness leg spends only enabledness, and the completeness leg realizes each classical firing at exactly the dwell floor. The flagged check comes back quiet: the face is independent of

the dwell assignment (`face_dwell_uniform`), so the uniform unit clock that transcription conventions pin is nowhere load-bearing — the transcription sentence of chapter 11, that the unit-dwell embedding is faithful for ordering-and-counting properties and deliberately not for metric-time properties an untimed net never had, is at this face a theorem rather than a convention’s gloss.

The wiring into the semantic universe lands in the record shape of the demonic corners (sections 12.13 and 12.15, and the scheduler-quantified face of section 12.18): each resolution’s law at a horizon is the Dirac law on its marking trajectory, runs shorter than the horizon idling at their final marking; the compiled presentation is a one-atom projection, so finite presentation is untouched by the resolution family’s infinitude; and the ideal point of a law models the compiled demonic presentation exactly when the law is a member of the demonic law-set (`su_vanilla_demonic_adequacy`, with its `mem`, `bottom`, and `slice` forms). One structural fact is stated in the module rather than dressed up: this species has no rows-level compilation in the tree — every kernel being Dirac, the scheduler seat’s resolutions are the whole mechanism — so the feeding lemma is definitional (`vanillaDemonicC_iff`) and the square’s content lives one retract below the law stratum, at the visited-marking face (`recordVisited_eq_face`). The headline, read at the universe’s judgment: a marking is visited by the support of some law whose ideal point models the presentation, at some horizon, exactly when it is classically reachable (`su_vanilla_reach_iff`) — and the horizon quantifier is honest rather than lossy, since every visited marking is visited at a finite index and prefixes of admissible runs are admissible, so the finite-run family carries the whole reachability face. Dwell-uniformity transports to the same stratum (`su_vanilla_dwell_uniform`), and safety transfers as the customer shape for invariant-class examinations — a marking predicate holds on everything the record semantics can visit precisely when it holds on the classically reachable set (`su_vanilla_safety`) — with one-safeness and unboundedness exercised concretely on a two-place toggle and a one-place source (the dwell floor bounds nothing). Liveness is withheld by design, not staged: the demonic clock carries no fairness, priority, or weight vocabulary, and the idling resolution is always seated, so the honest current semantics asserts reachability and safety, never progress. Set-firing beyond interleaving has its own face at the coloured development (section 12.18); inhibitor arcs and mixed race-plus-demonic systems are untouched at this entry. Mechanized, sorry-free (`lean/su-functor/`, module `SuFunctor.Adequacy.PetriVanilla`).

**Status.** Certification: a three-place producer-consumer ring saturates to exactly its ten-marking state space with coverability verdicts and a place-invariant checked per reachable marking; an overflow mutant is caught by a derived coverability fact, and ablating the equality discipline reproduces divergence as a loud non-termination, not a wrong answer. The hypothesis is stated in the theory itself, not in a side note: token counts live on a finite declared ladder, so fixpoint verdicts are decisions only up to that headroom — exact for structurally bounded nets, while unbounded nets refuse loudly and coverability positives remain semi-decidable witnesses. Semantics: the compiled route’s interleaving square — reachability, dwell-uniformity, and the safety transfer — is mechanized (the Adequacy paragraph above), with no certificate rows minted by the square itself. Open: the ladder-horizon side conditions pinned to their oracle; net-of-nets composition, which runs through the token-colouring closure of the composition inventory (section 13.13) via the coloured wing; the stochastic wing.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , `pop`, and profile conformance are defined in section 12.1.

$$\frac{\text{places, transitions declared finite sets} \quad \text{pre, post total flow matrices over places} \times \text{transitions} \quad \text{marking a first-class marking over places}}{\text{Petri}(\text{places}; \text{transitions}; \text{pre}; \text{post}; \text{marking}) : \text{System}} \quad \text{S-PETRI}$$

The incidence presentation, multiplicities native; totality over the declared sets is closedness consumed as a sort — every flow and marking entry names a declared place and transition by construction.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named fx\_\* or f.\* are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript system.petri

signature — ts/src/system.ts

```
/** S-PETRI: places, transitions, total flow matrices, marking (token
 * game as chase derivation). */
export const petri = profileFactory("petri");
```

construction — ts/tests/system-native.test.ts

```
// Petri flow row citing an undeclared endpoint: which endpoint.
const dangling = sys.petri({
  petri: {
    places: [{ id: 0 }],
    transitions: [{ id: 0 }],
    pre: [{ place: 7, transition: 0, mult: 1 }],
  },
});
assert.ok(!dangling.ok);
```

The TS corpus exercises this factory through its refusal geometry (the corpus's discipline that failure geometry is routing advice); the quoted construction is that witness, and the happy-path byte authority is the Rust corpus opposite.

### Rust mtk\_catalogue::system::presets::Petri::manifest

signature — crates/mtk-catalogue/src/system/presets.rs

```
preset!(
  /// System Catalogue: Petri net – places, transitions, total flow
  /// matrices, marking (token game as chase derivation).
  Petri, "Petri net",
  req = [Petri],
  allowed = [],
  extra = |m| {
    let p = m.petri.as_ref().expect("required by signature");
    let place_ok = |id: u32| p.places.iter().any(|d| d.id == id);
    let trans_ok = |id: u32| p.transitions.iter().any(|d| d.id == id);
    for (index, e) in p.pre.iter().chain(p.post.iter()).enumerate() {
      if !place_ok(e.place) || !trans_ok(e.transition) {
        return Err(Refusal::DanglingEndpoint {
          profile: "Petri net",
          index,
          place: (!place_ok(e.place)).then_some(e.place),
          transition: (!trans_ok(e.transition)).then_some(e.transition),
        });
      }
    }
    for (index, e) in p.marking.iter().enumerate() {
      if !place_ok(e.place) {
        return Err(Refusal::DanglingEndpoint {
          profile: "Petri net",
          index,
          place: Some(e.place),
          transition: None,
        });
      }
    }
    Ok(())
  }
);
```

construction — crates/mtk-catalogue/tests/builders.rs

```

let petri = MechanismBuilder::new()
  .petri(PetriDecl {
    places: vec![PlaceDecl {
      id: 0,
      name: None,
      capacity: None,
    }],
    transitions: vec![TransitionDecl { id: 0, name: None }],
    pre: vec![FlowEntry {
      place: 0,
      transition: 0,
      mult: WitInt::from(1),
    }],
    post: vec![],
    marking: vec![MarkEntry {
      place: 0,
      tokens: WitInt::from(3),
    }],
  })
  .finish();
Petri::manifest(petri).unwrap();

```

## 12.20 Probabilistic Graphical Models

The thesis lists three sub-frames — Bayesian networks, structural causal models, and Markov random fields — and the inventory carries the first and third through *one* rendering: probabilistic dependency graphs (Richardson and Halpern [209]) as wires over trivial dynamics, one wire per conditional table. Dependency graphs strictly subsume Bayesian networks and factor graphs, and here the subsumption is a *checked* sentence rather than a cited one: the classic sprinkler network is reproduced byte-identically through the dependency-graph rendering by two decorrelated pipelines, and a two-source graph certified *inconsistent* witnesses strictly-beyond-network capability — every network-image graph is exactly consistent, so a certified inconsistency is a certified exit from the network image. Structural causal models carry their own theory, below. All three sub-frames are static, and for each the supervisory-switch pattern (section 13.3) is inapplicable — the omission earned by argument from the frame’s own semantics, as the closure leg requires, rather than declared. The pattern’s composition data — per-location executions, a firing moment at which a trajectory reaches a guard, an exit state read at the source and initial data planted at the target — has no instantiation from a semantics whose denotation is a single static measure over joint assignments: a guard over such variables has a truth-value *distribution*, not a trajectory of truth values, so nothing ever fires. The rows can always be typed — wrap the variables as a zero-drift carrier and write a guard box — but the resulting composite denotes switched executions, which is no graphical model: any instantiation exits the frame’s semantic category before it composes. Interventions are no counterexample (model surgery selected by the query, never an event fired when a state condition becomes true), and temporal extensions — dynamic Bayesian networks, samplers run as chains — are different frames, with Markov-chain machinery, where switch composition is applicable and required.

### Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (crates/mtk-catalogue/src/system/), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{W : \overrightarrow{\text{StochasticWire}}}{\text{PDG}(W) : \text{System}} \text{ S-PDG}$$



$$\text{StochasticWire} ::= \text{conditional}(\text{sources}, \text{target}, \overrightarrow{(\text{given}, \text{outcome}, \text{prob})}) \\ | \text{potential}(\text{vars}, \overrightarrow{(\text{assignment}, \text{weight})})$$

One wire per conditional table; potential wires are raw nonnegative tables with no normalization row (their local scale derived, never stored) — the atom that carries Markov random fields through the same species.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.pdg`

signature — `ts/src/system.ts`

```
/** S-PDG: BN / MRF / PDG – stochastic wires over trivial dynamics. */
export const pdg = profileFactory("pdg");
```

The TS conformance corpus holds no construction through this factory at this preview — the cross-language byte-identity fixtures exercise six profiles, and this one's byte authority is the Rust corpus opposite; the factory takes the same init-object vocabulary as every other profile.

### Rust `mtk_catalogue::system::presets::Pdg::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```
preset!(
    /// System Catalogue: BN / MRF (factor graph) / PDG –
    /// probabilistic-dependency wires over trivial dynamics (the
    /// stochastic-wire species); strictly subsumes Bayesian networks
    /// and factor graphs.
    Pdg, "PDG",
    req = [StochasticWires],
    allowed = [],
    extra = ok
);
```

construction — `crates/mtk-catalogue/tests/builders.rs`

```
let pdg = MechanismBuilder::new()
    .stochastic_wire(StochasticWire::Conditional {
        id: 0,
        sources: vec![0],
        target: 1,
        rows: vec![CondRow {
            given: vec![0],
            outcome: 1,
            prob: r(1, 1),
        }],
    })
    .stochastic_wire(StochasticWire::Potential {
        id: 1,
        vars: vec![0, 1],
        rows: vec![PotRow {
            assignment: vec![0, 0],
            weight: r(7, 2),
        }],
    })
    .build();
```

```

    .finish();
    Pdg::manifest(pdg).unwrap();

```

### 12.20.1 Bayesian Networks

**The frame.** Directed acyclic dependency structure with conditional probability tables; the bread-and-butter question is a posterior, which is to say *conditioning*.

**Profile.** Conditional-table wires over finite variables; the joint is the wires' product. Sum-product inference is semiring tensor contraction [198], lowered to contraction rows (the same contraction discipline the substrate reference chapter specifies), and the well-formedness side condition on contractions turns out to be the shape-level shadow of table normalization.

**Semantics.** The joint distribution, exact at finite carrier; the scoring semantics of the dependency-graph rendering recovers it as the consistent case. The rendering's score is, in the universe's terms, a belief (level-of-inconsistency functional) — Definition 9.17 names this incompatibility score as one of its two sources — and the consistent case is its zero locus.

**Adequacy.** The compilation above is not merely well-typed; it is *adequate*: compiling and then denoting in the semantic universe agrees with interpreting classically and then including the classical object into the universe's precise face — the first such square off the time-line, with the topological order of the variables playing the role the horizon plays for chains. Stated self-contained, at the working scale of an exam problem:

**Problem 12.15** (the chain-rule square). Let  $X_0, \dots, X_n$  be finitely many variables with finite value domains, listed in a topological order: each  $X_i$  has a parent set  $\text{Pa}_i \subseteq \{X_j : j < i\}$ , and carries a rational conditional probability table  $P_i(\cdot \mid \text{pa})$ , one row per parent assignment, each row summing to one. On the record space  $\Omega$  of joint assignments write  $\mu_B$  for the chain-rule joint,

$$\mu_B(\sigma) = \prod_{i \leq n} P_i(\sigma_i \mid \sigma|_{\text{Pa}_i}),$$

the factorization semantics of Bayesian networks [141, 196]. Say a subprobability mass function  $\mu$  on  $\Omega$  satisfies the compiled wires when (i) for every  $\sigma$ , the mass of  $\{\sigma' : \sigma'_0 = \sigma_0\}$  equals the sourceless first wire's stored row  $P_0(\sigma_0)$ , an absent row read as 0; and (ii) for every later variable  $i$  and every  $\sigma$ , writing  $C_i(\sigma)$  for the event that a record agrees with  $\sigma$  at all variables of index  $< i$ , refinement by one variable spends exactly one stored row:  $\mu(C_{i+1}(\sigma)) = \mu(C_i(\sigma)) \cdot P_i(\sigma_i \mid \sigma|_{\text{Pa}_i})$ , an absent row again read as 0 — a constraint family whose multiplier reads only the parent coordinates, i.e. the ordered Markov property with the stored tables. Show that  $\mu$  satisfies the compiled wires if and only if  $\mu = \mu_B$  — in particular the constraints force total mass one, so the ambient subprobability slack is never taken up.

*Sketch.* ( $\Leftarrow$ ) Downward induction from full depth gives the closed form  $\mu_B(C_i(\sigma)) = \prod_{j < i} P_j(\sigma_j \mid \sigma|_{\text{Pa}_j})$ : partition  $C_i(\sigma)$  by the value at variable  $i$  and spend one row sum per variable — the multiplier is well-defined because parents sit strictly below the refinement frontier. ( $\Rightarrow$ ) Upward induction along the order pins  $\mu(C_i(\sigma))$  to the same prefix products; at full depth the prefix events are the atoms  $\{\sigma\}$ , so all weights agree, and summing the atoms puts the total mass at one. Two structural remarks carry content: domain nonemptiness is derived, not assumed — the row-sum obligation at a parentless root is over the (inhabited) empty parent assignment, and normalization forces inhabited domains upward along the order — and the constraint family depends on the chosen topological order while the law it pins does not, so every topological presentation of one network denotes the same joint.

In the universe's terms: the constraint family, read as a formal conjunction — one atom per stored wire, in topological order — is the compiled system's denotation at the per-element stratum of the credal stack, and the theorem says this loose arrow is isomorphic, by an invertible globular cell of the thin double category of per-element constraints, to the singleton inclusion of  $\mu_B$ : the precise face of Remark 9.22, whose ray form agrees with the singleton on the radially maximal slice that the verdict operators read. The pointwise equivalence is the chain rule for Bayesian networks mechanized as a pinning statement. The statement and both directions are mechanized, sorry-free, in the toolkit's Lean tree (`lean/su-functor/`, module

`SuFunctor.Adequacy.Pgm`): the pointwise equivalence is `satAtoms_iff`, the invertible cell is `pgm_adequacy`, and the mass identity is `jointLaw_mass`. The same module ties the exact-rational law to the standard monadic construction — ancestral sampling, the variables drawn in topological order by iterated `PMF.bind` against their conditional distributions — as `ancestralPMF_eq`, so the classical side of the square is the community’s mechanized object rather than a private restatement; the inclusion vocabulary is shared with the sibling frames of this catalogue (`SuFunctor.Adequacy.Inclusions`). The square covers the conditional-table face of the species — exactly the Bayesian-network sub-frame of this section; potential wires carry no normalization row, so their inclusion routes through a global normalization fact (the Gibbs distribution of the Markov-random-field sub-frame below) and is deliberately a separate statement.

**Status.** Certification, finite corner: exact-rational posterior certificates with an independent checker and a mutant battery, positivity of the evidence mass checked as a row rather than assumed — on the sprinkler network,  $P(\text{Rain} \mid \text{Wet}) = 509/719$  exactly. Representation: the network as dependency-graph rows, with the posterior recomputed from the rendering’s own minimizer masses [211] and accepted byte-identically by the unmodified network checker. Open: the continuous, score-carrying case — certified conditioning beyond finite enumeration is the named frontier this row shares with section 12.21 — and native ingestion.

## 12.20.2 Structural Causal Models

**The frame.** Structural equations over endogenous variables driven by exogenous sources; the questions that make the frame are interventional and *counterfactual*.

**Profile.** A shipped theory extending the value vocabulary: equations as polynomial rows; an intervention as mechanism surgery — a partial replacement function over equations; and the coupling of counterfactual worlds *structural* rather than disciplinary: one exogenous table per model, so the independent-worlds misreading of a counterfactual is unrepresentable, not merely discouraged.

**Semantics.** Twin-world substitution in the standard (Pearl [195]) sense — diagrammatic modern form in Lorenz and Tull [167] — with the lightest bridge set on the list: substitution, box-measure monotonicity and additivity, and a union bound — no stochastic analysis anywhere. The bridge set is machine-checked, sorry-free, as module `Qtsl.ScmTwin` in the toolkit’s Lean tree: twin-network substitution proven outright for acyclic models, with the box-measure and union-bound lemmas stated as the certificate classes the certifications below instantiate.

**Status.** Certification, in both truth directions, on the thesis’s own flagship specification — a bound on the probability of counterfactually caused harm, at the thesis’s stated  $10^{-4}$  threshold: one controller certified compliant (with the “would have been harmed anyway” cells genuinely exercised), a second certified *in violation* with a two-sided bracket  $3/256 \leq P \leq 17/256$  — inner boxes make violation certifiable, not merely non-certifiable — and a decoupling probe certifying that the same compliant system under the independent-worlds product reading admits  $P \geq 1/2048$ : the frame’s defining distinction, priced by certificates on both sides. Representation: the first instance rows to represent interventions and counterfactual coupling at all. Open: conditioning (counterfactuals given evidence — the full abduction step, joining the frontier named in section 12.20.1); latent confounders and non-Markovian structure (representable, unexercised); non-uniform exogenous densities; the ingestion gap; closure — for which the natural witness is a dependency-wire coupling of an exogenous block to a dynamical frame.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{U : \text{ExogenousTable} \quad E : \overrightarrow{\langle \text{var}, \text{rhs} \rangle} \text{ equations, each rhs polynomial} \quad I? : \overrightarrow{\langle \text{var}, \text{replacement}? \rangle} \text{ interventions (mechanism surgery)}}{\text{SCM}(E; U) : \text{System}} \text{ S-SCM}$$

The exogenous table is an *option, not a list* — one per mechanism, by the sort of  $U$ , which is the structural fact the twin-world pattern of chapter 13 leans on.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.scm`

signature — `ts/src/system.ts`

```
/** S-SCM: polynomial equations + interventions + the shared exogenous
 * table (structural: one per mechanism, by type). */
export const scm = profileFactory("scm");
```

The TS conformance corpus holds no construction through this factory at this preview — the cross-language byte-identity fixtures exercise six profiles, and this one’s byte authority is the Rust corpus opposite; the factory takes the same init-object vocabulary as every other profile.

### Rust `mtk_catalogue::system::presets::Scm::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```
preset!(
    /// System Catalogue: SCM — equations as polynomial rows,
    /// interventions as mechanism surgery, and the shared exogenous
    /// table STRUCTURAL (one per mechanism, by type).
    Scm, "SCM",
    req = [Equations, Exogenous],
    allowed = [Interventions],
    extra = ok
);
```

construction — `crates/mtk-catalogue/tests/builders.rs`

```
let scm = MechanismBuilder::new()
    .equation(0, mono(2, 1, 1, r(2, 1)))
    .intervention(InterventionRow {
        var: 0,
        replacement: Some(constant(2, r(0, 1))),
    })
    .exogenous(ExogenousTable {
        vars: vec![ExoVar {
            id: 1,
            name: Some("u".to_owned()),
            dist: Some(ExoDist::Discrete {
                rows: vec![
                    DiscreteWeight {
                        value: r(0, 1),
                        prob: r(1, 2),
                    },
                    DiscreteWeight {
                        value: r(1, 1),
                        prob: r(1, 2),
                    },
                ],
            })
        }],
    })
```

```

    },
    },
  },
  .finish();
Scm::manifest(scm).unwrap();

```

### 12.20.3 Markov Random Fields

**The frame.** Undirected factor graphs with unnormalized potentials; the bread-and-butter certified quantity is the partition function.

**Profile.** The dependency-graph rendering again, plus one new atom: potential wires — raw nonnegative tables with no normalization row, their local scale derived from entries rather than stored. The founding paper’s own factor-graph embedding is carried as instance rows.

**Semantics.** The Gibbs distribution, recovered as the rendering’s optimum at unit weight; the bracket below is, in those terms, a two-sided certificate on the optimal score — cited to the theorems of the founding paper and its loss-function sequel [209, 210] as a named obligation.

**Status.** Certification: a two-sided partition-function bracket  $\ell \leq Z \leq u$  on a small Ising model with all-rational potentials. The lower end is an explicit support list checked against the summed product directly — the support-restricted variational bound with every logarithm cancelled; the upper end is a tree-reweighted (Hölder) forest decomposition checked as a single rational inequality in which the fractional root never appears on the wire. The checker never computes  $Z$ ; test-side brute force confirms the truth sits inside a bracket whose ends lie at  $0.76\times$  and  $2.81\times$  the true value — displayed as measured rather than tuned. The factor-graph image is generically *inconsistent* as a dependency graph, so this sub-frame and section 12.20.1 together exercise both strata of the rendering’s semantics through one species — checked evidence that the unification is a unification, not an economy. Open: contraction-row evaluation of the forest group sums; the continuous case; the ingestion gap; closure.

## 12.21 Probabilistic Programs

**The frame.** Corecursive programs in a functional probabilistic programming language: sample and score, loops, and the measures they denote.

**Profile.** For the discrete-time slice: loops with sample are outcome-row systems, and the published-table campaign of section 12.11 is certified probabilistic-program verification for that slice — the swept objects are programs, including programs with piecewise-linear neural components. The general profile (higher-order corecursion, continuous distributions — the slice whose denotational home is the  $\omega$ PAP semantics [118]) is not yet rendered.

**Semantics.** The program’s measure semantics, two-tier: the kernel language denotes in the affine probability monad — the monad fork [57], once merely named here, is resolved, and the affine tier’s composition law is machine-checked, sorry-free, at the finite-outcome stratum: with disjoint noise rosters, the denotation of a sequential composite is the Kleisli composite of the stage denotations (`noiseDenot_seqComp` in module `SuFunctor.Stochastic.NoiseKleisli` of the toolkit’s Lean tree), and the copy-vs-resample separation of chapter 11 is carried positively as `copy_ne_kleisli`: a shared roster provably departs from the Kleisli composite, so the disjointness is content, not bookkeeping — while score-carrying measures enter only at the certificate boundary; conditioning denotes a normalized ratio whose positivity is itself an obligation.

**Status.** The forward, unconditioned slice is in place at the discrete tier, as above — and the former-level square is machine-checked, sorry-free: a program’s loop denotes stage kernels, the unrolled program text denotes the kernel-local Kleisli iterate of those kernels at every horizon (`noiseDenot_progTraj`, unconditional

at the finite-outcome stratum, every measurable stage admitted), and for a program whose stage kernels have exact-rational faces the semantic universe’s judgment on the compiled outcome-row presentation pins exactly the unrolled text’s denotation (`su_pprog_adequacy` in module `SuFunctor.Adequacy.ProbProg`) — so the formation rule below sharing the chain former’s data is a theorem-backed reading, not an analogy. Conditioning — posterior bounds for the operation most probabilistic programs exist to perform — is the named frontier, problem statement in hand, shared with section 12.20.1; the published yardsticks are the guaranteed posterior bounds for universal programs [19] and for discrete programs with loops [245]. *The large-language-model clause*: the thesis’s gloss (“including, notably, autoregressive large language models”) gets a scope statement rather than a status. In the small, the clause is subsumable today — a toy autoregressive sampler is a discrete-time chain over token prefixes, and section 12.11’s machinery applies verbatim. In the sense the thesis intends — certified claims about the checkpoints people actually run — it is a programme-scale undertaking well beyond this inventory, and this chapter declines to price it as a backlog row. Stating that plainly is this entry’s content.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ ,  $\text{pop}$ , and profile conformance are defined in section 12.1.

$$\frac{O \text{ unkeyed outcome rows, as for DTMC} \quad \iota? \text{ initial-distribution rows} \quad \tau? : \text{TickDecl}}{\text{ProbProg}(O) : \text{System}} \quad \text{S-PPROG}$$

Deliberately the chain former’s formation data verbatim: the discrete-time slice of a probabilistic program is an outcome-row kernel, and what distinguishes this entry is its reading (the mechanism presents a program’s loop) plus the named open leg — program-as-rows, the corecursive source itself as first-class data, which no premise above carries yet.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.probProg`

signature — `ts/src/system.ts`

```
/** S-PPROG: probabilistic program, discrete-time slice (a loop with
 * `sample` is a kernel, riding the outcome-row machinery). */
export const probProg = profileFactory("probProg");
```

The TS conformance corpus holds no construction through this factory at this preview — the cross-language byte-identity fixtures exercise six profiles, and this one’s byte authority is the Rust corpus opposite; the factory takes the same init-object vocabulary as every other profile.

### Rust `mtk_catalogue::system::presets::ProbProg::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`



```

preset!(
  /// System Catalogue: probabilistic program, discrete-time slice –
  /// a loop with `sample` is a kernel, riding the outcome-row
  /// machinery. Program-as-rows representation (the corecursive
  /// source as first-class data) is deliberately open; neural
  /// components
  /// ride the hypernet route.
  ProbProg, "probabilistic program",
  req = [Outcomes],
  allowed = [InitDist, Tick],
  extra = |m| outcomes_keyed("probabilistic program", m, false)
);

```

The discrete-time slice rides the outcome-row machinery; the corpus's construction of that machinery is the DTMC fixture, and no ProbProg-preset construction is in the conformance corpus at this preview.

## 12.22 Probabilistic Logic Programs

**The frame.** Definite clauses plus probabilistic facts; the distribution semantics — a measure over least models induced by independent fact coins — and certified brackets on a query's success probability.

**Profile.** Intended: probabilistic facts and clauses as rows over the substrate's own chase — which is already a geometric, Horn-clause engine — with derivation weights in a declared ordered-semiring algebra [174].

**Semantics.** Sato-style distribution semantics. What the frame adds beyond weighted derivations is the *independence structure* of the fact coins: derivations share coin outcomes, so query mass is a sum of products with shared variables, not a free semiring sum — and exactly there, certified *bounds* rather than exact weighted model counting become the achievable target.

**Status.** The frame integration — facts and clauses as rows over the substrate's own chase — remains unshipped, and the frame is graded absent on exactly that line: adjacent code, however promising, does not lift a frame out of absent (that line is what makes these statuses checkable). What the toolkit does carry is the stratum below: the declaration profile (the API block after the metanotation) and, over it, a producer-side certificate species — `problog-v0`, a certified bracket on a query's success probability computed from producer-found explanation and cut rows — whose own API block closes this section. The structural proximity noted above stands: the weight-algebra code exists, and wiring it into the chase is a named seam of the substrate rather than a research question.

### Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema's own (`crates/mtk-catalogue/src/system/`), and the former's mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , `pop`, and profile conformance are defined in section 12.1.

$$\frac{F : \langle \text{atom}, \text{weight}, \text{coin?} \rangle \text{ weighted facts} \quad \text{semiring a declared ordered semiring} \quad \text{program\_digest? a content-addressed program reference}}{\text{ProbLog(semiring; } F) : \text{System}} \quad \text{S-PROBLOG}$$

The coin index is the independence structure: derivations sharing a coin share its outcome, which is exactly what makes bounds rather than free semiring sums the achievable target.

### API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a

native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

## TypeScript `system.problog`

signature — `ts/src/system.ts`

```
/** S-PROBLOG: weighted logic programs over a declared ordered
 * semiring. */
export const problog = profileFactory("problog");
```

construction — `ts/tests/problog-native.test.ts`

```
return sys.problog({
  weightedProgram: {
    semiring: "probability",
    facts,
    programDigest: sha256(SMOKERS_SOURCE),
  },
});
```

facts are the smokers fixture's weighted rows (stress and influence facts sharing declared coins), built beside the quoted lines; the system constructed here feeds the certificate corpus's `fromSystem` path, whose emitted bytes are verified against the committed golden vectors.

## Rust `mtk_catalogue::system::presets::ProbLog::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```
preset!(
  /// System Catalogue: ProbLog / weighted logic programs – the
  /// chase over a declared ordered-semiring weight algebra +
  /// independence structure of fact coins.
  ProbLog, "ProbLog",
  req = [WeightedProgram],
  allowed = [],
  extra = ok
);
```

construction — `crates/mtk-catalogue/tests/builders.rs`

```
let problog = MechanismBuilder::new()
  .weighted_program(WeightedProgramDecl {
    semiring: "probability".to_owned(),
    facts: vec![WeightedFact {
      atom: "edge(a,b)".to_owned(),
      weight: r(3, 5),
      coin: Some(0),
      coin_domain: None,
    }],
    program_digest: None,
  })
  .finish();
ProbLog::manifest(problog).unwrap();
```

## API

Producer surface of the `mtk` toolkit for this certificate species: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

The claims stratum over the S-PROBLOG declaration: a problog-v0 certificate carries the ground program (coin, fact, and clause rows), the producer's search artifacts (explanation and cut rows), and the claimed bracket on the query's success probability — lower by inclusion–exclusion over coin-set unions, upper by min-over-cuts, exact by world enumeration where the coin count permits. The builder verifies every row it is handed (an explanation must derive the query on its own; a cut's joint absence must block it against every other coin) and computes the bracket itself; where the rows came from is the producer's business, and the checker re-derives every claim from the certificate bytes alone.

### TypeScript problog.builder / problog.fromSystem

signature — ts/src/problog.ts

```
/** Start a `problog-v0` certificate for `query` in the probability
 * semiring (declare the ground program with the chainable methods,
 * derive claims with `claimsFromRows`, then `build`). */
export function builder(query: string): CertBuilder {
  return new CertBuilder(new native.PlogCertBuilder(query));
}

/** Derive a certificate builder from a System whose manifest carries
 * the `weightedProgram` family (the S-PROBLOG profile's declaration):
 * coin probabilities come from fact weights (facts sharing a coin
 * must agree – refused with both fact indices otherwise), coinless
 * facts are deterministic and must carry weight 1, and the
 * declaration's program digest is required – it is what pins the
 * grounding correspondence. A System without the family throws
 * (programmer error – wrong manifest kind). */
export function fromSystem(
  system: System,
  clauses: ReadonlyArray<ClauseRow>,
  query: string,
): Result<CertBuilder, Refusal> {
  const r = native.problogBuilderFromManifest(
    system.nativeHandle,
    clausesToNative(closures),
    query,
  );
  return isRefused(r)
    ? refusedResult(narrowRefusal(r.refusal))
    : ok(new CertBuilder(r));
}
```

construction — ts/tests/problog-native.test.ts

```
const b = smokersBuilder(problog, sys);
const claims = problog.unwrap(b.claimsFromRows(ANTICHAIN, CUTS, true));
assert.equal(claims.exact?.toString(), "48861/125000");
assert.equal(claims.lower.toString(), "48861/125000");
assert.equal(claims.upper.toString(), "69/125");
const cert = problog.unwrap(b.build(claims));
```

smokersBuilder and its clause fixture are defined beside the quoted lines; the same file's die-chain exemplar exercises the direct-builder path (builder, mswCoin switch domains) against its own golden vector.

### Rust mtk\_catalogue::system::problog::ProblogCertBuilder

signature — crates/mtk-catalogue/src/system/problog.rs

```
pub fn new(query: impl Into<String>) -> Self

pub fn from_decl(
  decl: &WeightedProgramDecl,
  clauses: Vec<GroundClauseRow>,
  query: impl Into<String>,
) -> Result<Self, ProblogRefusal>

pub fn claims_from_rows(
  &self,
  explanations: Vec<Vec<u32>>,
  cuts: Vec<Vec<u32>>>,
```

```

        with_exact: bool,
    ) -> Result<ProblogClaims, ProblogRefusal>

    pub fn build(self, claims: ProblogClaims) -> Result<ProblogCert, ProblogRefusal>

```

construction — crates/mtk-catalogue/tests/problog\_builders.rs

```

// Manifest stratum: the S-PROBLOG profile accepts the declaration.
let decl = smokers_decl();
let manifest = MechanismBuilder::new()
    .weighted_program(decl.clone())
    .finish();
Problog::manifest(manifest).unwrap();

// Claims stratum: ground the program, hand over the search
// artifacts (explanation and cut rows), and let the builder verify
// them and compute the bracket — lower by inclusion-exclusion over
// coin-set unions, upper by min-over-cuts, exact by enumeration.
let builder = ProblogCertBuilder::from_decl(&decl, smokers_clauses(), "smokes(a)").unwrap();
let claims = builder.claims_from_rows(antichain(), cuts(), true).unwrap();
let cert = builder.build(claims).unwrap();
let bytes = cert.to_cbor().unwrap();

```

smokers\_decl, smokers\_clauses, antichain, and cuts are the corpus's shared fixtures (tests/problog\_common), each re-derived by the reference-number tests beside the quoted construction.

## API

Producer surface of the mtk toolkit for this certificate stratum: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named fx\_\* or f.\* are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

The exact-beyond-enumeration stratum of the problog-v0 certificate: past the world-enumeration cap, the producer commits its search as a leaf-cited decision tree — explanation and cut rows compiled so that every path through the coin axes closes at the first row that decides it, an undecidable cube refusing with the world as witness — and the exact probability is re-derived by the same verify-and-fold walk the checker runs, so the claim never rests on the producer's arithmetic. Both extensions of the wire (the compiled section and the even-depth Bonferroni truncation obligation) are optional and skip-if-absent: every earlier certificate's bytes decode unchanged.

## TypeScript problog.CertBuilder.compiledFromRows

signature — ts/src/problog.ts

```

/** Compile the full compiled section (exact-beyond-enumeration)
 * from producer-found rows: verifies each row, builds the
 * leaf-cited decision tree — refusing, with the undecided world as
 * witness, when the rows cannot decide every cube — and computes
 * `exact` by the same verify-and-fold walk the preflight and the
 * checker run. Attach the result as `claims.compiled` before
 * `build`. */
compiledFromRows(
    explanations: ReadonlyArray<ReadonlyArray<number>>,
    cuts: ReadonlyArray<ReadonlyArray<number>>,
): Result<CompiledSection, Refusal> {
    const r = this.h.compiledFromRows(
        explanations.map((e) => [...e]),
        cuts.map((e) => [...e]),
    );
    return isRefused(r)
        ? refusedResult(narrowRefusal(r.refusal))
        : ok(compiledFromNative(r));
}

```

construction — ts/tests/problog-native.test.ts

```
// The compiled section: producer-found rows (an explanation
// antichain plus cut rows covering every reject cube) compile to a
// leaf-cited decision tree whose exact-rational fold certifies the
// exact query probability with no world-count cap.
const b = dieBuilder(problog);
const compiled = problog.unwrap(
  b.compiledFromRows(DIE_ANTICHAIN, DIE_COMPILED_CUTS),
);
const claims = problog.unwrap(b.claimsFromRows(DIE_ANTICHAIN, DIE_CUTS, true));
const cert = problog.unwrap(b.build({ ...claims, compiled }));
```

dieBuilder and the row fixtures are defined beside the quoted lines; the byte-identity assertion against the committed compiled golden sits directly under the marked construction.

**Rust** mtk\_catalogue::system::problog::ProblogCertBuilder::compiled\_from\_rows

signature — crates/mtk-catalogue/src/system/problog.rs

```
pub fn compiled_from_rows(
    &self,
    explanations: Vec<Vec<u32>>,
    cuts: Vec<Vec<u32>>,
) -> Result<CompiledSection, ProblogRefusal>
```

construction — crates/mtk-catalogue/tests/problog\_compiled.rs

```
// The compiled section: producer-found rows (an explanation
// antichain plus cut rows covering every reject cube) compile to
// a leaf-cited decision tree whose exact-rational fold certifies
// the exact query probability with no world-count cap.
let builder = problog_common::die_builder();
let compiled = builder
  .compiled_from_rows(
    problog_common::die_antichain(),
    problog_common::die_compiled_cuts(),
  )
  .unwrap();
let mut claims = builder
  .claims_from_rows(
    problog_common::die_antichain(),
    problog_common::die_cuts(),
    true,
  )
  .unwrap();
claims.compiled = Some(compiled);
let cert = builder.build(claims).unwrap();
```

the shared fixtures (problog\_common) are the same corpus the claims-stratum block quotes; the hand-built switch-first tree beside the quoted test verifies the same certificate rows against an independent tree layout.

## 12.23 Clocked Boolean Circuits

**The frame.** Synchronous gate-level hardware (and-inverter graphs) as a first-class model type — one of two entries in production use that fall outside the 2023 roster, evidence the inventory is a living catalogue rather than homework against a fixed list.

**Profile.** Circuit content by digest — the ruled content-addressing idiom; gate internals stay un-row-ified — plus boot-prefix rows and per-lane input-configuration rows. Disturbance: demonic inputs as per-lane branch covers. Time: counter, on the machine clock, with a frame counter above it.

**Semantics.** The special case where the checker *is* the semantics: exact replay equivalence, bit for bit; claims read the credal-lower return (Definition 9.21) over the declared branch cover.

**Status.** Certification is the heavy leg: this is the type underneath the hardware-liveness and console campaigns of sections 2.5 and 2.6, where certified minimum-return claims run over branch covers at  $10^7$  ticks. Representation: declaration rows shipped (digest, boot prefix, input configurations). Open: the ingestion gap; closure.

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema's own (`crates/mtk-catalogue/src/system/`), and the former's mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{\text{digest a content-addressed circuit reference} \quad B : \overline{\langle \text{tick, wire, value} \rangle} \text{ boot-prefix rows} \quad C : \overline{\langle \text{lane, wire, value} \rangle} \text{ input-configuration rows} \quad \tau? : \text{TickDecl}}{\text{Circuit}(\text{digest}) : \text{System}} \quad \text{S-CIRCUIT}$$

Gate internals stay un-row-ified — content addressing is the ruled idiom, and the digest names the whole gate graph.

## API

Producer surface of the mtk toolkit for this model type: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.circuit`

signature — `ts/src/system.ts`

```
/** S-CIRCUIT: clocked boolean circuit (AIG by digest; exact replay IS
 * the semantics). */
export const circuit = profileFactory("circuit");
```

The TS conformance corpus holds no construction through this factory at this preview — the cross-language byte-identity fixtures exercise six profiles, and this one's byte authority is the Rust corpus opposite; the factory takes the same init-object vocabulary as every other profile.

### Rust `mtk_catalogue::system::presets::Circuit::manifest`

signature — `crates/mtk-catalogue/src/system/presets.rs`

```
preset!(
    /// System Catalogue: clocked boolean circuit (AIG by digest) –
    /// content-addressed circuit + boot-prefix rows +
    /// input-configuration rows; exact replay IS the semantics.
    Circuit, "clocked circuit",
    req = [Circuit],
    allowed = [Tick],
    extra = ok
);
```

construction — `crates/mtk-catalogue/tests/builders.rs`



```

let circuit = MechanismBuilder::new()
  .circuit(CircuitDecl {
    digest: vec![0xAB; 32],
    boot_prefix: vec![BootRow {
      tick: 0,
      wire: 0,
      value: true,
    }],
    input_configs: vec![InputConfigRow {
      lane: 0,
      wire: 1,
      value: false,
    }],
  })
  .finish();
Circuit::manifest(circuit).unwrap();

```

## 12.24 The i.i.d. Tier

**The entry.** Not a profile — a claim class that every sampleable model type carries, and the second production entry beyond the 2023 list. Where a system’s rollouts can be executed but no analytic species reaches its semantics — a compiled physics engine is the paradigm case — the catalogue switches to *confidence-qualified* judgments: claims of the form “the expected return of this pinned artefact is at least  $B$ , with confidence  $1 - \alpha$ ” (species U5 of section 2.1, always displayed as their own class, never mixed into a table of exact rows). The tier’s name states its subject: a pinned, *closed* composite, sampled under its own law episode after episode — independent and identically distributed draws of the artefact itself. This is the tier behind the physics-engine results of section 2.4, and it composes with the profile inventory rather than competing with it: the analytic tiers take over exactly where a frame’s rows stop being samples and start being algebra.

**The claim form is more general than it looks.** The expectation floor on a bounded per-episode output is the tier’s normal form, and claims that sound set-shaped compile into it. “The next landing stays inside the pad region with probability at least 0.99” is the statement that the mean of the landing indicator is at least 0.99; a quantile bound is a coverage bound at a fixed threshold, hence again an indicator mean; and a bound on a conditional value-at-risk is an expectation floor on a derived output — exhibiting any threshold  $z$  with  $z + \mathbb{E}[(X - z)^+]/(1 - \beta) \leq c$  certifies  $\text{CVaR}_\beta(X) \leq c$ , by the variational form of Rockafellar and Uryasev [213]. What the producer must arrange is that the property’s verdict or score be computed *inside* the sampled composite — the property logic wired in as one more component, its terminal output the episode’s return — and every claim above is then one expectation floor, carried by one certificate species.

**The guarantee class.** The certificates are anytime-valid lower confidence bounds in the game-theoretic school of sequential inference [207]: a betting martingale — concretely, the hedged-capital process for bounded means of Waudby-Smith and Ramdas [237] — is run over the recorded rollouts, the certificate is the resulting wealth ledger, sound by Ville’s inequality [234], and the checker replays the entire ledger in directed-rounding fixed point. This is not a new trust principle beside the rest of the book but the same one in a second mode: the supermartingale certificates of the analytic tiers *prove* the Ville bound against a known kernel, while this tier’s ledger *wagers* it against whatever law would violate the claim — the wealth process is a nonnegative supermartingale under every violating law by construction of the bets, so observed wealth crossing  $1/\alpha$  rejects every such law at confidence  $1 - \alpha$ , at any stopping time, with no penalty for peeking and none for extending the ledger. That anytime validity is what makes the certificate a stable public artefact rather than a frozen experiment, and the betting form is what an independent checker can re-verify line by line; sound statistical model checking [30] is the nearest published genre.

**The conformal sibling.** Conformal prediction [236] — lately including accept/reject guarantees for generative samplers under the name *conformal sampling* [161] — is the same distribution-free finite-sample school under a *weaker* assumption, exchangeability of the draws, with a different validity mechanism: rank statistics of a completed calibration batch, exact at a calibration size fixed in advance. The relation is precise. At the level of claims, a conformal coverage guarantee is an indicator-mean floor, so under this tier’s

stronger sampling assumption the ledger certifies the same statements — anytime-valid, and conditionally on the fitted predictor rather than on average over calibration draws. At the level of mechanisms the two are genuinely different: no adapted betting process can be simultaneously powerful under exchangeability alone — for binary sequences, a nonnegative supermartingale valid under every exchangeable law stays a supermartingale over their closed fork-convex hull, which contains every law on such sequences [208] — and this tier does not attempt that regime. It does not need to: its subject is a pinned closed artefact whose independent resampling is realized by the harness and attested on the certificate, not assumed of nature — the situation in which the stronger assumption costs nothing.

**Pinning.** A confidence bound on “the expected return of this artefact” means something only if there is one fixed law being sampled, and pinning is that premise made auditable — three ordinary ingredients rather than one new trust primitive. First, closure: the sampled composite has no dangling inputs, because an open system has no episode law to sample. Second, identity: trust rows name the sampled artefact by content digest — the policy and its determinism — the same content-addressing idiom the circuit former rules. Third, the one genuinely statistical residue, the sampling-law witness: the attestation that the deployed harness draws independent episodes of the digest-named artefact, with the seed provenance of every source of randomness as its evidence. All three are stated on the face of the certificate, never hidden; ledger production itself lives with the sampling harness rather than in these rows, and the almost-sure return range the martingale requires is itself a certified lemma about the environment’s code, not an assumption (Table 2.5).

## Computer Science Metanotation

as realized in the mechanized toolkit: the family vocabulary is the manifest schema’s own (`crates/mtk-catalogue/src/system/`), and the former’s mechanized ground truth is quoted in the API block below — the producer signatures and, for profile entries, the preset signature naming the populated families — extracted from the toolkit sources at every regeneration of this catalogue, so the two cannot drift apart silently. The formation rule here is typeset by hand for notational quality;  $\Sigma$ , pop, and profile conformance are defined in section 12.1.

$$\frac{\text{rows} : \overline{\langle \text{kind}, \text{detail?}, \text{digest?} \rangle} \text{ trust rows}}{\langle \text{rows} \rangle : \text{PinningDecl}} \quad \text{S-STAT}$$

Not a System former at this preview:  $\text{pinning} \in \Sigma$ , so the family is substrate, admitted alongside every former above, and the confidence-qualified judgment applies to any sampleable type whose subject is pinned. The catalogue’s design direction is for the sampled subject itself to become a first-class construction — a replication former on closed composites, in the Bag former’s genus, whose formation premise is the closure ingredient above and whose trust seam is exactly these rows; it enters the formal inventory with its producer surface, not before.

## API

Producer surface of the mtk toolkit for this entry: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript `system.PinningDecl`

signature — `ts/src/system.ts`

```
export interface PinningDecl {
  readonly rows: ReadonlyArray<TrustRow>;
}
```

```
export interface TrustRow {
  readonly kind: string;
  readonly detail?: string;
  readonly digest?: string;
}
```

Pinning is deliberately not a profile: any sampleable mechanism carries it, and the corpus holds no pinned construction at this preview — ledger production lives with the sampling harness.

**Rust** `mtk_catalogue::system::MechanismBuilder::pinning`

signature — `crates/mtk-catalogue/src/system/manifest.rs`

```
| pub fn pinning(mut self, p: PinningDecl) -> Self
```

Pinning is deliberately not a preset (any sampleable profile carries the family); the corpus holds no pinned construction at this preview — ledger production lives with the sampling harness.

## 12.25 Quantum Nets

**The frame.** Algebraic quantum field theory in its locally covariant formulation, at the kinematic fragment whose mechanization chapter 9 presents (section 9.2): observables indexed by causal diamonds, as they have been since Haag and Kastler [103]; an isotone net of local algebras with Einstein causality and the time-slice property, per Brunetti, Fredenhagen, and Verch [29]; and a positive record family whose Born rows are the measured content. The grade comes first, because a reader who hears “locally covariant quantum field theory” will expect type-III von Neumann algebras, microlocal spectrum conditions, and the continuum. This entry works at a deliberately finite stratum — a finite site, matrix algebras, the kinematic axioms, exact arithmetic — and “fragment” does the same work in this section that it does in the Lean names. Genuine dynamics (relative Cauchy evolution, the stress-energy tensor) lives on the far side of the fragment’s fence, which is where the locally covariant literature itself places it [76], and Fewster and Verch proved that “locally covariant” does not by itself mean “the same physics in all spacetimes” [77]. The bread-and-butter question is a certified bound on record probabilities holding uniformly over every admissible state, and the uniformity is no modelling flourish: the credal face below inherits it from the community’s own state doctrine. A two-qubit example runs through the section, with records the four projectors of the joint computational-basis measurement, so that the record index is a pair of bits: the Bell state prices the row  $(\frac{1}{2}, 0, 0, \frac{1}{2})$ , and the maximally mixed state prices  $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ .

**Profile.** A site kit — finitely many points with their chronology; diamonds, covers, and a distinguished ambient window are derived — with a per-window algebra presentation at matrix grade, generating families entered in  $\mathbb{Q}[i]$ ; matter data (worldlines, jointly exhaustive factors, a reversible mechanism), from which Cauchy determinacy follows as a theorem rather than a postulate; a declared record family at the ambient window — finitely many positive-semidefinite matrices summing to one, a positive operator-valued measure — and a state face, declaring either one distinguished state (the precise face) or a set of states (the credal face), each state carried with the exact-rational witness of its Born row. The carrier is the record space of the declared family, never the algebra: the  $C^*$ -algebra is presentation-tier data, as a Lagrangian is, and what the semantic universe consumes is its gauge-invariant shadow — probability rows over record outcomes. Initial conditions live in the state face.

**Sited time.** Every counter-time entry in this catalogue composes into real time through a tick declaration (section 12.2). This entry needs none: its windows are the site’s own causal diamonds, so the time semantics is the causal order itself — the ground storey of the semantic universe (section 9.1) — and a separate tick would have nothing to declare. Among the values the catalogue’s time-semantics column has needed so far (continuous, counter, hybrid), this one is new: sited.

**Semantics.** The record law of Gell-Mann and Hartle [86], which at a single window is the Born rule: a state  $\rho$  prices the record  $\alpha$  at  $\text{Re tr}(\rho R_\alpha)$ . The mechanized form of that law is the fragment’s own record shadow, so the bridge between the community’s formula and the toolkit’s is definitional rather than analytic; the connector lemma below is one line. Two doctrines from the locally covariant literature

enter as structure rather than gloss. First, the semantics factors through the state object: two states with the same restrictions at the ambient window price every record identically, so the square's left edge is a function of the state-presheaf fibre, as §5.1 of Brunetti–Fredenhagen–Verch demands [29]. Second, the state object is set-valued from birth: a functorial choice of single states is unrealizable — a theorem of the framework [29], re-proven model-independently under dynamical locality [77] — and the replacement that literature adopted, the folium, is structurally the flat shadow of this book's credal bodies (section 9.2 tells that story in full). The well-formedness hypotheses the squares consume are the community's own axioms: isotony, Einstein causality, the time-slice property at its two tiers, record positivity and exhaustiveness, state positivity and normalization.

**Adequacy (precise face).** The compilation stores the Born row and nothing else, so the square is the catalogue's leanest, and its textbook corner is unlike any other in the roster: every other entry's community-facing corner was built for its square, while this one consumes the mechanized fragment verbatim, because the fragment's fields already are the community's axioms. Stated self-contained, at exam scale:

**Problem 12.16** (the quantum-net square). Let  $\kappa$  be a finite record index,  $R: \kappa \rightarrow M_n(\mathbb{C})$  a family of positive-semidefinite matrices with  $\sum_{\alpha} R_{\alpha} = 1$ , and  $\rho \in M_n(\mathbb{C})$  a state: positive semidefinite, of trace one. Store the Born row as exact rationals,  $q: \kappa \rightarrow \mathbb{Q}$  with  $q_{\alpha} = \text{Re tr}(\rho R_{\alpha})$ , and say a subprobability mass function  $\mu$  on  $\kappa$  satisfies the compiled rows when  $\mu\{\alpha\} = q_{\alpha}$  for every  $\alpha$ . Show: (i)  $q$  is a probability row — every entry nonnegative, total mass one; (ii)  $\mu$  satisfies the compiled rows if and only if  $q$  is its weight function, and the uniqueness direction consumes neither an induction nor the exhaustiveness of  $R$ ; (iii) storing only a subfamily  $J \subseteq \kappa$  yields a row of mass  $\sum_{\alpha \in J} q_{\alpha}$  — exhibit an instance of mass exactly  $\frac{3}{4}$ ; (iv) positivity cannot be dropped: exhibit a Hermitian family summing to one, and a state, that together price an outcome negatively.

*Sketch.* (i) The trace of a product of positive-semidefinite matrices is real and nonnegative, and linearity spends the two normalizations:  $\sum_{\alpha} \text{Re tr}(\rho R_{\alpha}) = \text{Re tr}(\rho) = 1$ . (ii) The singleton probes read off every weight directly, and two subprobability mass functions with equal weights are equal — finite extensionality and nothing more. Compare the chain square (Problem 12.8), whose uniqueness runs an induction along the horizon and spends a row sum per step; exhaustiveness enters here only through the mass claim of (i). (iii) Sum the stored subfamily; for the instance, take the maximally mixed two-qubit state against three of the four basis projectors:  $3 \cdot \frac{1}{4} = \frac{3}{4}$ . (iv) One dimension suffices: the family  $(2, -1)$  on  $M_1(\mathbb{C})$  sums to one, and the unique state prices the second outcome at  $-1$ .

In the universe's terms: the compiled rows denote one probability-pinning constraint per record index, their formal conjunction a loose arrow at the per-element stratum of the credal stack, and the theorem exhibits an invertible globular cell to the singleton inclusion of the Born row — the precise face of Remark 9.22. The statement and both directions are mechanized, sorry-free, in the toolkit's Lean tree (`lean/su-functor/`, module `SuFunctor.Adequacy.QuantumNet`): the invertible cell is `quantum_net_adequacy`, the pointwise equivalence is `satAtoms_iff`, and the presentation structure (`QNetSystem`) consumes the fragment (`LCQFTFragment`, module `QtslAux.QuantumAssembly`) verbatim, together with a distinguished state and its exact-rational row witness. The community tie is cheaper here than anywhere else in the catalogue, and the entry says why rather than dressing it as more: the connector `shadowPMF_prob_eq_born` is the row witness itself — the mechanized record shadow is the literature's formula transcribed — and the mechanized probability library has no quantum machinery for a deeper tie to borrow, where the chain square could tie into the library's own monadic construction. What replaces depth is doctrine: `shadowPMF_eq_fragment_shadow` and `shadowPMF_class_invariant` factor the left edge through the state-presheaf fibre at the ambient window, so the classical semantics consumes the Brunetti–Fredenhagen–Verch state object rather than a bare density matrix. The mass theorems are `shadowLaw_mass` and `rowsLaw_mass_of_complete`, and the two-qubit instances tie to the fragment's landed shadows by `rfl(bellQNet_shadowPMF, mixedQNet_shadowPMF)`. Task (iii) is a first for the catalogue: the target stratum has carried a mass-loss allowance from birth, every stochastic entry's constraints force mass one (the chain problem says so explicitly), and `exists_incomplete_mass_lt_one` — three of the four projectors at the maximally mixed state — is the first compiled row set to exercise the allowance: an incomplete record family denotes a genuine subprobability law. Task (iv) is `exists_nonPSD_family_negative_shadow`, the kernel-visible reason record positivity appears among the hypotheses. The square also re-reads as a citizenship statement in the semantic universe's fibre over a one-wire record interface, by pushforward along the record embedding (`su_qnet_adequacy`); the wiring pattern is the assignment interface's, reused verbatim.

**Adequacy (credal face).** Every other imprecise face in this catalogue is chosen: a modeller declares a parameter box, or quantifies over policies or schedulers, because caution recommends it. Here the set-valuedness is a theorem of the frame's own literature — a functorial choice of single states is unrealizable [29, 77] — so the state face of a quantum-net presentation declares a set of states because the theory has nothing narrower to declare. Stated self-contained, at exam scale:

**Problem 12.17** (the credal square). Data as in Problem 12.16, except the state face: in place of one  $\rho$ , a set  $S$  of states is declared, and a *member witness* is a state of  $S$  together with the exact-rational row of its Born values. Write  $D$  for the set of member Born rows. On the compiled side the rows stay state-free: say  $\mu$  is *admitted* when the compiled rows of *some* member witness are satisfied by  $\mu$ . Show: (i)  $\mu$  is admitted if and only if  $\mu \in D$ ; (ii) every member's row has mass one, so the downward-closed credal presentation generated by the members' rays has  $D$  as its radially maximal slice, and both verdict polarities read as bounds uniform over every member; (iii) if  $S$  is closed under rational convex mixture then  $D$  is closed under rational mixtures of member rows, and the hypothesis is necessary: over the two-element set consisting of the Bell state and the maximally mixed state, the midpoint of the two rows prices the double-zero record at  $\frac{3}{8}$ , and no member does; (iv) record the degeneracies — with at least one member witness, the whole-set reading of a lower-bound atom refuses every positive threshold, and with none, both verdict kinds pass vacuously.

*Sketch.* (i) is the precise face's feeding lemma, run once per member, under the union of singletons. (ii) The record family's exhaustiveness prices every member at mass one, and the equal-mass reconciliation is the automaton problem's (Problem 12.13), reused. (iii) A quantum mixture's Born row is the same rational mixture of the members' rows, by linearity of the trace, so mixture-closure of  $S$  pushes to convexity of  $D$ ; for the negative half, the member witnesses pin the double-zero entry to  $\frac{1}{2}$  and  $\frac{1}{4}$ , and  $\frac{3}{8}$  is neither. (iv) The zero model lies on every ray, and a universal quantifier over an empty member index holds vacuously.

In the universe's terms: the admitted-by-some-member reading of the state-free compiled rows and the union-of-singletons inclusion of  $D$  are isomorphic by an invertible globular cell, exactly as at the decision and automaton formers — but where those faces quantify over an adversary the modeller invited, this one quantifies over the physics. The statement and both directions are mechanized, sorry-free, in the same module: the invertible cell is `quantum_credal_adequacy`, with admission predicate `qnetCredalC` and feeding lemma `qnetCredalC_iff` over member witnesses `RatShadowMember` (a member is carried with its certificate, never bare); the slice reconciliation is `quantum_maxSlice`, packaged as a cell by `quantum_ray_singleton_reconciliation`; the verdict transfers at both polarities are `satCredal_lower_quantumCredal` and `satCredal_upper_quantumCredal`, with law-set twins; the convexity theorem is `quantumLaws_mix`, consuming the fragment's mixture seam (`recordShadowPMF_mix`), with negative witness `mixLaw_not_mem_pair`; and the degeneracies are `not_satCredal_upper_quantumCredal_geC` and `satCredal_quantum_of_isEmpty`, recorded rather than smoothed. One more degeneracy is quoted deliberately: at matrix grade the folium is flat — every linear functional that is unital and positive on rank-one projectors is represented by a unique density matrix (`Qtsl.QuantumStates.existsUnique_isState_traceRepr`) — so this face's content lives in the set-valuedness and the credal band, never in folium distinctions; the strict openness of the Bell/mixed band is the fragment's `exists_shadow_lowerE_lt_upperE`. Deliberately separate strata, stated once: multi-window histories through the class-operator machinery of `QtslAux.QuantumRecords`, the named stretch tier; the quasilocal stratum, where folium distinctions become real and the algebras stop being matrix algebras; and dynamics beyond the kinematic fragment — relative Cauchy evolution and the stress-energy tensor [76] — outside this square by hypothesis rather than silently discharged.

**Refinement and abstraction.** The two squares above relate a System to its community semantics; this passage's statement-kind relates a System to a System. The community doctrine being transposed is effective field theory, and the grade comes first once more: a reader who hears the phrase will expect loop integrals, power counting, renormalization-group flow, and the continuum, and none of those live at this stratum. What survives the transposition — finite site, matrix algebras, exact arithmetic, kinematics plus one-step dynamics — is the procedure's logic, and the logic turns out to be the certifiable part. As its own standard references present it [34, 197], effective field theory is a two-ingredient move. Coarse-grain exactly: the Wilson action is what remains after the high-energy modes are integrated out once and for all — a pushforward, with nothing approximate about the step [34]. Then truncate with a declared tolerance: the effective theory keeps finitely many operators, their coefficients matched so that its predictions agree



with the pushed-forward theory up to the desired order [197]. An effective theory is never equal to the pushforward; it agrees with it inside a declared band, and the band is part of the statement. The move's reach is why it belongs in a catalogue of certifiable model types at all: it is effective-theory reasoning that justifies neglecting quantum corrections in macroscopic gravity while pricing them systematically [34] — “known physics” at any scale is an effective theory with a band — and a containment certificate is what that kind of validity claim looks like as an artefact.

Both ingredients are already on this entry's shelf, the first as the record-level coarse-graining calculus — record atoms merge along any map of indices, and pushing the law is measuring the merged observable family — and the second as a containment. Write  $d$  for the total-variation distance on outcome rows, half the  $\ell^1$  distance, exact rational at rational rows; inflate a law set by a tolerance  $\varepsilon \in \mathbb{Q}_{\geq 0}$ , admitting every row within distance  $\varepsilon$  of a member. The *certified-abstraction* statement between a fine System and a coarse one, along coarse-graining data  $f$ , is then a containment: every law the fine theory permits, pushed forward along  $f$ , lies in the coarse theory's  $\varepsilon$ -inflated law set. Error bars become imprecision, and “valid to accuracy  $\varepsilon$ ” becomes decidable arithmetic at finite index. Three choices inside that sentence carry the doctrine. The quantifier direction is soundness: the coarse law set may be strictly fatter than the pushforward — that is what “effective” means — but no behaviour the fine theory permits is ruled out, so the abstraction is epistemically honest: the coarse description's uncertainty band covers the fine truth. The tolerance lives on the law set rather than the state set, because matching in the literature is a statement about observables, and the checker's arithmetic lives on rows. And the tolerance is declared, never derived: power counting — the literature's dimensional-analysis estimate of what a given truncation achieves [34] — proposes  $\varepsilon$ , and the containment certificate checks it. Burgess's “Effective-Lagrangian Logic” [34] is a four-step procedure whose middle steps rest on power-counting estimates; here the estimates sit exactly where an untrusted heuristic belongs, in producer-side search, and trust-the-estimate is replaced by check-the-containment.

The worked exhibit is a block spin, and one design point governs its reading: the coarse theory is declared independently of the fine one. Had its presentation been defined as the coarsened fine family, coarse-graining would commute with compilation on the nose — the strict square is a landed theorem (`pushforward_ofPresentation`, module `QtslAux.QuantumCoarseGraining`) — and containment would hold at tolerance zero, exhibiting exact lumping rather than truncation; the effective-theory content is precisely that the declared coarse theory is simpler than the pushforward and provably within the band. Stated self-contained, at exam scale:

**Problem 12.18** (the block-spin containment). Data on the section's running two qubits, records the four joint computational-basis projectors, indexed by pairs of bits. The *fine* state face is the rational segment between the basis state at  $(0, 1)$  and the two-branch state with amplitudes  $\frac{24}{25}$  and  $-\frac{7}{25}$  on  $(0, 0)$  and  $(1, 1)$  — Born rows  $(0, 1, 0, 0)$  and  $(\frac{576}{625}, 0, 0, \frac{49}{625})$  respectively — a mixture-closed set, as the credal square requires; the fine mechanism is the reflection exchanging those two branch vectors, unitary, involutive, and rational-entried because  $7-24-25$  is a Pythagorean triple. The coarse-graining  $f$  keeps the first bit. The *coarse* theory, declared independently: records the two block-spin projectors (the fiber sums of the fine family), state face a single product state with row  $(1, 0)$ , mechanism the identity kernel — the block spin is 0 and nothing evolves. Show: (i) the segment member at parameter  $t$  pushes forward to the row  $(t + (1-t)\frac{576}{625}, (1-t)\frac{49}{625})$ , and containment holds at tolerance  $\frac{49}{625}$ ; (ii) the band is attained — any certifying tolerance is at least  $\frac{49}{625}$  — so containment at  $\varepsilon = 0$  is refuted: the truncation is genuine; (iii) on the mechanism's two-point orbit, the one-step defect — the distance between coarse-graining after one fine step and one coarse step after coarse-graining — equals  $\frac{49}{625}$  at both points, and every horizon  $k$  is then valid within  $k \cdot \frac{49}{625}$ ; (iv) the further merge to a single total record is contained at tolerance zero, and the composite fine-to-block-to-total containment holds at  $\frac{49}{625} + 0$ : budgets add along a tower.

*Sketch.* (i) The pushforward sums fibers and is affine in  $t$ : the first extreme pushes to  $(1, 0)$ , at distance zero, the second to  $(\frac{576}{625}, \frac{49}{625})$ , at distance  $\frac{49}{625}$ , and the member at  $t$  to distance  $(1-t)\frac{49}{625}$ . (ii) The coarse theory offers exactly one row, so the second extreme forces  $\varepsilon \geq \frac{49}{625}$  on any certificate, and a positive rational is not zero. (iii) The reflection swaps the two extreme states, so the pushed rows exchange  $(1, 0)$  and  $(\frac{576}{625}, \frac{49}{625})$  while the identity kernel holds each fixed; both defects are the distance already computed. For the horizon, the coarse step is a row-stochastic kernel, and such kernels contract total variation, so the  $k$ -step distance regroups through the triangle inequality into  $k$  one-step defects. (iv) Every state prices the single total record at trace one, so fine and coarse push to the same point row; merging twice is one merge along the composite map, and the triangle inequality adds the two budgets.



In the universe's terms: a refinement relation between two presentations, whose content at the credal stack is inclusion of law sets after inflation — so verdicts transfer across it with an  $\varepsilon$  margin, in the conservative direction. The vocabulary and the exhibit are mechanized, sorry-free, in the same module family as the squares (module `SuFunctor.Adequacy.QuantumContainment`): the distance is `tvDistQ`, with the metric lemmas and `tvDistQ_eq_zero_iff`; the inflation is `credalInflate`, monotone in the tolerance and in the set and commuting with the union-of-singletons form the credal face compiles; the statement is `Containment`, with monotonicity (`Containment.mono`) and additive vertical composition (`Containment.comp`); the row pushforward is the fiber sum (`SubPMF.map_wt_apply`), contracts the distance (`tvDistQ_map_le`), and measures the merged family (`recordShadow_coarsenFamily`). The exhibit's theorems are `uv_ir_containment`, `uv_ir_containment_lower`, and `not_uv_ir_containment_zero` for tasks (i) and (ii); `onestep_defect` and `horizon_bound`, instantiating the generic `lax_horizon` through the contraction `tvDistQ_stochAct_le`, for task (iii); and `ir_tot_containment` composing to `uv_tot_containment` for task (iv), whose zero leg consumes nothing but conservation of mass. The state face exercises the credal square's own convexity hypothesis (`uvSeg_mixClosed`, consumed by `uvLaws_mix_exercised`), and the two pushforward notions in the tree — Gram submatrix at measure level, fiber sum at row level — agree on the exhibit (`pushforward_probRow_bridge`). The qualitative half of the tower rides along: coarser is more classical, in the monotone sense that off-diagonal mass is antitone along tower legs (`eftTower_offDiagMass_antitone`) — though on this exhibit the dividend is degenerate, since projective records decohere every branch at the fine stage already (`eftStage_classical`); the strict decoherence drop lives on the library's three-stage diamond tower, cited rather than re-proven.

The fences, in the section's own manner. State-space compression, once the first fence here, is discharged at the same finite grade: the block spin whose state space genuinely shrinks lands as an isometry stratum and a trace-preserving Kraus channel with the compression/record-shadow interchange proved channel-wide (`recordShadow_kchanApply`); the partial trace acquires an explicit Kraus certificate (`kchanApply_ptrKraus_eq_ptr`), and the containment certificate transfers verbatim across the compression — the same declared band, still attained, still refuted at zero tolerance (`uv_smallIR_containment`). What stays fenced there is any compression-ratio or optimality claim: which compressions exist or are efficient is producer-side search, never certificate content. No power-counting claim: it enters as producer guidance only, and the checker never consumes an estimate as if it were a theorem. No continuum word: genuine Wilsonian renormalization — loop integrals, continuum limits, scheme independence — and perturbative error control made rigorous are research mathematics on the far side of this fence, and the literature's own error control is perturbative through and through, which is why the mechanized entry point is the finite stratum, where the error is one exact rational and there is no regularization scheme to hide in. What the fences protect is a modest, checkable form of the doctrine the literature itself teaches — every theory in this catalogue is somebody's effective theory [197], and certified budgets add along the descent.

**Status.** Semantics is the strong leg, and unusually strong for an entry this young: the kinematic mathematics (section 9.2) and both adequacy faces are mechanized end to end, with the negative witnesses — dropped positivity, dropped completeness, dropped mixture-closure — carried as theorems beside the positive ones. The refinement statement ships at the same grade — band proved, band attained, zero tolerance refuted — so the mechanized column runs both statement-kinds, System-to-semantics and System-to-System. Representation is design work: no producer surface exists at this preview, and the normative row grain is the adequacy module's own rows structure (`QNetRows`: sparse outcome rows over the record index; well-formedness as nonnegativity plus mass at most one; completeness deliberately separate), which a future declaration vocabulary and producer surface will mirror rather than reinvent. Certification: nothing shipped; the one genuinely open producer-side design point is exact positive-semidefiniteness certification over  $\mathbb{Q}[i]$ , where an exact  $LDL^*$  witness settles the generic case and the boundary-rank convention needs fixing before the first certificate does. Compositional closure: this version is a closed former, with no wiring interface, and the omission is argued from the frame rather than bare-declared — the physically native composition operations are spacelike tensor composition of nets, the named successor, a Composition Patterns sibling whose side condition is Einstein causality; and subregion embedding, which the site already carries internally. The deterministic-PDE entry set the precedent of landing former-level ahead of its composition story, and this entry follows it.

## Computer Science Metanotation

as mechanized in the toolkit's Lean tree: this entry's ground truth is the presentation structure and compiled row grain of its adequacy development (lean/su-functor/, module SuFunctor.Adequacy.QuantumNet, over the fragment structure of QtslAux.QuantumAssembly), rather than a producer extraction. No producer surface exists for this entry at this preview; the API block is omitted rather than invented, and the formation rule below is typeset by hand from the mechanized structure's fields.

$$\frac{K : \text{SiteKit} \quad \overrightarrow{N : \langle \text{window}, \text{gens} \rangle \text{ algebra rows}} \quad M : \text{MatterData} \quad \overrightarrow{R : \langle \text{index}, \text{matrix} \rangle \text{ record rows}} \quad \sigma : \text{StateFace}}{\text{QNet}(K; N; M; R; \sigma) : \text{System}} \text{ S-QNET}$$

The site kit lists finitely many points with their chronology; diamonds, covers, and the ambient window are derived. Algebra rows present one generating family per window, entries in  $\mathbb{Q}[i]$ ; matter data comprises worldlines, jointly exhaustive factors, and a reversible mechanism; record rows live at the ambient window. The state face declares one state with its exact-rational Born row (the precise face) or a set of states, each member carried with its row witness (the credal face). The well-formedness obligations are the community's own axioms — isotony, Einstein causality, the time-slice property, record positivity and exhaustiveness, state positivity and normalization — consumed by the mechanized textbook corner as the fragment's fields.

## 12.26 Future Work

The frames collected here are the catalogue's frontier: no producer surface at this preview — no rows, no API, no certificate. Encodings for some of them are in progress, and will graduate their entries to full catalogue sections as producer surfaces land. They stay in the catalogue as prose entries precisely so the chapter cannot quietly shrink its scope, and each keeps its full four-leg grading; what they do not get is an invented formal block, because a metanotation statement with no mechanized ground truth underneath would be exactly the coverage-matrix-of-green-ticks failure this chapter refuses. The quantum nets of section 12.25 are deliberately not collected here: that entry's semantics and adequacy squares are mechanized end to end, and only its producer surface remains future work.

### 12.26.1 Open Games

**The frame.** Compositional game theory: players, strategy sets, and utilities in the lens-shaped composition structure whose entire point is that games compose.

**Profile.** Intended: the composition structure preserved as the representation's grain — a rendering that flattens an open game to one normal-form matrix would fail the representation leg by construction. Not yet designed in rows.

**Semantics.** Equilibrium semantics under composition; the research content of this row lives here.

**Status.** Nothing shipped; the thesis's own bimatrix analysis is a *use* of game theory, not a subsumption witness, and we decline to count it. The concrete entry road is the certification leg: for polynomial games,  $\varepsilon$ -best-response inequalities are polynomial inequalities over strategy boxes — exactly the shape the kernel decides — so a certified  $\varepsilon$ -Nash artefact is buildable ahead of the representation and semantics legs that carry the real research weight. Graded absent until any leg ships.

### 12.26.2 Mean-Field Games

**The frame.** A representative agent coupled to the distribution of a population of itself; equilibria are forward-backward systems (Hamilton–Jacobi–Bellman backward, Fokker–Planck forward).

**Profile.** Intended: a representative-agent mechanism coupled to a population-distribution state. Not designed in rows.

**Semantics.** The equilibrium system itself; in the master-equation formulation, a PDE system — so the road runs through section 12.5, whose beachhead changes this row’s feasibility without yet shortening it much.

**Status.** Nothing shipped; this cell was not-covered in the 2023 SDCPN grading too, and it remains the list’s structurally hardest entry. The first tractable question is narrower than the frame: certified  $\varepsilon$ -consistency of a *given* candidate equilibrium pair, both components polynomial, may be reachable well before the frame itself. Graded absent.

### 12.26.3 Score-Based Generative Models

**The frame.** Diffusion models read as dynamics: a learned score field is the drift of a reverse-time stochastic differential equation, and the generative process is that equation run from noise.

**Profile.** Intended: an SDE mechanism (section 12.7’s profile) whose drift is a neural network, carried by content-addressed reference with piecewise-linear structure exposed to the kernel the same way the discrete campaigns already exercise.

**Semantics.** The generative path measure, through the same staged generator bridges as section 12.7.

**Status.** Nothing shipped for the frame as such — but this row is a *composite* gap, not a novel one: the multidimensional diffusion machinery is in place, the network-drift representation is in place elsewhere in the stack, and what remains is their composition. Graded absent, with its cost inherited almost entirely from parts that already exist.

## Systems: Composition Patterns

DRAFT

This chapter is the second wing of the System Catalogue — the first, chapter 12, inventories what a single mechanism can be; this one inventories how Systems combine. A specification is only as compositional as the model underneath it, so the System layer ships composition as a small, closed inventory of *patterns*: each is a syntactic row family with a typing rule, a semantics, and an explicit account of which certificates transport across the composite boundary and which are *refused*. The refusals carry real content: per-component stability certificates must not compose to composite stability, because that inference is false (re-entrant lines destabilize below capacity under plausible dispatch policies — Kumar–Seidman and kin [145]), and the pattern inventory encodes the refusal structurally rather than leaving it to the modeller’s vigilance.

A pattern, throughout, is a *usage of the wiring stratum plus its transport rules* — the row-level counterpart of composition by wiring diagram in operadic treatments of dynamical systems [162]. The composition data are ordinary rows — the manifest’s own composite-stratum descriptors, or one of four dedicated pattern row families — and the transport rules that move certificates across a composite boundary are rules of the Rule Catalogue, not machinery of this chapter. Each entry below states the pattern in prose (composition data, transport, witness status), then closes with the same two blocks as every System Catalogue entry. The *Computer Science Metanotation* block gives the entry’s *term former*: an inference rule whose premises are the formation data the pattern builders consume and refuse on, with their sorts, and whose conclusion is a primitive judgment at the sort System — or at the pattern’s own sort, where the mechanized constructor forms a pattern row citing its subjects rather than a System; the four dedicated row families do exactly that, and their rules say so. The *API* block carries producer surfaces extracted from the toolkit sources at every regeneration. Two reading conventions, stated once. First, the wiring-stratum rules display a *single* pattern row in the composite’s row slot: a composite carries any number of rows across the families, with each row’s premises checked independently, and the head form elides the ambient rows. Second, as in the first wing, optional formation data (marked ?) are consumed without appearing in the head form. The first four entries — the three wiring modes and nesting — are the composition machinery that the model-type chapter’s fourth grading leg, compositional closure, checks frames against; the remaining formal entries are instruments with landed witnesses. Two entries near the end withhold their formal blocks, by the same refusal that governs the first wing’s frontier entries: event ports, whose primitive’s producer surface has already landed, and state sharing, reserved outright. Between them sits supervision, whose semantic tier is mechanized ahead of its producer surface — its gating uses an arc kind the inventory already carries. The chapter closes with an entry that records a closure property rather than a pattern.

The composition tier underneath this inventory is mechanized, sorry-free, in the toolkit’s Lean tree (`lean/su-functor/`). The semantic universe’s decoration functor carries its compositor with every coherence obligation proved — transport functoriality, compositor naturality, associativity, and both unit laws, on the nose modulo the base’s definitional transports (module `SuFunctor.SingleSiteSemanticUniverse`) — and the possibilistic inclusion of hypernets composes through it exactly, with no determinism hypothesis (`SuFunctor.HypernetInclusion`). On the content-addressed wire stratum the calculus is strict, and failure is part of the account (module `SuFunctor.HypernetComposition`): composition of aligned interfaces is associative and unital on the nose (`AlignedLoose.seq_assoc` — hierarchy pays no coherence tax, which is the semantic face of the nesting entry’s freeness); components with equal semantic interfaces are interchangeable in any composite (`austereDnf_subst` — the congruence property every entry below presupposes when it speaks of transport across a boundary); an infeasible component makes its composite infeasible (`admRegion_append_empty_left`, iterating along the strict associativity above), while the converse fails by a mechanized witness (`share_contradiction`); and renaming a composite is the composite of the renamings (`theta_rename_seq`), so a certificate transported across a renamed boundary composes to the transported composite. What follows is therefore inventory over a proved foundation, and each entry’s transport story

stands on that tier rather than re-arguing it.

## 13.1 Parameter Setting

One system’s output fixes another’s parameter: an input wire bound to a parameter appearing in the consumer’s field polynomials. This is the “set” mode of the wiring stratum — directed by construction, so the row demands its from endpoint resolve to a declared output and its to endpoint to a declared input wherever the component’s interface is declared. Transport is the assume–guarantee square [38]: the wiring is certified for spec-to-spec entailment with slack, the leaves are ordinary sequents, and the slack arithmetic is checker-recomputed. **Witnessed** — this is the mode behind the inventory’s one cross-frame composition witness (the ODE plant wired to a discrete controller of section 12.4, every mutant of the composite refusing), and behind the inventory’s deliberate asymmetry: one mode witnessed, the rest below graded only as far as their evidence reaches.

### Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum’s own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former’s mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

$$\frac{S_1, \dots, S_k : \text{System} \quad \text{from, to : PortRef into the roster, from} \neq \text{to} \quad \text{dir(from) = output} \quad \text{dir(to) = input}}{\text{Composite}(S_1, \dots, S_k; \langle \text{set, from, to} \rangle) : \text{System}} \text{ S-SET}$$

PortRef =  $\langle \text{component, wire} \rangle$  addresses the roster by index. The direction premises resolve against declared *mechanism* interfaces; an endpoint whose component declares none (a composite; a mechanism without an interface binding) passes to the checker’s replay.

### API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

#### TypeScript ---

The pattern-level TypeScript binding is not yet arranged: the manifest-level wiring descriptors (`WiringRow`, `SwitchEdge`, `ScheduleAssumption`) already ride the TS system namespace (`system.composite`), while the pattern builders and the four pattern row families are Rust-only at this preview.

#### Rust `mtk_catalogue::compose::CompositeBuilder::set`

signature — `crates/mtk-catalogue/src/compose/builder.rs`

```
pub fn set(mut self, from: PortRef, to: PortRef) -> Self
pub fn build(self) -> Result<SystemManifest, Refusal>
```

construction — `crates/mtk-catalogue/tests/compose_common/mod.rs`

```

CompositeBuilder::new()
  .component(plant())
  .component(controller())
  .set(port(1, 1), port(0, 0))
  .build()
  .expect("set endpoints resolve against the declared interfaces")
  .named("set-loop")

```

## 13.2 Variable Sharing

Two systems read identified wires: the “share” mode identifies its two endpoints as one variable, with direction consumed by the causality check rather than declared in the row (stochastic wires remain directed-only at this preview). Transport: the same assume-guarantee square; the admissible-tier anchor is the small-gain arithmetic witness — the certified composite in which cross-supplies cancel identically. **Status:** machinery staged at the rule level; no heterogeneous share witness has landed yet. The mode’s *semantics*, by contrast, is already theorem-grade: admissibility of an identified composite is the conjunction of the parts’ admissibilities, mechanized with no determinism hypothesis at the catalogue stratum (`glue_admissible_iff`) and the semantic-universe stratum (`admRegion_append`, `theta_comp`); and identification is exactly where feasibility fails to compose — the minimal mechanized witness is a pair of individually feasible constant sources demanding different values of one shared variable (`share_feasible`, `share_contradiction`, module `SuFunctor.HypernetComposition`), while infeasibility, once present, propagates outward through further wiring (`admRegion_append_empty_left`).

### Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum’s own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former’s mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

$$\frac{S_1, \dots, S_k : \text{System} \quad \text{from, to : PortRef into the roster, declared where an interface exists} \quad \text{from} \neq \text{to}}{\text{Composite}(S_1, \dots, S_k; \langle \text{share, from, to} \rangle) : \text{System}} \text{ S-SHARE}$$

The row identifies rather than assigns, which is why, unlike S-SET, no direction premise appears: either side may be an input or an output, and the two wires denote one variable of the composite.

### API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

#### TypeScript ---

The pattern-level TypeScript binding is not yet arranged: the manifest-level wiring descriptors (`WiringRow`, `SwitchEdge`, `ScheduleAssumption`) already ride the TS system namespace (`system.composite`), while the pattern builders and the four pattern row families are Rust-only at this preview.

**Rust** `mtk_catalogue::compose::CompositeBuilder::share`

signature — `crates/mtk-catalogue/src/compose/builder.rs`



```
pub fn share(mut self, from: PortRef, to: PortRef) -> Self

construction — crates/mtk-catalogue/tests/compose_common/mod.rs

CompositeBuilder::new()
  .component(plant())
  .component(controller())
  .share(port(0, 1), port(1, 0))
  .build()
  .expect("share endpoints resolve against the declared interfaces")
  .named("shared-bus")
```

### 13.3 Mode Switching

A supervisory edge hands the active role between components. Switch edges live at the composite stratum in three species: guarded (guard content at the composite level), rate (stochastic hand-off at a state-dependent intensity — the species that makes switched diffusions composable), and forced (domain-exit at the composite stratum). Transport: the switch-composition rule, staged at the rule level with per-profile soundness lemmas in the inventory. **Status:** rule staged, witnesses pending; the static frames (the graphical-model rows of chapter 12) carry *inapplicability* arguments instead — a frame with no internal time axis has nothing for a supervisory switch to sequence — and that escape is earned per-frame by an argument from the frame’s own semantics (section 12.20 states it), never by a bare declaration.

#### Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum’s own (crates/mtk-catalogue/src/system/compose.rs and the pattern row families of crates/mtk-catalogue/src/compose/), and the former’s mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

$$\frac{S_1, \dots, S_k : \text{System} \quad i, j < k, i \neq j \quad \sigma : \text{species}}{\text{Composite}(S_1, \dots, S_k; \langle i \rightarrow j, \sigma \rangle) : \text{System}} \text{ S-SWITCH}$$

$$\sigma ::= \text{guarded}(\overrightarrow{\text{constraint}}) \mid \text{rate}(\text{poly}) \mid \text{forced}$$

$i \rightarrow j$  names roster indices — the component handing off and the component receiving — and the  $i \neq j$  premise is the no-self-switch refusal. The rate polynomial is exact-rational, like every field polynomial in the catalogue.

#### API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named fx\_\* or f.\* are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

#### TypeScript ---

The pattern-level TypeScript binding is not yet arranged: the manifest-level wiring descriptors (WiringRow, SwitchEdge, ScheduleAssumption) already ride the TS system namespace (system.composite), while the pattern builders and the four pattern row families are Rust-only at this preview.

**Rust** `mtk_catalogue::compose::CompositeBuilder::switch_guarded`signature — `crates/mtk-catalogue/src/compose/builder.rs`

```
pub fn switch_guarded(
    mut self,
    from_component: u32,
    to_component: u32,
    guard: Vec<DomainConstraint>,
) -> Self

pub fn switch_rate(mut self, from_component: u32, to_component: u32, rate: Poly) -> Self

pub fn switch_forced(mut self, from_component: u32, to_component: u32) -> Self
```

construction — `crates/mtk-catalogue/tests/compose_common/mod.rs`

```
let too_hot = DomainConstraint {
    poly: mono(1, 0, 1, r(1, 1)),
    op: "ge".to_owned(),
    rhs: r(100, 1),
};
CompositeBuilder::new()
    .component(thermal("heater", r(3, 1)))
    .component(thermal("cooler", r(-1, 10)))
    .switch_guarded(0, 1, vec![too_hot])
    .switch_rate(1, 0, constant(1, r(1, 20)))
    .switch_forced(0, 1)
    .build()
    .expect("switch edges cite roster components")
    .named("supervisor")
```

## 13.4 Nesting

Hierarchical composition is free in the theory: a component of a composite is any *System* — a mechanism or a built composite — so composition terms nest by construction, and no dedicated descriptor exists to get out of sync with the term structure. The flatten rule guarantees the discipline’s point: composites add structure and scalability, never expressiveness. Wire-level containment resolves against declared *mechanism* interfaces, and a composite component’s wires pass to the checker’s replay (composites declare no export map at this preview). **Status:** structural (the construction cannot fail to nest), but unwitnessed as a certified heterogeneous artefact; hierarchical control is the named first candidate.

## Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum’s own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former’s mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

No former of its own — deliberately. Nesting is the recursion already present in the core grammar: COMPOSITE’s premises (section 12.1) quantify over *System*, which a built composite inhabits exactly as a mechanism does, so `Composite(..., Composite(...), ...)` is well-formed by the rules already stated — cited here, not restated. A dedicated nesting rule would exist only to drift from the term grammar it duplicates.

## API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s

shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript ---

The pattern-level TypeScript binding is not yet arranged: the manifest-level wiring descriptors (`WiringRow`, `SwitchEdge`, `ScheduleAssumption`) already ride the TS system namespace (`system.composite`), while the pattern builders and the four pattern row families are Rust-only at this preview.

**Rust** `mtk_catalogue::compose::CompositeBuilder::component`

signature — `crates/mtk-catalogue/src/compose/builder.rs`

```
| pub fn component(mut self, sys: SystemManifest) -> Self
```

construction — `crates/mtk-catalogue/tests/compose_common/mod.rs`

```
| CompositeBuilder::new()
|   .component(set_composite())
|   .component(thermal("watchdog", r(-1, 100)))
|   .switch_forced(1, 0)
|   .build()
|   .expect("nesting is structural")
|   .named("hierarchical")
```

Nesting is structural: `component` accepts any manifest — a mechanism or a built composite — so composition terms nest by construction; no dedicated descriptor exists.

## 13.5 The Hybrid-Time Pattern

The pattern that discharges multi-clock composition in practice: put the discrete component's tick on a timer mechanism, prove the component's rows schedule-generically — per-tick, in its own counter — and carry the schedule class (period bounds, jitter bracket) in the *assumption slot* of the composition square. One value function, one cover, both time semantics — with the price of jitter certified, not assumed. **Witnessed:** the lander free-flight pair of section 12.2 (the same funnel under exact tick and jitter bracket from one table and one cover, jitter costing a certified factor 2.66 in ticks-to-funnel, with two exact infeasibility theorems marking where full dispersion is impossible), checked on an independent lane; the pattern text is the staging-independent recipe. The square behind the recipe is machine-checked, sorry-free, in the toolkit's Lean tree (module `SuFunctor.Adequacy.HybridTime`): compiling the rows onto the coarse clock, versus reading the per-tick semantics back through the declared clock morphism, the walled composite sits on the weak side in general (`hybridTime_square_subset`); the two readings agree whenever the transport relation is a function graph — the pattern's own tick seam is one (`tickSeam_hybridTime_square_eq`) — and at a jittered-coarse seam the inclusion is strict, which is why the assumption slot, and nothing else, carries the price of jitter.

### Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum's own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former's mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

$$\frac{S_1, \dots, S_k : \text{System} \quad i < k \quad \tau : \text{TickDecl}}{\text{Composite}(S_1, \dots, S_k; \langle i, \tau \rangle) : \text{System}} \text{S-HYBTIME}$$

At most one assumption per component — a duplicate is refused with both declarations serialized. `TickDecl` is section 12.2's grammar, here in the assumption slot of the composition square rather than on the mechanism.

## API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript ---

The pattern-level TypeScript binding is not yet arranged: the manifest-level wiring descriptors (`WiringRow`, `SwitchEdge`, `ScheduleAssumption`) already ride the TS system namespace (`system.composite`), while the pattern builders and the four pattern row families are Rust-only at this preview.

### Rust `mtk_catalogue::compose::CompositeBuilder::schedule`

signature — `crates/mtk-catalogue/src/compose/builder.rs`

```
| pub fn schedule(mut self, component: u32, tick: TickDecl) -> Self
```

construction — `crates/mtk-catalogue/tests/compose_common/mod.rs`

```
CompositeBuilder::new()
    .component(plant())
    .component(controller())
    .share(port(0, 1), port(1, 0))
    .set(port(1, 1), port(0, 0))
    .schedule(
        1,
        TickDecl::Jitter {
            lo: r(9, 100),
            hi: r(11, 100),
        },
    )
    .build()
    .expect("schedule assumption cites a roster component")
    .named("multi-clock")
```

## 13.6 The Funnel Pattern

Transit plus absorbing tail: an outer transit certificate (reach the gate within a budget, worst-case) composed with an inner invariance certificate (the gate region is forward-invariant and its holds are certified), citing the inner region as the outer target through the tower discipline. This is the certified shape of approximate dynamic programming with a terminal set — the outer table a value-supersolution on a cell graph, the inner tail the polynomial telescope. **Witnessed twice:** the pendulum swing-up row (a 24,467-cell supersolution, capture within 139 steps,  $\text{return} \geq -619.99$ , composed with the upright-hold invariance as absorbing tail) and the lander attitude funnel. The sort stated in the block below is forced, not chosen: a funnel that collapsed its parts into a one-slot composite could not be a lawful compositor, because collapsing composition breaks the unit law — a machine-checked refusal (`funnelComp_breaks_unit_coherence`, beside its no-tax counterpart `unitClass_transparent`, module `SuFunctor.Plain.Battery`) — so the pattern composes certificates about a System rather than forming one. The pattern's positive semantics is machine-checked in the same tree, sorry-free (module `SuFunctor.Adequacy.Funnel`): a funnel's composite satisfies the coupled transit-then-hold reading as a pure instantiation of the tower's composition theorem for budget sums with certified dips (`funnel_coupledSat_of_parts`, instantiating `coupledSat_comp_add_of_dips` with no new induction), and the tower-citation edge enters as an equation: assembly succeeds precisely when transit and hold cite one gate (`assemble_isSome_iff`), so a funnel whose parts disagree on the gate is unrepresentable rather than refused at replay.

## Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum's own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former's mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

$$\frac{\text{sys : SysRef} \quad \text{transit, hold : SpecRef} \quad \text{gate : RegionAtom carrying a certified family} \quad \text{budget} \in \mathbb{Q}_{>0}}{\text{Funnel(sys; transit; hold; gate; budget) : FunnelDecl}} \text{ S-FUNNEL}$$

The sort is forced: not a System former — the pattern composes two certificates about one subject System, cited by reference, and its row is the composition data the tower-citation edge consumes at replay. The gate premise excludes the bare box: the tower discipline cites *certified families* (sublevel or affine), and a box region carries none to place the citation on. The budget premise is the sign gate — a funnel with no transit budget asserts nothing about reaching the gate.

## API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

## TypeScript ---

The pattern-level TypeScript binding is not yet arranged: the manifest-level wiring descriptors (`WiringRow`, `SwitchEdge`, `ScheduleAssumption`) already ride the TS system namespace (`system.composite`), while the pattern builders and the four pattern row families are Rust-only at this preview.

## Rust `mtk_catalogue::compose::funnel`

signature — `crates/mtk-catalogue/src/compose/funnel.rs`

```
pub fn funnel(
    sys: SysRef,
    transit: SpecRef,
    hold: SpecRef,
    gate: RegionAtom,
    budget: WitRat,
) -> Result<FunnelDecl, Refusal>
```

construction — `crates/mtk-catalogue/tests/compose_common/mod.rs`

```
funnel(
    SysRef("sys:pendulum-swingup".to_owned()),
    SpecRef("spec:transit-to-gate".to_owned()),
    SpecRef("spec:upright-hold".to_owned()),
    RegionAtom::Sublevel {
        cert: CertFamRef("vfam:telescope".to_owned()),
        thr: r(1, 1),
    },
    r(139, 1),
)
.expect("fixture conforms to the funnel pattern")
```

## 13.7 The Twin-World Pattern

Two intervened copies of one model run against a single draw of the exogenous sources — the counterfactual composition. The pattern is *structural*: the causal-model profile owns one exogenous table per mechanism (an option, not a list — section 12.20.2), and the pattern row carries one base mechanism, so the decoupled independent-worlds reading is unrepresentable rather than discouraged. The certified decoupling probe prices exactly what that unrepresentability is worth (0 coupled against  $\geq 1/2048$  decoupled for the same controller). **Witnessed**: the counterfactual harm certificates in both truth directions (section 12.20.2). The pattern’s semantic bridge is machine-checked in the toolkit’s Lean tree (`lean/qlsl/`, module `Qtsl.ScmTwin`): twin-network substitution — the counterfactual event as an ordinary single-network event of the twin model, proven outright for acyclic models — together with the box-measure and union-bound certificate classes the harm certificates instantiate, and the worked instance’s landmark values as drift alarms; a sorry-free development, its axiom census the standard classical base.

### Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum’s own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former’s mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

$$\frac{\text{base : Mechanism} \quad \{ \text{equations, exogenous} \} \subseteq \text{pop}(\text{base}) \quad \text{base un-intervened} \quad \text{left, right : InterventionRow, each var defined by a base equation, no var twice per world}}{\text{TwinWorld}(\text{base; left; right}) : \text{TwinWorldDecl}} \quad \text{S-TWIN}$$

Two facts about the sort, both deliberate. The conclusion is a pattern row, not a System. And the base’s sort is Mechanism, not System: a composite base is unrepresentable, which is the one-exogenous-table structure stated as a premise sort — the coupling is *through* that table. A pre-intervened base is refused (world surgery is this pattern’s own left/right; anything else would smuggle a third world in). An empty intervention list is a legal world — the factual side of a harm query intervenes on one side only.

### API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

#### TypeScript ---

The pattern-level TypeScript binding is not yet arranged: the manifest-level wiring descriptors (`WiringRow`, `SwitchEdge`, `ScheduleAssumption`) already ride the TS system namespace (`system.composite`), while the pattern builders and the four pattern row families are Rust-only at this preview.

#### Rust `mtk_catalogue::compose::twin_world`

signature — `crates/mtk-catalogue/src/compose/twin.rs`

```
pub fn twin_world(
    base: Mechanism,
    left: Vec<InterventionRow>,
    right: Vec<InterventionRow>,
) -> Result<TwinWorldDecl, Refusal>
```



construction — crates/mtk-catalogue/tests/compose\_common/mod.rs

```
// Variable index space: endogenous x0, x1 then exogenous u (2).
let base = MechanismBuilder::new()
    .equation(0, mono(3, 2, 1, r(1, 1)))
    .equation(1, mono(3, 0, 1, r(1, 2)))
    .exogenous(ExogenousTable {
        vars: vec![ExoVar {
            id: 0,
            name: Some("noise".to_owned()),
            dist: Some(ExoDist::Box {
                region: BoxRegion {
                    axes: vec![[r(-1, 1), r(1, 1)]],
                },
            },
        )],
    })
    .finish();
twin_world(
    base,
    vec![InterventionRow {
        var: 1,
        replacement: Some(constant(3, r(1, 1))),
    }],
    vec![],
)
.expect("fixture conforms to the twin-world pattern")
```

## 13.8 The Stochastic-Wire Pattern

Systems coupled through probabilistic-dependency wires — one conditional-table wire per dependency: the frame family of the graphical-model row (section 12.20) and simultaneously a composition mode for exogenous blocks feeding dynamical frames. Single-system use is landed (the network rendering); **cross-system use awaits the marginal-compatibility discharge**, and the pattern row carries that residue explicitly as data: the citation field for the compatibility certificate is optional, and *empty means pending*. That this residue matters is itself machine-checked: the classical obstruction — pairwise-consistent marginal families with no joint law (Vorob'ev) — is mechanized over the subprobability stock (vorobev\_no\_joint, module SuFunctor.Adequacy.MarginalGlue: not even a mass-losing joint exists), records glue where laws do not (vorobev\_record\_glue), and an inconsistent family denotes the bottom credal object rather than crashing (credalCompile\_vorobevGlue) — so an absent  $\mu$  citation marks a pending composite, never a silently wrong one.

### Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum's own (crates/mtk-catalogue/src/system/compose.rs and the pattern row families of crates/mtk-catalogue/src/compose/), and the former's mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

$$\frac{S_1, \dots, S_k : \text{System} \quad W : \overrightarrow{\text{CrossWire}} \quad \mu? : \text{CertFamRef}}{\text{StochasticCoupling}(S_1, \dots, S_k; W) : \text{StochasticCouplingDecl}} \text{ S-STOCHWIRE}$$

$$\text{CrossWire} = \langle \text{sources} : \overrightarrow{\text{PortRef}}, \text{target} : \text{PortRef}, \text{rows} : \overrightarrow{\langle \text{given}, \text{outcome}, \text{prob} \rangle} \rangle$$

Per-wire formation data, checked wire by wire: sources resolve to outputs and the target to an input where an interface is declared (directed-only at this preview); the target is not among its own sources; each row's given arity equals the source count; an empty source list is a marginal (an exogenous block feeding a frame unconditionally); and — the exact-arithmetic gate — per-given probabilities sum to one, exactly, with

the computed sum as the refusal witness.  $\mu?$  is the marginal-compatibility citation, consumed without appearing in the head form; its absence *is* the pending residue, carried as data. The sort is deliberate: this is the one pattern that consumes Systems without forming one — the coupling is certified composition data over its roster, and the composite System awaits exactly the discharge  $\mu$  cites.

## API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript ---

The pattern-level TypeScript binding is not yet arranged: the manifest-level wiring descriptors (`WiringRow`, `SwitchEdge`, `ScheduleAssumption`) already ride the TS system namespace (`system.composite`), while the pattern builders and the four pattern row families are Rust-only at this preview.

### Rust `mtk_catalogue::compose::stochastic_coupling`

signature — `crates/mtk-catalogue/src/compose/stochwire.rs`

```
pub fn stochastic_coupling(
    components: Vec<SystemManifest>,
    wires: Vec<CrossWire>,
    marginal_compat: Option<CertFamRef>,
) -> Result<StochasticCouplingDecl, Refusal>
```

construction — `crates/mtk-catalogue/tests/compose_common/mod.rs`

```
let exo_block = {
    let m = MechanismBuilder::new()
        .interface(Interface {
            inputs: vec![],
            outputs: vec![wire(0, "w")],
        })
        .stochastic_wire(StochasticWire::Conditional {
            id: 0,
            sources: vec![],
            target: 0,
            rows: vec![
                CondRow {
                    given: vec![],
                    outcome: 0,
                    prob: r(1, 3),
                },
                CondRow {
                    given: vec![],
                    outcome: 1,
                    prob: r(2, 3),
                },
            ],
        })
        .finish();
    presets::Pdg::manifest(m)
        .expect("exogenous block conforms to the PDG profile")
        .named("weather")
};
let frame = {
    let m = MechanismBuilder::new()
        .state_var(0, Some("x"))
        .interface(Interface {
            inputs: vec![wire(0, "w")],
            outputs: vec![],
```

```

    })
    .drift(None, 0, mono(1, 0, 1, r(-1, 2)))
    .finish();
presets::Ode::manifest(m)
    .expect("frame conforms to the ODE profile")
    .named("field")
};
stochastic_coupling(
    vec![exo_block, frame],
    vec![CrossWire {
        sources: vec![port(0, 0)],
        target: port(1, 0),
        rows: vec![
            CondRow {
                given: vec![0],
                outcome: 0,
                prob: r(1, 3),
            },
            CondRow {
                given: vec![0],
                outcome: 1,
                prob: r(2, 3),
            },
            CondRow {
                given: vec![1],
                outcome: 0,
                prob: r(1, 1),
            },
        ],
    }],
    Some(CertFamRef("qmc:exo-frame".to_owned()))
)
.expect("fixture conforms to the stochastic-wire pattern")

```

## 13.9 Credal-Hull Certification

The two grades of open nondeterminism as one pattern. A system with unresolved choice — a policy slot, a scheduler slot — is certified either *at a declared resolution* (the resolution witness composed in; the closed object an ordinary citizen of the base frame's rows — the polling-automaton closure of section 12.15 is the landed instance) or *over the whole hull*: per-state-action row families asserting the generator inequality for *every* enabled choice, which covers deterministic, randomized (convexity — the rows are the simplex's extreme points), and history-dependent resolutions at once. The declared grade is the hull grade restricted to one vertex; the certified gap between them is real and priced (the demonic fixtures carry named breaking policies). Denotation: the credal lower expectation of the imprecise-semantics layer — and the semantics side of this pattern carries an adequacy receipt of its own: the demonic-face square of the decision-process section (section 12.13) mechanizes, at each finite horizon, that the compiled, policy-free presentation determines exactly the closed-loop law set over the memoryless randomized class, with the event-bound verdicts uniform over that class. **Witnessed**: a cliff-walk demonic pair (uniform bound with tightness witness), and the polling automaton's uniform re-certification, which thinned the declared grade's slack from  $5.13 \times 10^{-3}$  to eight parts per million without moving the bound.

## Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum's own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former's mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

$$\begin{array}{c}
\text{sys} : \text{SysRef} \quad m : \text{Mechanism the subject, } E(m) \neq \emptyset \\
\text{cert} : \text{CertFamRef} \quad \rho : \langle \text{state}, \text{action} \rangle, \text{ exactly one enabled action per open state} \\
\hline
\text{CredalHull}(\text{sys}; \text{cert}; \text{declared}(\rho)) : \text{CredalHullDecl} \quad \text{DECLARED}
\end{array}$$

$$\begin{array}{c}
\text{sys} : \text{SysRef} \quad m : \text{Mechanism the subject, } E(m) \neq \emptyset \\
\text{cert} : \text{CertFamRef} \quad h : \langle \text{state}, \text{action}, \text{driftBound} \rangle, \text{ exactly one row per pair of } E(m) \\
\hline
\text{CredalHull}(\text{sys}; \text{cert}; \text{quantified}(h)) : \text{CredalHullDecl} \quad \text{QUANTIFIED}
\end{array}$$

$E(m)$  is the subject's enabled choice set —  $\text{state} \mapsto \text{actions}$  read off its action-keyed outcome rows; a subject with no action-keyed structure is refused outright (no open choice — nothing for either grade to range over). The subject is consumed as formation data without appearing in the head form: it travels in the row by reference ( $\text{sys}$ ), and the coverage premises — one action per state for the vertex grade, since a partial resolution is not a vertex; one row per enabled pair for the hull grade, a gap's address serialized on refusal — are checked against the mechanism itself. Drift bounds are exact and sign-free (a supermartingale certificate's generator values are typically nonpositive); the generator-inequality *content* is the cited certificate family's business, recomputed at replay.

## API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named  $\text{fx}_*$  or  $\text{f}.*$  are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript ---

The pattern-level TypeScript binding is not yet arranged: the manifest-level wiring descriptors (`WiringRow`, `SwitchEdge`, `ScheduleAssumption`) already ride the TS system namespace (`system.composite`), while the pattern builders and the four pattern row families are Rust-only at this preview.

### Rust `mtk_catalogue::compose::quantified_hull`

signature — `crates/mtk-catalogue/src/compose/credal.rs`

```
pub fn declared_resolution(
    sys: SysRef,
    subject: &Mechanism,
    cert: CertFamRef,
    rows: Vec<ResolutionRow>,
) -> Result<CredalHullDecl, Refusal>

pub fn quantified_hull(
    sys: SysRef,
    subject: &Mechanism,
    cert: CertFamRef,
    rows: Vec<HullRow>,
) -> Result<CredalHullDecl, Refusal>
```

construction — `crates/mtk-catalogue/tests/compose_common/mod.rs`

```
let big = WitRat::new(
    -"1234567890123456789012345678901234567890"
    .parse::<BigInt>()
    .unwrap(),
    BigInt::from(7u8),
);
quantified_hull(
```

```

SysRef("sys:cliffwalk-2".to_owned()),
&mdp_subject(),
CertFamRef("vfam:cliffwalk-lower".to_owned()),
vec![
  HullRow {
    state: 0,
    action: 0,
    drift_bound: r(-1, 8),
  },
  HullRow {
    state: 0,
    action: 1,
    drift_bound: big,
  },
  HullRow {
    state: 1,
    action: 0,
    drift_bound: r(0, 1),
  },
],
1,
)
.expect("fixture conforms to the quantified grade")

```

The declared grade (one vertex of the hull) is the corpus's `credal_declared_decl` sibling fixture.

## 13.10 Event-Port Patterns

A finished design whose primitive now carries a producer surface. Jump synchronization — two mechanisms' events declared to fire together — is constructible today: the toolkit's composite stratum stores typed fusion rows (capability legs across components' population edges, one firing law per event under the single-home discipline below), the pattern-level builder gates them, the coloured-net decomposition emits them, and the API block at the end of this section quotes the conformance constructions in both languages. Everything downstream of the primitive — resource pools, broadcast, dispatch, rate reasoning at fused boundaries — still speaks in design tense, settled to typing-rule depth (for the fusion core and resource pools, to certificate-transport depth) and recorded here so the remaining vocabulary lands on a fixed target; until it lands, synchronized structure beyond raw fusion lives inside single mechanisms, a granularity cost priced in the design's deferral note with its re-pricing trigger named.

### 13.10.1 Fusion

The primitive. Mechanisms expose selected transitions as *event ports* on their interfaces; a fusion row names a set of bound ports across a wiring, and the named transitions fire as one joint jump — a rendezvous rather than a broadcast [44]. Enabling is conjunctive over the parties' local guards, evaluated on the pre-jump composite state (snapshot semantics: each party reads the snapshot and writes only its own state, so resets within a fusion commute by construction). Every port carries a *modality*: at most one party in a fusion is *active* — the timing owner, whose transition kind (rate law, forced guard, or immediate) times the joint jump — while passive parties contribute enabling conjuncts and resets only, and never fire outside a fusion. The design's worked example is operator 1 of the Mini-Fab model (the standard five-machine semiconductor benchmark, chapter 17), who serves both lithography and implantation: “load starts” is one jump across station and operator — machine ready  $\wedge$  lot present  $\wedge$  operator idle — with the operator's idle  $\rightarrow$  task reset riding the same jump. The fusion row has landed on the producer side, with the decomposition feeding it: the toolkit's coloured-net decomposition (section 12.18) derives, for each transition, a fused event over the incident places' population edges — the legs as (component, edge) pairs, enabling arcs riding as read legs, the one firing law on the row — and emits it as the serialized fusion row the composite stratum stores and the pattern builder gates.

### 13.10.2 Resource Pools

A packaged pattern over fusion, not a new primitive: a shared resource is a pool mechanism (a counter  $k \in \{0, \dots, C\}$ , or a one-place coloured pool) with passive *take/give* ports, and a resource-share row family definitionally expands into that mechanism plus fusion rows binding each user's seize and release transitions.

A crossing discipline (each user declares which of its modes count as *holding* the resource; seizes enter the held set, releases leave it, no other transition crosses) makes the pattern’s payoff a theorem rather than a convention — the *conservation law*: for a well-formed pool share, every jump of the composite preserves

$$k_{\text{pool}} + \sum_{u \in \text{users}} \#H_u = C,$$

where  $\#H_u$  counts user  $u$ ’s units in held modes. The invariant is a count row [172], free by construction; raw fusion rows can express the same model and can also silently break the invariant — the packaged rows cannot, and that refusal is what the package is for.

### 13.10.3 Broadcast Fan-Out

One sender, many receivers. The inventory deliberately carries no broadcast primitive; it carries the *fixed-roster reduction*: when the receiver roster is finite and statically known — which is every use the catalogue’s source models have produced — a broadcast is a finite family of rendezvous fusions, and optional-join broadcast (each receiver participates only when locally enabled) is a finite family indexed by the participating subsets. Genuinely dynamic rosters are priced and deferred with the rest of the stratum.

### 13.10.4 Dispatch: Control Through Selection

Fusion creates contention: two fusions sharing a passive party conflict, and the enabled-conflict set is derivable from the fusion rows alone. Absent further data the conflict resolves demonically [13] — sound, and exactly as weak as it sounds. A dispatch row installs a policy: a chooser over the conflict set, reading current aggregate observables. Certificates are dispatch-indexed, with one free lemma (any dispatched policy refines the demonic envelope, so demonic safety transports to every policy) and one structural refusal (recurrence-class conclusions never cross an undischarged boundary). The design’s worked example is first-come-first-served against setup-affinity dispatch on the Mini-Fab model: two policies over an identical composite, with per-policy certified throughput the target — the setup-affinity policy’s advantage being exactly the kind of claim that is only ever *per-policy*, since the demonic envelope proves neither. Dispatch is the *per-epoch* member of the inventory’s control trio; its persistent sibling — control through enablement, where a supervisor carves the enabled set itself rather than selecting within it — is the supervision pattern (section 13.11), and the two compose because they are disjoint by kind.

### 13.10.5 Delayed Trace and the Delay Media

Feedback with memory: a delayed-trace row [223] closes a composite’s output back onto its input through a delay, well-defined precisely because the delay breaks instantaneous self-reference [53]. Delays themselves are first-class composition elements [154] in two species with a shared degenerate case — an event species (a one-place coloured-token place whose tokens carry a time-to-go coordinate) and a signal species (a transport medium: pure advection over the delay interval), both degenerating, when time is discrete and the delay is one step, to the *tick register*: the unit delay, which is also the minimal medium a delayed trace needs. The family exists so that delays compose like everything else — a delay is a mechanism with ports, not an annotation on a wire. For the signal species in continuous time, the semantics tier is already receipted: the delay-line adequacy square of the delay-differential-equations section (section 12.6) mechanizes exactly this composite — an ODE cell closed through the transport medium — so the pattern arrives with its continuous-time semantics a theorem and its producer surface landed: the delayed trace is a wiring, and the toolkit’s conformance corpus constructs the same composite — cell, line, and the two trace wires — through the composition surface (a dedicated delayed-trace row stays deliberately absent; the wiring is the pattern). The design’s flagship construction is the partially observable decision process: environment mechanism, observation channel, and a belief-tracking controller closed through a delayed trace — partial observability as a *composition* pattern rather than a new model class (section 12.14 grades the frame) — with the first certificate, a lower bound on achieved value for the classic two-door problem living on the belief simplex, already exercised in the workshop.



### 13.10.6 Event-Rate Assume/Guarantee

Rate reasoning across fused boundaries: an event-rate atom (the specification catalogue’s envelope pair, staged — section 14.6.4) bounds the firing rate of a shared event, a mechanism’s guarantee about its side of a fusion becomes its neighbour’s assumption, and assume/guarantee chains propagate rate bounds through a composite without unfolding it [28]. The delay place enters as a *rate-preserving element* — tokens are conserved, so the departure rate equals the arrival rate shifted by the delay, and a rate assumption crosses a delay medium unchanged. This is the pattern that would let throughput claims localize: bound the bottleneck’s service rate, and let rate-preservation carry the bound to the boundary where the specification states it.

## Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum’s own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former’s mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

*Producer-landed, theory-pending*: the fusion row family now has a builder underneath (the API block below quotes it), but no term former is stated ahead of its certificate-schema vocabulary — the formal block enters with the event-port interface stratum’s staging, and asserting a rule now, with no loader beneath it to refuse on its premises, would be exactly the invented formal block the first wing’s frontier entries refuse.

## API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

The fusion primitive’s producer surface: a fusion row names capability legs (bag edges across a composite’s components, read legs included) that fire as one joint jump, with the event’s one firing law under the single-home discipline — exactly one active leg (its edge’s own law times the jump) xor a row-resident law over all-passive legs; both double-home and all-passive-lawless declarations refuse at build, along with leg resolution, membership (active-never-optional; a row-law fusion keeps a mandatory leg), and the delay basis (selection blocks per consuming/read leg, within-leg slot-exchange symmetry). The packaged patterns over fusion — resource pools, dispatch, event-rate atoms — remain design-tense and carry no API here.

**TypeScript** `compose.composite / system.composite (fusions)`

signature — `ts/src/system.ts`

```
/** S-EVENTPORT: one capability leg of a fused event — `edge` of
 * `components[component]`'s Bag self-loop roster. A leg whose edge
 * is `synced` is passive; an edge carrying its own local law is the
 * fusion's active leg (the timing owner). `membership` defaults to
 * `"mandatory"`; an `"optional"` leg joins only when locally enabled
 * (maximal join). */
export interface FusionLeg {
  readonly component: number;
  readonly edge: number;
  readonly membership?: "mandatory" | "optional";
}

/** S-EVENTPORT: the row-resident firing law of an all-passive fusion
 * — delay (rate per unordered selection across the consuming/read
 * legs' blocks, in leg order), guard (forced; every consuming/read
```

```

    * leg has delta 1), or immediate (the eager case, spelled
    * explicitly). */
export type FusionLaw =
  | { readonly kind: "delay"; readonly rate: Polynomial }
  | { readonly kind: "guard"; readonly guard: ReadonlyArray<DomainConstraint> }
  | { readonly kind: "immediate" };

/** S-EVENTPORT: one event-port fusion row – the named capability
 * legs fire as one joint jump (rendezvous), with the event's one
 * firing law under the single-home discipline: exactly one active
 * leg (`law` absent) xor all legs passive with `law` on the row. */
export interface FusionRow {
  readonly event: number;
  readonly legs: ReadonlyArray<FusionLeg>;
  readonly law?: FusionLaw;
}

```

construction — ts/tests/compose-native.test.ts

```

// The transfer rendezvous with an operator: a token leaves
// `upstream` and enters `downstream` as ONE joint jump, with an
// operator token read (never consumed). All legs are passive, so
// the law rides the row: a delay rate over the derived selection
// basis – the consumed token's colour, then the read operator's.
return compose.unwrap(
  compose.composite({
    name: "transfer-rendezvous",
    components: [
      bagStation(sys, "upstream", "destroy", 2),
      bagStation(sys, "downstream", "create", 1),
      bagStation(sys, "operators", "read", 1),
    ],
    fusions: [
      {
        event: 0,
        legs: [
          { component: 0, edge: 0 },
          { component: 1, edge: 0 },
          { component: 2, edge: 0 },
        ],
        law: { kind: "delay", rate: mono(2, 0, 1, "1/10") },
      },
    ],
  }),
);

```

bagStation and mono are the conformance corpus's shared fixtures, defined beside the quoted lines; the quoted construction's emitted bytes are the committed compose\_fusion golden vector, byte-identical across both language surfaces.

## Rust mtk\_catalogue::compose::CompositeBuilder::fuse

signature — crates/mtk-catalogue/src/system/compose.rs

```

/// System Catalogue: one event-port fusion row (`S-EVENTPORT`) – a
/// set of capability legs across the composite's components firing
/// as one joint jump (rendezvous, not broadcast), plus the event's
/// one firing law under the single-home discipline: either
/// exactly one leg is active (its edge's own law times the joint
/// jump; `law` absent) xor all legs are passive and `law` is
/// present on the row. Both double-home and all-passive-lawless
/// declarations refuse at [CompositeBuilder::build].
///
/// Enabling is conjunctive over the legs' local guards on the
/// pre-jump snapshot; each leg writes only its own component's
/// state (snapshot semantics, so reset commutation within a fusion
/// is a theorem – kernel-side, like every transport rule here).
///
/// [CompositeBuilder::build]: crate::compose::CompositeBuilder::build
#[derive(Debug, Clone, PartialEq, Serialize, Deserialize)]
pub struct FusionRow {
  /// Event id (roster-unique among the composite's fusion rows; for
  /// a decomposition image, the source SDCPN transition id).

```

```

    pub event: u32,
    /// The fused capability legs.
    pub legs: Vec<FusionLeg>,
    /// The row-resident firing law – present iff every leg is
    /// passive (the single-home XOR).
    #[serde(default, skip_serializing_if = "Option::is_none")]
    pub law: Option<FusionLaw>,
}

```

construction — crates/mtk-catalogue/tests/compose\_builders.rs

```

// The transfer rendezvous: a token leaves `a` and enters `b` as
// ONE joint jump. Both capability legs are passive (`Synced`
// edges), so the event's one firing law rides the row – a delay
// rate over the derived selection basis (the consumed token's
// colour coordinates).
let sys = CompositeBuilder::new()
    .component(bag_station("a", BagEdgeKind::Destroy, 2))
    .component(bag_station("b", BagEdgeKind::Create, 1))
    .fuse(FusionRow {
        event: 0,
        legs: vec![leg(0, 0), leg(1, 0)],
        law: Some(FusionLaw::Delay {
            rate: mono(1, 0, 1, r(1, 10)),
        }),
    })
    .build()
    .expect("legs resolve; all passive; law on the row");

```

`bag_station` and `mono` are the corpus's shared fixtures (tests/compose\_common); `FusionLeg`/`FusionLaw` and the membership column are declared beside the quoted row type, and the builder verb (`fuse`) runs the well-formedness gates at build.

## 13.11 Supervision: Control Through Enablement

Partially observed supervisory control [165, 206] over the coloured-net frame: a *supervisor* that can switch *some* declared plant transitions between enabled and disabled — not all of them, and never by firing them itself — that sees only declared observable functions of parts of the plant's marking, and that springs into action only when subscribed events occur. The entry's organizing fact is that the control seam costs nothing, because the inventory already carries it: **the gate is a place, and the gating is an enabling arc**. The policy is an ordinary jump-only System that owns places; a gated transition carries an ordinary weight-1 enabling arc (read-no-consume) from its gate place; enabling is minting the gate token and disabling is draining it — both ordinary jumps of the policy itself. No new arc kind, wire class, or row family exists for this pattern to add: enabling arcs are first-class in the coloured-net vocabulary at every stratum (section 12.18's arc table and the three faces of its adequacy development), and the inventory's seams for shaping a *live* component's behaviour thereby form a trio at three tempos — parameter setting (continuous, section 13.1), *enablement* (persistent, this entry), dispatch (per-epoch, section 13.10); the supervisory *edges* of mode switching (section 13.3) sit outside the trio, handing the active role between components rather than steering an active one. Gates carve the enabled set; dispatch selects within what survives; uncovered freedom stays demonic.

What the packaged row genuinely adds is bookkeeping with one new census clause. The package declares the ownership split — which places and which update transitions are the policy's — and derives the gated roster from it (every read arc sourced at a policy place; row-level, decidable, never separately declared). Its well-formedness census closes the information channels: policy inputs factor through the declared observables, subscriptions are the declared events, and — the clause the place-rendering itself demands — *arc separation*: plant transitions touch policy places through the declared gating read arcs only, with ordinary arcs refused in both directions (a plant transition producing into a gate place would be plant-side control of the policy's state), and policy updates carry no arcs on plant places at all — the policy acts on the plant through enablement, full stop. Raw rows remain lawful with the census as admission check, as for the inventory's other packaged rows. Zero-order hold between wakes is not even a lemma under this rendering: read arcs cannot move tokens, so no plant firing can touch a gate, and token presence has no drift to have. Gate places carry a Boolean discipline by default — capacity 1, gates as switches — and the discipline is deliberately omissible:  $k$  tokens in a gate place multiply a gated race's intensity by exactly

$k$  (the ordered census’s falling factorial at the gate place), so integer throttles come free by *not* imposing it, while fractional modulation remains parameter setting on rate content. The capacity declaration is accordingly a *must* wherever the model means Boolean gates: without it, an accidental second token silently doubles a rate.

The semantics per gated species is read off the landed faces rather than added to them. A gated delay transition is race thinning: gate closed, the ordered census vanishes (pre-enabling was always inside the count); gate open, the survivor law is the integrated-rate law, by memorylessness. A gated immediate transition is already inside the cascade square: gate coordinates are marking coordinates, so the tangible/vanishing classification, the scheduler discipline, and the cascade-rank certificate quantify over them unmodified — and the policy’s own wake is a cascade step like any other, so one structural rank discharges non-Zeno for every enabling-respecting scheduler, policy updates included. The whole pattern lives strictly inside the monotone fragment: disabling by *removing* a token is not a zero-test, enabledness stays upward closed (surplus gate tokens never disable a step), and the inhibitor exclusion is untouched — supervision is the monotone-safe rendering of control. Closure is on the nose: the supervised composite — plant places plus policy places, gating arcs, fused wakes — is again a coloured net of the same class, so supervision is composition *inside* the model class, and the fixed-policy adequacy statement is the net square applied to the supervised composite, no pattern-specific vertical. Mechanized at the uncoloured stratum (`SuFunctor.Adequacy.StochasticPetriSupervise`): the gating lemmas, the decidable arc-separation census with refusal witnesses in both directions, the Boolean-discipline preservation, and supervised micro-instances whose controls pin the closed, open, and graded race intensities as values, drive a two-step cascade *through* the policy’s own mint, and consume the landed cascade adequacy square at the supervised composite as an instance.

The pattern is a deliberate fourfold weakening of the probabilistic model checker’s scheduler — the adversary that tools in the PRISM and Storm family resolve [114, 148] — and each axis has a formal cash-out, landed or staged as the Status note records:

| axis          | scheduler (limit case)          | supervised variant                                                                                                                                                                                                            |
|---------------|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| control scope | resolves all nondeterminism     | read arcs on a declared roster; races stay stochastic, uncovered conflicts stay demonic — the controllable/uncontrollable partition is roster data                                                                            |
| effect        | the choice <i>is</i> the firing | the policy’s action type is its own marking: mint and drain in places it owns; the plant’s races and cascades do the firing — and the composite is again a net of the same class, where a scheduler lives outside its model   |
| observation   | reads full state                | reads declared observables of declared parts; measurability transposed into typing, checked at formation — adaptedness then a theorem about every well-formed composite, staged with the observation legs (Status note below) |
| activation    | consulted at every epoch        | wakes only at subscribed events; hold between wakes is read-no-consume; decision epochs are a declared sub-point-process of the plant’s event stream, non-Zeno by the shared rank certificate                                 |

Certificate polarity splits cleanly over the policy family, and the split is the entry’s payoff. *Safety is free across the whole family*: monotone enabledness makes refinement monotone in the gated roster, every well-formed policy’s closed-loop behaviour is contained in the demonic gate-envelope, so one safety certificate against the gate-demon transports to every partially observing policy — the information constraint costs nothing at this polarity. *Liveness over the unrestricted family is unprovable by construction*: the empty policy — no updates, gates never minted — is well-formed, so deliberate idling is expressible, and throughput claims are false over the family wherever a gated transition is load-bearing. That expressiveness is the commission, not a defect: deliberate idling is exactly the capability dispatch lacks (a chooser must choose; a supervisor may refuse), and it is why liveness certificates are per-policy here, exactly as they are per-dispatch in the previous entry, Kumar–Seidman teeth included [145]. The “some transitions, not all” clause has the same certificate-side reading: a total roster would make safety-under-supervision vacuous of plant content and

family liveness never provable; partial rosters keep both polarities informative.

One refusal survives, and it is the pattern's certificate-soundness story. At formation, gating is admitted on *every* transition species — including boundary-forced guard transitions, whose semantics stays total when disabled: a guard transition that is not pre-enabled imposes no forcing, and the run continues past its boundary untouched. What must not be gated, on pain of certificate unsoundness rather than semantic undefinedness, is a transition serving as a model's *box-discharge witness*: the certificate layer quantifies colours over compact rational boxes, the discharge obligation is what keeps the process inside the box the rows quantify over, and its default witness is itself an exit-guard transition. Gate that transition off and the process exits its box with positive probability while the certificate's rows still quantify over it. The obligation therefore lives at the certificate stratum, where the discharge already does: a gated transition may not serve as a box-discharge witness unless the model re-discharges by invariance or by stopped-process reading — a row-level lint on the package roster, decidable, and binding however the arcs were wired, because it is checked on the composite's rows rather than on any declared seam. On the semiconductor-fab model (chapter 17) the lint draws exactly the right line: a *suppressible deadline* is directly representable — gate an interior maintenance-window transition and the model means what it says — while the shift-boundary transitions are ungatable not because they are calendars but because they are their timer boxes' own discharge witnesses.

**Status:** the gating semantics and the arc-separation census are mechanized, sorry-free, at the uncoloured stratum (the module above, over the statics square and both dynamics faces), with refusal witnesses on every census clause and the graded-throttle identity pinned as a value; the coloured packaged row — observation legs as record-lens readouts, wakes as passive optional-join fusions — stages with the event-port interface stratum (section 13.10), and until it lands, supervised composites are hand-wired through the census door. The demonic gate-envelope face — the family-level object that safety certificates ride — is mechanized over the credal-hull machinery (section 13.9) at the uncoloured stratum: the certified policy seat is a dependent-index family (each member carrying its own termination rank — no automaton-level certificate can be uniform across the seat), its closed-loop laws inhabit one envelope on the common marking space, safety verdicts transport uniformly over the seat at both polarities, the envelope is monotone in the gated roster, and deliberate idling is a member — the mechanized face of the polarity split above.

## Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum's own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former's mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

*Former pending its strata; census landed.* The packaged row consumes an observation interface and a subscription list whose sorts stage with the event-port interface stratum (section 13.10), so no term former is asserted ahead of them — but unlike the entries that refuse their blocks outright, this pattern's one genuinely new formation premise already has a loader to refuse on it: the arc-separation census is a decidable row-level predicate, mechanized with refusal witnesses in both directions (a plant write into a gate place; a policy arc on a plant place). The gate map itself adds no premise sort at all — a gate is a weight-1 enabling arc from a policy-owned place, an arc kind every stratum of the coloured-net development already carries — and the ownership split is package data, invisible to the wire vocabulary.

## API

Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit's conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus's shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

**TypeScript** ---

As for Rust: enters with the event-port interface stratum.

**Rust** ---

None yet, and the absence is a sequencing decision rather than a gap: supervision is a packaged pattern over fusion (event-triggered policy wakes) and enabling arcs (the gating), so its producer surface is sequenced behind the event-port interface stratum rather than built ahead of it. The pattern’s semantic tier is already mechanized in the toolkit’s Lean tree — the gating lemmas, the arc-separation census, and the supervised micro-instance with its cascade controls (module `SuFunctor.Adequacy.StochasticPetriSupervise`, over the statics square and both dynamics faces of the stochastic-Petri development) — so the builder, when it lands, arrives to a fixed semantic target.

## 13.12 State Sharing

The reserved axis: gluing along shared *state* — a jointly driven continuous variable, common noise — rather than signal wires. The designed semantics is the generator’s additivity over transitions sharing state,

$$\mathcal{A}_{\text{composite}} = \sum_i \mathcal{A}_i,$$

so certificates compose by summing generator contributions; the design warrant is recorded, and the axis is reserved rather than staged. Its first customers: common-noise mean-field composition, should that frontier entry of chapter 12 ever earn a producer surface, and the shared-place face of net composition — two nets glued along one common place — which is the one part of “nets of nets” that genuinely lives on this axis (the rest turned out not to need it; section 13.13).

### 13.12.1 Continuous State Sharing: Port-Hamiltonian Interconnection

The one pattern in the inventory’s design space that is not reducible to fusion: two mechanisms continuously driving one shared state have no events to synchronize — the content is drift superposition. The disciplined form is port-Hamiltonian interconnection: each mechanism carries an energy function and a power port (a paired effort/flow boundary variable), and composition couples ports power-conservingly, so the composite’s energy is the sum and energy-based certificates compose along the interconnection. The designed elimination has two species — additive, for flow-summed sharing, and an exact linear solve with a determinant-guard witness, for constraint coupling — and the guard is where the design pays for itself: the composite kinetic metric is a sum of pullbacks of positive-definite component inertias, so the division guards that must today be proven per model become structural consequences of the pattern. The design’s worked example is cartpole recomposed — cart and pole as separate Hamiltonian mechanisms coupled through a power-conserving pivot junction, recovering the standard monolithic dynamics as a theorem rather than an authoring choice. This design is in flight; this section records its intended shape.

## Computer Science Metanotation

as realized in the mechanized toolkit: the row vocabulary is the composite stratum’s own (`crates/mtk-catalogue/src/system/compose.rs` and the pattern row families of `crates/mtk-catalogue/src/compose/`), and the former’s mechanized ground truth is the builder signature quoted in the API block below, extracted at every regeneration. The formation rule, where the entry states one, is typeset by hand for notational quality, its premises the formation data the pattern builders consume and refuse on; transport soundness, tiering, and the causality/simultaneity side conditions are rule- and kernel-side, catalogued in the Rule Catalogue.

*Reserved:* no term former is stated ahead of the axis’s vocabulary. The design warrant — generator additivity, and the power-conserving junction rows of the port stratum — is recorded in the prose above, asserted as nothing until the axis stages.

## API



Producer surface of the mtk toolkit for this composition pattern: the TypeScript arrangement (npm package @maitria/mtk) and the Rust catalogue crate (crates/mtk-catalogue) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are extracted from the toolkit sources at every regeneration of this catalogue; each construction is quoted verbatim from the toolkit’s conformance corpus, code whose emitted certificate bytes are verified byte-identical against the committed golden vectors (helper values named `fx_*` or `f.*` are that corpus’s shared fixtures, defined beside the quoted lines). This API constructs and emits; it never checks — the checker re-derives every claim independently.

### TypeScript ---

As for Rust: reserved for a later revision of the producer surface.

### Rust ---

None by design at this preview: state-sharing / net-of-nets (gluing along shared state — places, common noise — rather than signal wires; the generator’s additivity over transitions as the designed semantics) is reserved for a later revision of the producer surface, with its design warrant recorded; SDCPN net-of-nets and common-noise mean-field composition are its first customers.

## 13.13 Nets of Nets

A closure property rather than a pattern, recorded as its own entry because the catalogue once reserved “nets of nets” as future vocabulary and most of the reservation dissolved under the term formers now stated. The population former is parametric in its inner System: S-BAG’s premise (section 12.17) is  $S : \text{System}$ , with nothing about  $S$  consulted beyond its sort. And the coloured-net former concludes at exactly that sort:  $\text{SDCPN}(P; T; A) : \text{System}$  (section 12.18). So a net whose tokens are themselves nets is one substitution — put a coloured-net System in the token-System slot where an ODE mechanism would ordinarily go, and the outer net’s places hold inner nets as tokens: hierarchy by instantiating a premise, not by any new row family. The base rung of the same ladder is already in the catalogue as the population entry’s identity anchor — a Bag of ODE tokens is the single-place, zero-arc coloured net, two presentations of one System — and each further rung is the same substitution one level up.

What this entry deliberately does not claim: the certification story across nesting depth. The well-formedness is free — that is the finding — but no shipped certificate yet exercises a population-of-nets carrier end to end, and the grading discipline of chapter 12 applies to the nested composite exactly as to any frame: depth adds obligations, not exemptions. What remains genuinely reserved is different in kind — two nets glued along a *shared place*, which is state sharing (section 13.12), not token colouring, and stays on that axis’s reservation.

No metanotation or API block: there is no former here to state — the entry’s formal content is two already-stated rules composing, and a dedicated block would only restate them.

# Specification Language

DRAFT

Every certificate in this stack ultimately concludes a claim of the form *this system satisfies this specification*. The Mathematical Models catalogue (chapter 12) is the normative inventory of the left-hand side of that judgment; this chapter is its right-hand side: the *spec language* — the temporal-logic proposition formers a modeller actually states requirements in — together with the record that packages a requirement for composition, the derived spellings that cover recurring claim species without new grammar, the rewrite layer that lets the *spec as stated* be the *spec as certified*, the deliberate absences, and the judgment kinds: the satisfaction-side sorts that certificates conclude into.

The presentation is semantics-first, and its form follows the rest of this book. The grammar is not given as productions: each former is a *deduction rule* in Computer Science Metanotation [225], concluding a formation judgment — premises above the line are the former’s well-formedness conditions, side conditions stand beside the rule, and what used to be checked into existence is now visible in the rule that brings the term into existence at all. The discipline pays twice. First, the deliberate absences of section 14.9 stop being policy prose: a fence is the *absence of a rule*, and several of the absences carry a semantic theorem as their warrant (section 14.2). Second, each formation rule is the natural index of its semantics: every former carries a *compile clause* into the spec storey of the semantic universe (chapter 9), an *adequacy obligation* relating the compile to the former’s intended reading — stated with its current status, machine-checked where it is machine-checked and named where it is owed — a *witness* from the certified corpus, and the producer *API* that constructs it. The record-level compile is a structural recursion over the formation derivation, so the adequacy of a whole specification is the fold of its rules’ cells (section 14.3).

**Why the grammar exists at all.** Before these formers were typed rows, two claim species — expected return and bounded always — lived in certificate-header *sidecars*: prose fields beside the proof rows, spelling claims the grammar could not. A claim that lives in a sidecar is a claim no checker re-derives and no composition rule can consume. The discipline this chapter enforces is the retirement of that disease: **a spec claim is a typed row**, and the rows’ schema has a normative home in this repository — theories/Qtsl.geolog, the theory encoding this chapter’s vocabulary as data, with its well-formedness and mutant battery beside it. Everything below exists so that the statement a reader sees is the statement the checker replayed.

The grammar is deliberately small — threshold atoms over observables, Boolean structure, probability, expectation, and level thresholds, run-functional accumulation, and the temporal operators (*always*, *always up to a deadline*, *reach by a deadline*, *until*, metered Streett acceptance) — and the catalogue’s completeness claim is empirical and checkable: **every claim species asserted by a certificate in the certified corpus appears in this chapter**, either as a former, as a derived spelling, or as a judgment kind — with the exercising artefact named. A claim species with no catalogue entry is a defect of the catalogue, never a licensed side channel: the admission rule is that a new species gets its former, with a named kernel and a named semantics obligation, when its first certificate lane arrives.

## 14.1 The Compile Target: Spec Classes of the Semantic Universe

What a specification denotes is fixed once, in the semantic universe, and this chapter compiles into it rather than re-deriving it. The stock this chapter consumes (chapter 9, section 9.7): a *spec atom* is a safety face together with a finite set of liveness demands; its class in the hyperspace is a *crescent* — one closed containment face cut by finitely many open escape faces, a locally closed set — and a compound specification in disjunctive normal form denotes the union of its disjuncts’ crescents, a *constructible* class.

Satisfaction is the fibre order itself: a system's belief (level-of-inconsistency functional; chapter 9) satisfies a spec when the spec's belief lies below it (`SuFunctor.Universe.Sat`), and the crisp judgment against a class has two faces — *safety* is wall domination, the system's permitted region contained in the demanded one, and *liveness* is a dip, some admitted state permitted at a finite level of inconsistency under each demand (`wallOn_le_iff`, `dip_top_iff`). Where the model leaves openness — demonic inputs, worst-case initial conditions, imprecise kernels — the judgment reads against a *set* of behavioural laws through the credal envelope faces (`SatCredal`, in the adequacy layer), lower for guarantees; a singleton set recovers the plain reading, a purely demonic model recovers min-over-branches. This one convention removes any need for separate worst-case formers.

An earlier edition of this chapter fixed four denotation strata — region, path, credal,  $\omega$ -word — as primitives of its front matter. They survive, but as *views* of the landed semantics, not as axioms of the catalogue:

- the *region* view: a state formula's denotation is a set of states, entering the judgment as threshold walls — and nothing expressible is lost by compiling regions to walls, because rational threshold rows generate the flat universe (`iSup_boxRows_eq_self`);
- the *path* view: a path formula's denotation is a class of records over the site — occupied-stage classes over the record lens at the finitary stratum, trajectory classes at the clocked strata (section 14.5);
- the *credal* view: probability and expectation atoms read *lower* bounds against the whole set of admitted laws, through the envelope faces named above;
- the  $\omega$ -word view: the rewrite layer (section 14.8) reads formulas over infinite words — one carrier instance among those a former admits, the one on which the eighteen rewrite rules are machine-checked today.

**The properness presupposition.** One structural fact of the compile target shapes the entire chapter, so it is stated once, here. Every spec atom's liveness set contains the top demand — in the mechanization this is a *field* of the atom (`SpecAtom.top_mem`), so an atom without it is not merely refused but unrepresentable — and therefore **every specification's interpretation includes the trivial liveness bound**: eventually *something*, at a finite level of inconsistency. Satisfaction of any spec asserts *safe and proper*: to satisfy anything, first exist. This is a law of the semantics, not a default a producer opts into — the compile emits the mark unconditionally, and that emission is itself a theorem (`eLabSpec_top_mem`) — and it costs nothing in syntax: there is no conjunct to write and no flag to store, because the demand lives in the judgment.

What it buys is that infeasibility is observable. An inconsistent system — one admitting no behaviour at all — satisfies every safety demand vacuously and no liveness demand (`top_notMem_modelClass`: the inconsistent system belongs to no spec class); a pure-safety reading would let a bricked system satisfy the specification of a working one, and composition could manufacture such systems from proper components while every certificate stayed green. Under the presupposition the true constant is not vacuous: `Spec(SInit(STtrue))` — the initial-state injection wrapping the true constant (section 14.3) — is the *bare feasibility spec*, the minimal claim distinguishing a bricked system from a working one (section 14.4.1). The same guard appears at the other end of the judgment, where the Rule Catalogue's liveness verdicts carry their existential conjunct (chapter 15) — one guard, two ends. What it costs is one nonemptiness discharge per root satisfaction claim, discharged where the claim is consumed; and where infeasibility is genuinely expected in some conditions, the honest spelling is the assumption slot of the spec record — satisfaction relativized to conforming inputs, with nothing demanded off-assumption (section 14.10.2) — never a properness opt-out, which the grammar deliberately does not provide.

## 14.2 Formation Judgments

Eight sorts carry the grammar, and they are not a flat list: four of them arrange as a two-by-two grid — the *temporal* axis (state: atemporal, against path: trajectory) crossed with the *shape* axis (observable:  $\mathbb{Q}$ -valued, against pred: Boolean) — and the grid is capped by the polarity sorts — two faces and the attitude blend — and the record.

|                   | Observable ( $\mathbb{Q}$ -valued) | Pred (Boolean)      |
|-------------------|------------------------------------|---------------------|
| State (atemporal) | $o$ StateObservable                | $\varphi$ StatePred |
| Path (trajectory) | $R$ PathObservable                 | $\psi$ PathPred     |

|                     |                                                           |
|---------------------|-----------------------------------------------------------|
| $o$ StateObservable | $o$ is a well-formed atemporal observable                 |
| $\varphi$ StatePred | $\varphi$ is a well-formed atemporal predicate (a region) |
| $R$ PathObservable  | $R$ is a well-formed run-functional                       |
| $\psi$ PathPred     | $\psi$ is a well-formed trajectory event                  |
| $\Phi$ ForallSpec   | $\Phi$ is a well-formed safety-face component             |
| $E$ ExistsSpec      | $E$ is a well-formed liveness-mark component              |
| $H$ AttitudeSpec    | $H$ is a well-formed attitude blend (a design score)      |
| $s$ Spec            | $s$ is a well-formed spec record                          |

Three of the eight are this edition's names for sorts the catalogue always had: StatePred was StateF, PathPred was PathF, and PathObservable was Return — repairing a standing defect of the earlier edition, whose formation table omitted Return entirely while eight rules concluded into it. StateObservable names the atemporal quantitative stratum that was previously anonymous: a polynomial argument, a channel index, a gate citation — material the rules always consumed without a judgment to its name; its four injections are now formers with an entry of their own (section 14.4.2). The two polarity caps are new, and they are the point of the redesign; the attitude cap completes them, entered with the theory revision because the blend it carries fits *neither* face — its discipline is stated at its entry (section 14.10.18). One further sort stands beside the eight as pair vocabulary rather than as a grid cell:  $\pi$  StreettPair ( $\pi$  is a well-formed request/response pair), feeding the two acceptance formers. Its dissolution into per-pair premise structure was designed, and the theory revision settled the question the other way, deliberately: the pair spine persists at the row stratum as list-spine *encoding* — small hash-consed nodes, pairs-as-data — with the two pair constructors carrying the consumption-parity vocabulary, and no rule reads the pair sort as a judgment.

**The stratification law.** The State row admits no trajectory quantification and no credal quantification; the Path row lifts State-row material along trajectories; **all quantification over the credal body happens at the cap, and the cap is where polarity lives.** A ForallSpec component reads its bound against *every* admitted law — the safety face, compiling to a floor: wall domination, the closed containment face of the compile target (section 14.1) — while an ExistsSpec component asserts that *some* admitted law attains — a liveness mark, compiling to a dip: the open escape face. An AttitudeSpec component reads *both* envelopes and mixes them arithmetically before thresholding — no face, no participation in the floor/mark algebra, a design score rather than a guarantee; in one line, **a Spec mixes verdicts; an attitude mixes envelopes** (section 14.10.18). Polarity is thereby a *sort*, not an annotation: a specification cannot be written without declaring the face of every component, a mis-faced premise is a formation error rather than a semantic surprise, and a rule that needs a face restriction states it as a sort premise instead of a prose side condition. The judgment layer is untouched — one turnstile, no subscripts: the face of every component is read off the constructor vocabulary of the *arguments*.

**The formers are the grid's structure maps.** Every former of this chapter lands on exactly one cell or arrow of the diagram:

- *within-cell structure* (State–Pred): the constants and Boolean closure — STRUE, SFALSE, SAND, SOR, SNOT;
- *within-cell structure* (State–Observable): the provenance injections oCHAN, oGATE, oPOLY (section 14.4.2) — one observable, whatever its provenance;
- *thresholding*, Observable  $\rightarrow$  Pred at the State row: SATOMLE, SATOMGE;
- *indicator embedding*, Pred  $\rightarrow$  Observable at the State row: oIND, the evident square-filler ( $1_\varphi$  as an observable);
- *temporal lifting*, Pred column, State  $\rightarrow$  Path: PALWAYS, PALWAYST, PDEADLINE, PUNTIL, the metered acceptance pair PDWELLSTREETT and PCROSSSTREETT, PNOTPATH;
- *accumulation*, Observable column, State  $\rightarrow$  Path: the accumulators poSUM, poINT, poDISC, poAVG, poSUP, each over one observable — the earlier edition's provenance-fused run-functional constructors, re-founded as injections plus accumulation modes at unchanged denotations — beside the transition-count functional RETFIRECOUNT (section 14.5.6);
- *indicator embedding*, Pred  $\rightarrow$  Observable at the Path row: RETPATH;

- *thresholding*, Observable  $\rightarrow$  Pred at the Path row: PATOMLE, PATOMGE — pathwise accumulated-cost events, a claim species distinct from any expectation bound (section 14.5.7);
- *the cap*: SPROBGE, SPROBLE, SRETURNGE, SRETURNLE, SLEVELGE, SLEVELLE, the event-rate envelope pair SRATEGE, SRATELE (over transition rosters, horizon-parametric — section 14.6.4), the initial-state injection SINIT, and the regret certificate face SREGRETLE conclude ForallSpec; the possibility marks SPROBMAYBEGE, SPROBMAYBELE, SRETURNMAYBEGE, SRETURNMAYBELE, SLEVELMAYBEGE, SLEVELMAYBELE, and the regret query face SREGRETLEMARK conclude ExistsSpec; the attitude pair SHURWICZGE, SHURWICZLE concludes AttitudeSpec. The mark roster is *polarity-total*: each threshold family — probability, expectation, level — presents both faces (the level marks are new with this edition, completing the grid), and the regret pair is the same duality one seat up (section 14.6.5);
- *the record*: Spec packages face components (section 14.3); and spec-level disjunction ( $s_1 \vee s_2$ ) Spec for  $s_1, s_2$  Spec — its denotation is the union of the two record classes, constructible by definition, and it never asks the impossible merge (the no-single-floor and no-single-mark facts below are about *one* record's slots); discharge is by disjunct selection.

**The one deliberate absence.** The completed diagram has exactly one missing arrow family, and it is absent on purpose: the *descending* arrows,  $\text{Path}\{\text{Pred}, \text{Observable}\} \rightarrow \text{State}\{\text{Pred}, \text{Observable}\}$  — nested path quantification in the CTL style, where a path-quantified claim re-enters the atemporal row as a state label ( $\mathbb{P}_{\geq q}[\psi]$  as a state predicate; a value functional as a state observable). Its status is recorded in three facts. First, the *raw* formers are not cautiously deferred but fenced by the geometry: a single level of such re-entry already exits the constructible classes — neither face's rooted class is a crescent — and the negation fence forbids recovering the universal face from the existential one, so any admission would have to carry both faces as primitives (the analysis is mechanized in the adequacy layer's path-to-state development). Second, the pathway is open, not closed: the behavioural tier already *denotes* the descending arrows — the probability-threshold and expected-return sets of the compile clauses below are exactly path-to-state denotations — and the principled admission route is a *certified-citation* former: the tower discipline promoted from convention to formation rule, Observable column first, where certified envelopes carry the right one-sided semicontinuity. Third, that route is deliberately out of scope for this edition, planned for the language's second major revision (2026Q4) — a choice made with the open pathway in full view, which is what distinguishes a scoped fence from an impossibility. What the certified corpus actually needs of nesting is already served through the judgment stratum: a child claim's *certified region* enters a parent formula as an ordinary StatePred atom (entry SPROBGE; the reach-then-hold spelling of section 14.7.2 is the landed witness).

**Refinements.** Two further judgments are read off derivations rather than declared:  $\varphi \Theta_0$  — membership in the *base stratum* (nonstrict polynomial atoms, constants, and their and/or closure; nothing else), demanded wherever a consumer needs base-exact material — and the *carrier-requirement synthesis*  $\psi \triangleright \kappa$  with  $\kappa \subseteq \{\text{steps}, \text{duration}\}$ , which computes what temporal structure a former demands of the carrier it is interpreted on (per-tick accumulation demands steps; deadlines, discounts, and time-integral accumulation demand duration). Under the grid the synthesis is a *Path-row device*, and the stratification law gains a checkable spelling: every State-row former synthesizes  $\triangleright \emptyset$  — atemporality as a computation, not a convention. Carrier scope is *derived, never stored* — the consequence for equivalence judgments is drawn at entry QSPEC EQUIV (section 14.10.5).

One further stratum is deliberately *not* a formation judgment: consumability at parity. Negation is a surface former (SNOT, PNOTPATH), resolved by negation normal form, and whether a residue survives to a consumable position is *checked* — by the producer-side typing preflight, and independently by the checker — not stored in the term. In rule form the refusal table becomes a family of absences: there is no rule concluding consumability of a negated path atom at even parity, of negated Streett acceptance, of negated bounded until, or of negated unbounded always. (The table's historical first member, the negated *probability* atom, no longer needs a row at all: no probability former concludes StatePred any more, so SNot cannot reach one — the fence became a type; the path atom of the theory revision inherits its seat in the table.) These absences carry a semantic warrant, stated once and cited by the fence entries: **under the properness presupposition, no spec class has an expressible complement**. The proof is two lines — every class excludes the inconsistent system (`top_notMem_modelClass`), so every complement *contains* it,



and no union of crescents does — so the class algebra is a distributive lattice, irreducibly non-Boolean, and the refused residues are not missing engineering but missing mathematics. The dual machinery that would consume them (probability upper bounds as positive claims, Rabin acceptance, over-approximating towers) is deliberately absent until a certificate lane demands it (section 14.9).

The surface grammar, its normal forms, and its compile are themselves mechanized, and the mechanization consumes the formation discipline literally: state predicates and path predicates are inductive types whose constructors are the rules below (`SuFunctor.Universe.SpecElab` and the path-lens development beside it) — and the mechanized state-predicate grammar was already atemporal and atom-abstract before the grid named the discipline, so `StatePred` is a rename at that stratum, not a rebuild; negation normal form has *no negation node by type*, so unconsumable residues have no clause because they have no constructor; and the compile is one clause per formation rule, with the record-level adequacy statement the fold of the per-rule cells over the formation derivation — the same architecture as the Rule Catalogue’s soundness, one lemma per rule composed by induction over admitted derivations, transposed from the inference stratum to the formation stratum.

**Sequencing.** The grid was enacted grammar-first, and this section is its normative statement; the normative theory’s row schema now carries it whole (`theories/Qtsl.geolog`): the re-sorted conclusions, the level marks, the initial-state injection, spec-level disjunction, the embedding and threshold arrows, the metered acceptance pair, the running-sup and transition-count functionals, the attitude cap, and the component-record assembly are all rows of the landed revision, with their well-formedness and mutant battery beside them. Denotations are unchanged throughout, and every inhabitant of the earlier edition’s grammar has a total spelling transport — the re-homed cap formers keep their names and field labels with only sort membership moved; a fused run-functional becomes its accumulator over the matching injection; a polynomial atom becomes the same atom over `oPoly`; a bare root region becomes its explicit `SINRT` component — so no certified artefact and no published number moved with the re-sort. The producer surface has since followed: the toolkit’s revision landed the grid spellings, and the quoted API blocks below are extracted from that revised surface — the residual producer gaps (the record-top disjunction, the compiled verdict form, the regret certificate face) are named at the entries that own them.

**How to read an entry.** Each entry carries two blocks. The *Formation* block states the former’s deduction rules, with  $\Theta_0$  rules inline where the former preserves or demands base membership and  $\triangleright$  annotations beside the rules that demand carrier structure. The prose beneath it states the *compile clause* — what the former elaborates to in the semantic universe’s vocabulary — the *adequacy* mode and status (exact where the compile’s class provably equals the intended reading; directed where a sound implication is proved and no converse is claimed; per consuming battery where the obligation is discharged rule-side), and the *witness*: the landed artefact that exercises the former.

The *API* block is the producer surface of the mtk toolkit for that entry: the TypeScript arrangement (npm package `@maitria/mtk`) and the Rust catalogue crate (`crates/mtk-catalogue`) — one Rust implementation underneath, reached from TypeScript in-process through a native binding, so the two surfaces cannot diverge. Signatures and constructions are quoted *verbatim* from the toolkit sources and its conformance corpus: code whose emitted certificate bytes are verified byte-identical against committed golden vectors, from both languages (the TypeScript-driven path re-emits the same bytes the Rust-driven path committed). The quoted blocks are regenerated mechanically from the sources by the catalogue’s extraction tooling — a missing declaration or marker fails the regeneration loudly — so quoted code cannot drift silently. This API constructs and emits; it never checks: constructors refuse malformed *syntax* with the failure’s geometry attached (which field, which subterm, which value), the typing preflight is a courtesy image of the checker-side oracle, and the checker’s independent replay is the only authority.

Two vocabulary facts used throughout, stated once. *Exact scalars*: every threshold, deadline, probability, and coefficient is an exact rational (0 in TypeScript, an exact-rational witness type in Rust); no floating point enters a claim. *Positional variables*: an atom’s polynomial reads the paired system’s state coordinates positionally — or, for interface-indexed specs, the interface’s output wires in list order (guarantees) and input wires (assumptions); port names are presentation data.



## 14.3 The Spec Record

### 14.3.1 Spec: root, interface, assumptions, slack

#### Formation

$$\begin{array}{c}
 \frac{\varphi \text{ StatePred}}{\text{SInit}(\varphi) \text{ ForallSpec}} \text{ SINIT} \\
 \\
 \frac{\Phi_i \text{ ForallSpec} \quad (i < m) \quad E_j \text{ ExistsSpec} \quad (j < k) \quad H_l \text{ AttitudeSpec} \quad (l < r) \quad \text{widths}(\vec{\Phi}; \vec{E}; \vec{H}) \text{ agree pairwise}}{\text{Spec}(\vec{\Phi}; \vec{E}; \vec{H}) \text{ Spec}} \text{ SPEC-C} \\
 \\
 \frac{\begin{array}{c} \Phi_i \text{ ForallSpec} \quad (i < m) \\ E_j \text{ ExistsSpec} \quad (j < k) \quad H_l \text{ AttitudeSpec} \quad (l < r) \quad \text{width}(\vec{\Phi}; \vec{E}; \vec{H}) = \text{arity}_{\text{out}}(I) \\ \alpha \text{ StatePred} \quad \alpha \Theta_0 \quad \text{width}(\alpha) = \text{arity}_{\text{in}}(I) \end{array}}{\text{Spec}(\vec{\Phi}; \vec{E}; \vec{H}, I, \alpha) \text{ Spec}} \text{ SPEC-I} \\
 \\
 \frac{s_1 \text{ Spec} \quad s_2 \text{ Spec}}{(s_1 \vee s_2) \text{ Spec}} \text{ SPEC-OR}
 \end{array}$$

A specification is a package of *face components* — alone (SPEC-C, the closed form) or against an interface with an optional  $\Theta_0$  assumption on input wires (SPEC-I); an optional slack slot (a candidate gain function for quantitative composition) may accompany either form. The rule shape wears the compile target's own algebra: the ForallSpec components conjoin invertibly — floors merge by pointwise maximum, one slot — while the ExistsSpec components aggregate as a *list*, because marks do not merge (no single mark expresses two demands; the record holds a list exactly where the semantics refuses a merge); the AttitudeSpec components stand beside the faced ones and render as design scores, never as guarantees (section 14.10.18), taking no part in either the floor merge or the mark list. A record must carry at least one component — across the three families, or a disjunction — as an admission conformance fact: the empty record claims nothing, and the bare feasibility spec is its minimal inhabitant. The initial-state injection SINIT is the honest name for what the earlier edition attached silently to a bare region at spec root: SInit( $\varphi$ ) reads *every admitted initial state satisfies  $\varphi$*  — the  $\forall$ -face reading, now one visible rule — so the bare feasibility spec is spelled Spec(SInit(STrue)), its properness mark carried by the judgment exactly as before (eLabSpec\_top\_mem unchanged). Spec-level disjunction (SPEC-OR) sits at the record top (section 14.2): a union of record classes, discharged by disjunct selection — deliberately *above* the record, so it never asks two records to merge into one floor slot and mark list; a disjunction record carries *no* component rows of its own (the exclusivity is refuse-mode at the row stratum: its class is its arms'), and or-graph acyclicity is admission conformance. Three record-level disciplines are rule shapes rather than checks. There is no rule introducing an assumption without an interface — “a spec carrying an assumption must carry an interface” is the absence of an  $\alpha$ -with-no- $I$  rule, and in the mechanization the assumption slot lives *inside* the interface option, so the discipline holds by type. Assumptions are policed to the base stratum at formation (temporal assumptions would reintroduce assume–guarantee circularity, so they are fenced at the grammar). And closed versus open is an absence distinction: a spec with no interface row is closed, and the closed satisfaction judgment is the empty-interface instance of the one satisfaction kind (entry QSAT) — there is no second sort for interface-indexed satisfaction, deliberately. The slack slot's  $\mathcal{K}_\infty$ -class membership obligations (zero at zero, monotone, unbounded) are discharged where the slack is *consumed* — by the composition rules of chapter 15 — never stored as facts. There is also deliberately *no model binding* on a spec: pairing a spec with its system is the satisfaction judgment's job, and a spec that named its model would be the sidcar disease in miniature.

At the row stratum the record reads the same way, at the row schema's grain. A specification is an *entity*; its face components attach as relation rows, one family per cap sort — spec\_forall, spec\_exists, spec\_score — zero or more rows each, read per the algebra above, with component presence, pairwise width agreement

(arity arithmetic), and the disjunction row’s exclusivity all admission conformance rather than rows. The earlier edition’s single root *attribute* is retired with the re-sort: a bare-region root was the `SINRT` conflation, and a mixed-face conjunction root is what the component families now spell explicitly. The interface and the slack (a declared candidate gain function — a polynomial, in the row schema) attach as zero-or-one relation rows; assumptions attach as *zero-or-more* relation rows, read conjunctively — several assumption rows constrain jointly, and each row is policed to the base stratum by a load-time refusal: a provably nonbase assumption is a malformed record, not a false one. The assumption-implies-interface discipline itself is carried by the formation calculus and the mechanization’s option type; the row schema leaves it to them.

*Compile.* The record-level compile is `elabSpec`: *interface resolution, then structural recursion*. Resolution fixes the record’s positional geometry — the interface’s arities for an open spec, pairwise width agreement for a closed one — and is total on conformant records (`resolveArity_isOk_iff`), refusing otherwise *before any recursion* with the failure’s geometry attached: the offending atom, its found width, the expected arity (`resolveArity_error_names_offender`). Decoded records re-validate with the same checks — decoded data is data. The recursion is then one clause per formation rule, and the compile factors through resolution (`elabSpec_isOk_iff`); its class-level adequacy is the fold of the per-former cells over the formation derivation (`elabSpec_ok_modelClass`), with the properness mark carried unconditionally (`elabSpec_top_mem`). One deferral, fail-closed and named: the compile reads the *guarantee* only. An assumption is carried as data — its conditional-satisfaction semantics (entry `QSAT`, the interface-indexed reading) is deferred, never silently read as an unconditional strengthening, and any consumer that would need it refuses until the relativized compile lands.

*Witness:* every certificate in the corpus pairs a system with a record of this shape; the interface-indexed form is exercised by the liveness vector quoted below (two output wires, one input wire, a  $\Theta_0$  assumption).

## API

The record-top disjunction (`SPEC-OR`) remains a logged deferred former on the producer side; everything else below is the revised all-relations surface.

**TypeScript** `spec.build / spec.encode / spec.init`

signature — `ts/src/spec.ts`

```
export interface Spec {
  readonly name?: string;
  readonly forall?: ReadonlyArray<ForallSpec>;
  readonly exists?: ReadonlyArray<ExistsSpec>;
  readonly score?: ReadonlyArray<AttitudeSpec>;
  readonly iface?: Iface;
  readonly assume?: StatePred;
  readonly slack?: Slack;
}

export function build(record: Spec): Result<Spec, SpecRefusal>

export function encode(record: Spec): Result<Artifact, SpecRefusal>

export function init(arg: StatePred): ForallSpec
```

construction — `ts/tests/spec-native.test.ts`

```
const art = s.unwrap(
  s.encode({
    name: "arbiter-liveness",
    forall: [recur0, recur1],
    iface: {
      invars: [{ name: "u", shape: [] }],
      outvars: [
        { name: "grant0", shape: [] },
        { name: "grant1", shape: [] },
      ],
    },
    assume: s.polyLe(1n, s.unwrap(s.polyVar(0, 1))),
  })
```

```

    }},
  );

```

The component families (SINIT + the forall / exists / score lists) are driven end to end by the formers-tour vector (the same test file).

**Rust** `mtk_catalogue::spec::Spec::builder`

signature — `crates/mtk-catalogue/src/spec/formula.rs + crates/mtk-catalogue/src/spec/record.rs`

```

pub fn builder() -> SpecBuilder

pub fn forall(mut self, f: ForallSpec) -> Self

pub fn exists(mut self, e: ExistsSpec) -> Self

pub fn score(mut self, h: AttitudeSpec) -> Self

pub fn build(self) -> Result<Spec, Refusal>

pub fn init(arg: StatePred) -> ForallSpec

```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```

let spec = Spec::builder()
    .forall(recur0)
    .forall(recur1)
    .name("arbiter-liveness")
    .iface(iface)
    .assume(assume)
    .build()
    .unwrap();

```

The component families (SINIT + the forall / exists / score lists) are driven end to end by the formers-tour vector (the same test file).

## 14.4 State-Row Formers

This wing is the grid's State row, whole (section 14.2): both cells and both arrows between them. State observables (StateObservable) carry the row's quantitative material — one observable, whatever its provenance — and state predicates (StatePred) denote *regions* — sets of states from which the claim holds — compiling to spec atoms whose safety face is the region's wall, with the properness mark on every atom; thresholding (SATOM) passes from observables to predicates, and indicator embedding (OIND) passes back. Everything here is atemporal by the stratification law, checkably: every former of this wing synthesizes  $\triangleright \emptyset$ . Trajectories enter with the next wing, and the credal body only at the spec faces (section 14.6). Two moves organize everything. The tower discipline of chapter 15 is *certificate-mediated*: a certified child region enters a parent formula as an ordinary StatePred atom through the judgment stratum, never by syntactic embedding (entry SPROBGE). And the grammar is closed, nonstrict, and canonical by construction, so the checker never normalizes — producers do.

### 14.4.1 STTrue and SFalse: the constants

#### Formation

$$\frac{}{\text{STTrue StatePred}} \text{STTrue} \quad \frac{}{\text{SFalse StatePred}} \text{SFalse} \quad \frac{}{\text{STTrue, SFalse } \Theta_0} \Theta_0$$

*Compile.* STTrue elaborates to the one-atom presentation  $(0, \{\top\})$  — the free wall carrying the properness mark alone — and SFalse to the empty disjunct family. Exact at the class level: the true constant's class is the proper systems (`SpecDnf.modelClass_top`), the false constant's class is empty (`SpecDnf.modelClass_bot`). Under the properness presupposition the true constant is therefore not vacuous: `Spec(SInit(STTrue))` is the

bare feasibility spec — the minimal claim distinguishing a bricked system from a working one — and certifying it is certifying existence.

*Witness*: exercised everywhere; `STrue` is the trivially-true base-exact Streett request under the recurrence sugar (entry `PDWELLSTREETT` and the derived spellings below).

## API

**TypeScript** `spec.TRUE / spec.FALSE`

signature — `ts/src/spec.ts`

```
export const TRUE: StatePred = Object.freeze({ kind: "true" as const });
/** `SFalse` — the empty set. */
export const FALSE: StatePred = Object.freeze({ kind: "false" as const });
```

construction — `ts/tests/spec-native.test.ts`

```
const orPart = s.or([s.FALSE, s.not(s.polyLe(0n, x1))]);
```

`TRUE` rides the formers-tour root (the grammar-tour vector of the same test file).

**Rust** `mtk_catalogue::spec::StatePred::{True, False}`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
/// `STrue` — the full space.
True,
/// `SFalse` — the empty set.
False,
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
let or_part = StatePred::or(vec![
    StatePred::False,
    StatePred::not(StatePred::poly_le(q(0, 1), x1())),
]);
```

### 14.4.2 oChan, oGate, oPoly, oInd: the observable injections

#### Formation

$$\begin{array}{c}
 \frac{k \in \mathbb{N}}{\text{oChan}(k) \text{ StateObservable}} \text{oCHAN} \qquad \frac{g \text{ HypernetRef}}{\text{oGate}(g) \text{ StateObservable}} \text{oGATE} \\
 \frac{p \text{ Poly} \quad p \text{ canonical}}{\text{oPoly}(p) \text{ StateObservable}} \text{oPOLY} \qquad \frac{\varphi \text{ StatePred}}{\text{oInd}(\varphi) \text{ StateObservable}} \text{oIND}
 \end{array}$$

The State row's quantitative cell, and the stratum where reward *provenance* lives — so that everything downstream consumes *one observable, whatever its provenance*. A *channel* citation names a positional output wire of the paired subject's interface; channel-within-arity is record-level conformance at the pairing, where the interface is in hand, and the citation elaborates to the readout's gate there — computed, never trusted. A *gate* cites a scalar-codomain hypernet on the (possibly augmented) state carrier by content address — the carve vocabulary's gate object itself — with scalar-codomain-ness checked at admission, where the cited content is reachable; the demand is minted once, at the injection, and every gate-bearing consumer inherits it through this single locus. A *polynomial* is the view over the paired carrier's positional variables, carrying the canonicity premise (coefficients nonzero and exactly rational, exponent tuples of uniform width, strictly ascending in the canonical order — established by producer-side canonicalization rather than checked into existence). And `oIND` embeds a predicate as its characteristic observable ( $1_\varphi$ ) — the State row's `Pred` → `Observable` arrow, one of the grid's two indicator arrows (`RETPATH` is the other,

one per row): probability is the expectation of an indicator *by the diagram*, and the alias ties stay denotation lemmas at the mechanized stratum, never schema equations. Only the polynomial injection reaches the base stratum: a channel, gate, or indicator atom is nonbase by the refinement’s own negative rows, which is what keeps  $\Theta_0$  base-*exact* rather than base-shaped.

*Witness:* every polynomial atom in the corpus enters through oPOLY under the re-sort’s spelling transport; the channel and gate injections carry the provenance the run-functional formers of section 14.5.6 previously fused.

## API

**TypeScript** spec.oChan / oGate / oPoly / oInd

signature — ts/src/spec.ts

```
export function oChan(chan: number): StateObservable
export function oGate(digest: string, role?: string): StateObservable
export function oPoly(p: Poly): StateObservable
export function oInd(arg: StatePred): StateObservable
```

construction — ts/tests/spec-native.test.ts

```
const indAtom = s.atomGe(1n, s.oInd(band));
const chanObs = s.oChan(1);
```

**Rust** mtk\_catalogue::spec::StateObservable::{chan, gate, poly, ind}

signature — crates/mtk-catalogue/src/spec/formula.rs

```
pub fn chan(chan: u64) -> StateObservable
pub fn gate(gate: HypernetRef) -> StateObservable
pub fn poly(poly: Poly) -> StateObservable
pub fn ind(arg: StatePred) -> StateObservable
```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```
let ind_atom = StatePred::atom_ge(q(1, 1), StateObservable::ind(band.clone()));
let chan_obs = StateObservable::chan(1);
```

### 14.4.3 SAtomLE and SAtomGE: threshold atoms

#### Formation

$$\frac{c \in \mathbb{Q} \quad o \text{ StateObservable}}{\text{SAtomLE}(c, o) \text{ StatePred}} \text{ SATOMLE} \quad \frac{c \in \mathbb{Q} \quad o \text{ StateObservable}}{\text{SAtomGE}(c, o) \text{ StatePred}} \text{ SATOMGE}$$

$$\frac{c \in \mathbb{Q} \quad p \text{ Poly}}{\text{SAtomLE}(c, \text{oPoly}(p)), \text{SAtomGE}(c, \text{oPoly}(p)) \Theta_0} \Theta_0$$

The thresholding arrow of the grid: an atom bounds an *observable* against an exact rational, and the observable’s provenance is the previous entry’s business, not the atom’s. Over the polynomial injection this is the closed polynomial halfspace — the kernel’s native object, with no obligation standing between the atom and the exact arithmetic that checks it, and the only form the base stratum admits ( $\Theta_0$  reads the oPOLY image alone). Over a channel or gate injection the same atom thresholds the cited observable, its wall read at the pairing through the computed elaboration. The surface grammar is nonstrict *by construction*:

strict atoms are compiler-internal artifacts of negation normal form and are unrepresentable here — the schema’s testable negative; the negation of a closed atom is the strict open one, a normal-form-internal literal consumable at either parity (`den_compl_le`). Boolean wires need no dedicated former: “wire  $b$  is set” is spelled  $b \geq 1$  on a  $\{0, 1\}$ -valued wire (the derived spelling below cites the circuit suite, whose atoms compile to exactly this form).

*Compile.* The wall of the closed sublevel set, with the properness mark:  $\text{SAtomLE}(c, \text{oPoly}(p)) \mapsto (\text{wall}\{x : p(x) \leq c\}, \{\top\})$ , and symmetrically — channel and gate atoms compile the same wall species over the observable the pairing resolves. Exact: satisfaction of the compiled atom is containment of the permitted region in the halfspace *and* properness (`safetyAtom`, `coupledSat_safetyAtom_iff`, `safetyAtom_modelClass`), and nothing expressible is lost by compiling regions to threshold walls — rational threshold rows generate the flat universe (`iSup_boxRows_eq_self`), the generation theorem grounding the region view of section 14.1.

*Witness:* every certificate in the corpus.

## API

**TypeScript** `spec.atomLe / atomGe / polyLe / polyGe`

signature — `ts/src/spec.ts`

```
export function atomLe(c: QLike, o: StateObservable): StatePred
export function atomGe(c: QLike, o: StateObservable): StatePred
export function polyLe(c: QLike, p: Poly): StatePred
export function polyGe(c: QLike, p: Poly): StatePred
```

construction — `ts/tests/spec-native.test.ts`

```
const goal = s.polyGe(1n, s.unwrap(s.polyVar(0, 2)));
const cap = s.probGe(Q.of(3, 4), s.deadline(20n, goal));
```

`polyLe / polyGe` are the polynomial-atom conveniences (atom at the `oPoly` injection); polynomials are minted canonical by `poly / polyVar` (the same module).

**Rust** `mtk_catalogue::spec::StatePred::{atom_le, atom_ge}`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
pub fn atom_le(c: WitRat, o: StateObservable) -> StatePred
pub fn atom_ge(c: WitRat, o: StateObservable) -> StatePred
pub fn poly_le(c: WitRat, poly: Poly) -> StatePred
pub fn poly_ge(c: WitRat, poly: Poly) -> StatePred
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
let goal = StatePred::poly_ge(q(1, 1), Poly::var(0, 2).unwrap());
let cap =
    ForallSpec::prob_ge(q(3, 4), PathPred::deadline(q(20, 1), goal, false).unwrap()).unwrap();
```

Polynomials are minted canonical by `Poly::new / Poly::var` (`crates/mtk-catalogue/src/spec/poly.rs`).

### 14.4.4 SAnd and SOr: Boolean structure

#### Formation

$$\frac{\varphi_i \text{ StatePred} \quad (i < n)}{\text{SAnd}(\varphi_0, \dots, \varphi_{n-1}) \text{ StatePred}} \text{SAND} \quad \frac{\varphi_i \text{ StatePred} \quad (i < n)}{\text{SOr}(\varphi_0, \dots, \varphi_{n-1}) \text{ StatePred}} \text{SOR} \quad \frac{\varphi_i \Theta_0 \quad (i < n)}{\text{SAnd}(\vec{\varphi}), \text{SOr}(\vec{\varphi}) \Theta_0} \Theta_0$$



$n$ -ary as built; binary presentations are sugar, folding through associativity of the exact cells.

*Compile.* Conjunction is the *atom merge*, disjunct-family-wise: safety floors join by pointwise maximum, mark families union — and the merge is exact, its class the intersection of the classes (`SpecAtom.modelClass_merge`, `SpecDnf.modelClass_conj`; the safety face is the conjunction-is-max universal property, the order-theoretic kin of `sat_sup_iff`). Disjunction is the disjunct union outright, and its class is the plain union (`SpecDnf.modelClass_disj`). Every merge carries the properness mark by construction. One design fact deserves its theorem here, because it is why the compile’s semantics is *class* semantics: disjunction of presentations is deliberately *not* the pointwise-minimum coproduct of beliefs. A proper system can hold its permitted region inside the union of two regions while belonging to neither disjunct’s class — models of a disjunction may *hedge*, and the presentation semantics excludes hedging; the gap is machine-checked (`exists_hedging_gap`), and the region reading collapses to safe-and-proper against the denoted region exactly on the disjunction-free fragment (`elabNnf_modelClass_disjFree`).

*Witness:* exercised everywhere; the full-grammar tour vector drives both constructors directly.

## API

### TypeScript `spec.and / spec.or`

signature — `ts/src/spec.ts`

```
export function and(args: ReadonlyArray<StatePred>): StatePred
export function or(args: ReadonlyArray<StatePred>): StatePred
```

construction — `ts/tests/spec-native.test.ts`

```
const orPart = s.or([s.FALSE, s.not(s.polyLe(0n, x1))]);
```

### Rust `mtk_catalogue::spec::StatePred::{and, or}`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
pub fn and(args: Vec<StatePred>) -> StatePred
pub fn or(args: Vec<StatePred>) -> StatePred
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
let or_part = StatePred::or(vec![
    StatePred::False,
    StatePred::not(StatePred::poly_le(q(0, 1), x1())),
]);
```

## 14.4.5 SNot: negation under the closure discipline

### Formation

$$\frac{\varphi \text{ StatePred}}{\text{SNot } \varphi \text{ StatePred}} \text{ SNot}$$

Negation forms freely — and concludes nothing else: no rule puts a negation in the base stratum, and no compile clause consumes the residues that negation normal form cannot resolve.

*Compile.* By negation normal form. Negation is pushed to the leaves (De Morgan, double elimination), where a negated closed atom becomes its strict open literal; normal forms have no negation node *by type*, so the compile’s domain is a grammar in which unconsumable residues are unrepresentable, and what survives is compiled exactly — the resolution preserves the region semantics on the nose (`den_nnf`). The residues that do not survive — a negated path atom at even parity (the strict complement of an accumulated-cost event), negated Streett acceptance (Rabin), negated bounded until, negated unbounded always — are

absences of rules carrying the no-expressible-complement warrant of section 14.2: under the properness presupposition the class algebra has no complements to compile to. The producer-side typing preflight refuses each residue with a path address into the formula tree, and the unbounded-always refusal carries its remedy in the message: spell the intent as a producer-chosen deadline (PDEADLINE with the producerT flag).

*Witness:* the negation-duality suite and the rewrite layer’s normal forms; the preflight’s refusal geometry is test-pinned in both producer languages.

## API

### TypeScript spec.not

signature — ts/src/spec.ts

```
export function not(arg: StatePred): StatePred
export function preflightState(f: StatePred): Result<undefined, SpecRefusal>
```

construction — ts/tests/spec-native.test.ts

```
const orPart = s.or([s.FALSE, s.not(s.polyLe(0n, x1))]);
```

### Rust mtk\_catalogue::spec::StatePred::not

signature — crates/mtk-catalogue/src/spec/formula.rs + crates/mtk-catalogue/src/spec/typing.rs

```
pub fn not(arg: StatePred) -> StatePred
pub fn preflight_state_at(f: &StatePred, at: &FormulaPath) -> Result<(), Refusal>
```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```
let or_part = StatePred::or(vec![
    StatePred::False,
    StatePred::not(StatePred::poly_le(q(0, 1), x1())),
]);
```

## 14.5 Path-Row Formers

This wing is the grid’s Path row, whole (section 14.2): the temporal liftings that form trajectory events, the run-functional stratum that accumulates State-row observables along runs (PathObservable, entered after the liftings), and the row’s own thresholding arrow (PATOM) passing back from run-functionals to events. Path predicates (PathPred) denote sets of trajectories — *events*, and an event carries no face of its own: it enters a specification only through a cap former (the probability thresholds, or indicator accumulation via RETPATH into the prevision square), and the face — floor or mark — is declared by the consumer (section 14.6). One event, either face, and the sort says which is meant, every time. Their compile target is the *record* stratum of the semantic universe: at the finitary site, a path former reads the set of *occupied stages* of a record — and the site offers two readers of that set, the stage graph (the *touch* species) and the support of the occupation measure (the *dwell* species), with the record coupling condition saying exactly that the dwell reader is contained in the touch reader (recordCon\_iff\_stages). That one fact organizes the event formers’ semantics. Each former’s transport between the species is read off its *variance* in the occupied-stage set: invariance formers are antitone (a touch-certified claim transports to the dwell claim), reach formers are monotone (the transport flips), and the mixed formers — until, Streett — transport in *neither* direction, witnessed by explicit coupled separations (exists\_until0n\_species\_incomparable, exists\_streett0n\_species\_incomparable), so the absence of those transports is content, not omission. Polarity dictates the judgment face: invariance classes enter on the safety floor (the wall, and the landed record walls are the invariance compile’s floors — touchWall\_eq\_wall0n, dwellWall\_eq\_wall0n); reach classes enter on the liveness marks (mark0n, the mark twin of the wall), and the properness demand is the everything-goal instance of the localized liveness mark (mark0n\_univ): every “eventually” claim generalizes the bare existence claim by localizing its goal. The general record atom carries one safety class and one liveness class, and its satisfaction is *safe and proper-into-the-goal* (coupledSat\_pathAtom\_iff). At deterministic

records — the canonical lift of a stage graph — the two species coincide (`stages_uniformLift`) and the compile agrees with the trajectory clauses stated at each entry on the nose; the stochastic tier keeps only the directed transports. Windows at this stratum are natural numbers, for the same reason as the summed-return horizon: the finitary record site is tick-indexed and unitless; the duration-typed rational readings of the rules below keep their force at clocked continuum strata (validated-enclosure consumption), and the encoding question is the same one, unchanged.

### 14.5.1 PAlways: invariance

#### Formation

$$\frac{\varphi \text{ StatePred}}{\text{PAlways } \varphi \text{ PathPred}} \text{PALWAYS}$$

$G \varphi$ : the state formula holds at every time; no carrier requirement of its own.

*Compile.* The dwell class over the record site — records whose every occupied stage lies in the argument region (`alwaysOn`); trajectory clause  $\{\omega : \forall t. \omega(t) \in \llbracket \varphi \rrbracket\}$  at the deterministic cells. The class is closed (`isClosed_recordClass_always`), antitone in the occupied-stage set (touch certifies dwell), and enters the judgment as a safety floor. The safety battery consumes it (invariance and containment rows, chapter 15).

*Witness:* every avoid and invariance certificate in the corpus.

#### API

##### TypeScript `spec.always`

signature — `ts/src/spec.ts`

```
| export function always(arg: StatePred): PathPred
```

construction — `ts/tests/spec-native.test.ts`

```
| const alwaysPart = s.probGe(1n, s.always(band));
```

##### Rust `mtk_catalogue::spec::PathPred::always`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
| pub fn always(arg: StatePred) -> PathPred
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
| let always_part = ForallSpec::prob_ge(q(1, 1), PathPred::always(band.clone())).unwrap();
```

### 14.5.2 PAlwaysT: bounded always

#### Formation

$$\frac{T \in \mathbb{Q} \quad T > 0 \quad \varphi \text{ StatePred}}{\text{PAlwaysT}(T, \varphi) \text{ PathPred}} \text{PALWAYST}$$

$$\text{PAlwaysT}(T, \varphi) \triangleright \{\text{duration}\}$$

The state formula holds at every time up to the deadline — the claim shape of every validated-enclosure certificate (“the flow from the initial box stays in the tube over  $[0, T]$ ”), which is why the former exists: this catalogue holds that the spec as stated must be the spec certified, and this statement deserved a spelling.

*Compile.* The bounded dwell class: occupied stages within the window lie in the region (`windowAlwaysOn`; window a natural number at the tick-indexed stratum, per the wing preamble). The class is closed (`isClosed_recordClass_alwaysT`), antitone in the occupied-stage set and in the window — the window is an approximation source: a longer-window certificate consumes a shorter-window claim. Deterministic consumption is immediate (enclosure semantics is trajectory-wise); stochastic consumption at even parity awaits the over-approximating tower — the same fence as the probability upper bounds (section 14.9.3). Its normal-form dual *is* spellable: negated bounded always rewrites to bounded reach, so surface path negation over this former passes the preflight at both parities.

*Witness for the demand:* the enclosure fixture family (flow-tube certificates); the safety-until-deadline worked example; the LunarLander funnel holds.

## API

### TypeScript `spec.alwaysT`

signature — `ts/src/spec.ts`

```
| export function alwaysT(t: QLike, arg: StatePred): PathPred
```

construction — `ts/tests/spec-native.test.ts`

```
| const boundedPart = s.probGe(Q.of(1, 2), s.notPath(s.alwaysT(3n, band)));
```

### Rust `mtk_catalogue::spec::PathPred::always_t`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
| pub fn always_t(t: WitRat, arg: StatePred) -> Result<PathPred, Refusal>
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
| let bounded_part = ForallSpec::prob_ge(
|   q(1, 2),
|   PathPred::not_path(PathPred::always_t(q(3, 1), band.clone()).unwrap()),
| )
| .unwrap();
```

## 14.5.3 PDeadline: bounded reach

### Formation

$$\frac{T \in \mathbb{Q} \quad T > 0 \quad \varphi \text{ StatePred} \quad b \in \mathbb{B}}{\text{PDeadline}(T, \varphi, b) \text{ PathPred}} \text{PDEADLINE}$$

$$\text{PDeadline}(T, \varphi, b) \triangleright \{\text{duration}\}$$

The target region is reached by the deadline. The flag  $b$  (`producerT`) marks a producer-chosen horizon standing in for an unbounded reach — a sound under-approximation, carried *flagged* in the row so the reader of a certificate knows which kind of statement was made (the unbounded primitive is a deliberate absence; the derived spelling appears with the fences below).

*Compile.* The début (hitting) record class by the deadline: some occupied stage within the window lies in the target (`windowReachOn`); trajectory clause  $\{\omega : \exists t \leq T. \omega(t) \in \llbracket \varphi \rrbracket\}$  at the deterministic cells. The class is open (`isOpen_recordClass_deadline`), monotone in the occupied-stage set (dwell certifies touch —

the variance flip), and enters the judgment on the liveness face. Obligations: the deadline-battery kernels (geometric-decay and expectation-of-hitting arguments), machine-checked per the Rule Catalogue’s receipt columns.

*Witness:* reach-by-deadline certificates across the corpus; the certified-Gymnasium hitting waves (twenty-eight certificates of the quoted shape).

## API

### TypeScript `spec.deadline`

signature — `ts/src/spec.ts`

```
export function deadline(
  t: QLike,
  arg: StatePred,
  opts?: { producerT?: boolean },
): PathPred
```

construction — `ts/tests/spec-native.test.ts`

```
const goal = s.polyGe(1n, s.unwrap(s.polyVar(0, 2)));
const cap = s.probGe(Q.of(3, 4), s.deadline(20n, goal));
```

### Rust `mtk_catalogue::spec::PathPred::deadline`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
pub fn deadline(t: WitRat, arg: StatePred, producer_t: bool) -> Result<PathPred, Refusal>
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
let goal = StatePred::poly_ge(q(1, 1), Poly::var(0, 2).unwrap());
let cap =
  ForallSpec::prob_ge(q(3, 4), PathPred::deadline(q(20, 1), goal, false).unwrap()).unwrap();
```

## 14.5.4 PUntil: safety until a deadline

### Formation

$$\frac{T \in \mathbb{Q} \quad T > 0 \quad \varphi_{\text{safe}} \text{ StatePred} \quad \varphi_{\text{tgt}} \text{ StatePred}}{\text{PUntil}(T, \varphi_{\text{safe}}, \varphi_{\text{tgt}}) \text{ PathPred}} \text{P}_{\text{UNTIL}}$$

$$\text{PUntil}(T, \cdot, \cdot) \triangleright \{\text{duration}\}$$

Stay safe until reaching the target, by the deadline; trajectory clause  $\{\omega : \exists t \leq T. \omega(t) \in \llbracket \varphi_{\text{tgt}} \rrbracket \wedge \forall s < t. \omega(s) \in \llbracket \varphi_{\text{safe}} \rrbracket\}$ .

*Compile.* The hitting-with-dwell record class (`until0n`): reach the target within the window, occupied stages strictly before staying safe. The mixed variance shows up everywhere the pure formers were simple: the class transports between the species in *neither* direction (witnessed, per the wing preamble), and its citizenship is the grammar’s first genuinely constructible-not-closed-not-open class — a finite union of locally closed pieces by window decomposition (`recordClass_untilF_eq`, `isConstructibleUnion_recordClass_untilF`), landing it squarely in the constructible stratum the compile target promises. The union-bound kernel that consumes it is machine-checked; negation of a bounded until has no sound compile target in this edition and is refused by the preflight with a path address.

*Witness:* the until-battery fixtures and the worked safety-until-deadline example.

## API

**TypeScript** `spec.until`signature — `ts/src/spec.ts`| `export function until(t: QLike, safe: StatePred, target: StatePred): PathPred`construction — `ts/tests/spec-native.test.ts`| `const untilPart = s.probGe(Q.of(9, 10), s.until(5n, band, s.polyGe(2n, x1)));`**Rust** `mtk_catalogue::spec::PathPred::until`signature — `crates/mtk-catalogue/src/spec/formula.rs`| `pub fn until(t: WitRat, safe: StatePred, target: StatePred) -> Result<PathPred, Refusal>`construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```

let until_part = ForallSpec::prob_ge(
  q(9, 10),
  PathPred::until(q(5, 1), band.clone(), StatePred::poly_ge(q(2, 1), x1())).unwrap(),
)
.unwrap();

```

**14.5.5 PDwellStreett and PCrossStreett: acceptance, metered****Formation**

$$\begin{array}{c}
\frac{\rho \text{ StatePred} \quad \rho \Theta_0 \quad \gamma \text{ StatePred}}{\text{PairBase}(\rho, \gamma) \text{ StreettPair}} \text{PAIRBASE} \quad \frac{\rho \text{ StatePred} \quad \gamma \text{ StatePred}}{\text{PairCmpl}(\rho, \gamma) \text{ StreettPair}} \text{PAIRCMPL} \\
\\
\frac{\pi_i \text{ StreettPair} \quad (i < n)}{\text{PDwellStreett}(\pi_0, \dots, \pi_{n-1}) \text{ PathPred}} \text{PDWELLSTREETT} \\
\\
\frac{\pi_i \text{ StreettPair} \quad (i < n)}{\text{PCrossStreett}(\pi_0, \dots, \pi_{n-1}) \text{ PathPred}} \text{PCROSSSTREETT}
\end{array}$$

Streett acceptance [226] — for each pair, an unbounded request forces an unbounded response — as *two* formers, because “unbounded” has two meterings and the record lens proves them incomparable: the mixed formers transport between the stage-graph reader and the occupation reader in neither direction, even on coupled points (the wing preamble’s witnessed separations). A metering flag would leave one former whose every consuming rule and citizenship theorem still cased on the flag with the type system enforcing nothing; two formers make the meter part of the *event’s identity* — the same move as the polarity redesign, one stratum down: metering as former, not annotation. PDwellStreett is occupation metering — unbounded request *occupation* forces unbounded response occupation (counting measure at tick strata, duration at clocked strata); you dwell on a street. PCrossStreett is entry-event counting — unboundedly many request *crossings* (fresh contact after absence, the countable, payable unit of touching) force unboundedly many response crossings; you cross a street. Enter-and-stay separates the species even at unit ticks — occupation diverges while the entry counter stops at one — and the two meterings at one pair list are distinct events, refuse-mode disjoint at the row stratum. Everything step-metered the catalogue already had is the dwell former’s unit-tick instance *by identity*: the liveness batteries, the acceptance event, and the recurrence sugar migrate there whole, and nothing migrates to the cross species, whose consuming machinery composes with the crossing-budget family at the behavioural tier. Where the species provably coincide — deterministic records, where the two stage readers agree — either former may be formed and the collapse theorem transports.

The pair vocabulary is metering-independent, shared verbatim. The two pair constructors are the two request-consumption modes — *parity as constructor choice, never stored data*: PairBase consumes the request base-exact, whence its  $\Theta_0$  premise (whose deliberate absence from PairCmpl is the point); PairCmpl



consumes the request through the syntactic complement’s over-approximation, stored positive. The recurrence and persistence operators are definitional instances, not formers:  $GF\varphi$  is a single dwell pair with a trivially-true base-exact request, and  $FG\varphi$  is the dwell pair whose complement-consumed request firing unboundedly forces the impossible response (both appear as derived spellings below). The face of an acceptance claim stays at the cap that consumes the event — two meterings by two faces, all four spelled compositionally. Negated Streett is Rabin acceptance — the dual judgment, a deliberate absence under either metering.

*Compile.* Two tiers, kept distinct. At the behavioural tier the dwell acceptance event lives on the kernel path measure, and the quantitative certificate bundle concluding it is machine-checked end to end (`sound_QStreettBundle`, composing into the probability-threshold region at entry `SProbGE`); the cross event’s infinitary reading is the entry-counter family, whose very definedness at continuum strata is the crossing-budget machinery’s evidence. At the record stratum the compile is the per-pair implication over occupied stages (`streettOn`), each former reading its *own* species — the dwell former the occupation support, the cross former the stage graph — as the *occupation shadow* of the infinitary acceptance condition, stated as such: a finite record has no “unboundedly often,” and the pasting between the record tier and the kernel path measure is deliberately not claimed here (it is the finite-dimensional-distributions stratum, the same fence the behavioural tier names). Both record classes are constructible; species transport fails in both directions, witnessed — which is the split’s warrant, restated as compile fact.

*Witness:* the 219-task liveness suite (specs normalized to  $\bigwedge_i GF\gamma_i$  — dwell spellings — and certified against circuit models); the cross former is new with the theory revision, its first certificate lane the crossing-budget composition.

## API

**TypeScript** `spec.dwellStreett / crossStreett / pairBase / pairCmpl`

signature — `ts/src/spec.ts`

```
export function dwellStreett(pairs: ReadonlyArray<StreettPair>): PathPred
export function crossStreett(pairs: ReadonlyArray<StreettPair>): PathPred
export function pairBase(req: StatePred, resp: StatePred): StreettPair
export function pairCmpl(req: StatePred, resp: StatePred): StreettPair
```

construction — `ts/tests/spec-native.test.ts`

```
const dwellPart = s.probGe(
  1n,
  s.dwellStreett([s.pairCmpl(band, s.polyGe(1n, x1))]),
);
const crossPart = s.proble(
  Q.of(1, 2),
  s.crossStreett([s.pairBase(band, s.polyGe(1n, x1))]),
);
```

A base-exact pair is also driven (through the recurrence spelling) by the liveness vector quoted under the derived spellings.

**Rust** `mtk_catalogue::spec::PathPred::{dwell_streett, cross_streett}`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
pub fn dwell_streett(pairs: Vec<StreettPair>) -> PathPred
pub fn cross_streett(pairs: Vec<StreettPair>) -> PathPred
pub fn base(req: StatePred, resp: StatePred) -> Result<StreettPair, Refusal>
pub fn cmpl(req: StatePred, resp: StatePred) -> StreettPair
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```

let dwell_part = ForallSpec::prob_ge(
  q(1, 1),
  PathPred::dwell_streett(vec![StreettPair::cml(
    band.clone(),
    StatePred::poly_ge(q(1, 1), x1()),
  )]),
)
.unwrap();
let cross_part = ForallSpec::prob_le(
  q(1, 2),
  PathPred::cross_streett(vec![StreettPair::base(
    band.clone(),
    StatePred::poly_ge(q(1, 1), x1()),
  )
.unwrap()]),
)
.unwrap();

```

### 14.5.6 The Return stratum

The Observable cell of the Path row: a PathObservable (the sort the earlier edition called Return) is a *run-functional* — what a run is worth, packaged as data — consumed by the expectation, level, and regret faces of section 14.6 and thresholded back into events by PATOM below. The cell is reached by accumulation from State-row material, and the theory spells it so: reward *provenance* (channel, gate, polynomial view, indicator) is the injection stratum of section 14.4.2, and *accumulation* is five formers over one observable — the earlier edition’s provenance-fused constructor roster (retChanSum through retAvg) re-founded as injections plus accumulators, at unchanged denotations. Beside the accumulators stand the two citation formers: RETPATH, the Path row’s indicator arrow, and RETFIRECOUNT, the transition-count functional.

### Formation

$$\begin{array}{c}
\frac{n \in \mathbb{N} \quad n \geq 1 \quad o \text{ StateObservable}}{\text{poSum}(n, o) \text{ PathObservable}} \text{poSUM} \quad \frac{T \in \mathbb{Q} \quad T > 0 \quad o \text{ StateObservable}}{\text{poInt}(T, o) \text{ PathObservable}} \text{poINT} \\
\\
\frac{\theta \in \mathbb{Q} \quad \theta > 0 \quad o \text{ StateObservable}}{\text{poDisc}(\theta, o) \text{ PathObservable}} \text{poDISC} \quad \frac{o \text{ StateObservable}}{\text{poAvg}(o) \text{ PathObservable}} \text{poAVG} \\
\\
\frac{o \text{ StateObservable}}{\text{poSup}(o) \text{ PathObservable}} \text{poSUP} \quad \frac{\psi \text{ PathPred}}{\text{retPath}(\psi) \text{ PathObservable}} \text{RETPATH} \\
\\
\frac{R \text{ TrList} \quad R \neq \text{nil}}{\text{retFireCount}(R) \text{ PathObservable}} \text{RETFIRECOUNT}
\end{array}$$

$$\text{poSum} \triangleright \text{steps} \quad \text{poInt} \triangleright \text{duration}$$

The two finite-horizon accumulators carry the integrality content the mechanization forced, now by *shape*: a discrete sum’s upper index is a natural number, and the tick-indexed strata that consume summed accumulation are unitless — no clock exists there against which a rational horizon could be rounded, which also rules out any rounding law at those strata (a system clock would make one spec denote different claims per clock). Integrated accumulation keeps its free positive rational horizon, which keeps its meaning at clocked continuum strata. The encoding decision the earlier edition left open is settled the strong way: split constructors with a natural-number tick horizon, so a rational tick horizon is unrepresentable rather than checked — and with the horizon split carried by the constructors, an integrality side condition has nothing left to check.

The remaining accumulators carry no horizon. poDisc is infinite-horizon exponential discounting at rate  $\theta$ : weights  $e^{-\theta t}$  on the internalized real-time base, under which tick systems embed at integer times

with weights  $e^{-\theta n}$  — kept exact in the log-rational grade, the same abstract-exponential discipline that keeps every transcendental out of the exact stratum. `poAvg` is the long-run average (the Cesàro limit); its certificate species is the gain/bias pair — a Poisson-equation drift row, the same supermartingale-shaped row grammar as everything above. `poSup(o)` is the pathwise running supremum of the observable along the run — horizon-free, read in the extended-real scale, where an infinite supremum makes an upper-bound claim *false*, never ill-formed. Its consuming shape is the maximal function: `SReturnLE(B, poSup(o))` bounds  $\mathbb{E}[\sup_t o(x_t)]$  over every admitted law, for which Doob’s maximal inequality is the natural certificate engine and the supermartingale row kit the natural feed — named with its species, its rule rows not yet claimed. Deliberately *not* minted beside it: a sup-outside-expectation former —  $\sup_t \mathbb{E}[\cdot]$  is a forall- $t$  family of marginal-law judgments, not a run-functional — and since  $\mathbb{E}[\sup] \geq \sup_t \mathbb{E}$  always, the per-instant-uniform reading arrives as a free corollary of the expressible, stronger form. `retPath( $\psi$ )` cites a path event as its indicator run-functional; temporal claims compile *into* this stratum (monitor automaton, product kernel, compiled state-reward gate, accumulation), following the value-functions-as-certificates route [3], under which the reward designs for Streett acceptance are state-valued indicators on the product carrier and the ordinary value function of a satisfying policy *is*, affinely, the certificate. The exact deadline table of chapter 15 is the value function of the compiled deadline reward — a reading, not a new obligation — and reinforcement learning becomes a sanctioned *producer* for the battery rows: powerful untrusted producers, small trustworthy checkers, as everywhere.

`retFireCount( $R$ )` is the transition-count functional: the total number of firings of the roster’s transitions along the run — no horizon, values in  $[0, \infty]$ , the same extended reading as the running supremum on its nonnegative fragment. The roster  $R$  is a positional list of transition citations of the paired subject; nonemptiness is formation (an empty roster would be the zero functional — a constant claim in count clothing, and canonical form forbids re-spelling), while duplicate-freedom and roster membership are pairing-relative demands at the certificate stratum, where the subject is in hand. Counting is firing-metered per roster member — deliberately *not* region-entry counting (that is PCROSSSTREETT’s device, section 14.5.5) and deliberately not the instant-crossing compensator mass of the Zeno seam: different species, not interchangeable. The consuming shape this former exists for is the hard count ceiling, `SReturnLE(B, retFireCount( $R$ ))` — along every admitted law, the total count of roster firings is at most  $B$  (section 14.7.7). The  $\geq$  face — at least  $n$  events, a genuine liveness claim consuming survival mass — is deliberately absent from this functional: that is throughput territory, and it enters at the expectation stratum rather than here — the event-rate envelope family’s lower face bounds the expected count from below, horizon-parametrically, over the same rosters (section 14.6.4).

## API

**TypeScript** `spec.poSum / poInt / poDisc / poAvg / poSup / retPath / retFireCount`

signature — `ts/src/spec.ts`

```
export function poSum(horizon: number, o: StateObservable): PathObservable
export function poInt(horizon: QLike, o: StateObservable): PathObservable
export function poDisc(o: StateObservable, rate: QLike): PathObservable
export function poAvg(o: StateObservable): PathObservable
export function poSup(o: StateObservable): PathObservable
export function retPath(arg: PathPred): PathObservable
export function retFireCount(trs: ReadonlyArray<number>): PathObservable
```

construction — `ts/tests/spec-native.test.ts`

```
const retLe = s.returnLe(100n, s.poInt(50n, s.oPoly(x0)));
const discFloor = s.returnGe(-5n, s.poDisc(chanObs, Q.of(1, 10)));
const countCeiling = s.returnLe(40n, s.retFireCount([0, 2]));
```

**Rust** `mtk_catalogue::spec::PathObservable::{sum, int, disc, avg, sup, path, fire_count}`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```

pub fn sum(horizon: u64, o: StateObservable) -> Result<PathObservable, Refusal>

pub fn int(horizon: WitRat, o: StateObservable) -> Result<PathObservable, Refusal>

pub fn disc(o: StateObservable, rate: WitRat) -> Result<PathObservable, Refusal>

pub fn avg(o: StateObservable) -> PathObservable

pub fn sup(o: StateObservable) -> PathObservable

pub fn path(arg: PathPred) -> PathObservable

pub fn fire_count(trs: Vec<u64>) -> Result<PathObservable, Refusal>

```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```

let ret_le = ForallSpec::return_le(
  q(100, 1),
  PathObservable::int(q(50, 1), StateObservable::poly(x0())).unwrap(),
);
let disc_floor =
  ForallSpec::return_ge(q(-5, 1), PathObservable::disc(chan_obs, q(1, 10)).unwrap());
let count_ceiling =
  ForallSpec::return_le(q(40, 1), PathObservable::fire_count(vec![0, 2]).unwrap());

```

### 14.5.7 PAtomLE and PAtomGE: accumulated-cost events

#### Formation

$$\frac{c \in \mathbb{Q} \quad R \text{ PathObservable}}{\text{PAtomLE}(c, R) \text{ PathPred}} \text{PAtomLE} \quad \frac{c \in \mathbb{Q} \quad R \text{ PathObservable}}{\text{PAtomGE}(c, R) \text{ PathPred}} \text{PAtomGE}$$

The thresholding arrow at the Path row — the grid’s admitted square-filler, entered with the theory revision: the *event* that the accumulated value lands on the stated side of the threshold, the inner clause of  $\mathbb{P}_{\geq p}[\sum r \leq B]$ . This is a claim species distinct from any expectation bound, and the distinction is the grid read column-wise: SReturnLE( $B, R$ ) bounds the *prevision* of  $R$  — one number per law, capped at the credal body — while PAtomLE( $B, R$ ) names the set of *runs* on which  $R$  lands under  $B$ , an event like any other, whose face is then declared by the cap that consumes it (a probability threshold, or indicator accumulation via RETPATH). Carrier requirements are inherited, not minted: the synthesis reads the run-functional argument ( $\text{PAtomLE}(c, R) \triangleright \kappa$  exactly when  $R \triangleright \kappa$ ).

*Compile.* Owed with the former’s first certificate lane — formation and the traversal stratum are landed at the theory revision; the record-stratum compile clause and its adequacy cell are named obligations, and no battery consumes the event yet.

*Witness for the demand:* the count-ceiling and budget families, whose probabilistic faces are exactly  $\mathbb{P}$ -thresholds over accumulated-cost events.

#### API

##### TypeScript spec.pathAtomLe / pathAtomGe

signature — ts/src/spec.ts

```

export function pathAtomLe(c: QLike, ret: PathObservable): PathPred

export function pathAtomGe(c: QLike, ret: PathObservable): PathPred

```

construction — ts/tests/spec-native.test.ts

```

const pathEvent = s.pathAtomGe(0n, s.retPath(s.always(indAtom)));
const eventCap = s.probGe(Q.of(3, 4), pathEvent);

```

**Rust** `mtk_catalogue::spec::PathPred::{atom_le, atom_ge}`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
pub fn atom_le(c: WitRat, ret: PathObservable) -> PathPred
pub fn atom_ge(c: WitRat, ret: PathObservable) -> PathPred
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
let path_event = PathPred::atom_ge(
  q(0, 1),
  PathObservable::path(PathPred::always(ind_atom.clone())),
);
let event_cap = ForallSpec::prob_ge(q(3, 4), path_event).unwrap();
```

## 14.5.8 PNotPath: surface path negation

### Formation

$$\frac{\psi \text{ PathPred}}{\text{PNotPath } \psi \text{ PathPred}} \text{ PNotPath}$$

Surface-only path negation, resolved by the dualities in negation normal form.

*Compile.* By the normal-form dualities — and the fence table is *executable*: normal-form resolution is a total function into either a negation-free form or a named refusal, and each row of the table is a definitional equation. The one always-spellable dual resolves at both parities — negated bounded always is bounded reach of the complement (`pathNnf_notPath_alwaysT`), and the even parity of a bounded reach resolves through double negation (`pathNnf_notPath_notPath_deadLine`) — while the three residues refuse with their addresses: negated unbounded always, negated until, negated Streett (`pathNnf_notPath_always`, `pathNnf_notPath_untilF`, `pathNnf_notPath_streett`). The path-atom formers of the theory revision extend the refusal roster at the producer surface: a negated path atom at even parity is the strict complement of an accumulated-cost event — a normal-form artifact with no surface spelling — refused with the same path-address geometry, as a documented extension of the pinned oracle. One note on the warrants, because they differ by stratum: at the record-class level the complement of an invariance class *does* exist (`not_alwaysOn_iff` — some occupied stage escapes), so the refusal of negated unbounded always is grammar economy — there is no unbounded-reach former to spell the dual, itself a deliberate absence — whereas at the spec-class stratum one level up, where the judgment kinds live, the refusals carry the no-expressible-complement theorem of section 14.2.

*Witness:* the negation-duality suite; the full-grammar tour vector’s `PNotPath( $G_{\leq 3} \varphi$ )` passes the preflight at both parities.

### API

**TypeScript** `spec.notPath`

signature — `ts/src/spec.ts`

```
export function notPath(arg: PathPred): PathPred
```

construction — `ts/tests/spec-native.test.ts`

```
const boundedPart = s.probGe(0.of(1, 2), s.notPath(s.alwaysT(3n, band)));
```

**Rust** `mtk_catalogue::spec::PathPred::not_path`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
pub fn not_path(arg: PathPred) -> PathPred
```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```
let bounded_part = ForallSpec::prob_ge(
  q(1, 2),
  PathPred::not_path(PathPred::always_t(q(3, 1), band.clone()).unwrap()),
)
.unwrap();
```

## 14.6 Spec-Face Formers

The wing above the grid. Every former here consumes run-valued material — an event, a run-functional, or (for the rate envelopes) a transition roster read as a horizon-parametric counting process — reads it against the credal body, and concludes into a polarity cap sort, so this wing is where a claim first acquires a *face*: the stratification law of section 14.2 puts all quantification over admitted laws here and nowhere else. Formers concluding ForallSpec are the safety faces — the bound holds over *every* admitted law, compiling to the floor — and the Maybe and Mark formers conclude ExistsSpec, the possibility faces: *some* admitted law (or, for regret, some filler) attains, compiling to a mark. The roster is polarity-total — probability, expectation, and level each present both faces, and the regret pair is the same duality with the quantifier moved to the filler seat. Two cap formers live with their consumers instead of here: the initial-state injection SINIT, stated with the record it feeds (section 14.3), and the attitude pair, which mixes envelopes rather than declaring a face and closes the judgment-kind inventory (section 14.10.18).

### 14.6.1 SProbGE, SProbLE, and the probability marks

#### Formation

$$\frac{p \in \mathbb{Q} \quad 0 < p \leq 1 \quad \psi \text{ PathPred}}{\text{SProbGE}(p, \psi) \text{ ForallSpec}} \text{ SProbGE} \quad \frac{q \in \mathbb{Q} \quad 0 \leq q < 1 \quad \psi \text{ PathPred}}{\text{SProbLE}(q, \psi) \text{ ForallSpec}} \text{ SProbLE}$$

$$\frac{p \in \mathbb{Q} \quad 0 < p \leq 1 \quad \psi \text{ PathPred}}{\text{SProbMayBeGE}(p, \psi) \text{ ExistsSpec}} \text{ SProbMayBeGE}$$

$$\frac{q \in \mathbb{Q} \quad 0 \leq q < 1 \quad \psi \text{ PathPred}}{\text{SProbMayBeLE}(q, \psi) \text{ ExistsSpec}} \text{ SProbMayBeLE}$$

One species, four faces. SProbGE( $p, \psi$ ) — historically this stack’s only probability former, under the name SProb — asserts that the path event holds with probability at least  $p$ , read as *lower* probability against the credal view, over every admitted law — whence its ForallSpec sort; the initial-state datum enters the satisfaction judgment in hypothesis position, not the formula — so one convention serves stochastic kernels, demonic branching, and worst-case initial conditions without a case split. SProbLE( $q, \psi$ ) is the second orientation *within* the species, not a mirror species: the *upper* probability of the same compiled event is at most  $q$ . It is a closed constraint whose wall is a citizen of the semantic universe like any other, and emphatically *not* the negation of the GE face — that is the strict  $P_{<p}$ , which rightly remains a compiler artifact (the fences below keep that absence). The Maybe pair carries the *possibility marks*: some admitted law attains the stated side of the threshold — the credal-existential face of the atom, a liveness demand entering on the marks rather than the floor.

What the earlier edition advertised as nesting is stated, under the grid, at its true stratum: the tower discipline is *certificate-mediated*. A child claim is discharged as its own satisfaction judgment, and its *certified region* enters the parent formula as an ordinary StatePred atom — geometry with a certificate behind it, not a formula inside a formula. Syntactic nesting of cap formers (a probability atom inside another’s path argument) is not formable — Path-row formers take StatePred arguments — and the certified corpus never spelled such a term: its one advertised nesting use, the reach-then-hold funnel, already routes the child claim through the judgment stratum (section 14.7.2 quotes it). The grammar now matches what the tower always did.



The family is a family of *aliases*: an event probability is the expectation of an indicator gamble, so orientation for orientation

$$\text{SProbGE}(p, \psi) = \text{SReturnGE}(p, \text{retPath } \psi),$$

and likewise for the LE face and both marks (section 14.6.2). The alias ties are definitional-grade denotation lemmas at the mechanized stratum, never schema equations: the surface constructors remain distinct hash-consed syntax, and the elaboration is computed where consumed.

The interval premises are *canonicity rules, not semantic fences* — four instances of one rule: the vacuous end of either orientation re-spells a structural part of the atom, and canonical form forbids re-spelling. At  $p = 0$  the GE floor (and at  $q = 1$  the LE floor) would compile to the unit atom, which is exactly `STrue`’s compile (`SpecDnf.modelClass_top`); at  $p = 0$  the `MayBeGE` mark reads “some admitted law with  $\mu(\psi) \geq 0$ ” — the structural properness mark itself (`SpecAtom.top_mem`), as does the `MayBeLE` mark at  $q = 1$ . Bottom-refutes was never carried by threshold positivity: the credal-lower reading is vacuous at the zero lift for every constraint (`satCredal_lower_singletonC_zero`), and it is the properness presupposition that excludes the bricked system.

*Compile.* The GE face: the region of states from which the credal-lower probability of the compiled path event clears the threshold — mechanized for the kernel path measure as `probThresholdSet`, entering the judgment as that region’s safety atom. Adequacy is *directed*: region inclusion transports to class inclusion (`safetyAtom_modelClass_mono`, the wing’s directed-cell template; `sprobAtom_modelClass_mono`), and each consuming battery discharges the sound direction — the Ville [234] and optional-stopping arguments behind the safety, deadline, until, and Streett batteries of chapter 15, per that catalogue’s receipt columns. One end-to-end specimen is machine-checked at the behavioural tier: a quantitative Streett certificate bundle concludes membership in the probability-threshold region of the acceptance event (`sprob_sound_of_streettBundle`). No completeness is claimed anywhere in this family: a certificate can be arbitrarily loose. The LE face enters as the same wall species at the opposite orientation — the checker-anchor restriction theorem already lists *both* threshold orientations as closed half-spaces of laws, and the negation-fence remark names the orientation pair the sound residue of the failed duality. The marks enter on the atom’s mark list — the mechanized `Dip`: a strict undercut somewhere admitted. Formation is open on all four faces; *consumption* of the LE face at interior  $q$  keeps its over-approximating-tower fence (section 14.9.2), while  $q = 0$  discharges on the exact lane today: an exact backward-induction table is simultaneously an under- and over-approximation, so the possibilistic prohibition  $\bar{P}[\psi] = 0$  — the semantic universe’s own behavioural-prohibition face — is certified by the same table read in the super-solution direction. Two compatibility facts anchor the credal reading at the precise end: at the singleton lift of a nonzero law, the lower reading of an event atom is that law’s own event probability (`satCredal_lower_singletonC_gec`); and the zero-lift vacuity above.

*Witness:* the supermartingale corpus end to end; the reach-by-deadline vector quoted below is the certified-Gymnasium hitting shape ( $\mathbb{P}[\text{goal by } 20] \geq 3/4$ ); the exact-zero avoidance claims of the deterministic corpus spell natively on the LE face.

## API

**TypeScript** `spec.probGe / probLe / probMayBeGe / probMayBeLe`

signature — `ts/src/spec.ts`

```
export function probGe(p: QLike, path: PathPred): ForallSpec
export function probLe(q: QLike, path: PathPred): ForallSpec
export function probMayBeGe(p: QLike, path: PathPred): ExistsSpec
export function probMayBeLe(q: QLike, path: PathPred): ExistsSpec
```

construction — `ts/tests/spec-native.test.ts`

```
const goal = s.polyGe(1n, s.unwrap(s.polyVar(0, 2)));
const cap = s.probGe(Q.of(3, 4), s.deadline(20n, goal));
```

The MAYBE marks construct in the marks vector quoted under the seat-regret entry (the same test file).

**Rust** `mtk_catalogue::spec::ForallSpec::{\prob_ge, \prob_le}`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
pub fn prob_ge(p: WitRat, path: PathPred) -> Result<ForallSpec, Refusal>

pub fn prob_le(q: WitRat, path: PathPred) -> Result<ForallSpec, Refusal>

pub fn prob_may_be_ge(p: WitRat, path: PathPred) -> Result<ExistsSpec, Refusal>

pub fn prob_may_be_le(q: WitRat, path: PathPred) -> Result<ExistsSpec, Refusal>
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
let goal = StatePred::poly_ge(q(1, 1), Poly::var(0, 2).unwrap());
let cap =
    ForallSpec::prob_ge(q(3, 4), PathPred::deadline(q(20, 1), goal, false).unwrap()).unwrap();
```

The MAYBE marks construct in the marks vector quoted under the seat-regret entry (the same test file).

## 14.6.2 SReturnGE, SReturnLE, and the possibility marks

### Formation

$$\frac{B \in \mathbb{Q} \quad R \text{ PathObservable}}{\text{SReturnGE}(B, R) \text{ ForallSpec}} \text{SRETURNGE} \quad \frac{B \in \mathbb{Q} \quad R \text{ PathObservable}}{\text{SReturnLE}(B, R) \text{ ForallSpec}} \text{SRETURNLE}$$

$$\frac{B \in \mathbb{Q} \quad R \text{ PathObservable}}{\text{SReturnMayBeGE}(B, R) \text{ ExistsSpec}} \text{SRETURNMAYBEGE}$$

$$\frac{B \in \mathbb{Q} \quad R \text{ PathObservable}}{\text{SReturnMayBeLE}(B, R) \text{ ExistsSpec}} \text{SRETURNMAYBELE}$$

The *soft* former: the prevision square at a run-functional. The floors read the paired System’s credal body at the gamble  $R$  — credal-lower expectation at least  $B$  (GE), credal-upper expectation at most  $B$  (LE) — and the marks are the credal-existential faces: some admitted law attains the stated side. Nothing on the spec side carries a set of models: under the one-universe reading the System’s own denotation is the weighted set, and the spec carries only the pair  $(R, B)$ . This is where minimax weighted expected regret lands without new vocabulary: a weighted set of probability measures [108] is, member for member, a set of sub-probability laws ( $\alpha \text{Pr}$  for weight  $\alpha$ ), i.e. a credal body in this universe’s own imprecise atlas; the sup-normalization convention (some member carries weight one) is the properness presupposition; and the weighted expected regret of a nonnegative run-functional is the body’s upper prevision. “WER within  $[B_1, B_2]$ ” is therefore the two-component record  $\text{Spec}(\text{SReturnLE}(B_2, R); \text{SReturnMayBeGE}(B_1, R))$  — one floor, one mark, the same denotation the earlier edition spelled as a state-level conjunction, with the face of each component now visible in its sort; no interval sort exists: ranges are orientation pairs here as everywhere in this catalogue. Bounds are unconstrained rationals — returns are not probabilities, and the vacuous-end canonicity rule is a probability-face fact.

*Compile.* Credal-lower/upper expectation rows over the declared reward, lifted at the measurable stratum; the finite-horizon summed channel form is mechanized as `returnGESet` (natural-number horizon, as the split now forces by shape), entering the judgment as that region’s safety atom, with region monotonicity in reward, horizon, and bound as the directed cells (`returnGESet_mono_reward`, `returnGEAtom_modelClass_mono_reward`). Adequacy is *directed*, certificate to claim, and the two square families carrying it are machine-checked at the abstract behavioural tier: a geometrically decaying Lyapunov certificate — drift factor  $\beta$  off the target, barrier  $a$  — yields the expected indicator-return bound  $1 - \beta^n (\int L_0)/a$  at horizon  $n+1$  (`returnGE_sound_of_deadline`); and a geometric-drift certificate bounds the discounted expected return from above, uniformly in the horizon, under the spectral condition  $\gamma\beta < 1$

(discountedReturn\_le\_of\_drift, with integral\_le\_pow\_mul the iterated-drift engine). Both squares are proved on an abstract filtered probability space, where the certificate’s conditional-drift data lives; the specialization to the kernel path measure is a separate, named stratum — an obligation this edition names rather than claims — and the transfer from the deadline kernel to the discounted return forms climbs the same ladder. The marks enter on the atom’s mark list (Dip), exactly as the probability marks do; the long-run-average battery (gain/bias) is design vocabulary of this edition, named with its species and not yet claimed. This family is the catalogue’s highest-priority former because of what it retired: until it existed, expected-return certificates carried their claims in header sidecars.

*Witness for the demand:* twenty-five tabular expected-return certificates ( $\mathbb{E}_\pi[\sum r] \geq B$  — Cliff Walking —13 exactly; Taxi 793/100; Blackjack —3954908734551683826282634489/13<sup>26</sup>, the corpus’s own bignum, quoted below); the pendulum swing-up worst-case return row ( $\geq -619.99$  from every default-reset initial state); the Atari circuit rows (min-over-lanes return, exact); the LunarLander statistical rows (via QSATCONF below).

## API

**TypeScript** spec.returnGe / returnLe / returnMayBeGe / returnMayBeLe

signature — ts/src/spec.ts

```
export function returnGe(lo: QLike, ret: PathObservable): ForallSpec
export function returnLe(hi: QLike, ret: PathObservable): ForallSpec
export function returnMayBeGe(lo: QLike, ret: PathObservable): ExistsSpec
export function returnMayBeLe(hi: QLike, ret: PathObservable): ExistsSpec
```

construction — ts/tests/spec-native.test.ts

```
const ret = s.poSum(26, s.oChan(0));
const floor = s.returnGe(
  Q.of(-3954908734551683826282634489n, 13n ** 26n),
  ret,
);
```

The accumulator injections are quoted under the Return stratum; the poInt / poDisc faces and the possibility marks ride the formers-tour vector (the same test file).

**Rust** mtk\_catalogue::spec::ForallSpec::{return\_ge, return\_le}

signature — crates/mtk-catalogue/src/spec/formula.rs

```
pub fn return_ge(lo: WitRat, ret: PathObservable) -> ForallSpec
pub fn return_le(hi: WitRat, ret: PathObservable) -> ForallSpec
pub fn return_may_be_ge(lo: WitRat, ret: PathObservable) -> ExistsSpec
pub fn return_may_be_le(hi: WitRat, ret: PathObservable) -> ExistsSpec
```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```
let thirteen_pow_26 = (0..26).fold(BigInt::from(1), |a, _| a * 13);
let bound = WitRat::new(
  BigInt::parse_bytes(b"-3954908734551683826282634489", 10).unwrap(),
  thirteen_pow_26,
);
let ret = PathObservable::sum(26, StateObservable::chan(0)).unwrap();
let floor = ForallSpec::return_ge(bound, ret);
```

The accumulator injections are quoted under the Return stratum; the poInt / poDisc faces and the possibility marks ride the formers-tour vector (the same test file).

### 14.6.3 SLevelGE, SLevelLE, and the level marks

#### Formation

$$\begin{array}{c}
\frac{B \in \mathbb{Q} \quad R \text{ PathObservable} \quad thr \in \mathbb{Q}_{\geq 0} \cup \{\infty\}}{\text{SLevelGE}(B, R, thr) \text{ ForallSpec}} \text{SLEVELGE} \\
\\
\frac{B \in \mathbb{Q} \quad R \text{ PathObservable} \quad thr \in \mathbb{Q}_{\geq 0} \cup \{\infty\}}{\text{SLevelLE}(B, R, thr) \text{ ForallSpec}} \text{SLEVELLE} \\
\\
\frac{B \in \mathbb{Q} \quad R \text{ PathObservable} \quad thr \in \mathbb{Q}_{\geq 0} \cup \{\infty\}}{\text{SLevelMayBeGE}(B, R, thr) \text{ ExistsSpec}} \text{SLEVELMAYBEGE} \\
\\
\frac{B \in \mathbb{Q} \quad R \text{ PathObservable} \quad thr \in \mathbb{Q}_{\geq 0} \cup \{\infty\}}{\text{SLevelMayBeLE}(B, R, thr) \text{ ExistsSpec}} \text{SLEVELMAYBELE}
\end{array}$$

The *hard* former: for every full-mass law whose level of inconsistency is finite and at most  $thr$ , the expected value of  $R$  lies on the stated side of  $B$ . The threshold lives in the extended nonnegative rationals — syntactically a two-constructor sum (finite bound; infinity), so every consumer’s finite/instant case split is carried by shape — and the finiteness clause is one uniform formula, not a case split: redundant at finite thresholds, it does all the work at  $thr = \infty$ , where the quantifier ranges over the *support*, every behaviour not fully refused. That instance is the behavioural-prohibition face of the satisfaction judgment itself: take  $R$  the indicator of the bad event via `retPath`,  $thr = \infty$ ,  $B = 0$  on the LE side, and the probe says the upper probability of the bad event is zero. At  $thr = 0$  the quantifier reads the zero-inconsistency core. Every rung of the ladder is a lawful probe; no positivity premise exists beyond the threshold’s own nonnegativity.

The *Maybe* pair completes the family’s polarity: *some* full-mass admitted law within the level threshold attains the stated side of  $B$  — the level stratum’s possibility marks, new with this edition, read exactly as the probability and expectation marks (at  $thr = \infty$  the mark asserts attainment by some behaviour-law in the support). With them the mark roster of section 14.2 is polarity-total: every threshold family presents both faces.

Soft and hard are not rivals but the two dualities the semantic universe already stratifies. The soft former’s single-shot linear tests are a complete instrument, by Fenchel–Moreau duality, exactly on the convex composition core; level probes recover the whole quasiconvex blanket, by Penot–Volle level-set duality — and the gap has a witness: a quantile (value-at-risk) specification has every sublevel a half-space, hence is hard-expressible tolerance-by-tolerance, and is not convex, hence invisible to the soft former alone. The hinge between them has a machine-checked ancestor over the retired calculus-tier carrier: there, the finite-tilt family converges in order to the wall at the infinite endpoint (`livenessTilt_top` in the archived `su-functor-legacy` tree, with epi-convergence), which is this pair of formers stated at the threshold stratum; its restatement over the live semantic universe is owed alongside the level compile itself. The hard former’s completeness, by contrast, is a theorem the live universe already holds: a single instance denotes a rational-height gamble row, and finite conjunctions of exactly these, closed under directed suprema, generate every citizen of the universe (`General.iSup_gambleRows_eq_self`) — the level probe is the universe’s own generation species wearing surface syntax, its certificate discharge the polynomial-nonnegativity species this book’s rule catalogue already discharges elsewhere, re-aimed.

#### API

The mechanized compile remains owed with the tilt-hinge restatement above; the producer surface below is landed, all four faces.

**TypeScript** `spec.levelGe / levelLe / levelMayBeGe / levelMayBeLe`

signature — `ts/src/spec.ts`

```

export function levelGe(lo: QLike, ret: PathObservable, thr: EnnRat): ForallSpec

export function levelLe(hi: QLike, ret: PathObservable, thr: EnnRat): ForallSpec

export function levelMayBeGe(
  lo: QLike,
  ret: PathObservable,
  thr: EnnRat,
): ExistsSpec

export function levelMayBeLe(
  hi: QLike,
  ret: PathObservable,
  thr: EnnRat,
): ExistsSpec

export function ennFin(q: QLike): EnnRat

export function ennInf(): EnnRat

```

construction — ts/tests/spec-native.test.ts

```

const levelLo = s.levelGe(Q.of(1, 2), s.poAvg(s.oChan(0)), s.ennFin(3n));
const levelHi = s.levelLe(Q.of(7, 2), s.poSup(s.oPoly(x1)), s.ennInf());

```

**Rust** mtk\_catalogue::spec::ForallSpec::{level\_ge, level\_le}

signature — crates/mtk-catalogue/src/spec/formula.rs

```

pub fn level_ge(lo: WitRat, ret: PathObservable, thr: EnnRat) -> ForallSpec

pub fn level_le(hi: WitRat, ret: PathObservable, thr: EnnRat) -> ForallSpec

pub fn level_may_be_ge(lo: WitRat, ret: PathObservable, thr: EnnRat) -> ExistsSpec

pub fn level_may_be_le(hi: WitRat, ret: PathObservable, thr: EnnRat) -> ExistsSpec

pub fn fin(q: WitRat) -> Result<EnnRat, Refusal>

pub fn inf() -> EnnRat

```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```

let level_lo = ForallSpec::level_ge(
  q(1, 2),
  PathObservable::avg(StateObservable::chan(0)),
  EnnRat::fin(q(3, 1)).unwrap(),
);
let level_hi = ForallSpec::level_le(
  q(7, 2),
  PathObservable::sup(StateObservable::poly(x1())),
  EnnRat::inf(),
);

```

## 14.6.4 SRateLE and SRateGE: event-rate envelopes

### Formation

$$\frac{\rho \in \mathbb{Q}_{\geq 0} \quad \sigma \in \mathbb{Q}_{\geq 0} \quad R \text{ TrList} \quad R \neq \text{nil}}{\text{SRateLE}(\rho, \sigma, R) \text{ ForallSpec}} \text{SRateLE}$$

$$\frac{\rho \in \mathbb{Q}_{\geq 0} \quad \sigma \in \mathbb{Q}_{\geq 0} \quad R \text{ TrList} \quad R \neq \text{nil}}{\text{SRateGE}(\rho, \sigma, R) \text{ ForallSpec}} \text{SRateGE}$$

The from-init affine count envelope — the rate axis’s first expectation-class citizens. Write  $N_t^R$  for the number of firings of the roster’s transitions on  $[0, t]$ : the same firing-metered counting the transition-count

functional reads (section 14.5.6), here horizon-parametric rather than whole-run, and consumed directly — the envelope is a family of bounds indexed by the horizon, not a single run-functional, which is why these formers take the roster itself. SRateLE claims that along every admitted law the expected count respects the envelope at every horizon — credal-upper expectation of  $N_t^R$  at most  $\rho t + \sigma$  for all  $t \geq 0$  — and SRateGE, the throughput face, that the credal-lower expectation dominates  $\rho t - \sigma$ . Both faces are floors at the ForallSpec cap: each bounds the whole credal body, and the claim direction selects the envelope — an upper count bound reads the upper prevision (the most events any compatible resolution produces), a lower bound the lower (the fewest). Both are certification pessimism, and rate atoms carry no attitude knob: the direction is the whole story. The possibility faces are deliberately absent, a recorded choice rather than an oversight: a mark-face rate envelope — *some* admitted law’s expected counts obey the envelope — has no certificate row family and no consumer, and the mark roster’s polarity-totality is a fact about the threshold families, which the envelope pair is not one of; a later widening stays additive.

The affine-from-init shape is the primitive on purpose. A bare long-run limit ( $\limsup_t N_t/t \leq \rho$ ) carries zero finite-window content — a consumer sized against it can be overwhelmed on any finite horizon — so the limit reading is a corollary of the envelope ( $\overline{E}[N_t]/t \leq \rho + \sigma/t$ ), never the interface object. And interval content is carried by *pairs*: a two-sided pair at matched rate — upper  $(\rho, \sigma_u)$ , lower  $(\rho, \sigma_l)$  — yields the interval envelope  $N_{(s,t]} \leq \rho(t-s) + \sigma_u + \sigma_l$  by pathwise subtraction, no ergodic machinery, which is why the primitive can stay from-init without losing burstiness control: a one-sided envelope says nothing about increments; a pair does. The remaining free laws are of the same grain: weakening an envelope weakens its claim (the upper face is monotone in both parameters; the lower face trades them the other way), and disjoint rosters sum — upper envelopes over disjoint rosters conjoin to an upper envelope at the summed parameters on the merged roster, pathwise count additivity plus subadditivity of the upper prevision — while the converse split is refused: a summed bound cannot be divided between rosters without a per-roster certificate, since all the mass may ride one of them.

Refusals, each with its reason. No limsup or ergodic primitive (the zero-finite-window-content argument above — and a fortiori no almost-sure empirical-rate convergence, a law of large numbers needing recurrence structure the vocabulary deliberately does not assume). Nothing sharper than the pair-derived interval form: equilibrium-rate claims and interval budgets below  $\sigma_u + \sigma_l$  genuinely need that same ergodic machinery — the fence the occupation-time operators sit behind (section 14.9.5). Counted events are declared transitions only, never predicate crossings: a full-rank diffusion crosses a surface infinitely often, so region-entry counting stays PCROSSSTREETT’s metered device (section 14.5.5) — a different species, not interchangeable. No within-roster order content: the envelope counts class firings, and per-item ordering or latency stays aggregate-invisible by design. No policy-quantified or optimality rate claims (“no dispatch achieves a better rate” is a research frontier, not a former): rate judgments are fixed-dispatch or demonic, like every judgment in this catalogue. And no attitude knob, as above. The assume/guarantee half of the rate axis — rate bounds crossing fused boundaries, and the probabilistic token-bucket budget form that rides fusion — remains a catalogued deliberate absence with its own registry slot (section 14.9.1).

*Compile.* The certificate species is staged beside the formers: a supermartingale row kit per face, kin to the drift rows everywhere above. The upper face’s kit carries a per-cell polynomial potential  $V$  with interior rows bounding the generator’s action on  $V$  plus the counted transitions’ intensity by  $\rho$  on each mode cell, counted-transition guard rows absorbing the unit charge ( $V$  after the reset at most  $V - 1$ ), un-counted guard rows non-increasing, and nonnegativity on the reset closure — making  $V + N_t^R - \rho t$  a supermartingale under every compatible law, so the expected count is at most  $\rho t + V(x)$ , envelopes last, with the certified region the sublevel  $\{V \leq \sigma\}$ . The lower face mirrors it ( $V + \rho t - N_t^R$ ; the counted compensator credits the count’s expected growth), and its potential is the Foster-Lyapunov throughput shape: a work-in-progress potential whose forced descent pays out completions. A third, structural route concludes the upper face from clock structure alone — race, dwell, and instant-budget readings of the landed timing stock, no potential synthesized. No exhibit claims a rate battery yet; formed claims await theirs, as everywhere in this catalogue.

*Witness for the demand:* the throughput claim of the Mini-Fab tutorial (section 17.4, chapter 17) renders as this former’s lower face; until this family was staged, that chapter could record the claim only as an argument beside the vocabulary.

## API



**TypeScript** ---

The theory rows are staged; the producer surface enters with the toolkit’s next revision, and this entry gains its quoted constructions then.

**Rust** ---

Likewise.

**14.6.5 SRegretLE and SRegretLEMark: seat regret****Formation**

$$\frac{\varepsilon \in \mathbb{Q} \quad \varepsilon \geq 0 \quad \pi \text{ System} \quad R \text{ PathObservable}}{\text{SRegretLE}(\varepsilon, \pi, R) \text{ ForallSpec}} \text{SREGRETLE}$$

$$\frac{\varepsilon \in \mathbb{Q} \quad \varepsilon \geq 0 \quad R \text{ PathObservable}}{\text{SRegretLEMark}(\varepsilon, R) \text{ ExistsSpec}} \text{SREGRETLEMARK}$$

The subject of a regret claim is a *comb*: a composite with exactly one seat — a hole presenting a declared interface, in the system side’s own vocabulary (chapter 12) — and the comparison class is the seat itself: every filler presenting that interface. The comparison class is read off the *subject*, never re-declared per formula — re-declaration would double-count it, and silent demonic closure of an unfilled hole under an ordinary claim is exactly what the pairing-relative demands refuse. Achieved and comparison values are credal-*lower* expectations of the run-functional on the respective closed composites: residual nondeterminism stays demonic on both sides uniformly — maximin regret, deliberately *not* online-learning pathwise regret, which needs a shared weather trace no compositional semantics supplies. The certificate face carries its reference filler: no type-correct filler beats  $\pi$  by more than  $\varepsilon$ , each judged by its own guaranteed value. The query face carries none — it is well-formed in total ignorance of the answer, and its discharging selection witness *carries* the filler: the angelic idiom, in which the witness is the answer. The two faces are the floor/mark duality one level up, with the quantifier moved from the law seat to the filler seat — which is why the name is MARK and not MAYBE: MAYBE is the law-seat existential, and the two seats have just been separated. Regret is one-sided by definition (no GE mirror: a “regret at least” face would be a liveness claim about the seat’s *spread* — a different former, deliberately not minted), and  $\varepsilon = 0$  is lawful and canonical: the exact-optimality claim. Pairing-relative demands — the paired subject carries exactly one hole; the certificate face’s filler fills it — mint at the certificate stratum, where the pairing is in hand.

*Witness for the demand*: the certified-Gymnasium “optimal, exactly” family — optimality claims the earlier edition could state only as bare return bounds against a known optimum; the discharge lanes are named with their species, not yet claimed.

**API****TypeScript** `spec.regretLeMark`

signature — `ts/src/spec.ts`

```
| export function regretLeMark(eps: QLike, ret: PathObservable): ExistsSpec
```

construction — `ts/tests/spec-native.test.ts`

```
| const markGe = s.probMayBeGe(Q.of(1, 2), s.always(band));
| const markRet = s.returnMayBeLe(0n, s.poSum(7, s.oChan(0)));
| const markLevel = s.levelMayBeGe(Q.of(1, 4), s.poAvg(s.oChan(1)), s.enFin(1n));
| const regretMark = s.regretLeMark(
|   Q.of(1, 100),
|   s.poDisc(s.oChan(0), Q.of(1, 20)),
| );
```

The safety face SREGRETLE has no producer former yet; the MARK is the landed surface, and the marks vector quoted here also drives the probability, return, and level MAYBE marks.

**Rust** `mtk_catalogue::spec::ExistsSpec::regret_le_mark`

signature — `crates/mtk-catalogue/src/spec/formula.rs`

```
| pub fn regret_le_mark(eps: WitRat, ret: PathObservable) -> Result<ExistsSpec, Refusal>
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
let mark_ge = ExistsSpec::prob_may_be_ge(q(1, 2), PathPred::always(band.clone())).unwrap();
let mark_ret = ExistsSpec::return_may_be_le(
    q(0, 1),
    PathObservable::sum(7, StateObservable::chan(0)).unwrap(),
);
let mark_level = ExistsSpec::level_may_be_ge(
    q(1, 4),
    PathObservable::avg(StateObservable::chan(1)),
    EnnRat::fin(q(1, 1)).unwrap(),
);
let regret_mark = ExistsSpec::regret_le_mark(
    q(1, 100),
    PathObservable::disc(StateObservable::chan(0), q(1, 20)).unwrap(),
)
.unwrap();
```

The safety face SREGRETLE has no producer former yet; the MARK is the landed surface, and the marks vector quoted here also drives the probability, return, and level MAYBE marks.

## 14.7 Derived Spellings

Each entry here is a claim species from the certified corpus with its spelling in the grammar above — the catalogue’s evidence that *no further former is needed* for it. The helpers add no expressive power; where a helper has code, its definition is quoted, because the definition *is* the content. Two rows deliberately have no code in this wing, and say where their content lives instead. Each spelling owes the compile a coherence cell — the definitional rider saying the sugar’s class is the spelled form’s class. The first cells of this one-line class are machine-checked at the record site: bounded reach is trivially-safe until (`until0n_univ_safe`), and the recurrence sugar’s single trivial-request pair degenerates to reachability of the response region, conditioned on occupation, with the empty record exhibited as the one stage set where the readings part (`streett0n_singleton_univ_iff`, `streett0n_singleton_univ_empty`). The remaining spellings’ riders are named obligations of the same class, to land with the compile they annotate.

### 14.7.1 Capture within N steps

#### Spelling

$$\text{capture-within}(N, \varphi) = P_{\geq 1}[F_{\leq N} \varphi]$$

The credal view makes this worst-case over demonic branching and initial conditions — no dedicated former, no worst-case variant. *Witness*: the pendulum swing-up row — capture within 139 steps from every default-reset initial state.

#### API

**TypeScript** `spec.derived.captureWithin`

signature — `ts/src/spec.ts`

```
| captureWithin(steps: QLike, capture: StatePred): ForallSpec
```

definition — ts/src/spec.ts

```
/** Capture within `steps` — `P_{>=1}[F_{<=steps} capture]` as a
 * safety-face component (the credal stratum makes this worst-case
 * over demonic branching/init). */
captureWithin(steps: QLike, capture: StatePred): ForallSpec {
  return probGe(1n, deadline(steps, capture));
},
```

**Rust** mtk\_catalogue::spec::derived::capture\_within

signature — crates/mtk-catalogue/src/spec/derived.rs

```
pub fn capture_within(steps: WitRat, capture: StatePred) -> Result<ForallSpec, Refusal>
```

definition — crates/mtk-catalogue/src/spec/derived.rs

```
/// Wing C: capture within `steps` — `P_{>=1}[F_{<=steps} capture]`
/// (the credal stratum makes this worst-case over demonic
/// branching/init). Witness: the swing-up row — capture within 139
/// steps from every default-reset init.
pub fn capture_within(steps: WitRat, capture: StatePred) -> Result<ForallSpec, Refusal> {
  ForallSpec::prob_ge(rat_int(1), PathPred::deadline(steps, capture, false?))
}
```

## 14.7.2 Reach-then-hold (the funnel)

### Spelling

$\text{reach-then-hold}(p, T, \chi) = P_{\geq p}[F_{\leq T} \chi]$  where  $\chi$  is a child claim's certified invariance region

The outer transit is spelled here; the citation of the child's certified region ( $\chi$  as a region atom, concluded by a  $P_{\geq 1}[G \text{ band}]$  claim) belongs to the judgment stratum — the tower discipline again. *Witness*: the swing-up composition (outer transit into the held upright set); the LunarLander attitude funnel.

### API

**TypeScript** spec.derived.reachThenHold

signature — ts/src/spec.ts

```
reachThenHold(p: QLike, t: QLike, heldRegion: StatePred): ForallSpec
```

definition — ts/src/spec.ts

```
/** Reach-then-hold (funnel) — `P_{>=p}[F_{<=t} chil]` where `chi` is
 * a child claim's certified invariance region, cited as a region
 * atom (the citation itself rides the judgment stratum). */
reachThenHold(p: QLike, t: QLike, heldRegion: StatePred): ForallSpec {
  return probGe(p, deadline(t, heldRegion));
},
```

**Rust** mtk\_catalogue::spec::derived::reach\_then\_hold

signature — crates/mtk-catalogue/src/spec/derived.rs

```
pub fn reach_then_hold(
  p: WitRat,
  t: WitRat,
  held_region: StatePred,
) -> Result<ForallSpec, Refusal>
```

definition — crates/mtk-catalogue/src/spec/derived.rs

```

/// Wing C: reach-then-hold (funnel; band-hold) -
/// `P_{>=p}[F_{<=t} chi]` where `chi` is the *certified invariance
/// region* of a child claim (`P_{>=1}[G band]`), cited as a region
/// atom (tower citation: a child's certified region is a parent's
/// atom). This helper spells the outer transit; the citation itself
/// rides the judgment stratum. Witness: the swing-up composition;
/// the LunarLander attitude funnel.
pub fn reach_then_hold(
  p: WitRat,
  t: WitRat,
  held_region: StatePred,
) -> Result<ForallSpec, Refusal> {
  ForallSpec::prob_ge(p, PathPred::deadline(t, held_region, false?))
}

```

### 14.7.3 Safety until deadline

#### Spelling

$$\text{safe-until-deadline}(T, \text{safe}, \text{target}) = \text{safe } U_{\leq T} \text{target}$$

Directly the until former — or, for a probabilistic core, the two-story tower around it. Listed because the species recurs in the corpus under this name. *Witness*: the worked example; the until-battery fixtures.

#### API

##### TypeScript spec.derived.safetyUntilDeadline

signature — ts/src/spec.ts

```
| safetyUntilDeadline(t: QLike, safe: StatePred, target: StatePred): PathPred
```

definition — ts/src/spec.ts

```

/** Safety until deadline - `safe U_{<=t} target`, directly. */
safetyUntilDeadline(t: QLike, safe: StatePred, target: StatePred): PathPred {
  return until(t, safe, target);
},

```

##### Rust mtk\_catalogue::spec::derived::safety\_until\_deadline

signature — crates/mtk-catalogue/src/spec/derived.rs

```

pub fn safety_until_deadline(
  t: WitRat,
  safe: StatePred,
  target: StatePred,
) -> Result<PathPred, Refusal>

```

definition — crates/mtk-catalogue/src/spec/derived.rs

```

/// Wing C: safety until deadline - `safe U_{<=t} target`, directly
/// (or as the probability cap around it, which is a composition of
/// [`ForallSpec::prob_ge`] with this). Witness: the worked example;
/// until-battery fixtures.
pub fn safety_until_deadline(
  t: WitRat,
  safe: StatePred,
  target: StatePred,
) -> Result<PathPred, Refusal> {
  PathPred::until(t, safe, target)
}

```

## 14.7.4 Almost-sure recurrence under demonic input

### Spelling

$$\text{recurrence-as}(\gamma) = P_{\geq 1}[GF \gamma], \quad GF \gamma = PDwellStreett(PairBase(STrue, \gamma))$$

The dwell-metered Streett sugar at the demonic profile (frequently is step-frequently at unit ticks): the credal view reads the lower probability over the branch set, so almost-sure recurrence under an adversarial input needs no game former. *Witness*: the 219 circuit tasks, whose conjunctions of recurrence goals are built by exactly the quoted spelling.

### API

**TypeScript** `spec.derived.recurrenceAs`

signature — `ts/src/spec.ts`

```
recurrenceAs(gamma: StatePred): ForallSpec
gf(gamma: StatePred): PathPred
fg(phi: StatePred): PathPred
```

construction — `ts/tests/spec-native.test.ts`

```
const gamma0 = s.unwrap(s.derived.boolWireSet(0, 2));
const gamma1 = s.unwrap(s.derived.boolWireSet(1, 2));
const recur0 = s.derived.recurrenceAs(gamma0);
const recur1 = s.derived.recurrenceAs(gamma1);
```

**Rust** `mtk_catalogue::spec::derived::recurrence_as`

signature — `crates/mtk-catalogue/src/spec/derived.rs`

```
pub fn recurrence_as(gamma: StatePred) -> Result<ForallSpec, Refusal>
pub fn gf(gamma: StatePred) -> Result<PathPred, Refusal>
pub fn fg(phi: StatePred) -> PathPred
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
let gamma0 = derived::bool_wire_set(0, 2).unwrap();
let gamma1 = derived::bool_wire_set(1, 2).unwrap();
let recur0 = derived::recurrence_as(gamma0).unwrap();
let recur1 = derived::recurrence_as(gamma1).unwrap();
```

## 14.7.5 Min-return over enumerated branches

### Spelling

$$\text{min-return}(B, R) = SReturnGE(B, R) \quad \text{at the credal reading}$$

Lower expectation over a finite branch set is the minimum — the spelling is the plain former; the worst-case reading belongs to the substrate, not the syntax. *Witness*: the Atari circuit rows (per-lane returns, claimed bound equal to the minimum, exact).

### API

**TypeScript** `spec.derived.minReturnOverBranches`

signature — `ts/src/spec.ts`

```
| minReturnOverBranches(lo: QLike, ret: PathObservable): ForallSpec
```

```
definition — ts/src/spec.ts
```

```
/** Min-return over enumerated branches — `SReturnGE` at the credal
 * stratum (lower expectation over a finite branch set = the
 * minimum); the spelling is the plain former. */
minReturnOverBranches(lo: QLike, ret: PathObservable): ForallSpec {
  return returnGe(lo, ret);
},
```

**Rust** `mtk_catalogue::spec::derived::min_return_over_branches`

```
signature — crates/mtk-catalogue/src/spec/derived.rs
```

```
| pub fn min_return_over_branches(lo: WitRat, ret: PathObservable) -> ForallSpec
```

```
definition — crates/mtk-catalogue/src/spec/derived.rs
```

```
/// Wing C: min-return over enumerated branches — `SReturnGE` at the
/// credal stratum (lower expectation over a finite branch set = the
/// minimum); the *spelling* is the plain former over the given run
/// functional, the credal reading is the substrate's. Witness: the
/// Atari circuit rows (per-lane returns, claimed bound = the
/// minimum).
pub fn min_return_over_branches(lo: WitRat, ret: PathObservable) -> ForallSpec {
  ForallSpec::return_ge(lo, ret)
}
```

## 14.7.6 Worst-case-over-init

### Spelling

any former + a demonic initial box on the system side

No spec-side helper, by design: worst-case over initial conditions is a property of the *pairing*, spelled as a demonic initial box in the system's own vocabulary (chapter 12) and read through the credal view — the formula layer needs no worst-case former. *Witness*: the swing-up rows; the random-ODE uniform-over-parameter barrier (parameter box as an initial-box axis).

### API

**TypeScript** ---

Spelled on the system side (a demonic initial box declaration), not in this namespace; the credal reading supplies the worst case.

**Rust** ---

Likewise: a system-side declaration, deliberately without a formula-layer twin.

## 14.7.7 Hard event-count envelopes

### Spelling

$$\text{count-ceiling}(B, R) = \text{SReturnLE}(B, \text{retFireCount}(R))$$

The count ceiling is the plain expectation former at the transition-count functional (section 14.5.6): along every admitted law, the total count of roster firings is at most  $B$  — no new grammar, as this section's



charter demands. An earlier edition of this entry routed the species through the designed event-rate envelopes as their  $p = 1$  instance; the count spine landed first, the rate family's expectation atoms have since been staged in their own right (section 14.6.4), and the probabilistic budget form — the  $p = 1$  route — remains the deliberate absence it was (section 14.9.1); the ceiling stays the plain spelling above either way. *Witness for the demand*: structurally bounded resource-sharing net families; the plated-transfer discharge that concludes the ceiling is Rule Catalogue content.

## API

**TypeScript** `spec.returnLe / retFireCount`

The ceiling is the plain composition of landed formers, `returnLe(B, retFireCount(R))` — the transition-count injection's producer surface, both languages, lives with the run functionals (section 14.5.6).

**Rust** `mtk_catalogue::spec::{ForallSpec::return_le, PathObservable::fire_count}`

Likewise: `ForallSpec::return_le(B, PathObservable::fire_count(R))`, the empty roster refused at construction and at the decoded-artifact preflight alike.

### 14.7.8 Boolean-output atoms

## Spelling

$$\text{bool-wire-set}(i) = \text{SAtomGE}(1, x_i) \quad \text{bool-wire-clear}(i) = \text{SAtomLE}(0, x_i)$$

The 0/1 convention on Boolean-valued wires: no Boolean atom former exists or is needed. *Witness*: the circuit suite's atoms compile to exactly this form; the liveness vector's recurrence goals are built over these spellings.

## API

**TypeScript** `spec.derived.boolWireSet / boolWireClear`

signature — `ts/src/spec.ts`

```
boolWireSet(index: number, width: number): Result<StatePred, SpecRefusal>
boolWireClear(index: number, width: number): Result<StatePred, SpecRefusal>
```

construction — `ts/tests/spec-native.test.ts`

```
const gamma0 = s.unwrap(s.derived.boolWireSet(0, 2));
const gamma1 = s.unwrap(s.derived.boolWireSet(1, 2));
const recur0 = s.derived.recurrenceAs(gamma0);
const recur1 = s.derived.recurrenceAs(gamma1);
```

**Rust** `mtk_catalogue::spec::derived::bool_wire_set`

signature — `crates/mtk-catalogue/src/spec/derived.rs`

```
pub fn bool_wire_set(index: usize, width: usize) -> Result<StatePred, Refusal>
pub fn bool_wire_clear(index: usize, width: usize) -> Result<StatePred, Refusal>
```

construction — `crates/mtk-catalogue/tests/golden_spec.rs`

```
let gamma0 = derived::bool_wire_set(0, 2).unwrap();
let gamma1 = derived::bool_wire_set(1, 2).unwrap();
let recur0 = derived::recurrence_as(gamma0).unwrap();
let recur1 = derived::recurrence_as(gamma1).unwrap();
```

## 14.8 The Rewrite Layer

### 14.8.1 Rewrite chains, concluding spec equivalence

#### Rules

Eighteen rewrite rules, each a denotation *equality* over  $\omega$ -words, with congruence (rewriting at a position) and chain transport (a sequence of rewrites transports satisfaction verbatim, in both directions): implication elimination, right distribution, double negation, the two De Morgan laws, negation past F/G/X, and the polarity-absorption fusion family — the existential-over-time operators absorb  $\vee$  and the universal-over-time operators absorb  $\wedge$ . Its charter member is the GF-disjunction fusion

$$\text{GF}(\varphi \vee \psi) = \text{GF } \varphi \vee \text{GF } \psi$$

(infinitely-often of a union is a union of infinitely-often, by pigeonhole); its siblings are F of a disjunction, G of a conjunction, FG of a conjunction, until with disjointed targets

$$(\sigma \text{ U } \tau_1) \vee (\sigma \text{ U } \tau_2) = \sigma \text{ U } (\tau_1 \vee \tau_2)$$

(the target position is existential), until with conjoined safety arguments

$$(\sigma_1 \text{ U } \tau) \wedge (\sigma_2 \text{ U } \tau) = (\sigma_1 \wedge \sigma_2) \text{ U } \tau$$

(the safety position is universal), and the bounded-horizon ( $\leq T$ ) siblings of the four fusions that carry a horizon — the  $\leq T$  decorations carry through the witness arithmetic verbatim, while FG has no bounded row: its settling time is unbounded by construction. The absorption boundary is exact, and the fence is theorem-backed rather than asserted: the four wrong-polarity fusions — F of a conjunction, G of a disjunction, and their GF/FG counterparts — are machine-refuted by explicit  $\omega$ -word counterexample theorems in both provers (`no_f_and_fusion`, `no_g_or_fusion`, `no_gf_and_fusion`, `no_fg_or_fusion`). A chain is a typed citation: rule, position, direction, step by step; it concludes the judgment `QSpecEquiv(a, b)` below. The concluded judgment is already a row species of the normative schema; the chain citation's own row species is owed there, and until it lands a chain is a wire artifact of the producer surface.

All eighteen rules are proved in Lean *and* in HOL4, statement for statement, with no incomplete proofs; completeness is proven for the benchmark liveness fragment — every spec of that suite's grammar reaches the consumable normal form, witnessed mechanically by all 219 verbatim specs normalizing and re-validating in an independent checker. Carrier genericity varies by rule family: eight of the eighteen rules are per-position Boolean and duality laws whose equalities hold over any linearly ordered time carrier; the nine fusion rules beyond GF — five unbounded, four bounded-horizon — are finite-witness manipulations whose equalities hold over any unbounded linear time carrier; the GF-fusion remains the only pigeonhole rule, needing infinitely-often as cofinal occurrence; the next-step duality carries the step-structure requirement of its operator, which enters the row grammar together with that operator. The machine-checked proofs are the  $\omega$ -word instances; the carrier-generic restatements are small named obligations. One further restatement belongs to this chapter's own compile, and its ground floor is now machine-checked (`SuFunctor.Universe.SpecRewrite`): soundness at the universe means the compiled spec classes of a rule's two sides coincide, and that warrant is strictly stronger than denotation equality — a disjunction and an atom of the same denotation compile to different classes, because models may hedge (`exists_denEq_not_classEq`, the compiled image of the hedging gap) — so class-level soundness travels by term structure: the class semantics is a homomorphism of the normal-form lattice (commutation, association, distribution, and — against the hedging intuition — absorption, each landed as a class equality), and the normal-form dualities are sound for the sharper reason that they do not move the term at all (`classEquiv_of_nnf_eq`); the obstruction lives at the region-to-class seam and nowhere else. That development settles the state stratum. The eighteen temporal rules' own class-level restatement, one row up over the path formers' compile, remains an obligation this edition names rather than claims.

The producer surface *constructs* chains — it deliberately does not apply rewrites. The checker replays the chain; a chain the constructor accepts is a well-formed citation, not a verified equivalence; an empty chain refuses (reflexivity is not a chain). Application code is withheld on purpose while the deadline-fragment

deductive calculus (bounded-reach/always equivalences over clocked classes) remains live research: frozen application rows would re-mint the sidecar disease one level up.

Cross-carrier *application* is not a rewrite matter at all: a step-structured spec applied to a continuous-time system is expressed by composing the system with a sampler mechanism and asserting the spec of the sampled composite, with claims related across the pair by the simulation-transport rule of chapter 15 — zero new machinery. The certified LunarLander pair (one value table, both tick semantics, jitter priced at a factor 2.66 in ticks-to-funnel) is the landed witness of this pattern.

*Witness*: 219 rewrite-chain artifacts asserting spec equivalence for the liveness suite — the demand that shaped the row form quoted below.

## API

### TypeScript spec.rewriteChain

signature — ts/src/spec.ts

```
export function rewriteChain(
  a: SpecRef,
  b: SpecRef,
  steps: ReadonlyArray<RewriteStep>,
): Result<Artifact, SpecRefusal>
```

construction — ts/tests/spec-native.test.ts

```
const art = s.unwrap(
  s.rewriteChain(j.specRef("spec-as-stated"), j.specRef("spec-normal-form"), [
    {
      rule: "implicationElim",
      at: [{ kind: "specForall", i: 0 }],
      dir: "forward",
    },
    {
      rule: "gfDisjunctionFusion",
      at: [{ kind: "specForall", i: 0 }, { kind: "andArg", i: 1 }],
      dir: "forward",
    },
  ]),
);
```

### Rust mtk\_catalogue::spec::rewrite::RewriteChainV0::new

signature — crates/mtk-catalogue/src/spec/rewrite.rs

```
pub fn new(a: SpecRef, b: SpecRef, steps: Vec<RewriteStep>) -> Result<Self, Refusal>

pub fn concludes(&self) -> QSpecEquiv
```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```
let chain = RewriteChainV0::new(
  SpecRef("spec-as-stated".into()),
  SpecRef("spec-normal-form".into()),
  vec![
    RewriteStep {
      rule: RewriteRule::ImplicationElim,
      at: FormulaPath(vec![Step::SpecForall(0)]),
      dir: Direction::Forward,
    },
    RewriteStep {
      rule: RewriteRule::GfDisjunctionFusion,
      at: FormulaPath(vec![Step::SpecForall(0), Step::AndArg(1)]),
      dir: Direction::Forward,
    },
  ],
)
.unwrap();
let concluded = chain.concludes();
```

## 14.9 Deliberate Absences

A catalogue that only listed what exists would invite spelling every missing claim species as prose — the sidecar disease by another door. This section records what the formula layer deliberately does *not* provide, why, and what would unlock each. In rule form an absence is the absence of a formation rule (or of a compile clause for a formable term), and several are enforced in code, as typing-preflight refusals that name their fence; the negation-residue absences additionally carry the no-expressible-complement warrant of section 14.2 — they are theorems about the class algebra, not prudence. The admission rule stands: each of these gains its former when its first certificate lane arrives, and this catalogue is the registry where that admission is recorded.

### 14.9.1 Event-rate assume/guarantee; the probabilistic budget form

#### Absence

The expectation-envelope atoms of the rate family are staged (section 14.6.4); this slot keeps what remains absent, which is everything that crosses a fused boundary. Rate *assumptions* and *guarantees* as interface fields — a producer's guaranteed output envelope becoming a fused neighbour's assumed input envelope, with the transported bound recomputed by the checker as affine arithmetic in  $(\rho, \sigma)$  — wait on the event-port vocabulary of the system side and enter with the first fused-boundary composite, together with their composition rule and its acyclicity oracle. The *probabilistic* budget form — a time-uniform claim that the count never overdraws its budget, with probability at least  $p$  — is sugar over a token-bucket monitor mechanism fused into every counted firing, riding the safety kernel unchanged; it waits on the same fusion vocabulary, and the hard  $p = 1$  envelope with it. Cyclic rate reasoning across a feedback loop is refused even then: naive fixpoints are unsound, the sound discharge is induction over time, and that rule enters on its own warrant or not at all. None of this fences the staged atoms or the count spine — what this absence still fences is boundary-crossing rate reasoning and probabilistic budgets, not counts and not expectation envelopes.

#### API

##### TypeScript ---

No constructor exists for the boundary-crossing forms; the staged atoms carry the family's admission (section 14.6.4), and this entry is the residue's registry slot.

##### Rust ---

Likewise.

### 14.9.2 Interior upper-bound consumption; Rabin; negated Streett and until

#### Absence

Formation of probability upper bounds is open (SPROBLE above): the direct  $P_{\leq q}$  claim is a closed constraint of the same species as the GE face, and fencing its formation as if it were a complement over-read the negation fence. What remains absent is narrower, and two different things. *Consumption* of an upper bound at interior  $q$  over an under-determined model needs certificate towers that *over*-approximate denotations — the dual of the entire under-approximation machinery this stack ships; the exact lane discharges the  $q = 0$  instance today (the same backward-induction table, read in the super-solution direction), and interior  $q$  waits for the tower. Rabin acceptance (the negation of Streett) and path-negated Streett or bounded-until remain genuinely fenced by the negation residue: under the properness presupposition no spec class has an expressible complement (section 14.2), so there is no class-level object for those dual rules to conclude into — the missing rules are missing mathematics. The typing preflight still refuses each *negation* residue with a

path address (a complemented probability is lawful only as a Streett request consumed through PairCmpl); the direct LE former needs no refusal, because it was never a negation.

## API

**TypeScript** `spec.proble`

The LE former constructs (`proble`, quoted at its entry); the preflight refusals (`negPathAtomEvenPolarity`, `negStreett`, `negUntil`) remain the surface for the negation residues, whose entry points are quoted under `SNOT`.

**Rust** `mtk_catalogue::spec::ForallSpec::prob_le`

Likewise (`Refusal::NegPathAtomEvenPolarity` and `kin`, with formula-path geometry).

### 14.9.3 Stochastic bounded-always at even parity

## Absence

The former constructs today (`PALwaysT` above); what is absent is stochastic *consumption* at even parity — the same over-approximating tower machinery as the probability upper bounds. Deterministic consumption is immediate now; the spellability of the statement was the point of drafting the former ahead of its full battery.

## API

**TypeScript** ---

The former's constructor is quoted under `PALwaysT`; no additional surface exists for the fenced consumption.

**Rust** ---

Likewise.

### 14.9.4 Unbounded quantitative reach

## Absence

$$F\varphi \rightsquigarrow F_{\leq T}\varphi \quad \text{at a producer-chosen } T, \text{ flagged producer } T$$

No unbounded-reach primitive: the spelling is a producer-chosen deadline, sound as an under-approximation and *flagged as such* in the row, so a certificate reader can always tell a bounded claim made for its own sake from a bounded claim standing in for an unbounded one. A complete almost-sure-termination battery is the named unlock for the primitive form.

## API

**TypeScript** `spec.derived.eventuallyProducerHorizon`

signature — `ts/src/spec.ts`

| `eventuallyProducerHorizon(horizon: QLike, arg: StatePred): PathPred`

definition — `ts/src/spec.ts`

```

/** The `producerT` spelling of unbounded reach: `F phi` compiled as
 * `F_{<=horizon} phi` at a producer-chosen horizon – a sound
 * under-approximation, flagged as such in the row. */
eventuallyProducerHorizon(horizon: QLike, arg: StatePred): PathPred {
  return deadline(horizon, arg, { producerT: true });
},

```

**Rust** `mtk_catalogue::spec::derived::eventually_producer_horizon`

signature — `crates/mtk-catalogue/src/spec/derived.rs`

```
pub fn eventually_producer_horizon(horizon: WitRat, arg: StatePred) -> Result<PathPred, Refusal>
```

definition — `crates/mtk-catalogue/src/spec/derived.rs`

```

/// The `producerT` spelling of unbounded reach: `F phi` compiled as
/// `F_{<=horizon} phi` at a producer-chosen horizon – a sound
/// under-approximation, *flagged as such* in the row (Wing F keeps
/// unbounded quantitative reach out of the primitive grammar; S7.3).
pub fn eventually_producer_horizon(horizon: WitRat, arg: StatePred) -> Result<PathPred, Refusal> {
  PathPred::deadline(horizon, arg, true)
}

```

## 14.9.5 Occupation-time and duration operators

### Absence

Narrowed by the theory revision: much of what this absence used to fence is now spellable compositionally. Integrated occupation time is the time integral of a region’s indicator (`poInt` over `oInd`), the long-run occupation fraction is its average (`poAvg` over `oInd`), and dwell-metered acceptance is a former of its own (`PDWELLSTREETT`). What remains absent is *dedicated* sojourn and per-visit-duration operators — claims about individual visits rather than accumulated occupation — withheld pending a kernel theorem: occupation-measure mechanics are the natural substrate, and a former without its kernel would be a claim nothing can certify. The compositional spellings above share the constraint at one remove: their certificate lanes are named, not yet claimed.

### API

#### TypeScript ---

The compositional spellings construct on the landed injection and accumulator surfaces (quoted at their entries); the dedicated operators remain withheld pending the kernel theorem — nothing further to construct.

#### Rust ---

Likewise.

## 14.9.6 Nested path quantification

### Absence

The descending grid arrows: path-quantified material re-entering the atemporal row as a state label —  $\mathbb{P}_{\geq q}[\psi]$  as a state predicate, a value functional as a state observable — the CTL/PCTL nesting style. This is the grid’s one deliberate absence (section 14.2), and its warrant has two layers. The *raw* formers are fenced by the geometry: a single level of re-entry exits the constructible classes (neither face’s rooted class is a crescent), and the negation fence forbids deriving the universal face from the existential one — raw nesting is not a missing feature but a shape the class algebra refuses, and any future admission must carry both faces as primitives (the analysis is mechanized in the adequacy layer’s path-to-state development). The



*principled* route — certified-citation formers, the tower discipline promoted to a formation rule, Observable column first — is semantically supported (the behavioural tier already denotes the descending arrows) and deliberately out of scope for this edition: planned for the language’s second major revision (2026Q4), a scoped fence chosen with the open pathway in full view. The certified corpus’s nesting needs are served through the judgment stratum — a child claim’s certified region cited as a parent’s StatePred atom, the tower discipline (entry SPROBGE) — so the absence fences only the syntactic embedding, which no landed artefact ever spelled.

## API

### TypeScript ---

No constructor exists, deliberately; the certified-citation former family is scheduled with the second major revision.

### Rust ---

Likewise.

## 14.9.7 Strategy quantification

### Absence

Controlled and game forms are absent from the formula layer by design: open nondeterminism is absorbed by the credal reading, and policy-level quantification is a judgment-layer and system-side question (chapter 12, the decision-process rows). The formula layer stays closed-system — a strategy quantifier would bind at the wrong stratum.

## API

### TypeScript ---

No strategy quantifier exists in this namespace, by design.

### Rust ---

Likewise.

## 14.9.8 Delays

### Absence

Permanently out of the formula layer: a delay is a model-class change (history-dependent state), not a formula-class change. If it arrives, it arrives as a new mechanism species with its own certificate shapes on the system side — never as a former here.

## API

### TypeScript ---

Out of scope for the formula layer, permanently.

### Rust ---

Likewise.

## 14.10 Judgment Kinds

A *judgment kind* is a  $\models$ -side sort — the type a certificate’s rules conclude into — and each kind carries its own small sub-logic: the premises it takes, the machinery that discharges it (*rule*: a rule node concludes it; *oracle*: a registry-pinned recompute; *chase*: load-time axioms, in the row schema’s three modes — chase proper *derives* rows, report-mode axioms state loud constraints, and refuse-mode axioms gate admission), and the consumers that cite it. The row schema reads this section’s inventory as three verbs: a revision *states* a satisfaction claim (a system paired with a spec), *compiles* it into the two-polarity verdict form (its entry below), and *discharges* it through battery and rule-node rows a checker replays. Homing these at the judgment stratum is a design statement: the formula is about the system; these are about the claim’s kind and evidence. At the semantic universe the satisfaction sort’s meaning is the fibre order itself (SuFunctor.Universe.Sat: the spec’s belief lies below the system’s), with the properness presupposition carried structurally, per section 14.1; the splice of the certified checker’s acceptance judgment into that order is named in the assurance ledger rather than claimed here — its stated prerequisite, the certificate schema’s absorption of this chapter’s redesign, has landed. The citation discipline that every kind below leans on — a premise accepts base-tower proof leaves, or concluded-judgment references of a stated kind, or certificate data, and never a mixture — is deliberately not an entry here: it is the edge discipline whose warrant is the induction over admitted derivations itself, and it renders at the end of chapter 15 as the soundness theorem the kinds feed.

Producer-side, every kind below is a versioned wire artifact (a literal schema name and version, exact rationals as canonical text, unknown fields ignored on read), emitted through the one canonical encoder. What is deliberately *not* here, recorded so nobody re-adds it: no second sort for interface-indexed satisfaction (see the next two entries); no tower-step kind (its content is a region satisfaction consumed through a premise field); no mixing-rate kind (named open); no attitude judgment kind (the attitude blend is a record component that no rule concludes and nothing consumes — the closing entry).

### 14.10.1 QSat: root satisfaction

#### Formation

$$\frac{s \text{ Spec} \quad M \text{ System} \quad \text{states}(M) \simeq \text{wire object of } s}{M \models s \text{ Judgment}} \text{ Q}_{\text{SAT}}$$

Root satisfaction, closed *and* interface-indexed: one judgment kind for both readings. A spec with no interface row is closed; with one, the assumption region reads input wires, the guarantee reads output wires, and the spec’s interface must equal the system’s presented interface (an agreement axiom, chase-policed at the pairing). The formerly mooted second sort for interface-indexed satisfaction is *dissolved*: the closed judgment is the empty-interface instance, so one kind suffices and the loader cannot be handed a claim whose reading is ambiguous.

Under the properness presupposition a root satisfaction claim asserts *safe and proper*: to satisfy anything, first exist. The conjunct is the judgment’s presupposition, carried by the compile target’s type (SpecAtom.top\_mem; emitted unconditionally, elabSpec\_top\_mem) — zero syntax cost, one nonemptiness discharge per root claim, discharged at use sites, with emptiness certificates the optional schema slot. This is the spec-side face of the guard whose system-side face reads an inconsistent system as satisfying all safety specs and no liveness specs (top\_notMem\_modelClass; ex falso inverts structurally, never by a rule). Sub-logic: concluded from a region-satisfaction row plus exact rational evaluation on the initial region (the satisfaction-from-region rule of chapter 15); transported across wirings with slack arithmetic by the assume-guarantee composition rules.

One satisfaction-stratum variant deserves its sentence here, because it is certificate machinery and must not be mistaken for formula grammar: a *guarded* satisfaction claim conditions the guarantee on an initial-region antecedent — demanded of every admitted law whose initial marginal is supported in the cited region, with nothing promised off-region. The guard is a predicate *on* the subject’s initial data, not initial data

(the record still binds no model), and the class algebra stays complement-free: the antecedent is a ground condition on system data, never a behaviour class, which is why no implication former exists below the cap. Its introduction rule discharges the initial-membership hypothesis *into* the claim; its record row cites the certificate-side region vocabulary and lives with the certificate sections of the normative theory — the subject strata stay certificate-citation-free by the standing deliberate absence — and its introduction and elimination are Rule Catalogue content.

*Witness*: every root certificate; the composition chapters' worked examples exercise the interface-indexed reading end to end.

## API

### TypeScript judgment.sat

signature — ts/src/judgment.ts

```
export function sat(args: { sys: SysRef; spec: SpecRef }): Result<
  Artifact,
  JudgmentRefusal
>
```

construction — ts/tests/spec-native.test.ts

```
s.unwrap(j.sat({ sys: j.sysRef("sys-a"), spec: j.specRef("spec-a") })),
```

### Rust mtk\_catalogue::judgment::QSat::new

signature — crates/mtk-catalogue/src/judgment/kinds.rs

```
pub fn new(sys: SysRef, spec: SpecRef) -> Self
```

construction — crates/mtk-catalogue/tests/roundtrip\_spec.rs

```
let sat = QSat::new(SysRef("sys-a".into()), SpecRef("spec-a".into()));
```

## 14.10.2 Interface-indexed satisfaction, defined

### Definition

For a system  $\Sigma$  and an interface spec  $(I, \alpha, \gamma[\cdot, \kappa])$  — assumptions  $\alpha$  on input wires, guarantee  $\gamma$  on output wires, optional slack  $\kappa$  — the judgment  $\Sigma \models_I (I, \alpha, \gamma[\cdot, \kappa])$  asserts that every behaviour of  $\Sigma$  whose inputs conform to  $\alpha$  delivers  $\gamma$  (with margin per  $\kappa$  where declared); in the stochastic profiles, almost every behaviour, with failure measured *within the conforming event*:

$$\forall^\mu \omega, \omega \in \text{conform} \rightarrow \omega \in \text{sat}; \quad \text{quantitatively} \quad \mu(\text{conform} \cap \text{sat}^c) \leq \varepsilon.$$

Unlike the closed judgment  $\mathcal{M} \models s$ , which quantifies unconditionally over all behaviours from the initial region,  $\models_I$  quantifies conditionally — satisfaction relativized to the interface; the closed judgment is its empty-interface instance. The subscript is a roman kind-marker, not a parameter, following the global-stratum marker  $\models_D$  (entry QMORSE).

This reading is defined once, here, and has *no schema sort of its own*: its home is QSat whose spec carries an interface; assumptions and slack live on the spec record; interface agreement is the pairing's chase axiom. It earns an entry because it is a genuinely different satisfaction relation — conditional, with failure measured inside the conforming event — and because the composition rules' conclusions are exactly of this shape. Its mechanization status: the sequent substrate for the satisfaction judgment carries the properness conjunct as a theorem (the pair-sequent development, SuFunctor.Universe.PairSequent), and the record-level compile carries assumptions as data with their conditional semantics deferred fail-closed (section 14.3.1); the relativized reading stated at the semantic universe is a named obligation, not yet claimed.

## API

**TypeScript** ---

Carried by `judgment.sat` with an interface-carrying spec — deliberately no separate entry point.

**Rust** ---

Carried by `QSat` likewise; the dissolved second sort is recorded in the module documentation so nobody re-adds it.

**14.10.3 The compiled verdict form****Definition**

$$\text{QVerdictForm} : [\text{floors} : \text{QRegionSat}^*, \text{marks} : \text{QRegionSat}^*]$$

A compiled satisfaction verdict carries its demand atoms in *two* lists, and the split is schema content because it is not recoverable from the atoms alone: the same constraint species denotes differently per face. A *floor* holds over the whole credal body; a *mark* asserts that some admitted law attains — so folding the two into one list would erase exactly the distinction the denotation reads. Polarity is *declared* at formation — the cap sorts of section 14.2 — and *discharged* here: one home per stratum, connected by the compile. No battery or region sort carries a polarity flag; the two lists are the verdict-side home, with the floors compiled from the `ForallSpec` components and the marks from the `ExistsSpec` list plus the unconditional properness mark — attitude components take no part in the form, because a design score is not a verdict — and the form’s agreement with the declared faces is part of the recompute obligation below. The mark list is the verdict’s existential conjunct, first-class: an *empty* mark list is the all-safety claim, true of the zero-mass subject, and a form that derived liveness from floor content would silently certify a dead subject live — which is why the record-level compile emits the properness mark unconditionally (`elabSpec_top_mem`, section 14.3.1), so a form compiling a root specification carries a nonempty mark list. Liveness demands enter the mark list as *negation walls*: the demand “at least  $q$ ” enters as the mark of its complement’s failure — the wall/dip pair of the compile target (section 14.1), as rows. The form is value-like; its attachment to a root claim is a zero-or-one relation (a claim with no form is simply not compiled); and the agreement between the form and the specification it compiles is a *recompute obligation* at the ground seam, never a trusted row: the checker re-derives the form from the spec it cites.

*Witness*: the staged certificate probes of the normative theory’s battery; corpus-wide re-emission follows the producer-side emitter, which has not landed.

**API****TypeScript** ---

No constructor is exposed: the form is compiled, not authored, and no producer-side emission exists yet.

**Rust** ---

Likewise.

**14.10.4 QSatConf: the confidence qualifier****Definition**

$$\text{QSatConf} : [\text{sys} : \text{System}, \text{spec} : \text{Spec}, \alpha : \mathbb{Q}], \quad 0 < \alpha < 1$$

The spec holds *up to the evidence's error budget*: the claim is backed by a betting-martingale ledger whose wealth crossed  $1/\alpha$ , and soundness is time-uniform — Ville's inequality [234] for nonnegative supermartingales plus a randomized-last-look refinement, both machine-checked in the Lean auxiliary tree. This is a *distinct kind*, not a field on QSat: confidence-qualified rows are never silently mixed with exact or almost-sure rows — a sort distinction the loader cannot blur, rather than a rendering convention. Sub-logic: rule (ledger rows in, judgment out); the range demand on  $\alpha$  is a report-mode constraint at load, both ends strict ( $\alpha = 0$  would claim the exact tier in statistical clothes;  $\alpha = 1$  claims nothing). Ledger emission belongs to the statistical harness on the system side; this row is the claim it backs.

*Witness*: the LunarLander statistical rows —  $\mathbb{E}[R] \geq 236.577$  at  $\alpha = 1/20$  over  $10^6$  episodes, checked at nanoseconds per obligation; the continuous variant's certified floor exceeding the discrete variant's empirical mean.

## API

### TypeScript judgment.satConf

signature — ts/src/judgment.ts

```
export function satConf(args: {
  sys: SysRef;
  spec: SpecRef;
  alpha: QLike;
}): Result<Artifact, JudgmentRefusal>
```

construction — ts/tests/spec-native.test.ts

```
const art = s.unwrap(
  j.satConf({
    sys: j.sysRef("lunarlander-cont"),
    spec: j.specRef("return-floor-236577/1000"),
    alpha: Q.of(1, 20),
  }),
);
```

### Rust mtk\_catalogue::judgment::QSatConf::new

signature — crates/mtk-catalogue/src/judgment/kinds.rs

```
pub fn new(sys: SysRef, spec: SpecRef, alpha: WitRat) -> Result<Self, Refusal>
```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```
let row = QSatConf::new(
  SysRef("lunarlander-cont".into()),
  SpecRef("return-floor-236577/1000".into()),
  q(1, 20),
)
.unwrap();
```

## 14.10.5 QSpecEquiv: spec equivalence

### Definition

$$\text{QSpecEquiv} : [a : \text{Spec}, b : \text{Spec}]$$

Two specs denote the same property over every behaviour carrier on which both are interpretable. The judgment stores *no semantics-class field*, deliberately: a stored three-way class enum misstates the structure (an  $\omega$ -word is a càdlàg trajectory under the piecewise-constant-at-integer-times embedding, so the classes are not disjoint). Instead, **carrier scope is derived, never stored**: it is the carrier-requirement synthesis judgment of section 14.2 — each former's  $\triangleright$  annotation — with an equivalence judgment's scope the intersection of its two sides' requirements. An empty intersection is a conformance refusal at the consumer,

not a judgment. The Return stratum’s split horizons are this distinction already encoded: tick-summed returns demand steps, integrated returns demand duration. Sub-logic: concluded by the rewrite layer’s chains (rule); the derived-requirement computation is part of the producer surface, quoted below the tour entries.

*Witness:* the 219 rewrite-chain artifacts of the liveness suite, each concluding exactly this judgment.

## API

### TypeScript judgment.specEquiv

signature — ts/src/judgment.ts

```
export function specEquiv(args: { a: SpecRef; b: SpecRef }): Result<
  Artifact,
  JudgmentRefusal
>
```

construction — ts/tests/spec-native.test.ts

```
s.unwrap(j.specEquiv({ a: j.specRef("spec-a"), b: j.specRef("spec-b") })),
```

### Rust mtk\_catalogue::judgment::QSpecEquiv::new

signature — crates/mtk-catalogue/src/judgment/kinds.rs

```
pub fn new(a: SpecRef, b: SpecRef) -> Self
```

construction — crates/mtk-catalogue/tests/roundtrip\_spec.rs

```
let equiv = QSpecEquiv::new(SpecRef("spec-a".into()), SpecRef("spec-b".into()));
```

## 14.10.6 QRegionSat: the workhorse

### Definition

$$\text{QRegionSat} : [\text{formula} : \text{StatePred}, \text{region} : \text{QRegionAtom}] \quad \forall x \in R : x \models_{\mathcal{M}} \varphi$$

with the region-atom vocabulary

$$\text{qraSublevel}(fam, \theta) \mid \text{qraBox}(\text{intervals}) \mid \text{qraAffine}\left(\sum_i c_i V_i \leq \theta\right).$$

Every state in the cited region satisfies the formula, with the kernel region *under*-approximating the formula’s denotation — one orientation suffices, since complement consumption inherits the over-approximation for free. The region atoms are how certified regions are *cited*: a sublevel set of a certified function family, an exact-rational box, or a sublevel of an affine combination of certified families. Sub-logic: concluded by the battery rules (safety, deadline, until, Streett), basins, union bounds, the NNF alias family, and region-combination nodes; battery well-formedness policed by oracles; consumed by the tower-citation edge discipline (a child’s concluded region is a parent’s atom) and by satisfaction-from-region.

*Witness:* the entire supermartingale corpus; the three quoted constructions are the committed vectors for the three atom variants.

## API

### TypeScript judgment.regionSat

signature — ts/src/judgment.ts



```
export function regionSat(args: {
  formula: StatePred;
  region: RegionAtom;
}): Result<Artifact, JudgmentRefusal>
```

construction — ts/tests/spec-native.test.ts

```
const phi = s.polyLe(0n, s.unwrap(s.polyVar(1, 2)));

const sub = s.unwrap(
  j.regionSat({
    formula: phi,
    region: j.sublevel(j.certFamRef("Lyap-v1"), 0.of(7, 5)),
  }),
);
```

The box and affine region-atom variants ride sibling committed vectors in the same tests.

**Rust** mtk\_catalogue::judgment::QRegionSat::new

signature — crates/mtk-catalogue/src/judgment/kinds.rs

```
pub fn new(formula: StatePred, region: RegionAtom) -> Self
```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```
let phi = StatePred::poly_le(q(0, 1), Poly::var(1, 2).unwrap());
let sublevel = QRegionSat::new(
  phi.clone(),
  RegionAtom::Sublevel {
    cert: CertFamRef("Lyap-v1".into()),
    thr: q(7, 5),
  },
);
```

The box and affine region-atom variants ride sibling committed vectors in the same tests.

## 14.10.7 QCapSat: the cap-on-region vehicle

### Definition

$$\text{QCapSat} : [\text{cap} : \text{ForallSpec}, \text{region} : \text{QRegionAtom}] \quad \forall x \in R : x \models_{\mathcal{M}} \text{cap}$$

A probability cap certified from a region: every state in the cited region satisfies the safety-face component, with the region *under*-approximating the cap's denotation — the workhorse's orientation, one entry up the formula grid. The kind entered at revision event 4 as a deliberately *interim* vehicle: the previous edition nested the probability former under the region workhorse's state formula, the formation grid makes that shape unrepresentable (a cap is a component, never a state predicate), and so the cap-discharging rules — batteries, transports, reach tables — conclude this kind instead. Revision event 6 discharges the interim framing. The normative theory stores those conclusions as attributes of the battery proof nodes themselves and ties caps to spec components by keying each satisfaction claim to the component's own entity row: `qsat_warrant` relates the claim to a `QForallSat` row carrying the component and its region. The producer's rule rows are re-staged against that seam — warrants are emitted entity-keyed, under a versioned species binding that refuses pre-revision closures at the seam, loudly, at derivation time. What no producer emits yet is the compiled verdict form (section 14.10.3 records the same absence), so this kind remains the typed conclusion the cap-discharging rules place on the wire.

*Witness:* the committed vector (the quoted construction, both languages byte-identical); the battery conclusions across the corpus exercise the same shape.

### API

**TypeScript** judgment.capSat

signature — ts/src/judgment.ts

```
export function capSat(args: {
  cap: ForallSpec;
  region: RegionAtom;
}): Result<Artifact, JudgmentRefusal>
```

construction — ts/tests/spec-native.test.ts

```
s.unwrap(
  j.capSat({
    cap: s.probGe(Q.of(9, 10), s.always(phi)),
    region: j.sublevel(j.certFamRef("Lyap-v1"), Q.of(1, 10)),
  }),
),
```

Entered at revision event 4 as the interim typed conclusion for the cap-discharging rules (batteries, transports, reach tables); at revision event 6 the warrant seam is entity-keyed and the producer's rule rows are re-staged against it, and this kind remains the typed conclusion those rules place on the wire pending a verdict-form producer.

**Rust** mtk\_catalogue::judgment::QCapSat::new

signature — crates/mtk-catalogue/src/judgment/kinds.rs

```
pub fn new(cap: ForallSpec, region: RegionAtom) -> Self
```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```
let phi = StatePred::poly_le(q(0, 1), Poly::var(0, 1).unwrap());
let row = QCapSat::new(
  ForallSpec::prob_ge(q(9, 10), PathPred::always(phi)).unwrap(),
  RegionAtom::Sublevel {
    cert: CertFamRef("Lyap-v1".into()),
    thr: q(1, 10),
  },
);
```

Entered at revision event 4 as the interim typed conclusion for the cap-discharging rules (batteries, transports, reach tables); at revision event 6 the warrant seam is entity-keyed and the producer's rule rows are re-staged against it, and this kind remains the typed conclusion those rules place on the wire pending a verdict-form producer.

**14.10.8 QInvScope: invariance with scope****Definition**

$$\text{QInvScope} : [\text{region} : \text{QRegionAtom}, \text{init} : \text{QRegionAtom}] \quad S \text{ forward-invariant} \wedge X_0 \subseteq S$$

The region is forward-invariant and contains the initial set. Sub-logic: concluded by the block-invariance rule (strictly facet-inward cone rows plus containment) and its lattice alias; consumed by basins, by battery scope-warrant fields, and by the aggregate-scope kind's invariance route below. *Witness*: the invariance rows across the corpus; the Morse-stratum discharge kits cite it wherever a block's scope must be warranted.

**API****TypeScript** judgment.invScope

signature — ts/src/judgment.ts

```
export function invScope(args: {
  region: RegionAtom;
  init: RegionAtom;
}): Result<Artifact, JudgmentRefusal>
```

construction — ts/tests/spec-native.test.ts

```
| s.unwrap(j.invScope({ region: unitBox, init: j.box([[Q.of(1, 4), Q.of(1, 2)]] )})),
```

**Rust** mtk\_catalogue::judgment::QInvScope::new

signature — crates/mtk-catalogue/src/judgment/kinds.rs

```
| pub fn new(region: RegionAtom, init: RegionAtom) -> Self
```

construction — crates/mtk-catalogue/tests/roundtrip\_spec.rs

```
| let inv = QInvScope::new(
|   unit_box(),
|   RegionAtom::Box(IntervalBox(vec![Interval::new(q(1, 4), q(1, 2)).unwrap()])),
| );
```

## 14.10.9 QMorse: the global stratum

### Definition

$$\text{QMorse} : [\text{graph} : \text{MorseGraph}, \rho : \text{VertexRegion}^*, \text{arena} : \text{Arena}] \quad \mathcal{M} \models_D \text{Morse}(G, \rho)$$

Non-initialized, whole-system qualitative structure: a certified Morse graph with per-vertex readout regions. It is listed in this catalogue because a complete spec-language inventory must say where the non-formula-shaped claims live: the formula language and the global stratum share the region vocabulary and nothing else, and the  $D$  subscript marks the different satisfaction relation. The row asserts the graph and readouts against a declared *arena* — the grid complex the claim is read on; a discharged claim's arena is its lattice's, tied by a stated equation between hash-consed values (the tie is deliberately syntactic — semantic region equality is geometry the certificate does not owe). Acyclicity is the checker's to confirm, never this row's to assert, and the payload refuses out-of-range edge endpoints and vertex indices with the geometry named. Sub-logic: the Morse discharge and product rules of chapter 15, which carry the deepest mechanized ledger in the stack.

*Witness:* the Morse-graph certificate family across the corpus, up to certified graphs at the several-hundred-node scale.

### API

**TypeScript** judgment.morse

signature — ts/src/judgment.ts

```
| export function morse(args: {
|   vertices: number;
|   edges: ReadonlyArray<MorseEdge>;
|   rho: ReadonlyArray<VertexRegion>;
| }): Result<Artifact, JudgmentRefusal>
```

construction — ts/tests/spec-native.test.ts

```
| s.unwrap(
|   j.morse({
|     vertices: 2,
|     edges: [{ src: 0, dst: 1 }],
|     rho: [
|       { vertex: 0, region: phi },
|       { vertex: 1, region: s.TRUE },
|     ],
|   }),
| ),
```

**Rust** `mtk_catalogue::judgment::QMorse::new`

signature — `crates/mtk-catalogue/src/judgment/kinds.rs`

```
pub fn new(graph: MorseGraph, rho: Vec<VertexRegion>) -> Result<Self, Refusal>
```

construction — `crates/mtk-catalogue/tests/roundtrip_spec.rs`

```
let morse = QMorse::new(
  MorseGraph {
    vertices: 2,
    edges: vec![MorseEdge { src: 0, dst: 1 }],
  },
  vec![
    VertexRegion {
      vertex: 0,
      region: StatePred::poly_le(q(0, 1), Poly::var(0, 1).unwrap()),
    },
    VertexRegion {
      vertex: 1,
      region: StatePred::True,
    },
  ],
)
.unwrap();
```

#### 14.10.10 QWireCert: wiring certification

##### Definition

$$\text{QWireCert} : [w : \text{Wiring}, \text{premises} : \text{Spec}^*, \text{out} : \text{Spec}, \kappa : \text{Slack}]$$

The wiring  $w$  is certified for  $(\text{spec}_i) \Rightarrow \text{spec}_{\text{out}}$  with slack  $\kappa$  in the  $\mathcal{K}_{\infty}^0$  class. The premise list is positional against the wiring's inner-system order; lockstep with a consuming composition node's components is record-level conformance. Sub-logic: discharged from proof leaves for the two condition families of the composition calculus (mechanism certification; wiring-lens certification) [37], per region;  $\mathcal{K}_{\infty}$ -membership rows are proof-dischargeable; consumed by the wire-composition rule. The class membership and slack algebra are proven; the payload here freezes the API shape while the row's upstream schema staging follows the system-side work — stated so the reader knows which part is mathematics and which part is plumbing.

*Witness:* the assume–guarantee worked examples of chapter 15.

##### API

**TypeScript** `judgment.wireCert`

signature — `ts/src/judgment.ts`

```
export function wireCert(args: {
  wiring: WiringRef;
  premises: ReadonlyArray<SpecRef>;
  conclusion: SpecRef;
  gain: Poly;
}): Result<Artifact, JudgmentRefusal>
```

construction — `ts/tests/spec-native.test.ts`

```
s.unwrap(
  j.wireCert({
    wiring: j.wiringRef("w-loop"),
    premises: [j.specRef("inner-a"), j.specRef("inner-b")],
    conclusion: j.specRef("outer"),
    gain: s.unwrap(s.poly([ { expts: [1], coeff: 0.of(1, 2) } ])),
  }),
),
```

**Rust** `mtk_catalogue::judgment::QWireCert::new`

signature — `crates/mtk-catalogue/src/judgment/kinds.rs`

```
pub fn new(
    wiring: WiringRef,
    premises: Vec<SpecRef>,
    conclusion: SpecRef,
    slack: Slack,
) -> Self
```

construction — `crates/mtk-catalogue/tests/roundtrip_spec.rs`

```
let wire_cert = QWireCert::new(
    WiringRef("w-loop".into()),
    vec![SpecRef("inner-a".into()), SpecRef("inner-b".into())],
    SpecRef("outer".into()),
    Slack {
        gain: Poly::new(vec![(vec![1u32], q(1, 2))]).unwrap(),
    },
);
```

### 14.10.11 QDist: distance judgments

#### Definition

$$\text{QDist}[d](A, B) \leq \varepsilon \quad d : \text{a declared distance functor (registry entry)}, \varepsilon \in \mathbb{Q}_{\geq 0}$$

The declared distance functor — a registry entry naming subject sort, symmetry flag, licensed rules, and observable meaning — between two subjects (system manifests, distribution snapshots, or definite descriptions with their well-formedness licenses) is at most exact-rational  $\varepsilon$ . Judgment-primary by design: triangle, weakening, and symmetry-per-flag are judgment-level admissibles, while the discharge rules are their own catalogue entries — coupling-witnessed drift, contraction-witnessed fixed-point balls, and the transport cells that degrade a judgment across an approximate simulation with the  $n\varepsilon$  compounding law made explicit. *Schema status*: chartered design — this kind's row species is not yet part of the normative row schema (`theories/Qtsl.geolog`); it enters with its discharge lanes, and until then the entry stands as the registry slot for the demand.

*Witness*: the coupling and contraction fixture families; the finite-kernel transport tier.

#### API

##### TypeScript `judgment.dist`

signature — `ts/src/judgment.ts`

```
export function dist(args: {
    functor: string;
    a: DistSubject;
    b: DistSubject;
    eps: QLike;
}): Result<Artifact, JudgmentRefusal>
```

construction — `ts/tests/spec-native.test.ts`

```
s.unwrap(
    j.dist({
        functor: "wasserstein-1",
        a: { kind: "manifest", sys: j.sysRef("sys-a") },
        b: { kind: "snapshot", locator: "snap-7" },
        eps: Q.of(1, 100),
    }),
),
```

**Rust** `mtk_catalogue::judgment::QDist::new`

signature — `crates/mtk-catalogue/src/judgment/kinds.rs`

```
pub fn new(
    functor: impl Into<String>,
    a: DistSubject,
    b: DistSubject,
    eps: WitRat,
) -> Result<Self, Refusal>
```

construction — `crates/mtk-catalogue/tests/roundtrip_spec.rs`

```
let dist = QDist::new(
    "wasserstein-1",
    DistSubject::Manifest(SysRef("sys-a".into())),
    DistSubject::Snapshot("snap-7".into()),
    q(1, 100),
)
.unwrap();
```

### 14.10.12 The existence family: QZeroExists, QEquilibriumExists, QFixedPointExists

#### Definition

$$\text{QZeroExists} : [g : \text{Poly}^m, \text{cell} : \text{QRegionAtom}] \quad \exists x \in \text{cell} : g(x) = 0$$

with the equilibrium and fixed-point members reading the same shape over a drift field and a map ( $g = F - \text{id}$ , tied by an elaboration recompute downstream).

Poincaré–Miranda member claims: a zero, an equilibrium, a fixed point, *in a cell*. Existence only — nothing in this family concludes uniqueness, count, or non-existence; that refusal boundary is the family’s design. In this edition the cell is a box atom and the subject is equidimensional ( $m$  components over an  $m$ -dimensional cell), refused otherwise with the geometry named. Sub-logic: the shared face-sign battery (rule + pinned oracle), and a derived rule reading an inward invariance box as an equilibrium witness.

*Witness*: the face-sign battery fixtures; the equilibrium readouts of the Morse–stratum examples.

#### API

**TypeScript** `judgment.zeroExists / equilibriumExists / fixedPointExists`

signature — `ts/src/judgment.ts`

```
export function zeroExists(args: {
    g: ReadonlyArray<Poly>;
    cell: RegionAtom;
}): Result<Artifact, JudgmentRefusal>

export function equilibriumExists(args: {
    field: ReadonlyArray<Poly>;
    cell: RegionAtom;
}): Result<Artifact, JudgmentRefusal>

export function fixedPointExists(args: {
    map: ReadonlyArray<Poly>;
    cell: RegionAtom;
}): Result<Artifact, JudgmentRefusal>
```

construction — `ts/tests/spec-native.test.ts`

```
s.unwrap(j.zeroExists({ g: g1, cell: unitBox })),
```



The equilibrium and fixed-point members construct beside it in the same test.

**Rust** `mtk_catalogue::judgment::{QZeroExists, QEquilibriumExists, QFixedPointExists}::new`

signature — `crates/mtk-catalogue/src/judgment/kinds.rs`

```
pub fn new(g: Vec<Poly>, cell: RegionAtom) -> Result<Self, Refusal>

pub fn new(field: Vec<Poly>, cell: RegionAtom) -> Result<Self, Refusal>

pub fn new(map: Vec<Poly>, cell: RegionAtom) -> Result<Self, Refusal>
```

construction — `crates/mtk-catalogue/tests/roundtrip_spec.rs`

```
let zero = QZeroExists::new(gl(), unit_box()).unwrap();
```

The equilibrium and fixed-point members construct beside it in the same test.

### 14.10.13 QSim: exact simulation

#### Definition

$\text{QSim} : [\text{src} : \text{System}, \text{dst} : \text{System}, \sigma : (\mathbb{N} \times \text{Mode} \times \text{Poly})^*] \quad \sigma : S \rightarrow S' \text{ an exact simulation map}$

The refinement preorder, one-sided by design:  $\sigma$  carries the finer system onto the coarser, and simulations compose. The map is stored as flat (coordinate, mode, polynomial) triples — one polynomial per destination coordinate per source mode, the mode cited as an entity rather than by name string, so a piece citing a nonexistent mode is unrepresentable rather than a silent key mismatch. The producer payload carries the same content mode-indexed — the outer list ranges over the source's modes, each inner list is that mode's destination coordinates — matching the transport builder's convention, refusing an empty mode roster and empty per-mode maps. Sub-logic: discharge at the strict tier (compatibility squares as proof leaves plus a mode-coverage walk) and the kernel tier (intertwining of kernels, machine-checked at its pushforward lemma); consumed by the simulation-transport rule, whose simulation premise is an ordinary premise of this kind — no hypothesis asterisk. The approximate  $(\sigma, \varepsilon)$  class deliberately concludes QDist, never this kind: exactness is what makes transport lossless, and blurring the two sorts would launder approximation through an exact-looking judgment.

*Witness:* the simulation-transport worked examples; the LunarLander tick-semantics pair is carried by this transport.

#### API

**TypeScript** `judgment.sim`

signature — `ts/src/judgment.ts`

```
export function sim(args: {
  sigma: ReadonlyArray<ReadonlyArray<Poly>>;
  from: SysRef;
  to: SysRef;
}): Result<Artifact, JudgmentRefusal>
```

construction — `ts/tests/spec-native.test.ts`

```
s.unwrap(
  j.sim({
    // one mode, one destination coordinate
    sigma: [[s.unwrap(s.polyVar(0, 1))]],
    from: j.sysRef("fine"),
    to: j.sysRef("coarse"),
  }),
),
```

**Rust** `mtk_catalogue::judgment::QSim::new`

signature — `crates/mtk-catalogue/src/judgment/kinds.rs`

```
| pub fn new(sigma: Vec<Vec<Poly>>, from: SysRef, to: SysRef) -> Result<Self, Refusal>
```

construction — `crates/mtk-catalogue/tests/roundtrip_spec.rs`

```
| let sim = QSim::new(
|   vec![vec![Poly::var(0, 1).unwrap()]], // one mode, one coordinate
|   SysRef("fine".into()),
|   SysRef("coarse".into()),
| )
| .unwrap();
```

#### 14.10.14 QFlatEq: flattening

##### Definition

$$\text{QFlatEq} : [\text{sys} : \text{System}, \text{flat} : \text{Manifest}] \quad \text{flat} = \text{flatten}(\text{sys})$$

The monolithic manifest is the flattening of the composite. Sub-logic: one oracle-shaped discharge (the substitution and guard-product recompute, registry-pinned, refusing loudly when unpinned), with the shared-carrier and binary-wiring confinements as refusal boundaries — completeness costs, never soundness hypotheses: what the oracle cannot flatten it refuses to speak about, rather than asserting under an assumption. Consumed by the flattening substitution rule. The measurable-tier witness (the parallel-composition law) is machine-checked.

*Witness:* the flattening fixtures of the composition suite.

##### API

**TypeScript** `judgment.flatEq`

signature — `ts/src/judgment.ts`

```
| export function flatEq(args: {
|   composite: SysRef;
|   flat: SysRef;
| }): Result<Artifact, JudgmentRefusal>
```

construction — `ts/tests/spec-native.test.ts`

```
| s.unwrap(j.flatEq({ composite: j.sysRef("comp"), flat: j.sysRef("mono") })),
```

**Rust** `mtk_catalogue::judgment::QFlatEq::new`

signature — `crates/mtk-catalogue/src/judgment/kinds.rs`

```
| pub fn new(composite: SysRef, flat: SysRef) -> Self
```

construction — `crates/mtk-catalogue/tests/roundtrip_spec.rs`

```
| let flat = QFlatEq::new(SysRef("comp".into()), SysRef("mono".into()));
```

#### 14.10.15 QAggScope: aggregate scope

##### Definition

$$\text{QAggScope} : [N : \text{System}, \text{aggdim} : \mathbb{N}, K : \text{PolytopeRow}^*, \text{stop} : \{\text{reachable}, \text{stopped}\}]$$

The aggregate image of every in-scope reachable marking of the net  $N$  lies in the polytope  $K$ , under the declared scope semantics. Reachability is composition-unstable (so not chase) and semantic (so not oracle) — which is why this is a judgment kind with rule discharge: the three confinement routes (cite an invariance-scope row; exit-guard structure, by oracle walk over guard rows; stopped semantics, by declared flag with a conformance tie) each conclude it, alongside P-invariant conservation certificates ( $y \cdot C = 0$ , exact-rational, witness carried) and capacity rows. Consumed by the aggregation-reduction rule as an ordinary premise. *Schema status*: designed, not yet rows — the aggregation-reduction node's row kits are in the normative schema; the scope kind's own row species joins them when its rows land.

*Witness*: the structurally bounded net families of the aggregation suite.

## API

### TypeScript judgment.aggScope

signature — ts/src/judgment.ts

```
export function aggScope(args: {
  net: SysRef;
  aggDim: number;
  polytope: ReadonlyArray<{ coeffs: ReadonlyArray<QLike>; ub: QLike }>;
  stop: ScopeSemantics;
}): Result<Artifact, JudgmentRefusal>
```

construction — ts/tests/spec-native.test.ts

```
s.unwrap(
  j.aggScope({
    net: j.sysRef("net-a"),
    aggDim: 2,
    polytope: [{ coeffs: [1n, 1n], ub: 10n }],
    stop: "reachable",
  }),
),
```

### Rust mtk\_catalogue::judgment::QAggScope::new

signature — crates/mtk-catalogue/src/judgment/kinds.rs

```
pub fn new(
  net: SysRef,
  agg_dim: usize,
  polytope: Vec<PolytopeRow>,
  stop: ScopeSemantics,
) -> Result<Self, Refusal>
```

construction — crates/mtk-catalogue/tests/roundtrip\_spec.rs

```
let agg = QAggScope::new(
  SysRef("net-a".into()),
  2,
  vec![PolytopeRow {
    coeffs: vec![q(1, 1), q(1, 1)],
    ub: q(10, 1),
  }],
  ScopeSemantics::Reachable,
)
.unwrap();
```

## 14.10.16 QStateFinite: a classification judgment

### Definition

$\text{QStateFinite} : [\text{sys} : \text{System}]$     the subject's state type is finite

A *classification* judgment: a class membership of the rule’s subject, not a per-application fact. Sub-logic: chase-derivable by structural refinement over the state-type formers (enumerations are finite; finite products of finite factors are finite; nesting inherits; the tensor former derives nothing, a deliberate negative), with a decidable oracle as fallback. Consumed where a discharge rule needs finiteness as a premise (the coupling-drift finite tier), which retires a side-condition asterisk into an ordinary premise; a future countable-tier proof targets a weaker classification, display untouched. *Schema status*: designed, not yet rows in the normative schema; the classification enters with its chase-refinement family.

*Witness*: the finite-tier coupling fixtures.

## API

### TypeScript judgment.stateFinite

signature — ts/src/judgment.ts

```
export function stateFinite(args: { sys: SysRef }): Result<
  Artifact,
  JudgmentRefusal
>
```

construction — ts/tests/spec-native.test.ts

```
s.unwrap(j.stateFinite({ sys: j.sysRef("sys-a") })),
```

### Rust mtk\_catalogue::judgment::QStateFinite::new

signature — crates/mtk-catalogue/src/judgment/kinds.rs

```
pub fn new(sys: SysRef) -> Self
```

construction — crates/mtk-catalogue/tests/roundtrip\_spec.rs

```
let fin = QStateFinite::new(SysRef("sys-a".into()));
```

## 14.10.17 QLinGrowth: a growth-class judgment

### Definition

$$\text{QLinGrowth} : [f : \text{Poly}^*, a, b : \mathbb{Q}_{\geq 0}, \text{scope} : \text{Box}] \quad \|f(x)\| \leq a\|x\| + b \text{ on the scope}$$

Membership in the linear-growth class with the *constants carried in the judgment*, so downstream cells consume  $(a, b)$  arithmetically rather than a membership bit. The kind is load-bearing because the fence it polices is provably sharp: the transport law it feeds is *false* outside the fragment (the  $n\varepsilon$  compounding of the distance-transport cell is sharp without a contraction row), so growth control must arrive as evidence, not assumption. Sub-logic: certificate-carrying — affine-envelope and Bernstein rows discharge it for polynomial drifts; composition and weakening admissibles do constant arithmetic.

*Witness*: the distance-transport pushforward cells’ premise rows.

## API

### TypeScript judgment.linGrowth

signature — ts/src/judgment.ts

```
export function linGrowth(args: {
  f: ReadonlyArray<Poly>;
  a: QLike;
  b: QLike;
  scope: RegionAtom;
}): Result<Artifact, JudgmentRefusal>
```

construction — ts/tests/spec-native.test.ts

```
| s.unwrap(  
|   j.linGrowth({ f: g1, a: 2n, b: Q.of(1, 2), scope: unitBox })),  
| ),
```

**Rust** mtk\_catalogue::judgment::QLinGrowth::new

signature — crates/mtk-catalogue/src/judgment/kinds.rs

```
| pub fn new(f: Vec<Poly>, a: WitRat, b: WitRat, scope: RegionAtom) -> Result<Self, Refusal>
```

construction — crates/mtk-catalogue/tests/roundtrip\_spec.rs

```
| let growth = QLinGrowth::new(g1(), q(2, 1), q(1, 2), unit_box()).unwrap();
```

### 14.10.18 The attitude blend: SHurwiczGE and SHurwiczLE (not a judgment kind)

#### Formation

$$\frac{\alpha \in \mathbb{Q} \quad 0 \leq \alpha \leq 1 \quad B \in \mathbb{Q} \quad R \text{ PathObservable}}{\text{SHurwiczGE}(\alpha, B, R) \text{ AttitudeSpec}} \text{SHURWICZGE}$$

$$\frac{\alpha \in \mathbb{Q} \quad 0 \leq \alpha \leq 1 \quad B \in \mathbb{Q} \quad R \text{ PathObservable}}{\text{SHurwiczLE}(\alpha, B, R) \text{ AttitudeSpec}} \text{SHURWICZLE}$$

Certification is pessimism; design exploration wants the other end too, and occasionally a weighted blend — the Hurwicz optimism–pessimism criterion, transposed. Where the credal reading leaves a *set* of values at a run-functional, the blend mixes the two envelope readings with a declared attitude weight:  $\alpha = 0$  is the pure worst case (SReturnGE’s floor face, exactly),  $\alpha = 1$  the pure best case (SReturnMayBeGE’s witness face, exactly), and an interior  $\alpha$  mixes the two envelopes *arithmetically* into one scalar before thresholding, in checker-recomputed rational arithmetic. The pair therefore fits *neither* cap face, which is why it has its own sort instead of a mis-declared face — and the contrast with the record deserves its precise statement, because a specification is not envelope-pure either. A specification is *polarized*: every component carries a declared face, so its compiled verdict is truth-functional over per-component single-envelope discharges, monotone separately-per-slot with fixed sign, and verdict-mixing commutes with composition. The interior blend’s scalar has no sign in the refinement order — it mixes a quantity that shrinks under new admitted laws with one that grows — and it fails the marginal-extension tower property and equivalence with any single envelope bound (exact-rational countermodels are kernel-checked, both faces). In one line: **a Spec mixes verdicts; an attitude mixes envelopes** — and envelope-mixing is root-only, which is exactly where the record does its own mixing anyway.

Root-only-ness is now *type-level* — the fence-to-type upgrade of the theory revision: the pair concludes its own small cap sort, whose only consumer is the record’s score slot (spec\_score, section 14.3) — no connective, path former, or atom takes an AttitudeSpec argument, so occurrence below a root is unrepresentable rather than oracle-rejected, and the earlier edition’s whole-tree typing clause for this former is retired. What has *not* changed is the judgment-side exclusion, and it is the design: no rule concludes an attitude claim and no judgment sort exists for one — admitting the interior blend into the derivation calculus would break the citation discipline everything else preserves. Evaluation decomposes into the two extreme passes — one floor certificate, one witness certificate — mixed once at the root, and a verdict renders as a *design score*, never a guarantee. Nothing is lost at the endpoints: a nested blend at  $\alpha \in \{0, 1\}$  has its canonical spelling in the sibling formers.

*Witness*: the attitude fixtures of the producer surface; the credal-stratum examples of the semantic universe chapter (chapter 9, the section on agreement and the attitude operator).

#### API

**TypeScript** spec.hurwiczGe / hurwiczLe

signature — ts/src/judgment.ts + ts/src/spec.ts

```

export function hurwiczGe(
  alpha: QLike,
  lo: QLike,
  ret: PathObservable,
): AttitudeSpec

export function hurwiczLe(
  alpha: QLike,
  hi: QLike,
  ret: PathObservable,
): AttitudeSpec

export function hurwiczQuery(args: {
  alpha: QLike;
  root: SpecRef;
}): Result<Artifact, JudgmentRefusal>

```

construction — ts/tests/spec-native.test.ts

```

const hurwicz = s.hurwiczGe(Q.of(1, 3), 2n, s.poSum(11, s.oChan(1)));

```

The root-level query row (judgment.hurwiczQuery) remains beside the component formers as the judgment-tour vehicle for the blend.

**Rust** mtk\_catalogue::spec::AttitudeSpec::{hurwicz\_ge, hurwicz\_le}

signature — crates/mtk-catalogue/src/judgment/kinds.rs + crates/mtk-catalogue/src/spec/formula.rs

```

pub fn hurwicz_ge(
  alpha: WitRat,
  lo: WitRat,
  ret: PathObservable,
) -> Result<AttitudeSpec, Refusal>

pub fn hurwicz_le(
  alpha: WitRat,
  hi: WitRat,
  ret: PathObservable,
) -> Result<AttitudeSpec, Refusal>

pub fn new(alpha: WitRat, root: SpecRef) -> Result<Self, Refusal>

```

construction — crates/mtk-catalogue/tests/golden\_spec.rs

```

let hurwicz = AttitudeSpec::hurwicz_ge(
  q(1, 3),
  q(2, 1),
  PathObservable::sum(11, StateObservable::chan(1)).unwrap(),
)
.unwrap();

```

The root-level query row (HurwiczQueryV0) remains beside the component formers as the judgment-tour vehicle for the blend.



## Rule Catalogue

---

This chapter is the reference manual of the proof theory, and it is generated, not written: it incorporates the literate rule catalogue in full, emitted mechanically from the same sources the calculus is built from — the hand-authored rule cells, the Lean 4 development (statements extracted by declaration name and bound by hash at emission), the verifier-receipt store, the HOL4 development, and the per-rule proof files. Every rule of the proof theory has its section: statement in Steele’s metanotation, its producer API in TypeScript and Rust, commentary, mechanized realizations, receipts, and a proof at textbook detail; the coverage dashboard up front is the catalogue’s honest map of what the assurance edifice does and does not yet cover. Bernstein Logic appears here as what it has become in the finished design: the core fragment of one catalogue, not a separate system.

Inside this chapter the catalogue keeps its own conventions: one rule per entry, each starting on a fresh page, under stable catalogue rule numbers (indices of the roster, not section numbers — they also appear as PDF bookmarks), its own references at the end, and the page geometry it was designed at.

**What this catalogue is.** The proof rules of the certificate calculus, in two strata sharing one page discipline. The base stratum is **Bernstein Logic** — a small proof calculus for certificates about polynomial facts over networks of rational-tensor-valued wires: sequents  $\Phi \vdash \Psi$  asserting that every admissible valuation of the wires satisfying the premise  $\Phi$  satisfies the conclusion  $\Psi$ , with all trusted arithmetic exact over  $\mathbb{Q}$ . Its rules appear here in three tiers — core rules (kernel semantic content), admissible rules (verified unpackings into core rules), and alias rules (verified single-application spellings) — together with the composition rules (PROMISE, SYZYGY, SMALLGAIN, SMALLGAINN, PRODUCTLATTICE), which are Bernstein Logic outright. Above the base stratum sit the **QTSL rules**: rules concluding satisfaction judgments of the quantitative temporal specification language — a system satisfies a specification built from the specification chapter's formers — through the battery rules (safety, deadline, until, and Streett acceptance), the judgment-structure and specification-surface rules, the Morse-topology rules, composition and aggregation, distance transport, and the stored-chain certificate species (eventual-absorption brackets and generator-form contraction tails, recomputed exactly over stored chain rows), supported by discharge kits and arithmetic leaves that remain ordinary Bernstein Logic sequents. QTSL premises bottom out in Bernstein Logic sequents, and both strata are designed for one exact-replay discipline: a *certificate* is an acyclic derivation graph (judgment premises may be shared) whose nodes carry enough payload that a small checker can replay each rule's side conditions by exact rational computation — no search, no optimization, no floating point in the trusted base. Where a rule's row and checker arm are enacted, the checker replays it; the coverage dashboard names, rule by rule, what is enacted and what is not. Producers (LP/SDP pipelines, abstract interpreters, hand constructions) do whatever clever thing *finds* the certificate; the checker only replays. The catalogue opens with a coverage dashboard — one row per rule, covered or not, mapping the current level of mechanized verification of soundness, honest absences included — then a synopsis parade of every rule, then the rules grouped by subject.

The validity judgment itself is not free-floating: it is kernel-checked to be the possibilistic face of the semantic universe's satisfaction judgment (chapter 9), at the sequent's exact valuation carrier — a sequent is semantically valid iff the upper probability of its conclusion's complement vanishes under the credal compile of its premise cylinder's possibilistic wall (semanticallyValid\_iff\_face, module SuFunctor.Plain.CheckerFace) — so each soundness theorem in this catalogue is, through that identification, a statement about the semantic universe's own judgment.

Each rule of the calculus gets one section here, carrying five ingredients. (i) The rule itself, typeset in Steele's Computer Science Metanotation [225], with prose on what the rule does, when a producer uses it, and why each hypothesis is there. (ii) The Lean 4 declarations that realize it — the payload structure, the soundness theorem, and (where a side condition's meaning must be visible) the checker definitions in full — extracted mechanically from the sources at every build of this catalogue and bound by hash, so quoted text cannot drift from the tree silently. (iii) Verifier receipts: attestations that named checking layers — elaboration, kernel replay by the official external checker, a hardened screen, and a decorrelated second kernel — accepted the *exact build* being quoted, identified by the SHA-256 of the compiled module; every layer is a program the reader can re-run over the quoted sources. (iv) The HOL4 statement where the independent HOL4 development carries one, and a loud, reasoned absence where it does not. (v) A proof at textbook detail: ordinary mathematics for core rules, explicit rewriting recipes for admissible rules, filled-in applications of the target rule for alias rules. Where a rule's certificate content is carried by the baked theory, the section additionally quotes the theory rows — payload schema, side-condition demand relations, well-formedness axioms — mechanically from the theory source at every build, so a renamed or removed row fails the build rather than drifting silently.

**The theory as source.** The normative home of the rules' certificate content is the baked theory pair, shipped as data in theories/ beside this book: theory BernsteinLogic (the base calculus's proof tower, from ground dtypes through hypernets, formulas, and judgments to the proof species) and theory Qtsl (one omnibus extension carrying the specification language, the System stratum, the flavored cell schemas, the certificate sections — judgment kinds, proof-node payloads, batteries, the verdict form — and the certificate-shape stratum). Where this catalogue's statements and the theory disagree, the theory is the authority. Not every catalogued rule has a baked row home: entries whose certificate content is out of scope of the current baked theory say so outright — the rule remains catalogued, tiered, and proved at its stated scope, and its schema realization is named as a successor rather than implied.

**The certificate-shape stratum.** One stratum of the baked theory types certificates *as artifacts* rather than as derivations, and its consumers are the battery rules of this catalogue, so it is documented here. A *nondeterminism axis* (NDAxis) is an entity — identity is the choice source — carrying a total binding time and extent: btStep (re-chosen at every step; the axis enters drift reads cellwise as a region axis, and no envelope object is ever materialized) or btRun (committed once per run), with extent ndEnum (an enumerated branch count) or ndParam (a box enclosure of a parametric domain). A *certificate family* (CertFam) declares its own axis roster (cert\_axis); its branch-key space is the product of its enumerated run-bound axes in position order, keys riding the ground Shape sort as natural tuples — the scalar certificate is the nullary case (one branch under the width-zero key, not a second sort). Two refusals shape the roster: a family may not split over a step-bound axis, and may not enumerate a parametric one. Per-branch positional bodies (cert\_body) carry the certificate functions; position semantics belongs to the consuming rule. The existential face's slot is the id-keyed *selection* (cert\_selection: a named branch plus a properness witness; cert\_sel\_cell refines a parametric

domain, zero cell rows reading as the whole domain) — selections are mandatory by default, waived only when every consuming conclusion is safety-only. The universal face's economies are branch-discharge rows: an emptiness certificate (`cert_branch_empty`, the one optional slot), grade domination (`cert_dominates`, irreflexive and acyclic), and progressive-revelation hand-off (`cert_handoff`, prefix agreement between branch certificates). The stratum's structural demand is counting and ground comparison — admission, never geometry — and its deliberate absences are testable negatives: no envelope slot (a demonic envelope is a consuming rule's cellwise reading of the step axes, never an interface object), no verdict rows (those live with the certificate sections that consume them), no checker-arm vocabulary, and no branch-key structure sort.

**Why trust a green cell.** A green receipt row reduces to three things a skeptical reader can check independently: the statement printed on the page (hash-bound to the source file it was extracted from); the SHA-256 of the compiled artefact the receipt names; and the checker layers the receipt credits — each layer an independently runnable program (elaboration, official kernel replay, a hardened screen, a decorrelated second kernel), so a green cell never asks for trust in this book or its receipt store: rebuild the module, re-run the checkers, and compare. Staleness is always visible — a receipt attesting an older build, a statement that moved since its proof was written, a missing ingredient each render as an *amber* badge, never as silence. Red is reserved for outright verification failure. The coverage dashboard below records, rule by rule, which ingredients exist and which do not; the empty cells are deliberate, and are meant to be read.

**Naming conventions.** Judgment kinds carry a Q prefix (`QSat`, `QRegionSat`, ...): they are the satisfaction-judgment ( $\models$ -side) sorts of the extension theory, and the prefix is their namespace. Rules that conclude a satisfaction judgment carry the same Q; derivation-side ( $\vdash$ ) leaf rules stay bare or carry P, and mathematical eponyms are never forced into descriptive molds. Rule names end in a species suffix where one applies: *-Battery* — a certificate kit concluding a region judgment; *-Discharge* — evidence concluding a named judgment kind; *-Compose* — transport across a composition axis, named by the axis; *-Alias* — a verified one-application spelling; *-Rewrite* — an equivalence-concluding rewrite. A few rules carry a former name (marked “né”) from earlier editions of this catalogue.

Rendered 2026-08-04 03:24 UTC · receipts consulted from the store (checker-layer attestations) · tree 59ea8bd0e

## Coverage dashboard

One table over every rule of the catalogue, whether or not this edition covers it yet: a coverage dashboard that maps the current level of actual mechanized verification of soundness, rule by rule. Each covered section joins its ingredients from their single sources of truth: the rule cell from the hand-authored rules source (the same file the two-column diff pages render), the Lean statements from the sources by declaration name at every render, the receipts from the store (cited at render, never re-run), the HOL4 statement from the HOL4 development or a reasoned absence, and the proof from its hand-authored file. Rules with no catalogue entry yet render as all-absent rows, loudly. Some rows carry a Lean column without a catalogue section yet: their soundness theorems exist in the mechanized development (two sibling Lean packages — the core calculus and an auxiliary measure-theoretic library — both statement-bound by the same hash machinery), while their proof and HOL columns remain marked absent until those sections are authored. A module whose build has no attestation yet shows its receipt layers as none: statements are source-bound, and receipt coverage arrives when the checking gate next runs over that package.

Legend — one row per catalogue rule (every rules-page row carrying a tier), ordered by tier class: core rules first, then admissible, then alias, then the edge-discipline glue; within a class, catalogue order. The small gray mark after each rule names its source page (BL = the Bernstein Logic pages — the core roster and the composition rules, which are Bernstein Logic outright; QTSL = the QTSL rule roster). Lean: ✓ statement bound and fresh; an *amber* checkmark marks statements bound and fresh but warranting the rule *conditionally* — where the condition is a single named profile or fragment, the superscript says which (✓<sup>DTMC</sup> = proven at the discrete-kernel profile; the model-class names are the System chapter's), and the generic ✓<sup>hyp</sup> marks a warrant resting on a bag of named side conditions (the named hypotheses live on the rule's own catalogue page); ~ module moved since bind (~<sup>hyp</sup> likewise, for a conditional-warrant row — flag and label survive staleness), ✗ statement changed since bind (stale — the page badges say where), — no Lean binding. Corr: ✓ a reviewer-certified correspondence dictionary (how the Lean statements interpret the displayed rule) is current against the displayed cell, the statement texts, and its own prose; ~ a dictionary exists but its certification predates a change on some side; — no dictionary yet. The correspondence claim is reviewer-grade, not machine-checked — its machine-checked successor supersedes it per rule as soundness statements over rules-as-data land. Receipt layers (build/replay/hardened/decorrelated): **green** = a current receipt covers it, **amber** = stale receipts only, **grey** = none. HOL4: ✓ ported, ∅ out of scope by design, — absent; HOL rcpt: **doc** = documentary citation (no receipt store yet). (An Isabelle column is hidden while the Isabelle development is empty; it returns with its first entry.) Amber throughout means stale or absent; red is reserved for an outright verification failure. Empty cells are deliberate: the table records, rule by rule, what the assurance edifice does not yet cover.

| rule                     | tier | Lean stmt        | corr | receipts | HOL stmt | HOL rcpt | proof |
|--------------------------|------|------------------|------|----------|----------|----------|-------|
| Ax BL                    | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| Weaken BL                | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| Cut BL                   | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| AI BL                    | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| AE BL                    | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| VI BL                    | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| VE BL                    | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| EqElim BL                | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| EqRef BL                 | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| EqSym BL                 | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| Antisym BL               | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| ConvElim BL              | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| ConvIntro BL             | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| ArithAtom BL             | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| BernCone BL              | core | ✓                | ~    | brhd     | ✓        | doc      | ✓     |
| Syzygy BL                | core | ✓                | ✓    | brhd     | ✓        | doc      | ✓     |
| QGuardIntro QTSL         | core | ✓                | ✓    | brhd     | —        | —        | ✓     |
| QGuardElim QTSL          | core | ✓                | ✓    | brhd     | —        | —        | ✓     |
| QSafetyBattery QTSL      | core | ✓                | ~    | brhd     | —        | —        | ✓     |
| QDeadlineBattery QTSL    | core | ✓                | ~    | brhd     | —        | —        | ✓     |
| QUnionBound QTSL         | core | ✓                | ~    | brhd     | —        | —        | ✓     |
| QStreettBattery QTSL     | core | ✓                | ✓    | brhd     | —        | —        | ✓     |
| QAvgBattery QTSL         | core | —                | —    | brhd     | —        | —        | ✓     |
| QMorseDischarge QTSL     | core | ✓ <sup>hyp</sup> | ~    | brhd     | —        | —        | ✓     |
| QMorseProduct QTSL       | core | ✓                | —    | brhd     | —        | —        | ✓     |
| QBlockInv QTSL           | core | ✓ <sup>hyp</sup> | ~    | brhd     | ✓        | doc      | ✓     |
| QBasin QTSL              | core | ✓                | ~    | brhd     | —        | —        | ✓     |
| QZeroByMiranda QTSL      | core | —                | —    | brhd     | —        | —        | ✓     |
| QCoverByMiranda QTSL     | core | —                | —    | brhd     | —        | —        | ✓     |
| QWireCompose QTSL        | core | ✓ <sup>hyp</sup> | ~    | brhd     | —        | —        | ✓     |
| QWireCertDischarge QTSL  | core | ✓ <sup>hyp</sup> | ✓    | brhd     | —        | —        | ✓     |
| QWireComposeMarks QTSL   | core | ✓ <sup>hyp</sup> | ~    | brhd     | —        | —        | ✓     |
| QSimTransport QTSL       | core | ✓                | ✓    | brhd     | —        | —        | ✓     |
| QLinGrowthDischarge QTSL | core | —                | —    | brhd     | —        | —        | ✓     |
| QMarginalCompat QTSL     | core | —                | —    | brhd     | —        | —        | ✓     |

|                                 |       |          |   |      |   |     |   |
|---------------------------------|-------|----------|---|------|---|-----|---|
| QAggReduce QTSL                 | core  | ✓ hyp    | ✓ | brhd | — | —   | ✓ |
| QPAwaysFromFold QTSL            | core  | —        | — | brhd | — | —   | ✓ |
| QFireBoundFromFold QTSL         | core  | —        | — | brhd | — | —   | ✓ |
| QPAwaysTFromFold QTSL           | core  | —        | — | brhd | — | —   | ✓ |
| QRateFromClock QTSL             | core  | —        | — | brhd | — | —   | ✓ |
| QPAwaysTFromLineage QTSL        | core  | —        | — | brhd | — | —   | ✓ |
| QCoupleDrift QTSL               | core  | ✓ finite | ✓ | brhd | — | —   | ✓ |
| QFixBall QTSL                   | core  | ✓        | ✓ | brhd | — | —   | ✓ |
| QDistTransport QTSL             | core  | ✓ hyp    | ✓ | brhd | — | —   | ✓ |
| QInvRelativize QTSL             | core  | ✓        | — | brhd | ✓ | doc | ✓ |
| QFirstService QTSL              | core  | ✓        | — | brhd | ✓ | doc | ✓ |
| QFirstServiceSplit QTSL         | core  | ✓        | — | brhd | ✓ | doc | ✓ |
| DtmcAbsorbBracket QTSL          | core  | —        | — | brhd | — | —   | ✓ |
| DtmcPosteriorBracket QTSL       | core  | —        | — | brhd | — | —   | ✓ |
| QGenFormContraction QTSL        | core  | —        | — | brhd | — | —   | ✓ |
| Norm BL                         | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| Farkas BL                       | adm.  | ✓        | ~ | brhd | ✓ | doc | ✓ |
| Subdivide BL                    | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| LiRPA BL                        | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| ReLU-Pos BL                     | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| ReLU-Neg BL                     | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| ReLU-NN BL                      | adm.  | ✓        | ~ | brhd | ✓ | doc | ✓ |
| ReLU-Above BL                   | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| Cell-In-Interior BL             | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| Cell-Out BL                     | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| PConstExact BL                  | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| PsdCertTri BL                   | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| PsdCertDiag BL                  | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| PsdCertKernel BL                | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| ReLU-Split BL                   | adm.  | ✓        | — | brhd | — | —   | ✓ |
| Promise BL                      | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| ProductLattice BL               | adm.  | ✓        | ✓ | brhd | — | —   | ✓ |
| QUntilBattery QTSL              | adm.  | ✓        | ~ | brhd | — | —   | ✓ |
| QStreettCmplRequest QTSL        | adm.  | ✓ hyp    | ✓ | brhd | — | —   | ✓ |
| HypMultiplier QTSL              | adm.  | ✓        | ✓ | brhd | — | —   | ✓ |
| QVerifiedCover QTSL             | adm.  | ✓        | ✓ | brhd | — | —   | ✓ |
| QPositiveCombinationRoot QTSL   | adm.  | ✓        | ✓ | brhd | — | —   | ✓ |
| QClosureOutward QTSL            | adm.  | ✓        | ✓ | brhd | — | —   | ✓ |
| QClosureInwardEps QTSL          | adm.  | ✓        | ✓ | brhd | — | —   | ✓ |
| BoxBern QTSL                    | adm.  | ✓        | ✓ | brhd | — | —   | ✓ |
| PExpTypeEnclosure QTSL          | adm.  | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| QEquilibriumFromInward QTSL     | adm.  | —        | — | brhd | — | —   | ✓ |
| QFlatten QTSL                   | adm.  | ✓        | ✓ | brhd | — | —   | ✓ |
| PBound BL                       | alias | ✓        | ✓ | brhd | ✓ | doc | ✓ |
| SmallGain BL                    | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| SmallGainN BL                   | alias | ✓        | — | brhd | ∅ | —   | ✓ |
| QSatFromRegion QTSL             | alias | ✓        | ✓ | brhd | — | —   | ✓ |
| QStreettDegenerate QTSL         | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| QStreettTargetCellCollapse QTSL | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| QNnfRewrite QTSL                | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| QStreettRequestNnf QTSL         | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| QAvoid QTSL                     | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| QStreettGF QTSL                 | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| QStreettFG QTSL                 | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| QNegFAlias QTSL                 | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| QFullDomainWeaken QTSL          | alias | ✓        | ✓ | brhd | — | —   | ✓ |
| QClosureExact QTSL              | alias | ✓        | ✓ | brhd | — | —   | ✓ |
| TaylorBound QTSL                | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| RootEval QTSL                   | alias | ✓        | ✓ | brhd | ∅ | —   | ✓ |
| QLatticeInv QTSL                | alias | ✓        | ✓ | brhd | — | —   | ✓ |
| QCountAtomRounding QTSL         | alias | ✓        | ✓ | brhd | — | —   | ✓ |
| ProofWellFormedness QTSL        | glue  | ✓ hyp    | — | brhd | ∅ | —   | ✓ |

## 15.1 Rule Synopsis

Every rule of the catalogue at a glance: the rule as realized in the mechanized development, its name, and the page of its full entry. The closing entry, set off by a rule, is the assembly-level discipline the numbered rules feed.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| $\frac{}{\Phi \vdash \Phi}$ <b>AX</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 396 |
| $\frac{\Phi \vdash \Psi}{\Phi \wedge \chi \vdash \Psi}$ <b>WEAKEN</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 400 |
| $\frac{\Phi \vdash \chi \quad \Phi \wedge \chi \vdash \Psi}{\Phi \vdash \Psi}$ <b>CUT</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 403 |
| $\frac{\Phi \vdash \Psi_1 \quad \Phi \vdash \Psi_2}{\Phi \vdash \Psi_1 \wedge \Psi_2}$ <b><math>\wedge I</math></b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 407 |
| $\frac{\Phi \vdash \Psi_1 \wedge \Psi_2}{\Phi \vdash \Psi_i}$ <b><math>\wedge E</math></b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 410 |
| $\frac{\Phi \vdash \Psi_i}{\Phi \vdash \Psi_1 \vee \Psi_2}$ <b><math>\vee I</math></b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 414 |
| $\frac{\Phi_1 \vdash \Psi \quad \Phi_2 \vdash \Psi}{\Phi_1 \vee \Phi_2 \vdash \Psi}$ <b><math>\vee E</math></b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 418 |
| $\frac{\Phi \vdash t = u \quad \Phi \vdash \Psi[a := t] \quad \exists s. \text{bshape}_H(t) = s = \text{bshape}_H(u)}{\Phi \vdash \Psi[a := u]}$ <b>EQELIM</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 425 |
| $\frac{\exists s. \text{bshape}_H(t) = s \quad \text{wires}(t) \text{ finitely typed in } H}{\Phi \vdash t = t}$ <b>EQREFL</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 431 |
| $\frac{\Phi \vdash t = u}{\Phi \vdash u = t}$ <b>EQSYM</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 435 |
| $\frac{\Phi \vdash t \leq u \quad \Phi \vdash t \geq u}{\Phi \vdash t = u}$ <b>ANTISYM</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 439 |
| $\frac{\Phi \vdash x \in \mathcal{R}(\Delta, S)}{\Phi \vdash \bigwedge_{i=0}^n \text{canon}(\epsilon_i \lambda_i(x)) \leq 0 \quad \epsilon_i = +1 \text{ if } i \in S, -1 \text{ if } i \in \bar{S}}$ <b>CONVELIM</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 443 |
| $\frac{\Phi \vdash \bigwedge_{i=0}^n \text{canon}(\epsilon_i \lambda_i(x)) \leq 0 \quad \epsilon_i = +1 \text{ if } i \in S, -1 \text{ if } i \in \bar{S} \quad \text{bshape}_H(x) = [n]}{\Phi \vdash x \in \mathcal{R}(\Delta, S)}$ <b>CONVINTRO</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 450 |
| $\frac{\Phi \vdash (t_1 - t_2) \bowtie 0}{\Phi \vdash t_1 \bowtie t_2}$ <b>ARITHATOM</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |     |
| $\frac{\Phi \vdash t_1 \bowtie t_2}{\Phi \vdash (t_1 - t_2) \bowtie 0}$ <b>ARITHATOM<sup>-1</sup></b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |     |
| with $\bowtie \in \{<, \leq, =, \geq, >\}$ .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |     |
| $\frac{\Delta = \text{conv}(V_0, \dots, V_n) \subset \mathbb{R}^n \text{ an } n\text{-simplex}; \quad S \subsetneq \{0, 1, \dots, n\}$<br>$\Phi \vdash x \in \mathcal{R}(\Delta, S) \quad p: \mathbb{R}^n \rightarrow W \text{ polynomial map of degree } d, \text{ with coefficient tensor } (b_\alpha)_{ \alpha =d}$<br>$\text{wires}(p) = \emptyset \vee (x = w \wedge \text{wires}(p) \subseteq \{w\}) \quad S \neq \emptyset \Rightarrow P \text{ conic}; \quad S = \emptyset \Rightarrow P \text{ convex suffices}$<br>$(p \text{ idx-free, with } p \text{ constant } \vee n = 1) \vee (p \text{ projection-indexed, indices } < n, S = \emptyset)$<br>$\text{for every } j \in \mathbb{N}^S \text{ with }  j  \leq d, \text{ every degree-}d \text{ Bernstein coefficient of } P_j \text{ on } F_S \text{ satisfies } P$<br>$\Phi \vdash P(p(x))$ <b>BERNCONE</b> . . . . . | 454 |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 459 |



|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| $t_1, t_2$ polynomial terms in the same variables with equal monomial canonical forms over $\mathbb{Q}$<br>$\text{bshape}_H(t_1) = \text{bshape}_H(t_2) \neq \perp$ (a shared defined broadcast shape) $\text{wires}(t_1), \text{wires}(t_2)$ all finitely typed in $H$                                                                                                                                                                                                                                                                                                                    |     |
| $\Phi \vdash t_1 = t_2$<br><b>NORM</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 467 |
| $\{\ell_j \bowtie_j 0\}_j$ a sub-conjunction of $\Phi$ with each $\ell_j$ linear, $\bowtie_j \in \{\leq, =\}$<br>$\{\lambda_j \in \mathbb{Q}_{\geq 0}\}_j$ nonneg coefficients, with $\lambda_j$ free when $\bowtie_j$ is = $\text{bshape}_H(\sum_j \lambda_j \ell_j) \neq \perp$<br>$\text{wires}(\sum_j \lambda_j \ell_j)$ all finitely typed in $H$ $\text{wires}(\Phi) \cup \text{wires}(\sum_j \lambda_j \ell_j)$ all typed at positive-size shapes in $H$                                                                                                                            |     |
| $\Phi \vdash \sum_j \lambda_j \ell_j \leq 0$<br><b>FARKAS</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 472 |
| $\Phi \vdash x \in \mathcal{R}(\Delta, S)$<br>$\{(\Delta_k, S_k)\}_{k=1}^N$ finite, with cover side-condition $\mathcal{R}(\Delta, S) \subseteq \bigcup_k \mathcal{R}(\Delta_k, S_k)$ $\forall k. \Phi \wedge (x \in \mathcal{R}(\Delta_k, S_k)) \vdash \Psi$                                                                                                                                                                                                                                                                                                                              |     |
| $\Phi \vdash \Psi$<br><b>SUBDIVIDE</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 483 |
| $\Phi \vdash a \leq h$ $\Phi \vdash h \leq b$ $\text{relu}_{\{h\} \rightarrow \{y\}} \in H$ $a < 0$ $0 < b$<br>$\alpha a + \beta \leq 0$ $\beta \leq 0$ $\alpha b + \beta \leq b$<br>$0 \leq \gamma a + \delta$ $0 \leq \delta$ $b \leq \gamma b + \delta$ <b>LiRPA</b>                                                                                                                                                                                                                                                                                                                    |     |
| $\Phi \vdash (\alpha h + \beta) \leq y \wedge y \leq (\gamma h + \delta)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 490 |
| $\Phi \vdash h \geq 0$ $\text{relu}_{\{h\} \rightarrow \{y\}} \in H$<br>$\Phi \vdash y = h$ <b>ReLU-Pos</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 497 |
| $\Phi \vdash h \leq 0$ $\text{relu}_{\{h\} \rightarrow \{y\}} \in H$<br>$\Phi \vdash y = 0$ <b>ReLU-NEG</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 501 |
| $\text{relu}_{\{h\} \rightarrow \{y\}} \in H$<br>$\Phi \vdash y \geq 0$ <b>ReLU-NN</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |     |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 505 |
| $\text{relu}_{\{h\} \rightarrow \{y\}} \in H$<br>$\Phi \vdash y \geq h$ <b>ReLU-ABOVE</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 508 |
| $\Phi \vdash \bigwedge_v \lambda_v^{C_i}(x) \bowtie_v^\circ 0$ (strict barycentric interior atoms, flat vertex order; $<$ for extended $v$ , $>$ for core $v$ )                                                                                                                                                                                                                                                                                                                                                                                                                            |     |
| $\text{cell}[\Sigma]_{\{x\} \rightarrow \{b_1, \dots, b_N\}} \in H$<br>$\Phi \vdash b_i = 1 \wedge \bigwedge_{j \neq i} b_j = 0$<br><b>CELL-IN-INTERIOR</b>                                                                                                                                                                                                                                                                                                                                                                                                                                |     |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 511 |
| $\Phi \vdash \lambda_w^{C_j}(x) \bowtie^\circ 0$ (a single strict separating atom at a named witness vertex $w$ of $C_j$ ; $>$ for extended $w$ , $<$ for core $w$ )                                                                                                                                                                                                                                                                                                                                                                                                                       |     |
| $\text{cell}[\Sigma]_{\{x\} \rightarrow \{b_1, \dots, b_N\}} \in H$<br>$\Phi \vdash b_j = 0$<br><b>CELL-OUT</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |     |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 519 |
| $e_1, e_2$ wire-free (closed) terms $\bowtie \in \{<, \leq, =, \geq, >\}$ $e_1 \bowtie e_2$ holds under exact evaluation in $\mathbb{Q}$                                                                                                                                                                                                                                                                                                                                                                                                                                                   |     |
| $\Phi \vdash e_1 \bowtie e_2$<br><b>PCONSTEXACT</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 525 |
| $\{\Phi_i \vdash e_i \leq 0\}_{i=1}^n$ , $\Phi \supseteq \Phi_i$ conjunct-wise (checked) $\{g_k = 0\}_k$ equality sub-conjuncts of $\Phi$ $\lambda_i \in \mathbb{Q}_{\geq 0}$<br>$\nu_k \in \mathbb{Q}$ $c \equiv \sum_i \lambda_i e_i + \sum_k \nu_k g_k$ (exact canonical polynomial) $\text{bshape}_H(c) \neq \perp$ , $\text{wires}(c)$ all finitely typed in $H$<br>$\text{wires}(\Phi) \cup \text{wires}(c) \cup \bigcup_i \text{wires}(e_i)$ all typed at positive-size shapes in $H$<br>$\Psi \in \{c \leq 0, e \geq 0 \text{ where } c = -e, l \leq r \text{ where } c = l - r\}$ |     |
| $\Phi \vdash \Psi$<br><b>PBOUND</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 530 |
| $A \in \mathbb{Q}^{n \times n}$ symmetric<br>$Z$ upper-triangular, $Z_{ii} > 0$ $A \cdot Z$ lower-triangular, $(A \cdot Z)_{ii} > 0$ (recomputed exactly) $p_1, \dots, p_n$ payload terms<br>$c \equiv -\sum_{i,j} A_{ij} p_i p_j$ (exact canonical polynomial) $\Psi \in \{c \leq 0, Q \geq 0 \text{ where } c = -Q, l \leq r \text{ where } c = l - r\}$                                                                                                                                                                                                                                 |     |
| $\Phi \vdash \Psi$<br><b>PSDCERTTRI</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 535 |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| $A \in \mathbb{Q}^{n \times n}$ symmetric $Z$ upper-triangular, $Z_{ii} > 0$<br>$B := Z^\top A Z$ strictly row-diagonally dominant (recomputed exactly) $p_1, \dots, p_n$ payload terms<br>$c \equiv -\sum_{i,j} A_{ij} p_i p_j$ (exact canonical polynomial) $\Psi \in \{c \leq 0, Q \geq 0 \text{ where } c = -Q, l \leq r \text{ where } c = l - r\}$                                                                                                                                                                                                                                                                   |     |
| $\Phi \vdash \Psi$<br>PSDCERTDIAG                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | 542 |
| $A \in \mathbb{Q}^{(m+k) \times (m+k)}$ , kernel coordinates last $L := (I \mid N)$ , $N \in \mathbb{Q}^{m \times k}$<br>$A = L^\top B L$ entrywise exactly $B \in \mathbb{Q}^{m \times m}$ with a /T-or-/DD inner witness $Z$ $p_1, \dots, p_{m+k}$ payload terms<br>$c \equiv -\sum_{i,j} A_{ij} p_i p_j$ (exact canonical polynomial) $\Psi \in \{c \leq 0, Q \geq 0 \text{ where } c = -Q, l \leq r \text{ where } c = l - r\}$                                                                                                                                                                                        |     |
| $\Phi \vdash \Psi$<br>PSDCERTKERNEL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 547 |
| $\Phi \wedge (h \geq 0) \vdash \Psi[b := h]$ $\Phi \wedge (h \leq 0) \vdash \Psi[b := 0]$ $\text{relu}_{\{h\} \rightarrow \{y\}} \in H$ ( $y$ typed at rank 0)                                                                                                                                                                                                                                                                                                                                                                                                                                                             |     |
| $\Phi \vdash \Psi$<br>RELU-SPLIT                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 552 |
| $\Phi_1 \vdash Q$ $\Phi_2 \wedge Q \vdash \Psi$ $\text{wires}(Q) \subseteq Y$ $\text{wires}(\Phi_1) \cap (\text{wires}(\Phi_2) \cup \text{wires}(\Psi)) \subseteq Y$                                                                                                                                                                                                                                                                                                                                                                                                                                                       |     |
| $\Phi_1 \wedge \Phi_2 \vdash \Psi$<br>PROMISE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 556 |
| $\{\Phi_i \vdash e_i \leq 0\}_{i=1}^n$ $\{g_k = 0\}_k$ equality sub-conjuncts of $\Phi$ in wire variables $\lambda_i \in \mathbb{Q}_{\geq 0}$ $\nu_k \in \mathbb{Q}$<br>$c \equiv \sum_i \lambda_i e_i + \sum_k \nu_k g_k$ (exact canonical polynomial) $\Psi \in \{c \leq 0, e \geq 0 \text{ where } c = -e, l \leq r \text{ where } c = l - r\}$                                                                                                                                                                                                                                                                         |     |
| $\Phi \vdash \Psi$<br>SYZGY                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 562 |
| $\Phi_1 \vdash e_1 \leq 0$ $\Phi_2 \vdash e_2 \leq 0$ $\Phi_w \vdash \lambda \sigma_2 - (1-\theta) \alpha_1 \leq 0$<br>$\Phi_w \vdash \sigma_1 - \lambda(1-\theta) \alpha_2 \leq 0$ $\Phi \supseteq \Phi_1, \Phi_2, \Phi_w$ conjunct-wise (checked) $\lambda \in \mathbb{Q}_{>0}$ $\theta \in \mathbb{Q} \cap (0, 1)$                                                                                                                                                                                                                                                                                                      |     |
| $\Phi \vdash \mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2) \leq 0$<br>SMALLGAIN                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 570 |
| $\Phi \supseteq \Phi_1, \dots, \Phi_n, \Phi_w$ conjunct-wise (checked) $\{\Phi_i \vdash e_i \leq 0\}_{i=1}^n$ $\{\Phi_w \vdash m_i \leq 0\}_{i=1}^n$<br>$\lambda \in \mathbb{Q}_{>0}$ $B \in \mathbb{Q}_{\geq 0}^{n \times n}$ $\sum_i B_{ij} = \lambda_j \ (\forall j)$ $\theta \in \mathbb{Q} \cap (0, 1)$                                                                                                                                                                                                                                                                                                               |     |
| $\Phi \vdash \mathcal{L}(\sum_i \lambda_i V_i) + \theta \sum_j \lambda_j \alpha_j \leq 0$<br>SMALLGAINN                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 575 |
| $B_1$ a certified attracting-block lattice for $\varphi_1$ over $H_1$ $B_2$ likewise for $\varphi_2$ over $H_2$ $\text{wires}(H_1) \cap \text{wires}(H_2) = \emptyset$<br>$\{b_1 \times b_2 \mid b_1 \in B_1, b_2 \in B_2\}$ (the meet-generators) certified for $\varphi_1 \times \varphi_2$ over $H_1 \sqcup H_2$                                                                                                                                                                                                                                                                                                        |     |
| $\text{child} : \text{QRegionSat}(\varphi, R)$<br>$\text{eval} : \vdash x_0 \in R$ (BL leaf; sublevel $\rightarrow$ PCONSTEXACT, box $\rightarrow$ registry interval check) $\text{SInit}(\varphi) \in s.\forall$                                                                                                                                                                                                                                                                                                                                                                                                          |     |
| $\mathcal{M} \models s$<br>QSATFROMREGION                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 588 |
| $\text{child} : \text{QRegionSat}(\varphi, R)$ $\text{SInit}(\varphi) \in s.\forall$<br>$\mathcal{M} \models \text{Guard}(R, s)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 593 |
| $\mathcal{M} \models \text{Guard}(R, s)$ $\text{eval} : \vdash X_0 \subseteq R$<br>$\mathcal{M} \models s$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 597 |
| $V : \text{QCertFam}$<br>$\{D_m \vdash \mathcal{A}_m V_m \leq 0\}_m$ (interiors) $\{\text{slab}_j \vdash g_j \circ V_{m'_j} \leq V_{m_j}\}_j$ (guards, per jump $j$ : source mode $m_j$ , target $m'_j$ , reset $g_j$ )<br>$\{U \cap \bar{D}_m \vdash V_m \geq 1\}_m$ (floors) $\{\bar{D}_m \vdash V_m \geq 0\}_m$ (nonnegs)<br>$\text{scope\_warrant} : \text{QInvScope}$ (optional — the dissolved ScopeRule) $p + y \leq 1$ (threshold tie)                                                                                                                                                                             |     |
| $\mathcal{M}, x_0 \in \{V \leq y\} \models \text{SProbGE}(p, \text{PAlways } \varphi)$<br>QSAFETYBATTERY                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 600 |
| $V : \text{QCertFam}$ $R \subseteq \llbracket \varphi \rrbracket$ (target formula tie) $\{D_m \setminus R \vdash \mathcal{A}_m V_m \leq -\theta V_m\}_m$ (decay)<br>$\{\text{slab}_j \vdash g_j \circ V_{m'_j} \leq V_{m_j}\}_j$ (guards, per jump $j$ : source mode $m_j$ , target $m'_j$ , reset $g_j$ )<br>$\{\bar{D}_m \setminus R \vdash V_m \geq 1\}_m$ (floors, off-target only) $\{\bar{D}_m \vdash V_m \geq 0\}_m$ (nonnegs; bound)<br>$\vdash q \leq \sum_{k \leq n} (\theta T)^k / k!$ (one TAYLORBOUND leaf) $0 \leq \beta, \beta \leq e^{-\theta}, \theta \geq 0, 0 \leq q, p \leq 1$ (drift tie; sign gates) |     |
| $\mathcal{M}, x_0 \in \{V \leq (1-p)q\} \models \text{SProbGE}(p, \text{PDeadline}(T, \varphi, b))$<br>QDEADLINEBATTERY                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 622 |
| $\mu(E_1) \leq a$ $\mu(E_2) \leq b$ $\text{fail} \subseteq E_1 \cup E_2$<br>$\mu(\text{fail}) \leq a + b$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 632 |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $E_{\text{bad}} := \{\text{the first exit from } D \setminus (A \cup R) \text{ lands in } A \cup D^c\}, \quad E_{\text{miss}} := \{A \cup R \text{ not reached by } T\}$<br>$\llbracket \neg \varphi_s \rrbracket \subseteq A, \quad R \subseteq \llbracket \varphi \rrbracket \text{ (formula ties)}$<br>$\mu_x(E_{\text{bad}}) \leq V_a(x) \text{ (the QSAFETYBATTERY slice for } V_a \text{ at stop set } (A \cup R) \cup D^c \text{: floor on } A \text{ only)}$<br>$\mu_x(E_{\text{miss}}) \leq \beta^T V_d(x) \text{ (the QDEADLINEBATTERY slice for } V_d \text{ at target } A \cup R)$<br>$\llbracket \neg(\varphi_s \cup_{\leq T} \varphi) \rrbracket \subseteq E_{\text{bad}} \cup E_{\text{miss}} \text{ (the d  but decomposition; consumed by QUNIONBOUND)}$<br>$0 \leq \beta, \quad 0 \leq V_d(x), \quad \beta^T \leq c \text{ (sign gates; rate tie)}$                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                |
| $\mathcal{M}, x_0 \in \{V_a + c V_d \leq 1 - p\} \models \text{SProbGE}(p, \text{PUntil}(T, \varphi_s, \varphi))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | $\text{QUNTILBATTERY} \dots\dots\dots 636$                                                                                                                     |
| $V_0, (W_i)_{i \leq k}, (V_i)_{i \in A} : \text{QCertFam}; \quad A \subseteq \{1, \dots, k\} \text{ (the pairs posting absorbing certificates); } \varepsilon_0 \in \mathbb{Q}_{>0}; \quad (\gamma_i)_{i \in A} \in \mathbb{Q}_{>0}$<br>$\text{the QSAFETYBATTERY rows for } V_0 \text{ at } U := \text{cmpl}(I) \text{ (interior, guard, floor on } U \cap \bar{D}, \text{nonneg on } \bar{D})$<br>$\{S \setminus \llbracket \rho_i \rrbracket \vdash \mathcal{A}_m V_{i,m} \leq 0\}_{i \in A} \text{ (absorbing drift: off-request, on all of } S)$<br>$\{S \vdash 0 \leq V_i \leq C_i^a\}_{i \in A} \text{ (absorbing bounds, on all of } S), \quad \{\llbracket \rho_i \rrbracket \vdash V_i \geq 1\}_{i \in A} \text{ (absorbing floors)}$<br>$\{(\llbracket \varphi_i \rrbracket \cup J_i)^c \vdash \mathcal{A}_m W_{i,m} \leq -\varepsilon_0\}_{i \leq k} \text{ (ranking decay), } J_i := \{V_i \leq 1 - \gamma_i\} \cap (I \setminus \llbracket \rho_i \rrbracket) \text{ for } i \in A; \quad J_i := \emptyset \text{ for } i \notin A$<br>$W_i \equiv 0 \text{ on target cells (containment); } \{\bar{D} \vdash W_i \geq 0\}_{i \leq k} \text{ (ranking nonnegs)}$<br>$\{\bar{D} \vdash W_i \leq C_i\}_{i \leq k}, \quad C_i \in \mathbb{Q} \quad p + y \leq 1 \text{ (threshold tie)}$ |                                                                                                                                                                |
| $\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\rho_1, \varphi_1), \dots, \text{PairBase}(\rho_k, \varphi_k)))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | $\text{QSTREETTBATTERY} \dots\dots\dots 646$                                                                                                                   |
| $\text{drift}_m : R_m \vdash g - r_m - \mathcal{A}_m B_m \geq 0 \quad \overline{\text{guard}_j} \quad \overline{\text{floor}_m} \quad \overline{\text{nonneg}_m} \quad \text{tie} : \vdash g \bowtie c$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                |
| $\mathcal{M}, x_0 \in R \models \text{SReturn} \bowtie (c, \text{poAvg}(o))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | $\text{QAVGBATTERY} \dots\dots\dots 658$                                                                                                                       |
| $\text{TrMeter}(t, d) \quad \rho \cdot d \geq 1 \quad \sigma \geq 1 \quad \text{qtsl\_rate\_clock}(n)$<br>$\mathcal{M} \models \text{SRateLE}(\{t\}, \rho, \sigma)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | $\text{QRATEFROMCLOCK} \dots\dots\dots 660$                                                                                                                    |
| $\text{BoxInv}(B, \nu, x)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                |
| $\text{AService}(B, T, \chi \cap \nu, x) \leftrightarrow \text{AService}(B, T, \chi, x)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | $\text{QINVRELATIVIZE} \dots\dots\dots 663$                                                                                                                    |
| $B.\text{FusionClosed} \quad 0 < n \quad \forall i. \text{FirstVisits}(B, \alpha_i) \quad \forall i. \text{AService}(B, T_i, \alpha_i, x)$<br>$\text{AService}(B, \min_i T_i, \text{fsTarget}(\alpha, T), x)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | $\text{QFIRSTSERVICE} \dots\dots\dots 666$                                                                                                                     |
| $B.\text{FusionClosed} \quad 0 < n \quad \forall i. c < T_i \quad \forall i. \text{FirstVisits}(B, \alpha_i) \quad \forall i. \text{AService}(B, T_i, \alpha_i, x)$<br>$\text{AService}(B, c, \text{fsTarget}(\alpha, T) \cup \text{residualCell}(\alpha, T, c), x)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | $\text{QFIRSTSERVICE SPLIT} \dots\dots\dots 669$                                                                                                               |
| $\text{one QSTREETTBATTERY node at } \rho_i := \text{STrue} \text{ for every pair}$<br>$\text{(absorbing families and } J_i \text{ syntactically empty; ranking decay off } \llbracket \varphi_i \rrbracket \text{ alone)}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                |
| $\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\text{STrue}, \varphi_1), \dots, \text{PairBase}(\text{STrue}, \varphi_k)))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | $\text{QSTREETTDEGENERATE} \dots\dots\dots 672$                                                                                                                |
| $\text{a QSTREETTBATTERY kit at certified regions } (R_i, G_i)$<br>$\{\llbracket \text{cmpl } \rho_i \rrbracket \subseteq R_i\} \text{ (requests over-approximated)} \quad \{G_i \subseteq \llbracket \varphi_i \rrbracket\} \text{ (responses under-approximated)}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                |
| $\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairCmpl}(\rho_1, \varphi_1), \dots))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | $\text{QSTREETTCMPLREQUEST} \dots\dots\dots 677$                                                                                                               |
| $\mathcal{K}, \text{slab} \vdash W_i \geq 0$<br>$\mathcal{K}, \text{slab} \vdash g \circ W_i \leq W_i$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | $\text{QSTREETTTARGETCELLCOLLAPSE} \dots\dots\dots 683$                                                                                                        |
| $\overline{\llbracket \neg \neg \varphi \rrbracket} = \overline{\llbracket \varphi \rrbracket} \quad \overline{\llbracket \neg(\varphi \wedge \psi) \rrbracket} = \overline{\llbracket \neg \varphi \vee \neg \psi \rrbracket} \quad \overline{\llbracket \neg(\varphi \vee \psi) \rrbracket} = \overline{\llbracket \neg \varphi \wedge \neg \psi \rrbracket} \quad \overline{\llbracket \neg\{f \leq c\} \rrbracket} = \overline{\llbracket f > c \rrbracket}$<br>$\overline{\llbracket \neg\{f < c\} \rrbracket} = \overline{\llbracket f \geq c \rrbracket} \quad \overline{\llbracket \neg\{f \geq c\} \rrbracket} = \overline{\llbracket f < c \rrbracket} \quad \overline{\llbracket \neg\{f > c\} \rrbracket} = \overline{\llbracket f \leq c \rrbracket}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | $\text{QNNFREWITE} \dots\dots\dots 686$                                                                                                                        |
| $\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PDwellStreett}(\dots, \text{PairCmpl}(\text{elimNeg}(\text{nnf}(\rho_i)), \varphi_i), \dots))$<br>$\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PDwellStreett}(\dots, \text{PairCmpl}(\rho_i, \varphi_i), \dots))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | $\overline{\llbracket \text{elimNeg}(\text{nnf}(\rho)) \rrbracket} = \overline{\llbracket \rho \rrbracket}$<br>$\text{QSTREETTREQUESTNNF} \dots\dots\dots 690$ |
| $\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PAlways}(\text{SNot } \beta))$<br>$\mathcal{M}, x_0 \in R \models \text{P}_{\geq p}[\text{avoid } \beta]$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | $\text{QAVOID} \dots\dots\dots 694$                                                                                                                            |

|                                                                                                                                                                                                                  |                                                                       |     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|-----|
| $\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\text{STrue}, \varphi)))$                                                                                                 | paths total over the state space                                      |     |
| $\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{GF } \varphi)$                                                                                                                                           | $\text{QSTREETTGF}$                                                   | 697 |
| $\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\beta, \text{SFalse})))$                                                                                                  | $\text{QSTREETTFG}$                                                   | 700 |
| $\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PNotPath}(\text{PDeadline}(T, \varphi, \text{false})))$                                                                                                  | $\text{QNEGFAlias}$                                                   | 703 |
| $\vdash \lambda \geq 0 \text{ on cell} \quad \vdash N + \lambda g_q \leq 0 \text{ on cell}$                                                                                                                      | $\text{HYPMULTIPLIER}$                                                | 706 |
| $\vdash P \text{ on } D_m \quad \text{scope} \subseteq D_m \text{ (compile fact, not a posted premise)}$                                                                                                         | $\text{QFULLDOMAINWEAKEN}$                                            | 709 |
| $\vdash P \text{ on } C \text{ (the cover row)} \quad \{\vdash K_i \subseteq \text{Off}\}_i \text{ (per-component containments)} \quad D_m \setminus C \subseteq \bigcup_i K_i \text{ (residual decomposition)}$ | $\text{QVERIFIEDCOVER}$                                               | 712 |
| $\text{child}_1 : \text{QRegionSat}(\varphi_1, R_1) \quad \text{child}_2 : \text{QRegionSat}(\varphi_2, R_2)$                                                                                                    |                                                                       |     |
| $\mathcal{M}, x_0 \in R_1 \cap R_2 \models \varphi_1 \wedge \varphi_2$                                                                                                                                           |                                                                       |     |
| $\text{child}_1 : \text{QRegionSat}(\varphi_1, R_1) \quad \text{child}_2 : \text{QRegionSat}(\varphi_2, R_2)$                                                                                                    |                                                                       |     |
| $\mathcal{M}, x_0 \in R_1 \cup R_2 \models \varphi_1 \vee \varphi_2$                                                                                                                                             | $\text{QPOSITIVECOMBINATIONROOT}$                                     | 715 |
| $\text{cmpl}\{p < c\} = \{c \leq p\}$                                                                                                                                                                            | $\text{QCLOSUREEXACT}$                                                | 719 |
| $\text{cmpl}\{c < p\} = \{p \leq c\}$                                                                                                                                                                            |                                                                       |     |
| $\text{cmpl}\{p \leq c\} \subseteq \{c \leq p\}$                                                                                                                                                                 | $\text{QCLOSUREOUTWARD}$                                              | 722 |
| $\text{cmpl}\{c \leq p\} \subseteq \{p \leq c\}$                                                                                                                                                                 |                                                                       |     |
| $R \subseteq \llbracket \varphi \rrbracket$                                                                                                                                                                      | $\text{cmpl } \llbracket \varphi \rrbracket \subseteq \text{cmpl } R$ |     |
| $\varepsilon \in \mathbb{Q}_{>0}$                                                                                                                                                                                | $\text{QCLOSUREINWARD EPS}$                                           | 725 |
| $\{p \leq c - \varepsilon\} \subseteq \{p < c\}$                                                                                                                                                                 |                                                                       |     |
| $\varepsilon \in \mathbb{Q}_{>0}$                                                                                                                                                                                |                                                                       |     |
| $\{c + \varepsilon \leq p\} \subseteq \{c < p\}$                                                                                                                                                                 |                                                                       |     |
| $q \in \mathbb{Q}; \quad \theta \in \mathbb{Q}_{>0}; \quad T \in \mathbb{N}; \quad n \in \mathbb{N} \quad q \leq \sum_{k \leq n} (\theta T)^k / k! \text{ holds under exact evaluation in } \mathbb{Q}$          | $\text{TAYLORBOUND}$                                                  | 728 |
| $\Phi \vdash q \leq \sum_{k \leq n} (\theta T)^k / k!$                                                                                                                                                           |                                                                       |     |
| $R = \{x : V(x) \leq y\} \text{ (sublevel root atom)} \quad \vdash V(x_0) \leq y \text{ (one PCONSTEXACT leaf: the substituted comparison is closed)}$                                                           | $\vdash x_0 \in R$                                                    |     |
| $R = \prod_i [a_i, b_i] \text{ (box root atom)} \quad \forall i. a_i \leq (x_0)_i \leq b_i \text{ (recomputed in exact } \mathbb{Q})$                                                                            | $\vdash x_0 \in R$                                                    |     |
|                                                                                                                                                                                                                  | $\text{ROOTEVAL}$                                                     | 733 |
| $N \in \mathbb{Q}[z_1, \dots, z_n]; \quad D = \prod_v [a_v, b_v] \subseteq \mathbb{Q}^n$                                                                                                                         |                                                                       |     |
| $\varphi = (A, A^{-1}, b) : \quad A, A^{-1} : \mathbb{Q}^{[n] \times [n]}, \quad b : \mathbb{Q}^{[n]}; \quad \varphi(w) = A w + b \text{ (affine-rung witness; } \varphi \text{ omitted reads as id)}$           |                                                                       |     |
| $\vdash A^{-1} A = 1_{[n]} \text{ (the one certified product, exact)}$                                                                                                                                           |                                                                       |     |
| $\hat{N} = \varphi \circ N \text{ (exact substitution)} \quad \{b_\alpha \leq 0\}_\alpha \text{ (every tensor-product Bernstein coefficient of } \hat{N} \text{ over } D)$                                       | $D \vdash \hat{N} \leq 0$                                             |     |
|                                                                                                                                                                                                                  | $\text{BOXBERN}$                                                      | 738 |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|
| $f \in F \quad n \in \mathbb{N} \quad r, R \in \mathbb{Q}$<br>$a_k = a_k^f$ for $k = 0..n$ (exact coefficient recurrence) $0 \leq r < n + 2 \quad R \geq \frac{r^{n+1}}{(n+1)!} \cdot \frac{n+2}{n+2-r}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                  |
| $\forall x,  x  \leq r \Rightarrow \left  f(x) - \sum_{k \leq n} a_k x^k \right  \leq R$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | PEXPTypeENCLOSURE . . . . . 742  |
| $L : \text{QBlockLattice}$ (blocks over one rational grid)<br>$\{F_f \vdash \sigma_f \mathcal{A} \ell_f > 0\}_{f \in \text{facets}(L)}$ (facet-sign rows: $\ell_f$ the facet's normal coordinate, $\sigma_f \in \{\pm 1\}$ the declared crossing sign, $F_f$ the facet)<br>$h : \text{QCellMap}$ (cell $\rightarrow$ vertex $\sqcup$ {transit}) $\{ \vdash T(b) \subseteq \rho(h(b)) \}_{h(b) \neq \text{transit}}$ (containments, per vertex cell)<br>$\{T(b) \vdash \mathcal{A} V_b + \varepsilon_b \leq 0\}_{h(b) = \text{transit}}, \quad \varepsilon_b \in \mathbb{Q}_{>0}$ (transits, per transit cell)<br>$\{ \vdash \rho(u) \cap \rho(v) = \emptyset \}_{u \neq v \in G}, \quad \{ \vdash \rho(v) \subseteq D \}_{v \in G}$ ( $\rho$ -wf: rows where not box-aligned, recomputed where box-aligned)<br>$\text{CR}(\mathcal{M} _D) \subseteq \bigcup_{h(b) \neq \text{transit}} T(b)$ (cell-level chain-recurrence cover)<br>$\text{MorseAssembly}(L, h, G, \rho)$ (the assembly ledger, defined below) |                                  |
| $\mathcal{M} \models \text{Morse}(D, G, \rho)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | QMORSEDISCHARGE . . . . . 747    |
| $\mathcal{M}_1 \models \text{Morse}(D_1, G_1, \rho_1) \quad \mathcal{M}_2 \models \text{Morse}(D_2, G_2, \rho_2) \quad L_1, L_2 : \text{QBlockLattice}$ (the factor lattices)<br>$\mathcal{M}_1 \times \mathcal{M}_2 \models \text{Morse}(D_1 \times D_2, G_1 \times G_2, \rho_1 \times \rho_2)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                  |
| $\text{QMORSEPRODUCT}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | . . . . . 758                    |
| $N : \text{QRegionAtom}$ (the block)<br>$\{F_f \vdash \mathcal{A} \ell_f \geq \varepsilon_f\}_{f \in \text{facets}(N)}, \quad \varepsilon_f \in \mathbb{Q}_{>0}$ ( $\ell_f$ the facet's inward affine coordinate; $F_f$ its facet region)<br>$\text{FacetBridge}(\mathcal{M}, D, N, (F_f, \varepsilon_f)_f)$ (the bridge package, defined below) $\vdash X_0 \subseteq N$ (init_containment, a single BL leaf)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                  |
| $\mathcal{M} \models \text{QInvScope}(N, X_0)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | QBLOCKINV . . . . . 763          |
| $L : \text{QBlockLattice} \quad C$ (downset, a cell-set; membership and down-closedness recomputed)<br>$\mathcal{M} \models \text{QInvScope}(\bigcup_{i \in C} \text{blocks}(\text{emb } i), X_0)$ (the cited QBLOCKINV conclusion) $\vdash X_0 \subseteq \bigcup \downarrow C$ (init_containment)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                  |
| $\mathcal{M} \models \text{QInvScope}(\bigcup \downarrow C, X_0)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | QLATTICEINV . . . . . 771        |
| $\mathcal{M} \models \text{QInvScope}(S, X_0) \quad \{ \vdash S \cap \llbracket \beta_j \rrbracket = \emptyset \}_j$ (avoid_disjoint rows)<br>$\mathcal{M}, x_0 \in X_0 \models \text{SProbGE}(1, \text{PAlways}(\text{SNot } \beta_j))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                  |
| $\text{QBASIN}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | . . . . . 778                    |
| $b : \text{QMirandaBattery} \quad \text{qtsl\_miranda\_battery}(b, C, g, \Pi)$<br>$\models \text{QZeroExists}(g, C)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                  |
| $\text{QZEROBYMIRANDA}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | . . . . . 782                    |
| $\text{inv} : \text{QBlockInv}$ (strictly-inward facets, box block)<br>$\models \text{QEquilibriumExists}(f, C)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                  |
| $\text{QEQUILIBRIUMFROMINWARD}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | . . . . . 784                    |
| $b : \text{QMirandaBattery} \quad \text{qtsl\_miranda\_battery}(b, W, \sigma - \text{id}_Z, Z)$<br>$\models Z \subseteq \sigma(W)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                  |
| $\text{QCOVERBYMIRANDA}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | . . . . . 785                    |
| $\text{rank} : \{1, \dots, n\} \rightarrow \mathbb{N}; \quad \{\text{rank}_i < \text{rank}_j\}_{(i \rightarrow j) \in w}$ (the causality certificate: one rank per component)<br>$\{ \Sigma_i \models (I_i, \alpha_i, \gamma_i[\cdot, \kappa_i]) \}_{i=1}^n \quad \text{QWireCert}(w, (\text{spec}_i)_i \Rightarrow \text{spec}_{\text{out}})$<br>$\text{W0}$ : the composite martingale problem admits a càdlàg strong-Markov solution, unique in law, from every initial state (stochastic mechanism)<br>$\text{W1}$ : transition data through per-place colour multisets (SDCPN mechanisms; typing-uniformity)<br>$\text{W2}$ : the within-instant immediate-jump relation is well-founded, of uniform rank<br>$\text{W3}$ : simultaneous enableings carry a declared serialization or a certified reset commutation (a.s.)<br>$\kappa_{\text{out}} := \bigoplus_{\text{legs}} \kappa_{\text{leg}}, \quad \kappa_a \oplus \kappa_b := \kappa_a + \kappa_b$ (slack addition along lens composition; check)   |                                  |
| $\text{SysWire}(w, [\Sigma_1, \dots, \Sigma_n]) \models \text{spec}_{\text{out}}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | QWIRECOMPOSE . . . . . 787       |
| $\{R_\ell, \varphi(s) \wedge \alpha(v(s), a) \vdash \text{Lift}_s \varphi(u(s, a))\}_\ell, \quad \{R_\ell, \varphi(s) \vdash \gamma(v(s))\}_\ell$ (mechanism-certification legs; $s$ a mechanism state, $a$ an admissible input)<br>$\{R_\ell \vdash \alpha_2(w(o_1), a_2) + \kappa(\gamma_1(o_1)) \geq \alpha_1(w^\#(o_1, a_2))\}_\ell$ (assumption legs)<br>$\{R_\ell \vdash \gamma_1(o_1) \geq \gamma_2(w(o_1))\}_\ell$ (guarantee legs) $\vdash \kappa(0) = 0$ (closed evaluation)<br>$[0, \infty) \vdash \kappa' \geq 0$ (derivative-nonnegativity row) $\vdash \deg \kappa > 0, \text{lead}(\kappa) \geq 0$ (leading-coefficient row)                                                                                                                                                                                                                                                                                                                                                                    |                                  |
| $\text{QWireCert}(w, (\text{spec}_i)_i \Rightarrow \text{spec}_{\text{out}}, \kappa)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | QWIRECERTDISCHARGE . . . . . 801 |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                               |                                                                                                                                                                                                                                                                                                              |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                               | $\{\Sigma_i \models (I_i, A_i, E_i)\}_{i=1}^n$ where $A_i \equiv \bigwedge_{j \in \text{in}(i)} \text{ev}(g_j) \text{ PathPred}$ , $E_i \equiv g_i$<br>$\{\ell_i \text{ latency rows}\}_{i=1}^n$ (each premise ships its<br>$\text{QWireCert}(w, (\text{spec}_i)_i \Rightarrow \text{spec}_{\text{out}}, -)$ |
| $\text{MarkFlow}(w, (t_k)_k, (P_k)_{k \in \text{cyc}_0(w)})$ (the schedule walk, two columns: stage rows — difference-constraint feasibility at the labels; carve rows<br>$\text{W0}$ (composite properness — the existing family; the                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                               | SysWire( $u$ )                                                                                                                                                                                                                                                                                               |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                               | QWIRECOMPOSEMARKS . . . . . 806                                                                                                                                                                                                                                                                              |
| $\frac{S' \models \text{spec}' \quad \text{QSim}(\sigma, S, S')}{S \models \sigma^* \text{spec}'}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | QSIMTRANSPORT . . . . .       | 815                                                                                                                                                                                                                                                                                                          |
| $\frac{\text{QFlatEq}(\text{sys}, \text{flat}) \quad \mathcal{M}_{\text{flat}}, x_0 \in R \models \varphi \text{ (the closed-system pipeline's judgment)}}{\text{sys}, x_0 \in R \models \varphi}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                               | QFLATTEN . . . . . 821                                                                                                                                                                                                                                                                                       |
| $\frac{\text{env}_i : B \vdash \pm f_i(x) \leq a + b \ x\ }{\models \text{QLinGrowth}(f, a, b, B)}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | QLINGROWTHDISCHARGE . . . . . | 827                                                                                                                                                                                                                                                                                                          |
| $\frac{\text{dom}_k \text{ (per-contributor dominance rows)} \quad \text{qtsl\_stochwire\_shape}(cl, sp)}{\models \text{QMarginalCompat}(sp, cl)}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | QMARGINALCOMPAT . . . . .     | 829                                                                                                                                                                                                                                                                                                          |
| $(\vec{s}, \vec{k}) \in K \vdash G(\vec{s}, \vec{k}) \leq 0$ (one BERNCONE-family sequent over the aggregate coordinates)<br>$s_j^P := \sum_{t:P} \phi_j(c_t)$ , $k_P := \#P$ (the aggregate coordinates, defined)<br>$\{\vdash l_j k_P \leq s_j^P \leq u_j k_P\}_{j,P}$ (coupling rows; $[l_j, u_j]$ a Bernstein enclosure of $\phi_j$ on $B_P$ )<br>$\{\vdash 0 \leq k_P \leq M_P\}_P$ (count rows)<br>$\{\vdash y^\top \vec{k} = y^\top \vec{k}(x_0)\}_{y \in Y}$ (P-invariant conservation rows: each cited witness $y \geq 0$ , $y^\top C = 0$ verified against the uncoloured skeleton's incid<br>$\{\vdash (s_\alpha^P)^2 \leq k_P s_{2\alpha}^P\}_{\alpha,P}$ (moment rows, optional tightening) $\forall x \in \text{scope}, \forall t:P \text{ in } x : c_t \in B_P$ (colour-bo |                               | $x \in \text{scope} \vdash G(\vec{s}(x), \vec{k}(x)) \leq 0$<br>QAGGREduce . . . . . 831                                                                                                                                                                                                                     |
| $k_P$ a count observable, integer-valued at every reachable marking $j \in \mathbb{Z}$<br>$\text{cmpl}\{k_P \leq j\} \rightsquigarrow \{k_P \geq j+1\}$ (hypothesis positions, reachable scope)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                               | QCOUNTATOMROUNDING . . . . . 836                                                                                                                                                                                                                                                                             |
| $\text{upper} : \text{qpaf\_upper}(n, c_u) \quad \text{lower} : \text{qpaf\_lower}(n, c_l) \quad \text{qtsl\_palways\_fold}(n)$<br>$\mathcal{M} \models \text{SProbGE}(p, \text{PAlways}(\text{SAnd}[\text{SAtomLE}(c, V), \text{SAtomGE}(c, V)]))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                               | QPALWAYSFROMFOLD . . . . . 840                                                                                                                                                                                                                                                                               |
| $\text{cert} : \text{HCert} \quad d_{\min} > 0 \quad \text{qtsl\_firebound\_fold}(n)$<br>$\mathcal{M} \models \text{SReturnLE}(B, \text{retFireCount}(trs))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                               | QFIREBOUNDFROMFOLD . . . . . 842                                                                                                                                                                                                                                                                             |
| $n \text{ horizon} = T \quad \text{coverage\_lb}(c) \quad \text{flow\_lb}(c) \quad \forall r. u_r > 0 \Rightarrow \text{TrMeter}(tr(r), d_r) \quad \text{qtsl\_palwayst\_fold}(n)$<br>$\mathcal{M} \models \text{SProbGE}(p, \text{PAlwaysT}(T, \text{cmpl}(\text{goal})))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                               | QPALWAYSTFROMFOLD . . . . . 844                                                                                                                                                                                                                                                                              |
| $\text{unit\_lineage}(c) \quad \{\text{hrpGap}(1), \text{hrpDropAtMost}(1)\}$ per advancing row on $\chi = -\psi$ (pathwise species)<br>$\text{TrMeter}(tr(r), d)$ per advancing row $\forall \tau \in m_0. \psi(\tau) \leq 0 \quad T < k \cdot d \quad \text{qtsl\_palwayst\_lineage}(n)$<br>$\mathcal{M} \models \text{SProbGE}(p, \text{PAlwaysT}(T, \bigwedge_{q: \text{depth}(q) \geq k} \text{count}_q = 0))$                                                                                                                                                                                                                                                                                                                                                                       |                               | QPALWAYSTFROMLINEAGE . . . . . 847                                                                                                                                                                                                                                                                           |
| $\{\vdash K(p, \cdot) \geq 0, \sum_q K(p, q) = 1, \pi_{i\#} K(p, \cdot) = \hat{K}_i(p_i, \cdot)\}$ (coupling rows, exact $\mathbb{Q}$ ; $\hat{K}_i$ the completed component kernels)<br>$\{\vdash K(p, p) = 1 \text{ for both-absorbed } p\}$ (freeze rows)<br>$\{\vdash \sum_q K(p, q) V(q) \leq V(p)\}$ (relational supermartingale generator row, every product state)<br>$\vdash V \geq 0 \quad \{\vdash V \geq 1 \text{ (disagreeing), resp. } V \geq  v_1 - v_2 , \text{ on both-absorbed states}\}$ (terminal domination)                                                                                                                                                                                                                                                          |                               | absorption: per-component reach criterion; $\mu_0$ a probability law with $\pi_{i\#}\mu_0 =$ the component initial laws (initial marginal rows); $\vdash \sum_p \mu$<br>$\text{QDist}[d](\mathcal{M}_1, \mathcal{M}_2) \leq b \quad d \in \{\text{TV, gap-pseudometrics}\}$<br>QCOUPLEDRIFT . . . . . 850    |
| $d : \text{QDistRegistry}(X_d); \quad (X_d, d) \text{ complete} \quad \vdash \rho < 1 \text{ two-point on } \bar{B}(\hat{\mu}, r) \text{ (contraction row)}$<br>$\vdash d(T\hat{\mu}, \hat{\mu}) \leq \delta \text{ (residual row)} \quad \vdash \delta + \rho r \leq r \text{ (radius row, division-free)}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                               | $\vdash \text{QDist}[d](\hat{\mu}, \text{fix } T) \leq r$<br>QFIXBALL . . . . . 857                                                                                                                                                                                                                          |



|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             |                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| $J = (\mathcal{M}_c \models s) \text{ (strict, law-level) or } J = (\text{QDist}[\text{TV}](\mu_c, \nu) \leq \delta), \text{ on the coarse side}$ $\{\vdash \text{tv}(\pi_{\#} K(s, \cdot), Q(\pi(s), \cdot)) \leq \varepsilon\}_{s \in S_f} \text{ } (\sigma\text{-witness rows, one per fine state; } \varepsilon = 0 \text{ is exact intertwining)}$ $\vdash \mu_c^0 = \pi_{\#} \mu_f^0 \text{ (initial row) } \quad n \in \mathbb{N} \text{ (declared horizon; } \varepsilon\text{-cell only)}$ |                                                                                                                             |                                                                 |
| exact cell: $J$ transports unchanged                                                                                                                                                                                                                                                                                                                                                                                                                                                                | $\varepsilon$ -cell: $\text{QDist}[\text{TV}](\pi_{\#}(\text{fine law at } n), \text{coarse law at } n) \leq n \varepsilon$ |                                                                 |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             | $\text{QDISTTRANSPORT} \dots\dots\dots 862$                     |
| $q : \text{DAbsQuery}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | $\text{qtsl\_dabs\_wf}(q)$                                                                                                  | stored rows of $q.\text{chain}$ (dprob, dinit, dclass, dreject) |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             | $\text{qtsl\_dabs\_absorb\_bracket}(q, c, lo, hi)$              |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             | $\text{DTMCABSORBBRACKET} \dots\dots\dots 867$                  |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             | $q : \text{DAbsQuery}$                                          |
| $\text{qtsl\_dabs\_wf}(q)$                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | stored rows of $q.\text{chain}$ (dprob, dinit, dclass, dreject)                                                             | $S > 0$ (recomputed; refusal otherwise)                         |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             | $\text{qtsl\_dabs\_posterior\_bracket}(q, c, lo, hi)$           |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             | $\text{DTMCPSTERIORBRACKET} \dots\dots\dots 871$                |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             | $w : \text{DGenContraction}$                                    |
| stored rows: dweight ( $A = \text{supp } V$ , nonempty), dprob, dinit                                                                                                                                                                                                                                                                                                                                                                                                                               | $\sum_{y \in A} K(x, y) V(y) \leq c V(x) \quad \forall x \in A$                                                             |                                                                 |
| $K(x, y) = 0$ for stored rows $x \notin A, y \in A$                                                                                                                                                                                                                                                                                                                                                                                                                                                 | $\min_A V > 0$                                                                                                              | $\text{qtsl\_dtmc\_genform\_contraction}(w)$                    |
| each claimed row of $w$ : $\text{qtsl\_dgc\_partial\_lo}, \text{qtsl\_dgc\_ev\_bracket}, \text{qtsl\_dgc\_survival\_tail}, \text{qtsl\_dgc\_z\_one}$                                                                                                                                                                                                                                                                                                                                                |                                                                                                                             |                                                                 |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             | $\text{QGENFORMCONTRACTION} \dots\dots\dots 873$                |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             |                                                                 |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                             | $\text{PROOFWELLFORMEDNESS} \dots\dots\dots 877$                |

## 15.2 Core Structural Rules

### Rule 1. Ax (core rule)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Ax*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{}{\Phi \vdash \Phi} \text{Ax}$$

Ax is the leaf of every derivation: it discharges a sequent whose conclusion is literally its premise,  $\Phi \vdash \Phi$ . A producer uses it wherever a derivation bottoms out at an assumption — the demonstration certificate cited under the comparator note below begins with an Ax node discharging its composite premise before a conjunction elimination extracts the region fact a Bernstein-cone closure consumes.

What the checker recomputes is one thing: that the premise and conclusion are the *same formula tree*, compared node for node — no normalization, no reassociation, no matching. Formulas here are ordered trees, so Ax does not close  $\Phi \vdash \Phi'$  for a logically equivalent but differently shaped  $\Phi'$ ; that is re-derived through the conjunction rules when needed.

The single hypothesis is load-bearing in the obvious, total way: drop the equality and the premise-free rule concludes any sequent at all —  $\top \vdash \perp$  is the sharpest instance. Notably absent are well-typedness side conditions, and their absence is correct: whatever satisfaction amounts to for  $\Phi$  — even a  $\Phi$  mentioning untyped wires or atoms that never evaluate — it sits on both sides of the turnstile. The proof section contrasts this with *EqREFL*, which concludes a freshly minted *atom* and therefore does need typing fields.

#### Lean statement

BernsteinLogic.Soundness . **sound\_ax**

statement bound

```
theorem sound_ax {H : Hypernet} {seq : Sequent}
  (h : seq.pre = seq.post) : seq.semanticallyValid H
```

**How to read the payload.** A certificate node for this rule carries an inhabitant of the following inductive: its proof-obligation fields (here, one equality of formulas) are the rule's entire side-condition surface, and the soundness lemma above takes exactly those fields as hypotheses. The six other structural sections' payloads read the same way — sub-derivations plus structural equalities, nothing else.

BernsteinLogic.Core.Catalogue . **PAXData**

statement bound

```
inductive PAXData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (preEqPost : seq.pre = seq.post) : PAXData H seq
```

#### Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

- the judgment frame, shared by every structural dictionary below: a sequent is the pair of its premise and conclusion formulas, so a displayed judgment  $\Phi \vdash \Psi \mapsto \text{seq} : \text{Sequent}$  with `seq.pre`, `seq.post` the two sides, and asserting it  $\mapsto$  the goal `seq.semanticallyValid H` — at every admissible valuation of the ambient hypernet  $H$ , satisfaction of `seq.pre` implies satisfaction of `seq.post`.

- the display's single metavariable  $\Phi$ , appearing on both sides of the turnstile,  $\mapsto$  the sole hypothesis  $h : \text{seq.pre} = \text{seq.post}$  — sharing by field equation on one sequent binder, not by a shared metavariable: the statement quantifies over *all* sequents and consumes the identity as its only input.
- premise-freeness  $\mapsto$  silence: no other hypothesis exists;  $H$  and  $\text{seq}$  are arbitrary.
- the payload row `PAXData` (certified alongside) carries the same single tie as certificate data (`preEqPost`), so admission layer and soundness statement coincide exactly on this rule.

No width difference in either direction: one hypothesis, consumed by rewriting; nothing else is stated or needed.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: exercised via the demonstration fixture the composite demonstration certificate (an axiom node, a conjunction elimination, a Bernstein-cone closure) runs as a fixture through the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted), so this rule's node rides every execution path of that harness.

## HOL4 statement

theory `bern_fam_ax`, HOL4 development script `bern_fam_axScript.sml` script bound

### Definition `checkFam_ax_def`

```

Definition checkFam_ax_def:
  checkFam_ax (cols:columns) (ct:ctx) (i:num) ⇔
    case conclRef cols ct i of
    | NONE => F
    | SOME cj =>
      case poolTagAt cols ct cj of
      | NONE => F
      | SOME t =>
        if t ≠ (48:num) then F else
        case poolField cols ct cj 0 of
        | NONE => F
        | SOME preR =>
          case poolField cols ct cj 1 of
          | NONE => F

```

```

| SOME postR =>
  if preR = postR then T
  else
    case pv cols ct cj of
    | SOME (PJudg prew postw) =>
      (expandSph FTt prew = expandSph FTt postw)
    | _ => F
End

```

**Theorem famEq\_ax**

```

Theorem famEq_ax:
  wfColumns cols ct ∧ mkCtx cols = SOME ct ∧
  nodeCorr cols ct i (CertPf sq nd) ∧
  FST (THE (tagOf cols i)) ∈ {0}
  ⇒
  (checkFam_ax cols ct i ⇔ (sq.pre = sq.post))

```

**HOL verifier receipts**

**documentary** Documentary citations. Spine census: `bern_joins` GREEN under `hol-strict` (55 theorems, 0 offences, 208-theory ancestry including this theory), `checkJoins_eq_checkPf` verified verbatim in the built signature. The cited `famEq` lemma is the kernel-checked per-node reduction: under well-formed columns (`wfColumns/mkCtx/nodeCorr`) at wire tag 0, the columnar family check holds iff the node's sequent satisfies the rule's own side condition (`sq.pre = sq.post`). There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twin `sound_ax` (theory `bern_tr_soundstruct`: `hol-strict` clean, 9 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

**Proof**

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail. This section also fixes the semantic setting shared by the seven structural sections (`AX`, `WEAKEN`, `CUT`, `∧I`, `∧E`, `∨I`, `∨E`); the later six abbreviate it as “the `Ax` setting”.

**Setting (shared by the structural sections).** Fix a hypernet  $H$  — the ambient computation graph: typed wires, each declaring a tensor shape and a dtype, and primitive applications connecting them. A *valuation*  $\sigma$  assigns each typed wire a tensor of values from its declared value set;  $\sigma$  is *admissible* when every primitive application in  $H$  satisfies its semantic relation. A formula holds at  $\sigma$  by structural recursion: an atom  $t_1 \bowtie t_2$  holds when — and only when — both terms evaluate — exact rational arithmetic, wires read whole, evaluation failing on untyped wires, on shape-incompatible arithmetic, and on any  $\pm\infty$ -valued wire mention — to tensors compared entrywise under  $\bowtie$  (a rank-0 side broadcasts; incompatible shapes never hold), so an atom either of whose terms fails to evaluate does not hold; a region atom holds when its term evaluates to a rational point of the named region;  $\Phi_1 \wedge \Phi_2$  holds when both conjuncts hold,  $\Phi_1 \vee \Phi_2$  when at least one disjunct holds,  $\top$  always,  $\perp$  never. A sequent  $\Phi \vdash \Psi$  is *semantically valid* over  $H$  when every admissible  $\sigma$  satisfying  $\Phi$  satisfies  $\Psi$ .

Two facts frame all seven structural proofs. First, formulas are *ordered trees*:  $\wedge$  and  $\vee$  are binary constructors, and every side condition of a structural rule is an identity of trees, checked structurally — no normalization, no reassociation, no commuting. Wherever a proof below “rewrites by” a payload field, it is replacing a formula by itself. Second, none of the seven proofs consumes admissibility: each threads it, unexamined, to its sub-derivations, so the structural fragment is sound over *arbitrary* valuations — premise and conclusion validity both read over all valuations, admissible or not — and the admissibility hypothesis rides along only because the validity statement quantifies over it. Where a counterexample is claimed in these sections, read it over the one-wire hypernet ( $w$  rational, rank-0, no applications): there every valuation is admissible, so an unsound conclusion genuinely falsifies.

**Claim (sound\_ax).** The payload certifies one equality, `pre = post` (`preEqPost`); under it the premise-free rule concluding  $\Phi \vdash \Phi$  is sound.

**Theorem R1.1** (soundness of `Ax`). *If  $\text{seq.pre} = \text{seq.post}$ , then  $\text{seq}$  is semantically valid over every hypernet.*

*Proof.* One step. Let  $\sigma$  be an admissible valuation with  $\sigma \models \text{seq.pre}$ . The certified equality says `seq.post` is `seq.pre` — the same tree, not merely an equivalent formula — so  $\sigma \models \text{seq.post}$ .  $\square$

**Remark R1.2** (the one hypothesis is everything). `Ax` has no premises, so the payload equality is its entire content. Drop `preEqPost` and the degenerate rule concludes an arbitrary sequent; the sharpest instance is  $\top \vdash \perp$  — every valuation satisfies the premise and none the conclusion.

**Remark R1.3** (identity is free; reflexivity is not). `Ax` carries no well-typedness side conditions, and this is correct rather than an oversight:  $\Phi \vdash \Phi$  is valid even when  $\Phi$  mentions untyped wires or atoms that can never evaluate, because satisfaction — whatever

it turns out to be — sits on both sides of the turnstile. Contrast  $\text{EqREFL}$ , which concludes the *atom*  $t = t$  under an arbitrary premise and does need shape and finiteness fields: an atom is a claim about evaluation, and an ill-typed  $t$  never evaluates, so there the conclusion can fail at valuations the premise never constrained.  $\text{Ax}$  never crosses that line because it introduces no content.

*Remark R1.4* (structural identity, not logical equivalence). The checker compares the two formula trees node for node.  $\Phi \vdash \Phi'$  for a logically equivalent but differently shaped  $\Phi' \multimap B \wedge A$  against  $A \wedge B$ , say — is *not* an  $\text{Ax}$  instance; where needed it is re-derived through  $\wedge\text{E}$  and  $\wedge\text{I}$ . In the lattice reading of the structural fragment (developed at the  $\vee\text{E}$  section),  $\text{Ax}$  is reflexivity of the entailment order — the Gentzen-style leaf [87], discharged semantically rather than by proof search.

| proof step                   | formal anchor                                                                                                                                               |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Theorem <b>R1.1</b>          | <code>sound_ax</code> ; payload <code>PaxData</code> (field <code>preEqPost</code> ); wired as <code>RuleData.ax</code> into <code>ProofObject.sound</code> |
| ordered-tree formula grammar | <code>Formula</code> (constructors and/or); satisfaction <code>Formula.holds</code>                                                                         |
| validity statement           | <code>Sequent.semanticallyValid</code>                                                                                                                      |
| lattice reading              | <code>M3.le_refl</code> (the $\vee\text{E}$ section's countermodel kit)                                                                                     |

## Rule 2. WEAKEN (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Weaken*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash \Psi}{\Phi \wedge \chi \vdash \Psi} \text{WEAKEN}$$

WEAKEN moves a proven sequent under a larger set of assumptions: from  $\Phi \vdash \Psi$  to  $\Phi \wedge \chi \vdash \Psi$ . A producer reaches for it when contexts must be made to agree — the structural rules with two premises (CUT,  $\wedge$ I,  $\vee$ E) and the composition rule's containment check all compare premise trees structurally, so a sequent proven from fewer assumptions is weakened into the context the next node expects.

The checker recomputes two tree identities: the conclusion's premise is exactly the sub-derivation's premise conjoined *on the right* with the named conjunct  $\chi$  (one new  $\wedge$ -node — position is data, since formulas are ordered trees), and the conclusion is carried over unchanged. Naming  $\chi$  in the payload keeps this a comparison, never a search.

Each hypothesis earns its keep. Drop the premise identity and weakening could silently *replace* assumptions rather than extend them — the proof section's counterexample derives  $\top \vdash w \leq 0$  from the valid  $w \leq 0 \vdash w \leq 0$ . Drop the conclusion identity and the conclusion is free. Nothing at all is demanded of  $\chi$  itself: an unsatisfiable or ill-typed conjunct only makes the new premise harder to satisfy, which costs applicability, never soundness — the rule only discards information.

### Lean statement

BernsteinLogic.Soundness . **sound\_weaken**

statement bound

```
theorem sound_weaken {H : Hypernet} {seq premise : Sequent} {extra : Formula}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = .and premise.pre extra)
  (hpost : seq.post = premise.post) : seq.semanticallyValid H
```

Payload, read as in the Ax section: a sub-derivation plus structural equalities, nothing else.

BernsteinLogic.Core.Catalogue . **PWeakenData**

statement bound

```
inductive PWeakenData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (extra : Formula)
  (preEq : seq.pre = .and premise.pre extra)
  (postEq : seq.post = premise.post)
  : PWeakenData H seq
```

### Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax.

- the premise judgment  $\Phi \vdash \Psi \mapsto ih : \text{premise.semanticallyValid } H$  — in the proof-object recursion this hypothesis is the sub-derivation's induction hypothesis; the payload (`PWeakenData`, certified alongside) carries the sub-derivation itself as a `ProofObject` field beside the same two ties.
- $\Phi \mapsto \text{premise.pre}$ ;  $\Psi \mapsto \text{premise.post}$ ;  $\chi \mapsto$  the binder `extra : Formula`, of which nothing is demanded.



- conclusion  $\Phi \wedge \chi \vdash \Psi \mapsto$  the goal under  $\text{hpre} : \text{seq.pre} = .$  and  $\text{premise.pre extra}$  — the displayed conjunction in the displayed order ( $\Phi$  left,  $\chi$  right; formulas are ordered trees, so the position is part of the certified statement) — and  $\text{hpost} : \text{seq.post} = \text{premise.post}$ .

No width difference in either direction: every hypothesis is consumed, and the statement proves exactly the displayed rule at every hypernet and sequent.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_fam\_resid, HOL4 development script bern\_fam\_residScript.sml script bound

### Definition checkFam\_weaken\_def

```

Definition checkFam_weaken_def:
  checkFam_weaken (cols:columns) (ct:ctx) (i:num) ⇔
    case frSeq cols ct i of
    | NONE => F
    | SOME sq =>
      case frChildSeq cols ct i 0 of
      | NONE => F
      | SOME ps =>
        case frFormula cols ct i 0 of
        | NONE => F
        | SOME extra =>
          sq.pre = FAnd ps.pre extra ∧ sq.post = ps.post
End

```

### Theorem famEq\_weaken

```

Theorem famEq_weaken:
  wfColumns cols ct ∧ mkCtx cols = SOME ct ∧

```

```

parseCert (cols.hdrImage ++ rest) = SOME (ws, pf) ∧
nodeCorr cols ct i (CertPf sq nd) ∧
FST (THE (tagOf cols i)) = 1 ⇒
(checkFam_weaken cols ct i ⇒ armWeaken ws sq nd)

```

## HOL verifier receipts

**documentary** Documentary citations. Family validation: `bern_fam_resid` built GREEN with all ten `famEq` arms validated, `hol-strict` clean; spine census: `bern_joins` GREEN under `hol-strict` (55 theorems, 0 offences, 208-theory ancestry including this theory), `checkJoins_eq_checkPf` verified verbatim in the built signature. The cited `famEq` lemma is the kernel-checked per-node equivalence: under well-formed columns (`wfColumns/mkCtx/nodeCorr`) at this rule's wire tag, the columnar family check coincides with the flat reference walk's arm for this rule. There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twin `sound_weaken` (theory `bern_tr_soundstruct`: `hol-strict` clean, 9 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1). Setting as in the `Ax` section: valuations, admissibility, entrywise atom satisfaction, semantic validity; counterexamples read over the one-wire hypernet.

**Claim (`sound_weaken`).** The payload supplies a premise sequent  $\Phi \vdash \Psi$  with its sub-derivation, the added conjunct  $\chi$  (extra), and two tree identities: the conclusion's premise is  $\Phi \wedge \chi$  (`preEq`) and its conclusion is  $\Psi$  (`postEq`). If  $\Phi \vdash \Psi$  is semantically valid, so is  $\Phi \wedge \chi \vdash \Psi$ .

**Theorem R2.1 (soundness of `WEAKEN`).** *Under `preEq` and `postEq`, semantic validity of the premise sequent implies semantic validity of the conclusion sequent, over every hypernet.*

*Proof.* Let  $\sigma$  be admissible with  $\sigma \models \text{seq.pre}$  — by `preEq`, the tree  $\Phi \wedge \chi$ .

*Step 1 (project).* The  $\wedge$ -clause of satisfaction is meta-conjunction, so  $\sigma \models \Phi$  and  $\sigma \models \chi$ ; keep the left component and discard the right — this rule's whole activity is licensed forgetting.

*Step 2 (apply the premise).* Premise validity at  $\sigma$  gives  $\sigma \models \Psi$ , which `postEq` pins as the conclusion's post. □

**Remark R2.2** (`preEq` forces extension, not replacement). The identity `seq.pre` =  $\Phi \wedge \chi$  makes the conclusion's premise contain the sub-derivation's premise *verbatim* as the left conjunct of one new  $\wedge$ -node, with the named  $\chi$  on the right. Drop it and “weakening” could *replace* the assumptions instead of extending them: from  $w \leq 0 \vdash w \leq 0$  (an `Ax` instance) conclude  $\top \vdash w \leq 0$  — false at  $\sigma(w) = 1$ . This context-swap family is the standard counterexample to every dropped `preEq` in the structural sections; we cite it from here on rather than restating it.

**Remark R2.3** (positional discipline).  $\chi$  arrives as the right child of a single new  $\wedge$ -node:  $\Phi \wedge \chi$ , never  $\chi \wedge \Phi$ , and never merged into  $\Phi$ 's own tree. Any further placement is  $\wedge E/\wedge I$  work. Naming  $\chi$  in the payload keeps the checker's comparison pure structure — one tree identity, no matching or search.

**Remark R2.4** ( $\chi$  is unconstrained, harmlessly). Like `Ax`, the rule carries no typing side conditions on what it touches:  $\chi$  may be unsatisfiable ( $\perp$ ), ill-typed, or mention untyped wires. At worst the conclusion's premise becomes harder or impossible to satisfy — vacuous validity, never unsoundness. Weakening only discards information; rules that must *evaluate* are where typing fields live.

**Remark R2.5** (monotonicity, locally and wholesale). In the lattice reading (the  $\vee E$  section), meet is a lower bound: the interpretation of  $\Phi \wedge \chi$  lies below that of  $\Phi$ , which lies below that of  $\Psi$ . `WEAKEN` is the one-step premise-monotonicity of the consequence relation; the composition rule's conjunct-wise premise-containment check (`Szygy's preContains`) is the same monotonicity used wholesale.

| proof step                       | formal anchor                                                                                                                                                               |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Steps 1–2                        | <code>sound_weaken</code> ; payload <code>PWeakenData</code> (fields <code>extra</code> , <code>preEq</code> , <code>postEq</code> ); wired as <code>RuleData.weaken</code> |
| $\wedge$ -clause of satisfaction | <code>Formula.holds</code> (case and)                                                                                                                                       |
| lattice reading                  | <code>M3.weaken_kit</code>                                                                                                                                                  |

## Rule 3. CUT (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Cut*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash \chi \quad \Phi \wedge \chi \vdash \Psi}{\Phi \vdash \Psi} \text{CUT}$$

CUT is the lemma-staging rule: prove  $\Phi \vdash \chi$  once, prove  $\Phi \wedge \chi \vdash \Psi$  under the enlarged context, conclude  $\Phi \vdash \Psi$ . A producer uses it to factor a certificate the way a paper factors a proof — a bound or region fact is established once and then consumed as an assumption downstream; iterated cuts stage a chain of such lemmas.

The checker recomputes four tree identities: the conclusion's premise and conclusion are inherited from the left and right sub-derivations respectively; the left sub-derivation concludes exactly the named cut formula  $\chi$ ; and the right sub-derivation's premise is exactly the left one's premise conjoined with  $\chi$  — the same context tree, verbatim. The payload must name  $\chi$  explicitly because the conclusion mentions it nowhere: CUT is the one structural rule whose certified content is invisible in its conclusion, so there is nothing to reconstruct it from.

The two interesting hypotheses fail independently, with near-twin counterexamples walked in the proof section: unshare the context and a fact proven under foreign assumptions leaks into the conclusion ( $\top \vdash w \leq 0$  from two individually valid premises); unmoor  $\chi$  from the left conclusion and the right premise consumes an assumption nobody earned (same false conclusion, different route). The proof also records why no cut-elimination enters anywhere: soundness of the rule is direct, and the checker never normalizes a proof.

### Lean statement

BernsteinLogic.Soundness . **sound\_cut**

statement bound

```
theorem sound_cut {H : Hypernet} {seq leftPremise rightPremise : Sequent}
  {cutFormula : Formula}
  (ihl : leftPremise.semanticallyValid H)
  (ihr : rightPremise.semanticallyValid H)
  (hpre : seq.pre = leftPremise.pre)
  (hpost : seq.post = rightPremise.post)
  (hcutL : leftPremise.post = cutFormula)
  (hcutR : rightPremise.pre = .and leftPremise.pre cutFormula) :
  seq.semanticallyValid H
```

Payload, read as in the Ax section: sub-derivations plus structural equalities, with the cut formula as explicit data.

BernsteinLogic.Core.Catalogue . **PCutData**

statement bound

```
inductive PCutData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (cutFormula : Formula)
  (leftPremise : Sequent) (leftProof : ProofObject H leftPremise)
  (rightPremise : Sequent) (rightProof : ProofObject H rightPremise)
  (preEq : seq.pre = leftPremise.pre)
  (postEq : seq.post = rightPremise.post)
  (cutLeft : leftPremise.post = cutFormula)
  (cutRight : rightPremise.pre = .and leftPremise.pre cutFormula)
  : PCutData H seq
```

### Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber

until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in **AX**; sub-derivation premises as at **WEAKEN**.

- the two premise judgments  $\Phi \vdash \chi$  and  $\Phi \wedge \chi \vdash \Psi \mapsto \text{ihl} : \text{leftPremise.semanticallyValid} \text{ } H \text{ and } \text{ihR} : \text{rightPremise.semanticallyValid} \text{ } H$ .
- $\Phi \mapsto \text{leftPremise.pre}; \Psi \mapsto \text{rightPremise.post}; \chi \mapsto$  the binder `cutFormula`, tied to the left conclusion by `hcutL : leftPremise.post = cutFormula` — the explicit naming the commentary explains (the conclusion mentions  $\chi$  nowhere).
- the second premise's displayed context  $\Phi \wedge \chi \mapsto \text{hcutR} : \text{rightPremise.pre} = .\text{and leftPremise.pre cutFormula}$  — the same context tree verbatim, cut formula conjoined on the right.
- conclusion  $\Phi \vdash \Psi \mapsto$  the goal under `hpre : seq.pre = leftPremise.pre` and `hpost : seq.post = rightPremise.post`.

No width difference in either direction: the six hypotheses are exactly the display's two premises and four structural identities, each consumed.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

**gate-layer coverage of the bound build:**

`build` (lake build (elaboration)) current

`replay` (leanchecker (official kernel replay)) stale only

`hardened` (safe\_verify (hardened screen)) stale only

`decorrelated` (lean4export + nanoda (decorrelated kernel)) current

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory `bern_fam_resid`, HOL4 development script `bern_fam_residScript.sml` script bound

**Definition** `checkFam_cut_def`

```

Definition checkFam_cut_def:
  checkFam_cut (cols:columns) (ct:ctx) (i:num) =
    case frSeq cols ct i of
    | NONE => F
    | SOME sq =>
      case frChildSeq cols ct i 0 of
      | NONE => F
      | SOME ls =>
        case frChildSeq cols ct i 1 of

```

```

| NONE => F
| SOME rs =>
  case frFormula cols ct i 0 of
  | NONE => F
  | SOME cutf =>
    sq.pre = ls.pre ∧ sq.post = rs.post ∧
    ls.post = cutf ∧ rs.pre = FAnd ls.pre cutf
End

```

**Theorem famEq\_cut**

```

Theorem famEq_cut:
  wfColumns cols ct ∧ mkCtx cols = SOME ct ∧
  parseCert (cols.hdrImage ++ rest) = SOME (ws, pf) ∧
  nodeCorr cols ct i (CertPf sq nd) ∧
  FST (THE (tagOf cols i)) = 2 =>
  (checkFam_cut cols ct i ⇔ armCut ws sq nd)

```

**HOL verifier receipts**

**documentary** Documentary citations. Family validation: `bern_fam_resid` built GREEN with all ten `famEq` arms validated, `hol-strict` clean; spine census: `bern_joins` GREEN under `hol-strict` (55 theorems, 0 offences, 208-theory ancestry including this theory), `checkJoins_eq_checkPf` verified verbatim in the built signature. The cited `famEq` lemma is the kernel-checked per-node equivalence: under well-formed columns (`wfColumns/mkCtx/nodeCorr`) at this rule's wire tag, the columnar family check coincides with the flat reference walk's arm for this rule. There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twin `sound_cut` (theory `bern_tr_soundstruct`: `hol-strict` clean, 9 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

**Proof**

Genre: core rule (tier 1). Setting as in the `Ax` section; counterexamples read over the one-wire hypernet.

**Claim (sound\_cut).** The payload supplies the cut formula  $\chi$  (`cutFormula`), a left premise  $\Phi \vdash \chi$  and a right premise  $\Phi \wedge \chi \vdash \Psi$  with their sub-derivations, and four tree identities: the conclusion's premise is the left premise's premise (`preEq`), its conclusion is the right premise's conclusion (`postEq`), the left premise concludes exactly  $\chi$  (`cutLeft`), and the right premise's premise is exactly  $\Phi \wedge \chi$  over the *same*  $\Phi$  (`cutRight`). If both premise sequents are semantically valid, so is  $\Phi \vdash \Psi$ .

**Theorem R3.1** (soundness of `CUT`). *Under the four payload identities, semantic validity of both premise sequents implies semantic validity of the conclusion, over every hypernet.*

*Proof.* Let  $\sigma$  be admissible with  $\sigma \models \Phi$ .

*Step 1 (earn the cut formula).* By `preEq`,  $\sigma$  satisfies the left premise's premise; left validity gives  $\sigma \models \chi$  (`cutLeft` identifies the left conclusion with  $\chi$ ).

*Step 2 (assemble the enlarged context).* The  $\wedge$ -clause of satisfaction is meta-conjunction, so the standing assumption and Step 1 give  $\sigma \models \Phi \wedge \chi$  — and by `cutRight` this tree is the right premise's premise.

*Step 3 (apply the right premise).* Right validity gives  $\sigma \models \Psi$ , pinned as the conclusion's post by `postEq`.  $\square$

**Remark R3.2** (context sharing is load-bearing). `cutRight` forces the right premise's context to be the left premise's context — the context-sharing (*additive*) format [94]. Allow the right context to be foreign, say  $\Phi' \wedge \chi$ , and cut breaks: take the left premise  $\top \vdash \top$  (an `Ax` instance) and the right premise  $(w \leq 0) \wedge \top \vdash w \leq 0$  (valid: satisfaction of the conjunction projects); the degenerate rule would conclude  $\top \vdash w \leq 0$  — false at  $\sigma(w) = 1$ .

**Remark R3.3** (`cutLeft` moors the cut formula). Keep the sharing but unmoor  $\chi$  from what the left premise actually proved, and the right premise may consume an assumption nobody earned: from the left premise  $\top \vdash \top$  and the right premise  $\top \wedge (w \leq 0) \vdash w \leq 0$  (valid, again by projection), the degenerate rule would conclude  $\top \vdash w \leq 0$ . The two counterexamples are deliberately near-twins: `cutRight` guards *whose context*, `cutLeft` guards *whose lemma*, and each fails independently. `preEq`'s removal admits the context-swap family (the `WEAKEN` section); `postEq`'s removal frees the conclusion outright.

**Remark R3.4** (the cut formula leaves no trace). The conclusion  $\Phi \vdash \Psi$  mentions  $\chi$  nowhere — the conclusion does not determine the certified content (a hiddenness `CUT` shares with  $\wedge$ 's discarded conjunct), so the payload names `cutFormula` explicitly. That is a design choice rather than a forced move — the checker could equally read  $\chi$  off the left premise's post — and the explicit field keeps the check local. The same invisibility is what the rule is *for*: certificate producers stage lemmas — prove a bound or region fact once, then continue under the enlarged context  $\Phi \wedge \chi$  — and iterated cuts stage a chain of them.

*Remark R3.5* (semantic cut costs nothing). Gentzen’s Hauptsatz [87] eliminates cut from derivations at, in general, non-elementary blow-up in proof size [191, 224]; nothing here needs it. The soundness argument is the three steps above, the checker never normalizes a proof, and the calculus keeps CUT as a first-class node because producers want staged lemmas, not cut-free normal forms. In the lattice reading: if  $x \leq m$  then  $x = x \sqcap m$ , so  $x \sqcap m \leq y$  gives  $x \leq y$  — transitivity worn as meet bookkeeping.

| proof step                       | formal anchor                                                                                            |
|----------------------------------|----------------------------------------------------------------------------------------------------------|
| Steps 1–3                        | sound_cut; payload PCutData (fields cutFormula, preEq, postEq, cutLeft, cutRight); wired as RuleData.cut |
| $\wedge$ -clause of satisfaction | Formula.holds (case and)                                                                                 |
| lattice reading                  | M3.cut_kit                                                                                               |



## Rule 4. $\wedge I$ (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` —  $\wedge I$ . The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash \Psi_1 \quad \Phi \vdash \Psi_2}{\Phi \vdash \Psi_1 \wedge \Psi_2} \wedge I$$

$\wedge I$  pairs two conclusions proven from the *same* assumptions: from  $\Phi \vdash \Psi_1$  and  $\Phi \vdash \Psi_2$ , conclude  $\Phi \vdash \Psi_1 \wedge \Psi_2$ . A producer uses it to assemble composite claims from separately established facts — two one-sided bounds into a two-sided one, the conjuncts a later node's premise-form expects — and, with  $\wedge E$ , to rebuild conjunction trees in the shape a structural comparison demands (the core never reassociates a conjunction on its own).

The checker recomputes three tree identities: the two sub-derivations' premises are the same tree, the conclusion inherits that premise, and the conclusion's post is one new  $\wedge$ -node with the left sub-derivation's conclusion on the left. Order is data:  $\Psi_1 \wedge \Psi_2$  and  $\Psi_2 \wedge \Psi_1$  are different conclusions, obtained by swapping which sub-derivation rides which payload slot.

The context-sharing identity is the load-bearing one. Drop it and the rule pairs facts earned under different assumptions, keeping only one set: from the valid  $\top \vdash \top$  and  $w \leq 0 \vdash w \leq 0$  it would conclude  $\top \vdash \top \wedge (w \leq 0)$ , false wherever  $w$  is positive. The proof section also records the structural observation the  $\vee E$  section turns into a theorem:  $\wedge I$  and  $\wedge E$  are the two directions of the universal property of meet, so the conjunction fragment adds nothing beyond the lattice structure of entailment.

### Lean statement

BernsteinLogic.Soundness . **sound\_andI**

statement bound

```
theorem sound_andI {H : Hypernet} {seq leftPremise rightPremise : Sequent}
  (ihl : leftPremise.semanticallyValid H)
  (ihr : rightPremise.semanticallyValid H)
  (hpreMatch : leftPremise.pre = rightPremise.pre)
  (hpre : seq.pre = leftPremise.pre)
  (hpost : seq.post = .and leftPremise.post rightPremise.post) :
  seq.semanticallyValid H
```

Payload, read as in the Ax section: sub-derivations plus structural equalities, nothing else.

BernsteinLogic.Core.Catalogue . **PAndIData**

statement bound

```
inductive PAndIData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (leftPremise : Sequent) (leftProof : ProofObject H leftPremise)
  (rightPremise : Sequent) (rightProof : ProofObject H rightPremise)
  (preMatch : leftPremise.pre = rightPremise.pre)
  (preEq : seq.pre = leftPremise.pre)
  (postEq : seq.post = .and leftPremise.post rightPremise.post)
  : PAndIData H seq
```

### Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; sub-derivation premises as at **WEAKEN**.

- the two premise judgments  $\mapsto$  `ihl/ihr` over the binders `leftPremise/rightPremise`.

- the display writes one context  $\Phi$  twice; the statement binds two sequents and demands  $\text{hpreMatch} : \text{leftPremise.pre} = \text{rightPremise.pre}$  — sharing by equation rather than by a common binder, the pattern of every two-premise rule on this page.
- $\Phi \mapsto \text{leftPremise.pre}$  (hence, through the match,  $\text{rightPremise.pre}$ );  $\Psi_1 \mapsto \text{leftPremise.post}$ ;  $\Psi_2 \mapsto \text{rightPremise.post}$ .
- conclusion  $\Phi \vdash \Psi_1 \wedge \Psi_2 \mapsto$  the goal under  $\text{hpre} : \text{seq.pre} = \text{leftPremise.pre}$  and  $\text{hpost} : \text{seq.post} = \text{rightPremise.post}$  — left sub-derivation on the left, as displayed.

No width difference in either direction; every hypothesis is consumed.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_fam\_resid, HOL4 development script bern\_fam\_residScript.sml script bound

### Definition checkFam\_andi\_def

```

Definition checkFam_andi_def:
  checkFam_andi (cols:columns) (ct:ctx) (i:num) ↔
    case frSeq cols ct i of
    | NONE => F
    | SOME sq =>
      case frChildSeq cols ct i 0 of
      | NONE => F
      | SOME ls =>
        case frChildSeq cols ct i 1 of
        | NONE => F
        | SOME rs =>
          ls.pre = rs.pre ∧ sq.pre = ls.pre ∧
          sq.post = FAnd ls.post rs.post
    End

```

## Theorem famEq\_andI

```
Theorem famEq_andI:
  wfColumns cols ct  $\wedge$  mkCtx cols = SOME ct  $\wedge$ 
  parseCert (cols.hdrImage ++ rest) = SOME (ws, pf)  $\wedge$ 
  nodeCorr cols ct i (CertPf sq nd)  $\wedge$ 
  FST (THE (tagOf cols i)) = 3  $\Rightarrow$ 
  (checkFam_andI cols ct i  $\Leftrightarrow$  armAndI ws sq nd)
```

## HOL verifier receipts

**documentary** Documentary citations. Family validation: `bern_fam_resid` built GREEN with all ten `famEq` arms validated, `hol-strict` clean; spine census: `bern_joins` GREEN under `hol-strict` (55 theorems, 0 offences, 208-theory ancestry including this theory), `checkJoins_eq_checkPf` verified verbatim in the built signature. The cited `famEq` lemma is the kernel-checked per-node equivalence: under well-formed columns (`wfColumns/mkCtx/nodeCorr`) at this rule's wire tag, the columnar family check coincides with the flat reference walk's arm for this rule. There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twin `sound_andI` (theory `bern_tr_soundstruct`: `hol-strict` clean, 9 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1). Setting as in the `Ax` section; counterexamples read over the one-wire hypernet.

**Claim (`sound_andI`).** The payload supplies two premise sequents  $\Phi \vdash \Psi_1$  and  $\Phi \vdash \Psi_2$  with their sub-derivations and three tree identities: the two premises share one context (`preMatch`), the conclusion's premise is that context (`preEq`), and the conclusion's post is  $\Psi_1 \wedge \Psi_2$  with the left premise's conclusion as left conjunct (`postEq`). If both premise sequents are semantically valid, so is  $\Phi \vdash \Psi_1 \wedge \Psi_2$ .

**Theorem R4.1** (soundness of  $\wedge$ I). *Under the three payload identities, semantic validity of both premise sequents implies semantic validity of the conclusion, over every hypernet.*

*Proof.* Let  $\sigma$  be admissible with  $\sigma \models \Phi$ .

*Step 1 (left component).* By `preEq`,  $\sigma$  satisfies the left premise's premise; left validity gives  $\sigma \models \Psi_1$ .

*Step 2 (right component).* `preMatch` transports  $\sigma \models \Phi$  verbatim to the right premise's premise — the same tree, so the same satisfaction fact; right validity gives  $\sigma \models \Psi_2$ .

*Step 3 (pair).* The  $\wedge$ -clause of satisfaction is meta-conjunction, so  $\sigma \models \Psi_1 \wedge \Psi_2$ , pinned by `postEq`. □

**Remark R4.2** (`preMatch` is load-bearing). The rule may only pair conclusions drawn from the *same* assumptions — context sharing again, as at `CUT` [94]. Drop `preMatch` and take the left premise  $\top \vdash \top$  and the right premise  $w \leq 0 \vdash w \leq 0$  (both `Ax` instances): the degenerate rule concludes  $\top \vdash \top \wedge (w \leq 0)$  — false at  $\sigma(w) = 1$ , because the right conjunct was earned under an assumption the conclusion no longer carries. `preEq`'s removal admits the context-swap family (the `WEAKEN` section); `postEq`'s removal frees the conclusion outright.

**Remark R4.3** (order and association are data). `postEq` places the left premise's conclusion on the left of one new  $\wedge$ -node. The conjunction is an ordered tree: nothing in the core commutes or reassociates it, so  $\Psi_1 \wedge \Psi_2$  and  $\Psi_2 \wedge \Psi_1$  are different conclusions, each derivable from the same premises by choosing which sub-derivation rides which payload slot.

**Remark R4.4** (the meet adjunction, split across two rules). In the lattice reading,  $\wedge I$  is one direction of the universal property of meet —  $x \leq y$  and  $x \leq z$  iff  $x \leq y \sqcap z$  — and  $\wedge E$  is the other. Together the two rules say the  $\wedge$ -fragment of the calculus adds nothing beyond the lattice structure of entailment; the  $\vee E$  section makes this precise (and load-bearing) through a countermodel over exactly that structure. With  $\wedge E$  the pair is also computationally lossless at the formula-tree level: the two projections undo pairing, and pairing the two projections rebuilds the identical formula tree.

| proof step                       | formal anchor                                                                                                                                                            |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Steps 1–3                        | <code>sound_andI</code> ; payload <code>PAndIData</code> (fields <code>preMatch</code> , <code>preEq</code> , <code>postEq</code> ); wired as <code>RuleData.andI</code> |
| $\wedge$ -clause of satisfaction | <code>Formula.holds</code> (case and)                                                                                                                                    |
| lattice reading                  | <code>M3.andI_kit</code>                                                                                                                                                 |

## Rule 5. $\wedge E$ (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` —  $\wedge E$ . The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash \Psi_1 \wedge \Psi_2}{\Phi \vdash \Psi_i} \wedge E$$

$\wedge E$  projects a conjunction: from  $\Phi \vdash \Psi_1 \wedge \Psi_2$ , conclude  $\Phi \vdash \Psi_i$ . A producer uses it to take composite facts apart — the demonstration certificate applies it right after its axiom leaf, extracting from a composite premise the region conjunct that the Bernstein-cone closure needs — and, with  $\wedge I$ , to reshape conjunction trees for the structural comparisons other rules perform.

The rule is realized as two constructors, one per side; the paper's index  $i \in \{1, 2\}$  is the constructor choice itself, so there is no unchecked side flag. The checker recomputes two tree identities: shared context, and that the sub-derivation's conclusion is exactly the conclusion's post conjoined with the *named* discarded conjunct — on the right for the left constructor, on the left for the right one. Naming the discarded conjunct in the payload makes the side condition a single comparison; the checker never searches a conjunction for the target, and a certificate claiming the wrong side simply fails the identity.

Dropping the conclusion identity frees the conclusion outright; dropping the context identity admits the context-swap counterexamples walked at **WEAKEN**. The projection itself needs no defence: discarding a conjunct only weakens a claim, and the proof section notes the pairing with  $\wedge I$  under which the fragment is lossless at the formula-tree level.

### Lean statement

BernsteinLogic.Soundness . **sound\_andE\_left**

statement bound

```
theorem sound_andE_left {H : Hypernet} {seq premise : Sequent} {other : Formula}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hpost : premise.post = .and seq.post other) :
  seq.semanticallyValid H
```

BernsteinLogic.Soundness . **sound\_andE\_right**

statement bound

```
theorem sound_andE_right {H : Hypernet} {seq premise : Sequent} {other : Formula}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hpost : premise.post = .and other seq.post) :
  seq.semanticallyValid H
```

Payload, read as in the **Ax** section: one constructor per projection, each a sub-derivation plus structural equalities with the discarded conjunct as explicit data.

BernsteinLogic.Core.Catalogue . **PAndEData**

statement bound

```
inductive PAndEData : Hypernet → Sequent → Type where
| left (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (other : Formula)
  (preEq : seq.pre = premise.pre)
  (postEq : premise.post = .and seq.post other)
  : PAndEData H seq
| right (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (other : Formula)
  (preEq : seq.pre = premise.pre)
  (postEq : premise.post = .and other seq.post)
  : PAndEData H seq
```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in **Ax**; sub-derivation premise as at **WEAKEN**.

- the display's index  $i \in \{1, 2\} \mapsto$  the *choice of lemma*: `sound_andE_left` is  $i = 1$ , `sound_andE_right` is  $i = 2$ , mirroring the payload's two constructors — the side is a constructor choice, never an unchecked flag.
- premise judgment  $\Phi \vdash \Psi_1 \wedge \Psi_2 \mapsto$  `ih`, with the conjunction shape carried by `hpost`: in the left lemma `premise.post = .and seq.post other` — the surviving conjunct  $\Psi_i$  is the conclusion's own post field, and the discarded conjunct  $\Psi_{3-i} \mapsto$  the binder `other` (right lemma: mirrored, `.and other seq.post`).
- $\Phi \mapsto$  `premise.pre`, shared via `hpre : seq.pre = premise.pre`; conclusion  $\Phi \vdash \Psi_i \mapsto$  the goal `seq.semanticallyValid H`.

No width difference in either direction: the lemma pair covers exactly the displayed  $i$ -indexed family, and every hypothesis is consumed.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

gate-layer coverage of the bound build:

|                                                           |            |
|-----------------------------------------------------------|------------|
| build (lake build (elaboration))                          | current    |
| replay (leanchecker (official kernel replay))             | stale only |
| hardened (safe_verify (hardened screen))                  | stale only |
| decorrelated (lean4export + nanoda (decorrelated kernel)) | current    |

**comparator: exercised via the demonstration fixture** the composite demonstration certificate (an axiom node, a conjunction elimination, a Bernstein-cone closure) runs as a fixture through the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted), so this rule's node rides every execution path of that harness.

## HOL4 statement

theory `bern_fam_ande`, HOL4 development script `bern_fam_andeScript.sml` **script bound**

**Definition** `checkFam_ande_def`

```
Definition checkFam_ande_def:
  checkFam_ande (cols:columns) (ct:ctx) (i:num)  $\leftrightarrow$ 
    case tag0f cols i of
```

```

| NONE => F
| SOME (r, v) =>
  r = 4 ∧
  (case childOrd ct i 0 of
  | NONE => F
  | SOME k =>
    case conclRef cols ct i of
    | NONE => F
    | SOME cj =>
      case conclRef cols ct k of
      | NONE => F
      | SOME ccj =>
        case pv cols ct cj of
        | SOME (PJudg prew postw) =>
          (case pv cols ct ccj of
          | SOME (PJudg cprew cpostw) =>
            (case prefField cols ct i 0 of
            | NONE => F
            | SOME ot =>
              case pv cols ct ot of
              | SOME (PFormula otw) =>
                (* pre-chain, expanded level (design note N1) *)
                expandSph FTt prew = expandSph FTt cprew ∧
                (* child post = f.and; conclusion = taken side,
                other = untaken side, by variant *)
                expandSph FTt cpostw =
                  (if v = 0 then
                   FAnd (expandSph FTt postw) (expandSph FTt otw)
                   else
                   FAnd (expandSph FTt otw) (expandSph FTt postw))
                | _ => F)
            | _ => F)
          | _ => F)
  | _ => F)
End

```

**Theorem famEq\_ande**

```

Theorem famEq_ande:
  ∀cols ct i sq nd ws.
    wfColumns cols ct ∧ mkCtx cols = SOME ct ∧
    nodeCorr cols ct i (CertPf sq nd) ∧
    FST (THE (tagOf cols i)) ∈ {4} ⇒
    (checkFam_ande cols ct i ⇔ checkPf1_ande_arm ws sq nd)

```

**HOL verifier receipts**

[documentary](#) Documentary citations. Spine census: `bern_joins` GREEN under `hol-strict` (55 theorems, 0 offences, 208-theory ancestry including this theory), `checkJoins_eq_checkPf` verified verbatim in the built signature. The cited `famEq` lemma is the kernel-checked per-theory equivalence: under well-formed columns (`wfColumns/mkCtx/nodeCorr`) at this rule's wire tag, the columnar family check coincides with the flat reference walk's arm for this rule. There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twins `sound_andE_left` / `sound_andE_right` (theory `bern_tr_soundstruct`: `hol-strict` clean, 9 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified- fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

**Proof**

Genre: core rule (tier 1). Setting as in the `Ax` section; counterexamples read over the one-wire hypernet.

**Claim (`sound_andE_left` / `sound_andE_right`).** The rule is realized as two constructors, one per projection; the payload of either supplies a premise sequent with its sub-derivation, the *discarded* conjunct (*other*), and two tree identities: shared context (`preEq`) and, for the left constructor, that the premise's conclusion is `post ∧ other` — for the right, `other ∧ post` (`postEq`). If the premise sequent  $\Phi \vdash \Psi_1 \wedge \Psi_2$  is semantically valid, so is  $\Phi \vdash \Psi_i$ .



**Theorem R5.1** (soundness of  $\wedge E$ ). *Under the payload identities of either constructor, semantic validity of the premise sequent implies semantic validity of the conclusion, over every hypernet.*

*Proof.* Left constructor; the right is its mirror image.

*Step 1 (apply the premise).* Let  $\sigma$  be admissible with  $\sigma \models \Phi$ ; by `preEq` and premise validity,  $\sigma$  satisfies the premise’s conclusion.

*Step 2 (read it as a pair and project).* `postEq` says that conclusion is the tree `seq.post`  $\wedge$  other; the  $\wedge$ -clause of satisfaction is meta-conjunction, so  $\sigma$  satisfies both conjuncts. Keep the left one:  $\sigma \models \text{seq.post}$ .  $\square$

*Remark R5.2* (the discarded conjunct is named, so nothing is searched). Carrying other in the payload turns the side condition into one structural identity: the checker never hunts through the premise’s conclusion for an occurrence of the target — it compares two named trees. Orientation is checked by the same identity: a left-constructor certificate claiming the *right* conjunct fails `postEq` unless the two conjuncts happen to be the same tree, in which case the distinction is empty anyway.

*Remark R5.3* (the index is a constructor, not a flag). The paper’s  $i \in \{1, 2\}$  is realized as the choice between two constructors rather than an integer field — there is no unchecked side flag to corrupt, and each constructor’s soundness lemma is its own two-line proof. Dropping `postEq` in either frees the conclusion outright; dropping `preEq` admits the context-swap family (the `WEAKEN` section).

*Remark R5.4* (one-way information loss, two-way fragment). Projection discards its other irrecoverably — the conclusion carries no trace of it (compare `CUT`, whose conclusion hides the cut formula; here the forgetting is the visible point of the rule). Together with  $\wedge I$  the fragment is lossless at the formula-tree level: see the meet-adjunction remark in the  $\wedge I$  section. In the lattice reading these are `M3.andE_l_kit`/`M3.andE_r_kit`, the upper direction of meet’s universal property.

| proof step       | formal anchor                                                                                                                              |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Steps 1–2 (left) | <code>sound_andE_left</code> ; payload <code>PAndEData.left</code> (fields <code>other</code> , <code>preEq</code> , <code>postEq</code> ) |
| mirror (right)   | <code>sound_andE_right</code> ; payload <code>PAndEData.right</code>                                                                       |
| dispatch         | wired as <code>RuleData.andE</code> into <code>ProofObject.sound</code>                                                                    |
| lattice reading  | <code>M3.andE_l_kit</code> , <code>M3.andE_r_kit</code>                                                                                    |

## Rule 6. $\vee I$ (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` —  $\vee I$ . The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash \Psi_i}{\Phi \vdash \Psi_1 \vee \Psi_2} \vee I$$

$\vee I$  injects a proven conclusion into a disjunction: from  $\Phi \vdash \Psi_i$ , conclude  $\Phi \vdash \Psi_1 \vee \Psi_2$  — conclusion-side weakening, the mirror of  $\text{WEAKEN}$ 's premise side. A producer mostly reaches for it to make the branches of a disjunction elimination agree: each branch proves its own branch-specific fact and  $\vee I$ -injects it into the common disjunction the elimination's conclusion-matching check demands. The separation instance discussed at  $\vee E$  is the miniature — both of its branch derivations are an axiom node followed by one  $\vee I$  node.

Like  $\wedge E$ , the rule is realized as two constructors, one per side; the injected side is the constructor choice. The checker recomputes two tree identities: shared context, and that the conclusion's post is one new  $\vee$ -node with the sub-derivation's conclusion on the constructor's side and the *named* added disjunct on the other.

Nothing whatsoever is demanded of the added disjunct — it may be unsatisfiable, ill-typed, or mention untyped wires — because satisfaction of a disjunction demands only the witnessed side, and the satisfaction relation carries no typing side conditions of its own; typing enters the calculus only through rules that must evaluate. Dropping the conclusion identity frees the conclusion outright; dropping the context identity admits the context-swap family walked at  $\text{WEAKEN}$ .

### Lean statement

BernsteinLogic.Soundness . **sound\_orI\_left**

statement bound

```
theorem sound_orI_left {H : Hypernet} {seq premise : Sequent} {other : Formula}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hpost : seq.post = .or premise.post other) :
  seq.semanticallyValid H
```

BernsteinLogic.Soundness . **sound\_orI\_right**

statement bound

```
theorem sound_orI_right {H : Hypernet} {seq premise : Sequent} {other : Formula}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hpost : seq.post = .or other premise.post) :
  seq.semanticallyValid H
```

Payload, read as in the  $Ax$  section: one constructor per injection, each a sub-derivation plus structural equalities with the added disjunct as explicit data.

BernsteinLogic.Core.Catalogue . **P0rIData**

statement bound

```
inductive P0rIData : Hypernet → Sequent → Type where
| left (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (other : Formula)
  (preEq : seq.pre = premise.pre)
  (postEq : seq.post = .or premise.post other)
  : P0rIData H seq
| right (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (other : Formula)
  (preEq : seq.pre = premise.pre)
  (postEq : seq.post = .or other premise.post)
  : P0rIData H seq
```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in **Ax**; sub-derivation premise as at **WEAKEN**.

- the display's index  $i \in \{1, 2\} \mapsto$  the choice of lemma, exactly as at  $\wedge E$ : **sound\_orI\_left** is  $i = 1$ , **sound\_orI\_right** is  $i = 2$ , mirroring the payload's two constructors.
- premise judgment  $\Phi \vdash \Psi_i \mapsto$  **ih**; the injected disjunct  $\Psi_i$  is **premise.post**, and the added disjunct  $\Psi_{3-i} \mapsto$  the binder **other**, of which nothing is demanded.
- conclusion  $\Phi \vdash \Psi_1 \vee \Psi_2 \mapsto$  the goal under **hpost**: in the left lemma **seq.post** = **.or premise.post other** (right lemma mirrored);  $\Phi \mapsto$  **premise.pre**, shared via **hpre**.

No width difference in either direction: the lemma pair covers exactly the displayed  $i$ -indexed family, and every hypothesis is consumed.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **stale only**

hardened (safe\_verify (hardened screen)) **stale only**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_fam\_resid, HOL4 development script bern\_fam\_residScript.sml **script bound**

**Definition checkFam\_ori\_def**

```

Definition checkFam_ori_def:
  checkFam_ori (cols:columns) (ct:ctx) (i:num) =
    case tagOf cols i of
    | NONE => F

```

```

| SOME (r, v) =>
  case frSeq cols ct i of
  | NONE => F
  | SOME sq =>
    case frChildSeq cols ct i 0 of
    | NONE => F
    | SOME ps =>
      case frFormula cols ct i 0 of
      | NONE => F
      | SOME other =>
        sq.pre = ps.pre ∧
        sq.post = (if v = 0 then F0r ps.post other
                     else F0r other ps.post)
End

```

**Theorem famEq\_ori**

```

Theorem famEq_ori:
  wfColumns cols ct ∧ mkCtx cols = SOME ct ∧
  parseCert (cols.hdrImage ++ rest) = SOME (ws, pf) ∧
  nodeCorr cols ct i (CertPf sq nd) ∧
  FST (THE (tagOf cols i)) = 5 ⇒
  (checkFam_ori cols ct i ⇒ arm0rI ws sq nd)

```

**HOL verifier receipts**

**documentary** Documentary citations. Family validation: `bern_fam_resid` built GREEN with all ten `famEq` arms validated, `hol-strict` clean; spine census: `bern_joins` GREEN under `hol-strict` (55 theorems, 0 offences, 208-theory ancestry including this theory), `checkJoins_eq_checkPf` verified verbatim in the built signature. The cited `famEq` lemma is the kernel-checked per-node equivalence: under well-formed columns (`wfColumns/mkCtx/nodeCorr`) at this rule's wire tag, the columnar family check coincides with the flat reference walk's arm for this rule. There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twins `sound_orI_left` / `sound_orI_right` (theory `bern_tr_soundstruct`: `hol-strict` clean, 9 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified- fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

**Proof**

Genre: core rule (tier 1). Setting as in the `Ax` section; counterexamples read over the one-wire hypernet.

**Claim (`sound_orI_left` / `sound_orI_right`).** Two constructors, one per injection; the payload of either supplies a premise sequent  $\Phi \vdash \Psi_i$  with its sub-derivation, the *added* disjunct (*other*), and two tree identities: shared context (`preEq`) and, for the left constructor, that the conclusion's post is  $\Psi_i \vee \text{other}$  — for the right, *other*  $\vee \Psi_i$  (`postEq`). If the premise sequent is semantically valid, so is  $\Phi \vdash \Psi_1 \vee \Psi_2$ .

**Theorem R6.1** (soundness of  $\vee$ I). *Under the payload identities of either constructor, semantic validity of the premise sequent implies semantic validity of the conclusion, over every hypernet.*

*Proof.* Left constructor; the right is its mirror image.

*Step 1 (apply the premise).* Let  $\sigma$  be admissible with  $\sigma \models \Phi$ ; by `preEq` and premise validity,  $\sigma \models \Psi_i$ .

*Step 2 (inject).* The  $\vee$ -clause of satisfaction is meta-disjunction, and a disjunction is satisfied by satisfying either side; the witnessed side is  $\Psi_i$ , and `postEq` pins the conclusion's post as the tree  $\Psi_i \vee \text{other}$ .  $\square$

**Remark R6.2** (the added disjunct is wholly unconstrained). *other* may be unsatisfiable, ill-typed, or mention untyped wires; nothing is demanded of it, because satisfaction of a disjunction demands only the witnessed side and the satisfaction relation carries no typing side conditions of its own — typing enters the calculus only through rules that must evaluate.  $\vee$ I is conclusion-side weakening, the mirror of `WEAKEN`'s premise side: both discard strength, and discarding is always semantically safe. Dropping `postEq` frees the conclusion outright; dropping `preEq` admits the context-swap family (the `WEAKEN` section).

**Remark R6.3** (where a producer actually reaches for it). Mostly to make two branches agree: the branches of a disjunction elimination must conclude a *common* formula, so each branch  $\vee$ I-injects its own branch-specific conclusion into the shared disjunction. The separation instance quoted in the  $\vee$ E section is the miniature: both of its branch derivations are an `Ax` node followed by one  $\vee$ I node (left in one branch, right in the other).

*Remark R6.4* (lattice reading).  $x \leq y$  implies  $x \leq y \sqcup z$ : the upper-bound half of join’s universal property (`M3.orI_l_kit`/`M3.orI_r_kit`); the other half is exactly  $\vee E$ . The four  $\wedge/\vee$  rules are the meet and join adjunctions in full, which is the observation the  $\vee E$  section’s countermodel turns into a theorem about what the core *cannot* derive.

| proof step       | formal anchor                                                                                                                            |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| Steps 1–2 (left) | <code>sound_orI_left</code> ; payload <code>P0rIData.left</code> (fields <code>other</code> , <code>preEq</code> , <code>postEq</code> ) |
| mirror (right)   | <code>sound_orI_right</code> ; payload <code>P0rIData.right</code>                                                                       |
| dispatch         | wired as <code>RuleData.orI</code> into <code>ProofObject.sound</code>                                                                   |
| lattice reading  | <code>M3.orI_l_kit</code> , <code>M3.orI_r_kit</code>                                                                                    |

## Rule 7. $\vee E$ (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` —  $\vee E$ . The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi_1 \vdash \Psi \quad \Phi_2 \vdash \Psi}{\Phi_1 \vee \Phi_2 \vdash \Psi} \vee E$$

$\vee E$  is the calculus's only branching rule — case analysis: from  $\Phi_1 \vdash \Psi$  and  $\Phi_2 \vdash \Psi$ , conclude  $\Phi_1 \vee \Phi_2 \vdash \Psi$ . The disjunctions worth eliminating come from the hypernet's synthesized primitive facts — a cell indicator's finite case list, a rectifier's sign split — and a producer eliminates them by proving the goal once per case over the correspondingly strengthened premise.

The checker recomputes three tree identities: the two branch sub-derivations conclude the same formula, the conclusion's premise is exactly the  $\vee$ -node of the two branch premises, and the conclusion's post is the common branch conclusion. The conclusion-matching identity is the load-bearing one — drop it and a fact proven only in one branch escapes the case split (the proof section's counterexample); the premise identity's removal admits the context-swap family.

The essential caveat of this section, though, is the rule's *reach*: the quoted form eliminates a disjunction only when it is the *whole* premise, while in practice every eliminable disjunction arrives as a conjunct inside the premise — and the core provably cannot restructure its way there (distribution of  $\wedge$  over  $\vee$  is underivable; the proof section exhibits the mechanized lattice countermodel). The context-carrying generalization — same rule, plus a conjunct path addressing the disjunction inside the premise's  $\wedge$ -tree — is proven sound, proven to be a proper extension, and proven to collapse onto the quoted form at the identity path; the accompanying development treats it as the intended elimination rule, with the quoted plain form destined to become its boundary alias.

### Lean statement

BernsteinLogic.Soundness . **sound\_orE**

statement bound

```
theorem sound_orE {H : Hypernet} {seq leftPremise rightPremise : Sequent}
  (ihl : leftPremise.semanticallyValid H)
  (ihr : rightPremise.semanticallyValid H)
  (hpostMatch : leftPremise.post = rightPremise.post)
  (hpre : seq.pre = .or leftPremise.pre rightPremise.pre)
  (hpost : seq.post = leftPremise.post) :
  seq.semanticallyValid H
```

Payload of the plain form, read as in the Ax section: two branch sub-derivations plus structural equalities.

BernsteinLogic.Core.Catalogue . **P0rEData**

statement bound

```
inductive P0rEData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (leftPremise : Sequent) (leftProof : ProofObject H leftPremise)
  (rightPremise : Sequent) (rightProof : ProofObject H rightPremise)
  (postMatch : leftPremise.post = rightPremise.post)
  (preEq : seq.pre = .or leftPremise.pre rightPremise.pre)
  (postEq : seq.post = leftPremise.post)
  : P0rEData H seq
```

**The context-carrying satellite.** The four declarations below locate the context-carrying generalization precisely: its applicability data (a conjunct path with read/replace obligations; the plain rule is the  $\pi =$  here case), its semantic soundness over every hypernet, the concrete distribution instance no core derivation concludes, and the resulting impossibility of elaborating the generalization into the plain core. The proof section walks the countermodel argument these rest on.

BernsteinLogic.Admissible . **0rEctxStep**

statement bound

```
-- Applicability data of context-carrying disjunction elimination: a
-- conjunct path into the conclusion's premise addresses a disjunction;
-- each branch premise is the conclusion's premise with that conjunct
-- replaced by the respective disjunct; both branches conclude the common
```



```

conclusion. Field-for-field, these are the reference checker's
obligations for the rule (path resolves, path addresses a disjunction,
the two premise-replacement equations, the two conclusion equations).
The primitive elimination rule is the `path = here` case. -/
structure OrECtxStep where
  seq      : Sequent
  leftPremise : Sequent
  rightPremise : Sequent
  path     : ConjPath
  d1       : Formula
  d2       : Formula
  pathReads : seq.pre.conjGet? path = some (.or d1 d2)
  leftPre   : seq.pre.conjReplace? path d1 = some leftPremise.pre
  rightPre  : seq.pre.conjReplace? path d2 = some rightPremise.pre
  leftPost  : leftPremise.post = seq.post
  rightPost : rightPremise.post = seq.post

```

BernsteinLogic.Admissible.**sound\_orECtx**

statement bound

```

/-- Context-carrying disjunction elimination is sound: whenever both
branch premises are semantically valid, so is the conclusion – over
every hypernet. This is the warrant for adding the rule as an
admissible extension: every sequent the extended calculus derives is
semantically valid (per-rule soundness of the core + this lemma).

```

```

The proof is the conjunct-path read lemma (extract the disjunction),
the case split of the primitive rule, and the conjunct-path replace
lemma (transport the branch's strengthened premise). -/
theorem sound_orECtx {H : Hypernet} (st : OrECtxStep)
  (ihl : st.leftPremise.semanticallyValid H)
  (ihr : st.rightPremise.semanticallyValid H) :
  st.seq.semanticallyValid H

```

BernsteinLogic.Admissible.**cx\_not\_derivable**

statement bound

```

/-- The instance's conclusion is NOT derivable in the catalogue: its
interpretation drops from an atom to  $\perp$  across the turnstile, and the
metatheorem says no derivation does that. -/
theorem cx_not_derivable : IsEmpty (ProofObject cxH cxSeq)

```

BernsteinLogic.Admissible.**orECtx\_not\_elaborable**

statement bound

```

/-- Context-carrying disjunction elimination does not elaborate into
the core calculus: no elaboration function can exist, because this
instance has derivable branch premises and an underivable conclusion.
(Contrast `OrECtxStep.elabHere`: at the boundary path the elaboration
exists; the failure is specific to genuinely deep paths.) -/
theorem orECtx_not_elaborable :
  ¬ (∀ (st : OrECtxStep) (H : Hypernet),
    Nonempty (ProofObject H st.leftPremise) →
    Nonempty (ProofObject H st.rightPremise) →
    Nonempty (ProofObject H st.seq))

```

BernsteinLogic.Admissible.**ProofObject.m3sound**

statement bound

```

/-- Monotone replacement at a conjunct path: substituting a
lattice-smaller formula at any conjunct address shrinks the M3 value.
Meet monotonicity only – no distributivity – which is why Promise,
unlike vE-ctx, is lattice-sound and core-eligible. -/
theorem mval_conjReplace_mono :
  ∀ {π : ConjPath} {Γ Γ' x q : Formula},
  Γ.conjGet? π = some q → Γ.conjReplace? π x = some Γ' →
  M3.le (mval x) (mval q) = true →
  M3.le (mval Γ') (mval Γ) = true := by
  intro π
  induction π with

```

```

| here =>
  intro  $\Gamma$   $\Gamma'$  x q hget hrep hle
  simp only [Formula.conjGet?] at hget
  simp only [Formula.conjReplace?] at hrep
  injection hget with h1
  injection hrep with h2
  rw [← h2, h1]
  exact hle
| left p ih =>
  intro  $\Gamma$   $\Gamma'$  x q hget hrep hle
  cases  $\Gamma$  with
  | and l r =>
    simp only [Formula.conjGet?] at hget
    simp only [Formula.conjReplace?] at hrep
    cases hmap : l.conjReplace? p x with
    | none => rw [hmap] at hrep; nomatch hrep
    | some l' =>
      rw [hmap] at hrep
      injection hrep with h2
      rw [← h2]
      show M3.le ((mval l').inf (mval r)) ((mval l).inf (mval r)) = true
      exact M3.inf_mono_l _ _ _ (ih hget hmap hle)
  | atom b tl tr => nomatch hrep
  | inRegion t  $\Delta$  S => nomatch hrep
  | or l r => nomatch hrep
  | tt => nomatch hrep
  | ff => nomatch hrep
| right p ih =>
  intro  $\Gamma$   $\Gamma'$  x q hget hrep hle
  cases  $\Gamma$  with
  | and l r =>
    simp only [Formula.conjGet?] at hget
    simp only [Formula.conjReplace?] at hrep
    cases hmap : r.conjReplace? p x with
    | none => rw [hmap] at hrep; nomatch hrep
    | some r' =>
      rw [hmap] at hrep
      injection hrep with h2
      rw [← h2]
      show M3.le ((mval l).inf (mval r')) ((mval l).inf (mval r)) = true
      exact M3.inf_mono_r _ _ _ (ih hget hmap hle)
  | atom b tl tr => nomatch hrep
  | inRegion t  $\Delta$  S => nomatch hrep
  | or l r => nomatch hrep
  | tt => nomatch hrep
  | ff => nomatch hrep

```

/-! ## §A9 The metatheorem: derivability is lattice-sound

Over a hypernet with no typed wires, every derivable sequent satisfies  $\llbracket \text{pre} \rrbracket \leq \llbracket \text{post} \rrbracket$  in  $M_3$ . One case per rule of the catalogue; the propositional rules are the lattice axioms, equality elimination is the substitution invariance (its typing field forces both terms closed over an untyped hypernet), the region/arithmetic-closure rules are handled by their conclusion shapes or payload absurdities, and the empty-payload rules are vacuous. -/

```

theorem ProofObject.m3sound :  $\forall$  {H : Hypernet} {seq : Sequent},
  ( $\forall$  w, H.wireType w = none)  $\rightarrow$  ProofObject H seq  $\rightarrow$ 
  M3.le (mval seq.pre) (mval seq.post) = true
| _, _, _, .mk _ _ _ (.ax _ _ (.mk _ _ hpp)) => by
  rw [hpp]; exact M3.le_refl _
| _, _, hE, .mk _ _ _ (.weaken _ _ (.mk _ _ _ pproof _ hpre hpost)) => by
  rw [hpre, hpost]
  exact M3.weaken_kit _ _ _ (pproof.m3sound hE)
| _, _, hE, .mk _ _ _ (.cut _ _
  (.mk _ _ _ _ lproof _ rproof hpre hpost hcl hcr)) => by

```

have hl

BernsteinLogic.Admissible.cxStep

statement bound

```

/-- The separation instance's applicability data: all five obligations
hold by computation. -/
def cxStep : OrECtxStep

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; sub-derivation premises as at WEAKEN. The displayed rule corresponds to sound\_orE:

- the branch judgments  $\Phi_1 \vdash \Psi$  and  $\Phi_2 \vdash \Psi \mapsto \text{ihl}/\text{ih}$ ; the shared conclusion  $\Psi$  is shared by equation,  $\text{hpostMatch} : \text{leftPremise.post} = \text{rightPremise.post}$  — the load-bearing identity the commentary singles out.
- $\Phi_1 \mapsto \text{leftPremise.pre}$ ;  $\Phi_2 \mapsto \text{rightPremise.pre}$ ; the conclusion's premise  $\Phi_1 \vee \Phi_2 \mapsto \text{hpre} : \text{seq.pre} = .\text{or leftPremise.pre rightPremise.pre}$  — the disjunction must be the *whole* premise tree, which is precisely the reach limitation the commentary discusses.
- conclusion post  $\mapsto \text{hpost} : \text{seq.post} = \text{leftPremise.post}$ ; the judgment  $\mapsto$  the goal  $\text{seq.semanticallyValid H}$ .

Width note, deliberate and one-sided: the certified statement set also carries the context-carrying satellite (`OrECtxStep`, `sound_orECtx`, `cx_not_derivable`, `orECtx_not_elaborable`). Those statements are *about the generalization*, not the displayed rule: `sound_orECtx` is strictly wider (a conjunct path addresses a disjunction *inside* the premise tree; the displayed rule is its identity-path case), and the two negative statements certify that the extension is proper — its instance is underivable in, and not elaborable into, the plain core. The displayed rule itself corresponds to `sound_orE` with no width gap in either direction.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Admissible; bound build 52e5099ba522... (bound from receipt c17abf2e103fe745, 2026-07-21T05:28:22Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt | minted | layers | status |
|---------|--------|--------|--------|
|---------|--------|--------|--------|

|                  |                      |                                    |         |
|------------------|----------------------|------------------------------------|---------|
| 7f3043c54f717472 | 2026-07-20T07:43:38Z | build+decorrelated                 | current |
| acaf0e8871731343 | 2026-07-20T20:44:28Z | build+decorrelated                 | current |
| 62974466ea131d89 | 2026-07-20T21:00:57Z | build+replay+hardened+decorrelated | current |
| c17abf2e103fe745 | 2026-07-21T05:28:22Z | build+replay+hardened+decorrelated | current |

6 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_fam\_ore, HOL4 development script bern\_fam\_oreScript.sml **script bound**

#### Definition checkFam\_ore\_def

```

Definition checkFam_ore_def:
  checkFam_ore (cols:columns) (ct:ctx) (i:num) ⇔
    case nodeSq cols ct i of
    | NONE => F
    | SOME sq =>
      case child0rd ct i 0 of
      | NONE => F
      | SOME kl =>
        case child0rd ct i 1 of
        | NONE => F
        | SOME kr =>
          case nodeSq cols ct kl of
          | NONE => F
          | SOME ls =>
            case nodeSq cols ct kr of
            | NONE => F
            | SOME rs =>
              case pathArgs cols ct i of
              | NONE => F
              | SOME (path, ifc) =>
                case conjGet sq.pre path of
                | SOME (F0r d1 d2) =>
                  (conjReplace sq.pre path d1 = SOME ls.pre ∧
                   conjReplace sq.pre path d2 = SOME rs.pre ∧
                   ls.post = sq.post ∧ rs.post = sq.post)
                | _ => F
    End

```

#### Theorem famEq\_ore

```

Theorem famEq_ore:
  ∀cols ct i sq nd ws.
    wfColumns cols ct ∧ mkCtx cols = SOME ct ∧
    nodeCorr cols ct i (CertPf sq nd) ∧
    FST (THE (tagOf cols i)) ∈ {6:num} ⇒
    (checkFam_ore cols ct i ⇔ oreArm ws sq nd)

```

## HOL verifier receipts

**documentary** Documentary citations. Spine census: bern\_joins GREEN under hol-strict (55 theorems, 0 offences, 208-theory ancestry including this theory), checkJoins\_eq\_checkPf verified verbatim in the built signature. The cited famEq lemma is the kernel-checked per-node equivalence: under well-formed columns (wfColumns/mkCtx/nodeCorr) at this rule's wire tag, the columnar family check coincides with the flat reference walk's arm for this rule. There is no signed HOL receipt store yet — these citations

are documentary. Semantic stratum, also ported: the soundness twins `sound_orE` (theory `bern_tr_soundstruct`: `hol-strict` clean, 9 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources) and `sound_orEctx` (theory `bern_tr_kitd`, 9 theorems, 0 offences, same run — the context from the wire's tag carries), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named `future work`.

## Proof

Genre: core rule (tier 1). Setting as in the `Ax` section; counterexamples read over the one-wire hypernet. This section also carries the fork note on the rule's *reach*: the plain form quoted here, its mechanized limitation, and the context-carrying generalization of which it is the boundary case.

**Claim (`sound_orE`).** The payload supplies two branch sequents  $\Phi_1 \vdash \Psi$  and  $\Phi_2 \vdash \Psi$  with their sub-derivations and three tree identities: the branches share one conclusion (`postMatch`), the conclusion's premise is the disjunction  $\Phi_1 \vee \Phi_2$  of the branch premises (`preEq`), and the conclusion's post is the left branch's post (`postEq`). If both branch sequents are semantically valid, so is  $\Phi_1 \vee \Phi_2 \vdash \Psi$ .

**Theorem R7.1** (soundness of  $\vee E$ , plain form). *Under the three payload identities, semantic validity of both branch sequents implies semantic validity of the conclusion, over every hypernet.*

*Proof.* Let  $\sigma$  be admissible with  $\sigma$  satisfying the conclusion's premise, which `preEq` pins as the tree  $\Phi_1 \vee \Phi_2$ .

*Step 1 (case split).* The  $\vee$ -clause of satisfaction is meta-disjunction:  $\sigma \models \Phi_1$  or  $\sigma \models \Phi_2$ .

*Step 2 (either branch delivers).* In the first case, left branch validity gives  $\sigma \models \Psi$  directly (`postEq`). In the second, right branch validity gives satisfaction of the right branch's conclusion, which `postMatch` identifies — as a tree — with the left's, hence with the conclusion's post.  $\square$

**Remark R7.2** (`postMatch` is load-bearing). The branches must agree on what they conclude *before* the split is trusted. Drop `postMatch` and take the left branch  $w \leq 0 \vdash w \leq 0$  and the right branch  $\top \vdash \top$  (both `Ax` instances): the degenerate rule, reading its conclusion from the left branch, concludes  $(w \leq 0) \vee \top \vdash w \leq 0$  — false at  $\sigma(w) = 1$ , where the premise holds through the right disjunct. `preEq`'s removal admits the context-swap family (the `WEAKEN` section); `postEq`'s removal frees the conclusion outright. In the lattice reading the rule is the bounding half of join's universal property —  $x \leq z$  and  $y \leq z$  iff  $x \sqcup y \leq z$  — whose introducing half is  $\vee I$ .

**Remark R7.3** (the reach problem, and why it is a theorem). `preEq` demands that the eliminated disjunction *be* the whole premise. But in this calculus the disjunctions worth eliminating almost never are: a sequent's premise in practice has the shape  $\Gamma \wedge SP$  — a context conjoined with the synthesized case facts of the hypernet's primitives (a cell indicator's finite case list, a rectifier's sign split) — so the disjunction arrives as a *conjunct* of the premise, one or more  $\wedge$ -nodes deep. The plain rule cannot reach it, and the core cannot restructure its way there, because distributing  $\wedge$  over  $\vee$  is underivable there. That is a theorem, not a design suspicion. Interpret formulas into  $M_3$ , the five-element diamond — three pairwise-incomparable atoms between  $\perp$  and  $\top$ , the canonical modular non-distributive lattice [21] — in full:  $\wedge$  to meet,  $\vee$  to join,  $\top$  to top,  $\perp$  to bottom, three disequality atoms on designated wires to the three lattice atoms, region atoms to bottom, and every remaining atom to top. Work over a hypernet all of whose wires are untyped; that hypothesis is what discharges every non-structural rule (typing packs force closed terms there, region and arithmetic payloads are handled by their conclusion shapes or turn absurd, and the payload-free rules are vacuous). Then a derivable sequent's premise always interprets  $\leq$  its conclusion — by induction on derivations, one case per rule, the structural cases being precisely the lattice axioms of order, meet, and join: the sections `Ax` through  `$\vee E$`  in this book, one kit lemma each (`ProofObject.m3sound`). No rule of the *plain core* — the calculus as quoted in these sections, prior to the context-carrying extension of **Remark R7.4**, whose whole point is otherwise — consumes distributivity. Now take the distribution instance  $P \wedge (Q \vee R) \vdash (P \wedge Q) \vee (P \wedge R)$ : its premise interprets to  $a \sqcap (b \sqcup c) = a \sqcap \top = a$ , its conclusion to  $(a \sqcap b) \sqcup (a \sqcap c) = \perp \sqcup \perp = \perp$ , and  $a \not\leq \perp$  — so no core derivation concludes it (`cx_not_derivable`), even though both branch premises  $P \wedge Q \vdash \cdot$  and  $P \wedge R \vdash \cdot$  are two-step core derivations (an `Ax` node and a  $\vee I$  node each).

**Remark R7.4** (the context-carrying form, and the fork discipline). The repair is not restructuring but a stronger elimination. The context-carrying rule's payload adds a *conjunct path*  $\pi$  into the premise's  $\wedge$ -tree, with read and replace obligations: the conjunct at  $\pi$  is a disjunction  $\Phi_1 \vee \Phi_2$ , each branch premise is the conclusion's premise with that conjunct replaced by the respective disjunct, and both branches conclude the common conclusion (`orEctxStep`). Three mechanized facts locate it precisely. *Sound*: whenever both branch premises are semantically valid, so is the conclusion, over every hypernet (`sound_orEctx` — one path-read lemma to extract the disjunction, the case split of Theorem R7.1, one path-replace lemma to transport each branch's strengthened premise). *Proper*: no elaboration of the context-carrying rule into the core exists (`orEctx_not_elaborable`) — concretely, at the conjunct path addressing  $Q \vee R$  as the premise's right conjunct, the read obligation returns that disjunction and the replace obligations return the branch premises  $P \wedge Q$  and  $P \wedge R$  of **Remark R7.3**'s instance (`cxStep`); its branches are derivable there and its conclusion is not, so the extension proves something true that the core cannot reach. *Conservative at the boundary*: at  $\pi =$  here the rule elaborates into the plain constructor (`orEctxStep.elabHere`) — the form proven sound above is exactly the identity-path case. The cost side of the fork is small and bounded: the certificate carries the path, the checker two small path functions (`read-at`, `replace-at`), and the soundness proof one lemma about conjunct-path replacement. The alternative — adding  $\wedge$ -restructuring rules to the core (rules whose premise and conclusion rearrange the same conjuncts: reassociations, permutations) — cannot substitute: such rules are sound for every lattice interpretation, so the induction above absorbs them — as it absorbs any rule that preserves

$M_3$ -order — and the countermodel stands; closing the gap requires a rule whose soundness genuinely uses distributivity, which is the context-carrying elimination itself. The accompanying development accordingly treats the context-carrying form as the intended elimination rule; the constructor quoted in this section’s Lean cell is the plain form, destined to become its boundary alias.

| proof step                                     | formal anchor                                                                        |
|------------------------------------------------|--------------------------------------------------------------------------------------|
| Steps 1–2                                      | sound_orE; payload P0rEData (fields postMatch, preEq, postEq); wired as RuleData.orE |
| $\vee$ -clause of satisfaction                 | Formula.holds (case or)                                                              |
| lattice reading                                | M3.orE_kit                                                                           |
| Remark R7.3: order-preservation metatheorem    | ProofObject.m3sound (BernsteinLogic/Admissible.lean)                                 |
| Remark R7.3: underivable distribution instance | cxStep, cx_not_derivable                                                             |
| Remark R7.4: extension payload + soundness     | OrECtxStep, sound_orECtx                                                             |
| Remark R7.4: properness                        | orECtx_not_elaborable                                                                |
| Remark R7.4: boundary elaboration              | OrECtxStep.elabHere                                                                  |

## Rule 8. EQELIM (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *EqElim*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash t = u \quad \Phi \vdash \Psi[a := t] \quad \exists s. \text{bshape}_H(t) = s = \text{bshape}_H(u)}{\Phi \vdash \Psi[a := u]} \text{EQELIM}$$

EQELIM is Leibniz substitution as a proof rule: an established equality  $t = u$  licenses replacing  $t$  by  $u$  at a designated hole in an established formula. The payload names the hole (a wire  $a$ ), the body formula  $\Psi$ , and the two terms; premises prove  $t = u$  and  $\Psi[a := t]$  over the shared context, and the conclusion is  $\Psi[a := u]$ . A producer reaches for it wherever a derived fact must be rewritten along an equality: pushing a resolved value through a bound (the ReLU rules conclude equalities such as  $y = h$  exactly so that this rule can consume them), transporting an atom across a NORM-derived respelling, or assembling transitivity — which is not primitive in this calculus: a one-node derivation through this rule (the proof section's closing remark spells it out; EQREFL and EQSYM complete the kit).

What the checker recomputes: both substitution instances — it applies the (binder-free, hence capture-free) substitution function to  $(\Psi, a, t)$  and to  $(\Psi, a, u)$  and demands syntactic equality with the cited premise and conclusion formulas — plus the premise-sharing equations, and the typing side condition:  $t$  and  $u$  carry a *common defined* broadcast shape in the ambient hypernet. Nothing semantic is consulted; every side condition is a decidable syntactic equation.

The typing condition is the load-bearing hypothesis. Equality atoms hold *elementwise under broadcast*, so a scalar can stand equal to a vector whose every entry it matches — a genuine, derivable, usable fact that is nonetheless not a license to interchange the two terms: in a region atom, which constrains the shape of its term, the substitution flips satisfiability. Dropping the condition admits a concrete derivation of an unsatisfiable conclusion from a satisfiable premise (the proof section constructs it); with the condition, that certificate is refused because no common shape exists. Dropping either substitution equation instead would let the conclusion be an arbitrary formula; dropping the premise-match would combine facts proven under different contexts.

notes Producer-side, equational obligations are typically discharged by congruence-closure engines [188, 189]; the certificate carries not the closure computation but its residue — a chain of EQREFL/EQSYM/EQELIM nodes, each checked by recomputing one substitution.

### Lean statement

BernsteinLogic.Soundness . Term.evalQ\_subst\_congr

statement bound

```
theorem Term.evalQ_subst_congr {H : Hypernet} (σ : H.Valuation)
  {w : Wire} {u₁ u₂ : Term} (hu : Term.evalQ σ u₁ = Term.evalQ σ u₂) :
  ∀ (t : Term),
    Term.evalQ σ (Term.subst w u₁ t) = Term.evalQ σ (Term.subst w u₂ t)
```

BernsteinLogic.Soundness . Formula.holds\_subst\_congr

statement bound

```
theorem Formula.holds_subst_congr {H : Hypernet} (σ : H.Valuation)
  {w : Wire} {u₁ u₂ : Term} (hu : Term.evalQ σ u₁ = Term.evalQ σ u₂) :
  ∀ (F : Formula), (Formula.holds σ (F.subst w u₁) ↔ Formula.holds σ (F.subst w u₂))
```

BernsteinLogic.Soundness . Term.evalQ\_bshape

statement bound

```
/-- bshape soundness: a defined evaluation takes the broadcast shape. -/
theorem Term.evalQ_bshape {H : Hypernet} (σ : H.Valuation) :
  ∀ (t : Term) (s : Shape) (q : QTensor),
    t.bshape H = some s → Term.evalQ σ t = some q → q.shape = s
```

BernsteinLogic.Soundness . QTensor.eq\_of\_rel2

statement bound

```
/-- rel2 with an equality-forcing relation at equal shapes forces tensor
equality (the PROPOSED-B13 EqElim extraction: without equal shapes the
broadcast branches of rel2 defeat this — the header's counterexample). -/
theorem QTensor.eq_of_rel2 {r : Q → Q → Prop} {a b : QTensor}
  (hr : ∀ x y, r x y → x = y) (hs : a.shape = b.shape)
  (h : QTensor.rel2 r a b) : a = b
```



BernsteinLogic.Soundness . **sound\_eqElim**

statement bound

```

theorem sound_eqElim {H : Hypernet} {seq eqPremise bodyPremise : Sequent}
  {wire : Wire} {termT termU : Term} {bodyForm : Formula}
  (ihEq : eqPremise.semanticallyValid H)
  (ihBody : bodyPremise.semanticallyValid H)
  (hpre : seq.pre = eqPremise.pre)
  (hpreMatch : eqPremise.pre = bodyPremise.pre)
  (heqPost : eqPremise.post = .atom .eq termT termU)
  (htyEq : ∃ s, termT.bshape H = some s ∧ termU.bshape H = some s)
  (hbodySubT : substituteInFormula bodyForm wire termT bodyPremise.post)
  (hconclSubU : substituteInFormula bodyForm wire termU seq.post) :
  seq.semanticallyValid H

```

**The substitution the rule trusts, in full.** The payload fields `bodySubT` and `conclSubU` are equations against the substitution function displayed below: plain leaf replacement over a grammar with no binders, so no capture or freshness side condition exists anywhere. A skeptical reader should confirm by eye that `substituteInFormula` is nothing but that equation — the rule manipulates formulas syntactically and consults no valuation; semantic truth enters only through `sound_eqElim`, from the premise validity hypotheses.

BernsteinLogic.Core.Catalogue . **Term.subst**

definition bound

```

def Term.subst (w : Wire) (t : Term) : Term → Term
| .var w' => if w' = w then t else .var w'
| .const c => .const c
| .add l r => .add (Term.subst w t l) (Term.subst w t r)
| .mul l r => .mul (Term.subst w t l) (Term.subst w t r)
| .sub l r => .sub (Term.subst w t l) (Term.subst w t r)
| .neg u => .neg (Term.subst w t u)
| .idx u i => .idx (Term.subst w t u) i

```

BernsteinLogic.Core.Catalogue . **Formula.subst**

definition bound

```

def Formula.subst (w : Wire) (t : Term) : Formula → Formula
| .atom b l r => .atom b (Term.subst w t l) (Term.subst w t r)
| .inRegion u simplex S => .inRegion (Term.subst w t u) simplex S
| .and l r => .and (Formula.subst w t l) (Formula.subst w t r)
| .or l r => .or (Formula.subst w t l) (Formula.subst w t r)
| .tt => .tt
| .ff => .ff

```

BernsteinLogic.Core.Catalogue . **substituteInFormula**

definition bound

```

def substituteInFormula (F : Formula) (w : Wire) (t : Term) (F' : Formula) : Prop :=
  F' = F.subst w t

```

BernsteinLogic.Core.Catalogue . **PEqElimData**

definition bound

```

inductive PEqElimData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (eqPremise : Sequent) (eqProof : ProofObject H eqPremise)
  (bodyPremise : Sequent) (bodyProof : ProofObject H bodyPremise)
  (wire : Wire) (termT termU : Term) (bodyForm : Formula)
  (preEq : seq.pre = eqPremise.pre)
  (preMatch : eqPremise.pre = bodyPremise.pre)
  (eqPost : eqPremise.post = .atom .eq termT termU)
  (tyEq : ∃ s, termT.bshape H = some s ∧ termU.bshape H = some s)
  (bodySubT : substituteInFormula bodyForm wire termT bodyPremise.post)
  (conclSubU : substituteInFormula bodyForm wire termU seq.post)
  : PEqElimData H seq

```

## Correspondence (Steele display ↔ Lean)

reviewer-certified against the displayed texts

The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display

conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in **Ax**; sub-derivation premises as at **WEAKEN**.

- $t, u \mapsto \text{termT} \text{ termU} : \text{Term}$ ; the hole  $a \mapsto \text{wire} : \text{Wire}$ ; the body  $\Psi \mapsto \text{bodyForm} : \text{Formula}$ ;  $\Phi \mapsto \text{seq.pre}$ , shared with both premise sequents through **hpre** and **hpreMatch**.
- the equality premise  $\Phi \vdash t = u \mapsto \text{ihEq}$  with the shape tie **heqPost** : **eqPremise.post** = **.atom .eq termT termU**.
- the body premise  $\Phi \vdash \Psi[a := t] \mapsto \text{ihBody} : \text{bodyPremise.semanticallyValid H}$ , its post tied to  $\Psi[a := t]$  by **hbodySubT**; the goal post is tied to  $\Psi[a := u]$  by **hconclSubU**. Both ties run through **substituteInFormula**, quoted in full below: the one-line equation  $F' = F.\text{subst } w \text{ } t$  over plain leaf replacement — the grammar has no binders, so no capture or freshness side condition exists to omit.
- the typing premise  $\exists s. \text{bshape}_H(t) = s = \text{bshape}_H(u) \mapsto \text{htyEq}$ : existence of one shape  $s$  with **termT.bshape H = some s** and **termU.bshape H = some s** — the **some** carries the definedness the display's equation chain asserts by letting  $s$  range over shapes.
- conclusion  $\Phi \vdash \Psi[a := u] \mapsto$  the goal **seq.semanticallyValid H**. Certified alongside as the semantic engine: **Term.evalQ\_subst\_congr** and **Formula.holds\_subst\_congr** — substitution congruence along equal evaluations, the two lemmas the soundness proof routes through.

No width difference in either direction: statement and display carry the same typing gate (the commentary's counterexample is what its absence admits), and every hypothesis is consumed.

## Lean verifier receipts

module **BernsteinLogic.Soundness**; bound build **a20d90e66c13...** (bound from receipt **a9c0e34d3e3c9c8c**, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                 | minted               | layers             | status         |
|-------------------------|----------------------|--------------------|----------------|
| <b>a9c0e34d3e3c9c8c</b> | 2026-08-03T21:30:36Z | build+decorrelated | <b>current</b> |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module **BernsteinLogic.Core.Catalogue**; bound build **cbc92e976a1d...** (bound from receipt **5abdb3589707b5df**, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                 | minted               | layers             | status         |
|-------------------------|----------------------|--------------------|----------------|
| <b>24804e66edba0bc9</b> | 2026-08-03T20:25:09Z | build+decorrelated | <b>current</b> |
| <b>5abdb3589707b5df</b> | 2026-08-03T21:30:33Z | build+decorrelated | <b>current</b> |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

**gate-layer coverage of the bound build:**

**build** (lake build (elaboration)) **current**

**replay** (leanchecker (official kernel replay)) **stale only**

**hardened** (safe\_verify (hardened screen)) **stale only**

**decorrelated** (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory **bern\_catalogue**, **HOL4** development script **bern\_catalogueScript.sm**l **script bound**

**Definition termSubst\_def**

```

Definition termSubst_def:
  termSubst w t (TmVar w') = (if w' = w then t else TmVar w') ∧
  termSubst w t (TmConst c) = TmConst c ∧
  termSubst w t (TmAdd l r) = TmAdd (termSubst w t l) (termSubst w t r) ∧
  termSubst w t (TmMul l r) = TmMul (termSubst w t l) (termSubst w t r) ∧
  termSubst w t (TmSub l r) = TmSub (termSubst w t l) (termSubst w t r) ∧
  termSubst w t (TmNeg u) = TmNeg (termSubst w t u) ∧
  termSubst w t (TmIdx u i) = TmIdx (termSubst w t u) i
End

```

#### Definition formulaSubst\_def

```

Definition formulaSubst_def:
  formulaSubst w t (FAtom b l r) =
    FAtom b (termSubst w t l) (termSubst w t r) ∧
  formulaSubst w t (FInRegion u sx es) = FInRegion (termSubst w t u) sx es ∧
  formulaSubst w t (FAnd l r) =
    FAnd (formulaSubst w t l) (formulaSubst w t r) ∧
  formulaSubst w t (FOr l r) =
    FOr (formulaSubst w t l) (formulaSubst w t r) ∧
  formulaSubst w t FTt = FTt ∧
  formulaSubst w t FFf = FFf
End

```

#### Definition substituteInFormulaB\_def

```

Definition substituteInFormulaB_def:
  substituteInFormulaB fm w t fm' ↔ fm' = formulaSubst w t fm
End

```

## HOL verifier receipts

**documentary** hol-strict CLEAN census over the ported checker theory: `bern_catalogue`, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: `DISK_THM`; the four `bool` axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here — the earlier stale-run caveat (a 2026-07-18 script revision postdating the prior census) is retired by this run, which attests the current text. There is no signed HOL receipt store yet — these citations are documentary. Scope note, honest: the substitution core of this rule's check is what is ported; the broadcast-shape side condition (`tyEq`) rides the hypernet typing island, which the port deliberately defers to its proof-object stage. Semantic stratum, also ported: the soundness twin `sound_eqElim` (theory `bern_tr_fragclose`: `hol-strict clean`, 26 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail. Every hypothesis of the rule is used below and remarked on; the formal-anchor table maps each step to the Lean declaration that carries it. This section also fixes the semantic setting that the other equality and conversion sections cite.

**Setting.** Fix a hypernet  $H$ . A *valuation*  $\sigma$  assigns each *typed* wire a tensor of its declared shape over its declared dtype (an extended dtype admits infinite entries; untyped wires are assigned nothing), and  $\sigma$  is *admissible* when every primitive application in  $H$  satisfies its semantic relation — as at the `Ax` setting. The rational term evaluation  $\text{ev}_\sigma(t)$  is *partial* and the evaluator is structural and *strict* — a compound term evaluates exactly when its parts evaluate and the constructor's own side condition is met: shapes joining for the binary operations, nothing further for negation, a rank-1 operand with in-range index for projection — so a mention of an untyped wire, a wire holding an infinite value of an extended dtype, a broadcast-incompatible operation, or a projection whose operand is not rank-1 or whose index is out of range yields no value. A comparison atom  $t_1 \bowtie t_2$  *holds* at  $\sigma$  when both sides evaluate and the evaluated tensors stand in the relation  $\bowtie$  *elementwise under broadcast*: equal shapes compare entry by entry; a rank-0 (scalar) tensor against a tensor of any shape compares against every entry; two unequal positive-rank shapes never compare. A region atom holds when its term evaluates to a rank-1 tensor of extent the region's ambient dimension whose entries form a rational point of the region — a rank-0 evaluation is never coerced to a one-dimensional point. A sequent  $\Phi \vdash \Psi$  is *semantically valid* when every admissible  $\sigma$  satisfying  $\Phi$  satisfies  $\Psi$ ; counterexamples in the equality and conversion sections read, as at the `Ax` setting, over application-free hypernets, where every valuation is admissible, so an unsound conclusion genuinely falsifies. Note two consequences that drive everything below: equality atoms are *evaluation-level* facts, not syntactic ones; and a holding equality

between tensors of *different* broadcast shapes (a scalar against a vector) does not make the two sides interchangeable — it only says the scalar matches every entry.

**Claim** (sound\_eqElim). The rule’s payload supplies a hole wire  $a$ , terms  $t, u$ , a body formula  $\Psi$ , and checked side conditions: the two premises share the composite premise  $\Phi$ ; the first premise concludes the atom  $t = u$ ; the second premise concludes  $\Psi[a := t]$ ; the conclusion is  $\Psi[a := u]$ ; and (the typing condition)  $\exists s. \text{bshape}_H(t) = s = \text{bshape}_H(u)$  — both terms carry a *common defined* broadcast shape in  $H$ . If both premises are semantically valid, so is  $\Phi \vdash \Psi[a := u]$ .

**Lemma R8.1** (substitution congruence, term level). *Fix  $\sigma$ , and suppose  $\text{ev}_\sigma(t) = \text{ev}_\sigma(u)$  as partial values (both undefined, or both defined and equal). Then for every term  $e$ ,  $\text{ev}_\sigma(e[a := t]) = \text{ev}_\sigma(e[a := u])$ .*

*Proof.* Structural induction on  $e$ . The base case  $e = a$  is the hypothesis; the base cases  $e$  a constant or a different wire are reflexivity; every compound constructor is a congruence of the evaluator (the evaluator of a sum, product, difference, negation, or projection is a function of the evaluations of the parts). Two remarks. First, the hypothesis and conclusion are equalities of *partial* values — the lemma never manufactures definedness, it only transports whatever is there; in the theorem, definedness will come out of the equality premise. Second, the term grammar has no binders, so substitution is plain leaf replacement and no capture condition is needed anywhere.  $\square$

**Lemma R8.2** (substitution congruence, formula level). *Under the hypothesis of Lemma R8.1: for every formula  $F$ ,  $\sigma \models F[a := t]$  iff  $\sigma \models F[a := u]$ .*

*Proof.* Induction on  $F$ . Comparison atoms substitute in both sides and satisfy the same relation by Lemma R8.1; a region atom substitutes in its term slot only (its simplex and extension-set data carry no wires), so its holding — an assertion about the evaluated tensor — transports the same way;  $\wedge$  and  $\vee$  are the inductive steps;  $\top$  and  $\perp$  are untouched by substitution.  $\square$

**Lemma R8.3** (the shape pin). *Suppose  $\sigma \models t = u$ , and suppose  $t$  and  $u$  carry a common defined broadcast shape  $s$  in  $H$ . Then  $\text{ev}_\sigma(t) = \text{ev}_\sigma(u)$  — equality of tensors, not merely the elementwise relation.*

*Proof.* The holding atom yields evaluations  $A$  of  $t$  and  $B$  of  $u$  in the elementwise-equality relation. Broadcast-shape soundness (Term.evalQ\_bshape) pins  $A.\text{shape} = s = B.\text{shape}$ : a defined evaluation of a bshape-typed term lands exactly at the declared broadcast shape. Elementwise equality at *equal* shapes is equality of the (equal-length) entry lists, position by position, hence equality of the tensors. This is the one step that consumes the typing condition, and it is false without it: the elementwise relation between a rank-0 tensor and a vector says only that the scalar matches every entry (Remark R8.5).  $\square$

**Theorem R8.4** (soundness of EQELIM). *Under the payload’s checked hypotheses, if  $\Phi \vdash t = u$  and  $\Phi \vdash \Psi[a := t]$  are semantically valid, then so is  $\Phi \vdash \Psi[a := u]$ .*

*Proof.* Fix an admissible  $\sigma$  with  $\sigma \models \Phi$ . By validity of the first premise,  $\sigma \models t = u$ ; with the typing condition, Lemma R8.3 upgrades this to  $\text{ev}_\sigma(t) = \text{ev}_\sigma(u)$  (both defined and equal, so in particular equal as partial values). By validity of the second premise — its premise formula is  $\Phi$  itself, by the checked premise match —  $\sigma \models \Psi[a := t]$ . Lemma R8.2 transports this to  $\sigma \models \Psi[a := u]$ , which is the conclusion formula by the checked substitution equation.  $\square$

**Remark R8.5** (why the typing condition is not decoration). Drop the common-shape condition and the rule is *unsound* (orientation of the equality atom flipped here to spare an EQSYM step). Let  $v$  be a wire of shape  $[1]$  over  $\mathbb{Q}$ , let  $\Delta$  be the segment  $\text{conv}(1, 2) \subset \mathbb{Q}^1$ , and take

$$\Phi := (v = \frac{3}{2}) \wedge (v \in \mathcal{R}(\Delta, \emptyset)).$$

$\Phi$  is satisfiable: set  $v := [3/2]$ ; the equality holds by rank-0 broadcast (the scalar  $3/2$  matches the single entry), and the region atom holds.  $\text{Ax}$  and  $\wedge\text{E}$  derive  $\Phi \vdash v = 3/2$  and  $\Phi \vdash v \in \mathcal{R}(\Delta, \emptyset)$ ; the un-gated rule with body  $\Psi := (a \in \mathcal{R}(\Delta, \emptyset)), t := v, u := 3/2$  then concludes  $\Phi \vdash (\frac{3}{2} : \text{Term}) \in \mathcal{R}(\Delta, \emptyset)$  — which is *unsatisfiable*: a constant term evaluates to a rank-0 tensor, and a region atom demands a rank-1 tensor of extent equal to the ambient dimension. A satisfiable premise, an unsatisfiable conclusion. The root cause is precisely the gap between “elementwise equal under broadcast” and “interchangeable”:  $3/2 = [3/2]$  holds without the terms being exchangeable in region contexts. With the typing condition the payload is uninhabitable here —  $\text{bshape}_H(\frac{3}{2}) = []$  while  $\text{bshape}_H(v) = [1]$ , and no common  $s$  exists — so the certificate refuses. Note what the repair does *not* do: broadcast equalities remain derivable and usable as facts; what is forbidden is exactly using one as a Leibniz license.

**Remark R8.6** (the equational kit; relation to practice). EQELIM with EQREFL and EQSYM (next two sections) completes a certificate-side equational kit: reflexivity and symmetry are primitive, and transitivity is a one-node derivation — from  $\Phi \vdash t = u$  and  $\Phi \vdash u = w$ , apply EQELIM with body  $\Psi := (t = a)$ , the hole wire  $a$  chosen fresh for  $t$  (substitution replaces *every* occurrence of  $a$ , so a hole occurring in  $t$  would rewrite  $t$  itself), rewriting the  $u$  in the *first* premise (taken as body premise,  $\Psi[a := u]$ ) to  $w$ ; the derived node inherits the common-shape typing obligation on  $(u, w)$  — broadcast equality is not transitive without it (two tensors of unequal positive-rank shapes never compare, yet each can stand equal to one scalar between them:  $[0] = 0$  and  $0 = [0 \ 0]$  both hold while  $[0] = [0 \ 0]$  does not). On the producer side, equational goals are typically discharged by congruence-closure engines [188, 189]; what ships in a certificate is not the closure computation but its residue: a chain of EQREFL/EQSYM/EQELIM nodes, each checked by syntactic recomputation of one substitution.

#### proof step

Setting: valuation model, partial strict evaluation, atom and region semantics

#### formal anchor

Hypernet.Valuation, Term.evalQ, QTensor.rel2, Formula.holds (Layer 10)

Lemma R8.1  
 Lemma R8.2  
 Lemma R8.3  
 substitution equations  
 premise sharing  
 Theorem R8.4  
 Remark R8.5

Term.evalQ\_subst\_congr  
 Formula.holds\_subst\_congr  
 Term.evalQ\_bshape; QTensor.eq\_of\_rel2; payload field tyEq  
 substituteInFormula = Formula.subst (payload fields bodySubT, conclSubU)  
 payload fields preEq, preMatch  
 sound\_eqElim; wired as core RuleData.eqElim into ProofObject.sound  
 the counterexample inventory, Soundness.lean header

## Rule 9. EQREFL (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *EqRefl*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\exists s. \text{bshape}_H(t) = s \quad \text{wires}(t) \text{ finitely typed in } H}{\Phi \vdash t = t} \text{EQREFL}$$

EQREFL derives  $\Phi \vdash t = t$ , premise-free, for an arbitrary context  $\Phi$  (a fact true in the empty context is true under any premise; the proof never reads  $\Phi$ ). A producer uses it to seed equalities: the starting point of an EQELIM rewrite chain (transitivity, in particular, derives from these three rules and begins here), and the  $t = t$  obligations that alias unpacks and worked derivations demand.

That the rule has side conditions at all is its content. Equality atoms in this calculus are facts about *evaluation*, and evaluation is partial — a wire of an extended dtype can hold  $\pm\infty$  and fail the rational read; a shape-incompatible sum never evaluates. So “ $t = t$ ” costs something: the checker verifies the two typing conditions the semantics actually needs — the term's broadcast shape is *defined* in the hypernet, and every free wire is typed at a *finite* dtype. Under exactly these, evaluation is total at every valuation and reflexivity is free; drop either and the un-gated rule asserts an invalid sequent under the empty premise — in one case a conclusion failing only at extended-valued assignments, in the other one unsatisfiable outright (the proof section gives one concrete refutation per field). Nothing else is consulted: no premise, no admissibility, no evaluation — both conditions are static typing checks against the hypernet.

**notes** Where NORM applies, EQREFL is its diagonal; it is kept primitive so the equational kit stands in fragments that omit the polynomial engine entirely.

### Lean statement

BernsteinLogic.Soundness . Term.evalQ\_defined\_of\_bshape\_fin

statement bound

```
-- Totality of evaluation under the B13 typing pack: a `bshape`-typed
-- term over finitely-typed wires evaluates. Structural – no polynomial
-- view, so idx projections are covered (their rank-1 + range gates are
-- part of `bshape`). -/
theorem Term.evalQ_defined_of_bshape_fin {H : Hypernet} (σ : H.Valuation) :
  ∀ (t : Term) (s : Shape), t.bshape H = some s → t.finWires H →
  ∃ q, Term.evalQ σ t = some q ∧ q.shape = s
```

BernsteinLogic.Soundness . QTensor.rel2\_refl

statement bound

```
-- Reflexive relations lift to `rel2` on any tensor against itself. -/
theorem QTensor.rel2_refl {r : Q → Q → Prop} (hr : ∀ x, r x x)
  (a : QTensor) : QTensor.rel2 r a a
```

BernsteinLogic.Soundness . sound\_eqRefl

statement bound

```
-- EqRefl: under the B13 typing pack, t evaluates, and elementwise
-- equality against itself holds. -/
theorem sound_eqRefl {H : Hypernet} {seq : Sequent} {t : Term}
  (htyDef : ∃ s, t.bshape H = some s) (hfin : t.finWires H)
  (hpost : seq.post = .atom .eq t t) :
  seq.semanticallyValid H
```

**The two typing predicates, in full.** Both payload fields are static checks against the hypernet's wire typing — no valuation appears. `Term.bshape` is the broadcast-tolerant shape synthesis (scalars join with any shape; the projection case carries its own rank and range gates); `Term.finWires` demands a finite (non-extended) dtype at every free wire.

BernsteinLogic.Core.Catalogue . Term.bshape

definition bound

```
-- Broadcast-tolerant shape of a term: shapes must agree up to rank-0
-- broadcast (a scalar joins with any shape); dtypes are NOT consulted. -/
def Term.bshape (H : Hypernet) : Term → Option Shape
```

```

| .var w => (H.wireType w).map (·.shape)
| .const _ => some []
| .add l r | .mul l r | .sub l r =>
  match l.bshape H, r.bshape H with
  | some s1, some s2 =>
    if s1 = s2 then some s1
    else if s1 = [] then some s2
    else if s2 = [] then some s1
    else none
  | _, _ => none
| .neg t => t.bshape H
| .idx t i =>
  -- rank-1 argument, in-range index, scalar result (matches `evalQ`'s
  -- projection semantics – G1).
  match t.bshape H with
  | some [e] => if i < e then some [] else none
  | _ => none

```

BernsteinLogic.Core.Catalogue . **Term.finWires**

definition bound

```

/-- Every wire of the term is typed at a finite (non-ext) dtype – the
carrier of such a wire contains no  $\pm\infty$ , so entry reads are exact
rationals. -/
def Term.finWires (H : Hypernet) (t : Term) : Prop :=
  ∀ w ∈ t.freeWires, ∃ τ, H.wireType w = some τ ∧ τ.dtype.ext = Ext.fin

```

BernsteinLogic.Core.Catalogue . **PEqRefldata**

definition bound

```

inductive PEqRefldata : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (term : Term)
  (tyDef : ∃ s, term.bshape H = some s)
  (fin : term.finWires H)
  (postEq : seq.post = .atom .eq term term)
  : PEqRefldata H seq

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax.

- $t \mapsto$  the binder  $t : \text{Term}$ ; the two displayed typing premises  $\mapsto$  `htyDef`: existence of a shape  $s$  with  $t.bshape\ H = \text{some } s$  (defined broadcast shape — the `some` is the definedness), and `hfin` :  $t.finWires\ H$  (every free wire typed at a finite dtype, quoted below). Both are static checks against the hypernet; no valuation appears.
- conclusion  $\Phi \vdash t = t \mapsto$  the goal under `hpost` : `seq.post = .atom .eq t t`; the arbitrary context  $\Phi$  is premise-freeness by silence — no hypothesis mentions `seq.pre`, so the statement holds under every premise (the same representation the `PConstExact` dictionary describes).
- certified alongside as the semantic engine: `Term.evalQ_defined_of_bshape_fin` — under exactly the two typing conditions, evaluation is total at every valuation, at the synthesized shape — and `QTensor.rel2_refl`, the reflexivity of the elementwise comparison lifting.

No width difference in either direction: both typing hypotheses are consumed, and the proof section refutes the rule with either one dropped.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust



the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_fam\_resid, HOL4 development script bern\_fam\_residScript.sml script bound

#### Definition checkFam\_eqrefl\_def

```

Definition checkFam_eqrefl_def:
  checkFam_eqrefl (cols:columns) (ct:ctx) (i:num) ↔
    case frSeq cols ct i of
    | NONE => F
    | SOME sq =>
      case frTerm cols ct i 0 of
      | NONE => F
      | SOME t =>
        IS_SOME (termBshape ct.ws t) ∧ finWiresB ct.ws t ∧
        sq.post = FAtom BowEq t t
End

```

#### Theorem famEq\_eqrefl

```

Theorem famEq_eqrefl:
  wfColumns cols ct ∧ mkCtx cols = SOME ct ∧
  parseCert (cols.hdrImage ++ rest) = SOME (ws, pf) ∧
  nodeCorr cols ct i (CertPf sq nd) ∧
  FST (THE (tagOf cols i)) = 23 ⇒
  (checkFam_eqrefl cols ct i ↔ armEqRefl ws sq nd)

```

## HOL verifier receipts

documentary Documentary citations. Family validation: bern\_fam\_resid built GREEN with all ten famEq arms validated, hol-strict clean; spine census: bern\_joins GREEN under hol-strict (55 theorems, 0 offences, 208-theory ancestry including this theory), checkJoins\_eq\_checkPf verified verbatim in the built signature. The cited famEq lemma is the kernel-checked per-node equivalence: under well-formed columns (wfColumns/mkCtx/nodeCorr) at this rule's wire tag, the columnar family check coincides with the flat reference walk's arm for this rule. There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twin sound\_eqRefl (theory bern\_tr\_ready8: hol-strict clean, 8 theorems, 0 offences,

verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (checkPf\_sound\_frag, theory bern\_tr\_splice, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1). Setting (valuations, partial evaluation, elementwise-under-broadcast atom semantics) as fixed in the EQELIM section.

**Claim** (sound\_eqRef1). The payload supplies a term  $t$  and two checked typing conditions:  $\text{bshape}_H(t)$  is *defined* (some shape  $s$ ), and every free wire of  $t$  is typed at a *finite* (non-extended) dtype. Then  $\Phi \vdash t = t$  is semantically valid for an *arbitrary* premise  $\Phi$  — the rule is premise-free, and  $\Phi$  rides along unread (a fact true in the empty context is true under any premise). Indeed the proof below never reads  $\Phi$ , the admissibility of  $\sigma$ , or the premise-satisfaction hypothesis at all: the conclusion holds at *every* valuation.

That anything must be checked here at all is the point of the section. Equality atoms are evaluation-level facts (EQELIM setting), so “ $t = t$ ” is not free: it asserts that  $t$  *evaluates* — and evaluation is partial.

**Lemma R9.1** (totality under the typing pack). *If  $\text{bshape}_H(t) = s$  and every free wire of  $t$  is finitely typed, then at every valuation  $\sigma$  the evaluation  $\text{ev}_\sigma(t)$  is defined, with shape  $s$ .*

*Proof.* Structural induction on  $t$ ; both typing conditions descend to subterms (a subterm’s free wires are among  $t$ ’s, and  $\text{bshape}$  is synthesized bottom-up, so a defined  $\text{bshape}$  for  $t$  is defined on every subterm). A wire read succeeds because the wire is typed at a finite dtype — an untyped wire declares no dtype at all, so the finite-wires condition already excludes it: the valuation’s entries lie in the dtype’s value set, which for a finite dtype contains no  $\pm\infty$ , so the whole-tensor rational read succeeds, at the declared shape. A constant evaluates to its rank-0 tensor. For a compound term, the induction hypotheses give defined operands whose shapes are, by the definition of  $\text{bshape}$  (quoted in full beside this section), either equal or joinable through rank-0 broadcast — and on such shapes the elementwise operations are total; the result lands at the join. A projection’s rank-1-and-in-range gates are part of  $\text{bshape}$ ’s definedness, so the projected read is in range. (Term.evalQ\_defined\_of\_bshape\_fin.)  $\square$

**Lemma R9.2** (reflexivity lifts). *For any tensor  $A$ , the elementwise-under-broadcast equality relation holds between  $A$  and  $A$ .*

*Proof.* The shapes are equal, so the relation reads the entry list zipped against itself: every pair on the diagonal is  $(x, x)$ . (QTensor.rel2\_refl.)  $\square$

**Theorem R9.3** (soundness of EQREFL). *Under the payload’s typing conditions,  $\Phi \vdash t = t$  is semantically valid for every  $\Phi$ .*

*Proof.* Fix  $\sigma$ . Lemma R9.1 gives  $\text{ev}_\sigma(t) = A$  for some tensor  $A$ ; Lemma R9.2 closes the atom with both sides read as  $A$ .  $\square$

**Remark R9.4** (why each typing condition is load-bearing). Drop the finite-wires condition: let  $w$  be a wire of an *extended* dtype (its value set admits  $+\infty$ ), and let  $\sigma$  assign it a tensor with an infinite entry — admissible, since nothing constrains a bare wire. The rational read of  $w$  fails,  $t := w$  does not evaluate, and the atom  $w = w$  is *false* at  $\sigma$ ; with  $\Phi := \top$  the sequent  $\top \vdash w = w$  is refuted. Drop shape-definedness: take  $t := w_1 + w_2$  with  $w_1$  of shape [2] and  $w_2$  of shape [3], both finite — every wire reads, but the elementwise sum refuses the incompatible shapes,  $t$  never evaluates, and  $\vdash t = t$  is again *false* everywhere. In both cases the un-gated rule would assert an invalid sequent under a satisfiable (indeed empty) premise — in the first case the conclusion fails at the exhibited valuation (a finite assignment would satisfy it); in the second it is unsatisfiable outright. The typing pack is exactly the semantic price of equality-by-evaluation; it is the same field family that repairs NORM, there compounded with canonical-form equality.

**Remark R9.5** (taxonomy note). Wherever NORM applies, EQREFL is its diagonal ( $t_1 = t_2 = t$ , canonical forms trivially equal). It is nonetheless kept primitive: the development’s frozen minimal fragment (v0 in the sources) excludes the polynomial engine outright, and the equational kit must stand without it.

| proof step                | formal anchor                                                       |
|---------------------------|---------------------------------------------------------------------|
| Lemma R9.1                | Term.evalQ_defined_of_bshape_fin; payload fields tyDef, fin         |
| finite-wire reads succeed | Hypernet.Valuation.wireQ_isSome                                     |
| Lemma R9.2                | QTensor.rel2_refl                                                   |
| Theorem R9.3              | sound_eqRef1; wired as core RuleData.eqRef1 into Proof0Object.sound |
| typing predicates         | Term.bshape, Term.finWires (Core/Catalogue)                         |

## Rule 10. EQSYM (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *EqSym*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash t = u}{\Phi \vdash u = t} \text{EQSYM}$$

EQSYM flips an equality atom: from  $\Phi \vdash t = u$  conclude  $\Phi \vdash u = t$ . The need is real because downstream consumers of equalities match atoms *syntactically* with a fixed orientation — EQELIM's equality premise, in particular, names its two terms in order — while only the combination rules expose both polarities of an equality in their own matching. A producer whose congruence engine emits  $t = u$  but whose rewrite needs  $u = t$  inserts this one node.

What the checker recomputes: three syntactic form equations (premise atom, conclusion atom, shared context) — nothing else. The rule carries *no* typing fields, and that absence is principled twice over: the premise's holding atom already witnesses both evaluations (nothing must be manufactured, contrast EQREFL), and the elementwise-under-broadcast equality relation is symmetric branch by branch, so no shape agreement is needed (contrast EQELIM, whose typing condition guards transport into a *different* formula — exactly what a flip never does).

Every payload field is structural; there is no droppable hypothesis to exhibit a counterexample against. The rule is core rather than admissible because its would-be unpack (through EQREFL + EQELIM) demands typing side data the premise atom does not syntactically carry — the unpack is not computable from the node it would replace.

### Lean statement

BernsteinLogic.Soundness . QTensor.rel2\_symm

statement bound

```
-- `rel2` of a symmetric relation is symmetric (all three broadcast
branches flip pointwise). -/
theorem QTensor.rel2_symm {r : Q → Q → Prop}
  (hr : ∀ x y, r x y → r y x) {a b : QTensor}
  (h : QTensor.rel2 r a b) : QTensor.rel2 r b a
```

BernsteinLogic.Soundness . sound\_eqSym

statement bound

```
-- EqSym: flip the premise atom's two evaluations and the symmetric
`rel2`. -/
theorem sound_eqSym {H : Hypernet} {seq premise : Sequent} {tT tU : Term}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hprePost : premise.post = .atom .eq tT tU)
  (hpost : seq.post = .atom .eq tU tT) :
  seq.semanticallyValid H
```

BernsteinLogic.Core.Catalogue . PEqSymData

definition bound

```
inductive PEqSymData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (termT termU : Term)
  (preEq      : seq.pre = premise.pre)
  (premisePost : premise.post = .atom .eq termT termU)
  (postEq     : seq.post = .atom .eq termU termT)
  : PEqSymData H seq
```

### Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display

conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in **Ax**; sub-derivation premise as at **WEAKEN**.

- $t, u \mapsto \text{tT } \text{tU} : \text{Term}$ ; the premise judgment  $\Phi \vdash t = u \mapsto \text{ih}$  with the shape tie  $\text{hpremPost} : \text{premise.post} = .\text{atom} .\text{eq } \text{tT } \text{tU}$ ;  $\Phi \mapsto \text{premise.pre}$ , shared via  $\text{hpre}$ .
- conclusion  $\Phi \vdash u = t \mapsto$  the goal under  $\text{hpost} : \text{seq.post} = .\text{atom} .\text{eq } \text{tU } \text{tT}$  — the same two term binders, transposed; nothing else changes.
- certified alongside: `QTensor.re12_symm`, the branchwise symmetry of the elementwise comparison lifting — the reason no typing field exists (the premise's holding atom already witnesses both evaluations, as the commentary explains).

No width difference in either direction; every hypothesis is consumed.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

`build` (lake build (elaboration)) current

`replay` (leanchecker (official kernel replay)) stale only

`hardened` (safe\_verify (hardened screen)) stale only

`decorrelated` (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory `bern_fam_resid`, HOL4 development script `bern_fam_residScript.sml` script bound

### Definition `checkFam_eqsym_def`

```

Definition checkFam_eqsym_def:
  checkFam_eqsym (cols:columns) (ct:ctx) (i:num) ⇔
    case frSeq cols ct i of
    | NONE => F
    | SOME sq =>
      case frChildSeq cols ct i 0 of
      | NONE => F
      | SOME ps =>
        case frTerm cols ct i 0 of

```

```

| NONE => F
| SOME tT =>
  case frTerm cols ct i 1 of
  | NONE => F
  | SOME tU =>
    sq.pre = ps.pre ∧
    ps.post = FAtom BowEq tT tU ∧
    sq.post = FAtom BowEq tU tT
End

```

**Theorem famEq\_eqsym**

```

Theorem famEq_eqsym:
  wfColumns cols ct ∧ mkCtx cols = SOME ct ∧
  parseCert (cols.hdrImage ++ rest) = SOME (ws, pf) ∧
  nodeCorr cols ct i (CertPf sq nd) ∧
  FST (THE (tagOf cols i)) = 24 ⇒
  (checkFam_eqsym cols ct i ⇔ armEqSym ws sq nd)

```

**HOL verifier receipts**

[documentary](#) Documentary citations. Family validation: `bern_fam_resid` built GREEN with all ten `famEq` arms validated, `hol-strict clean`; spine census: `bern_joins` GREEN under `hol-strict` (55 theorems, 0 offences, 208-theory ancestry including this theory), `checkJoins_eq_checkPf` verified verbatim in the built signature. The cited `famEq` lemma is the kernel-checked per-node equivalence: under well-formed columns (`wfColumns/mkCtx/nodeCorr`) at this rule's wire tag, the columnar family check coincides with the flat reference walk's arm for this rule. There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twin `sound_eqSym` (theory `bern_tr_b14`: `hol-strict clean`, 2 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

**Proof**

Genre: core rule (tier 1). Setting as fixed in the `EqELIM` section.

**Claim** (`sound_eqSym`). The payload supplies terms  $t, u$ ; the premise concludes the atom  $t = u$  over premise formula  $\Phi$ , shared with the conclusion sequent; the conclusion is the atom  $u = t$ . If the premise is semantically valid, so is  $\Phi \vdash u = t$ .

**Lemma R10.1** (the elementwise relation of a symmetric relation is symmetric). *Let  $r$  be a symmetric binary relation on  $\mathbb{Q}$ , and suppose tensors  $A, B$  stand in the elementwise-under-broadcast lift of  $r$ . Then  $B, A$  stand in it too.*

*Proof.* The lift has three live branches, and each is separately symmetric — this per-branch analysis is the entire content, so we spell it out. *Equal shapes*: the relation reads the zipped entry lists; the swapped claim reads the zip in the other order, and each swapped pair is closed by symmetry of  $r$ . *Rank-0  $A$  against positive-rank  $B$* : the hypothesis says  $r(A_{\text{scalar}}, y)$  for every entry  $y$  of  $B$ ; the swapped claim is judged in the *other* scalar-broadcast branch (rank-0 on the right), which demands  $r(y, A_{\text{scalar}})$  for every  $y$  — symmetry of  $r$ , pointwise. *Rank-0  $B$* : mirror image. Two unequal positive-rank shapes never stand in the lift, so there is nothing to prove there. (`qTensor.rel2_symm`)  $\square$

**Theorem R10.2** (soundness of `EqSYM`). *If  $\Phi \vdash t = u$  is semantically valid, so is  $\Phi \vdash u = t$ .*

*Proof.* Fix an admissible  $\sigma \models \Phi$ . The premise yields evaluations  $A$  of  $t$  and  $B$  of  $u$  in the elementwise equality lift. Present the same two evaluations in the opposite roles:  $u$  evaluates to  $B$ ,  $t$  to  $A$ , and Lemma R10.1 (with  $r$  the symmetric relation  $=$ ) closes the atom  $u = t$ .  $\square$

**Remark R10.3** (the negative space: what this rule does *not* need). The rule has no droppable side condition — every payload field is structural (the three form equations) — so the honest remarks here concern the two fields conspicuously absent. No typing pack (`EqREFL`'s `tyDef/fin`): the premise's holding atom already *witnesses* both evaluations, and symmetry only re-uses the premise's own evidence; nothing must be manufactured. No common-shape condition (`EqELIM`'s `tyEq`): Lemma R10.1 is shape-agnostic — a broadcast equality flips to the mirror broadcast equality, same content, no interchangeability claim ever made. The contrast triangulates `EqELIM`'s repair: the typing condition there guards *transport into a different formula*, which is exactly what `EqSYM` never does.

**Remark R10.4** (why a primitive, when a two-node unpack exists on paper). From  $\Phi \vdash t = u$  one can derive  $\Phi \vdash u = t$  through `EqREFL` + `EqELIM`: body  $\Psi := (a = t)$  with the hole wire  $a$  fresh for  $t$  — substitution replaces *every* occurrence of  $a$ , and freshness is what makes  $\Psi[a := t]$  the atom  $t = t$  and  $\Psi[a := u]$  the atom  $u = t$  — rewriting the left occurrence of  $t$  in  $t = t$  to  $u$ . But the

derivation demands EQREFL’s typing pack on  $t$  and EQELIM’s common-shape condition on  $(t, u)$  — side data the premise atom does not syntactically carry, and which a producer cannot always reconstruct at the flip site. The primitive constructor instead consumes the premise’s semantic evidence directly (the proof above), making the flip unconditional. In the catalogue’s three-tier taxonomy that obstruction is what sorts EQSYM core rather than admissible: the unpack is not payload-computable from the node it would replace.

| proof step                                     | formal anchor                                                                                                                                   |
|------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Lemma R10.1<br>Theorem R10.2<br>form equations | QTensor.rel2_symm<br>sound_eqSym; wired as core RuleData.eqSym into ProofObject.sound<br>payload fields preEq, premisePost, postEq (PEqSymData) |

## Rule 11. ANTISYM (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Antisym*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash t \leq u \quad \Phi \vdash t \geq u}{\Phi \vdash t = u} \text{ ANTISYM}$$

ANTISYM squeezes an equality out of two opposite bounds: from  $\Phi \vdash t \leq u$  and  $\Phi \vdash t \geq u$  — the same two terms, the same context — conclude  $\Phi \vdash t = u$ . A producer reaches for it wherever a pinching argument closes: an optimization pass that certifies matching upper and lower bounds on the same quantity, a fixed-point equality assembled from two directional inequalities, or any pipeline whose native outputs are order facts but whose consumer (EQELIM's rewrite chains, in particular) wants an equality atom.

The rule earns its place in the core by a non-derivability fact, not by convenience: equality-from-inequalities has no derivation in the remainder of the calculus. Provably — an induction over derivations shows that every equality concluded from an equality-atom-free context equates terms with identical polynomial normal forms — so no chain of existing rules can ever conclude  $t = u$  from the two bounds when the sides are genuinely different terms. Historically the calculus carried this inference only inside two bespoke rules for the ReLU function, which were thereby load-bearing for something much more general than ReLU; with antisymmetry available as its own rule, that monopoly ends.

The side conditions are purely structural: two premise-atom form equations, the conclusion-atom form equation, and the two context equations tying each premise sequent to the conclusion's context. No typing fields — and the absence is principled (contrast EQREFL, which must *manufacture* an evaluation and so carries the typing pack): each premise atom holds only where both sides already evaluate finitely at compatible shapes, and that is exactly the evaluation evidence the equality atom's semantics consumes. Dropping either premise leaves a one-sided bound, and any strictly-below evaluation pair refutes the un-gated rule.

One reading discipline, stated here because the conclusion invites over-reading: the concluded equality holds in the calculus's elementwise-under-broadcast sense, which suffices for the truth of the atom and for every consumer that reads it semantically — but it is not a license to interchange  $t$  and  $u$  inside an arbitrary formula. Substitution rides EQELIM, whose common-defined-shape typing condition remains in force unchanged (that section constructs the refutation its absence would admit).

**notes** This rule is *staged, not enacted*: what exists today is the statement register — the soundness theorem below, proven in Lean with a HOL twin — while the certificate format knows nothing of it yet. No payload constructor, no wire tag, no checker arm: a certificate cannot currently carry an ANTISYM node, and this section will gain its checker-side anchors when enactment lands. The mathematics is settled; the plumbing is queued.

### Lean statement

BernsteinLogic.Soundness . QTensor.rel2\_conj

statement bound

```
/-- Two `rel2` facts at the SAME tensor pair combine pointwise: the
broadcast branch taken is a function of the two shapes alone, so the
quantified domains coincide and the pointwise conjunction lifts.
(Stated for `sound_antisym`; any pointwise combination rides it.) -/
theorem QTensor.rel2_conj {r s t : Q → Q → Prop}
  (hrst : ∀ x y, r x y → s x y → t x y) {a b : QTensor}
  (hr : QTensor.rel2 r a b) (hs : QTensor.rel2 s a b) :
  QTensor.rel2 t a b
```

BernsteinLogic.Soundness . sound\_antisym

statement bound

```
/-- Antisym: `≤` and `≥` premise validities at the same term pair force
the equality atom — entrywise `le_antisymm` lifted through `rel2`
(`QTensor.rel2_conj`: both premises constrain the same evaluated pair,
so the broadcast branches align). Like `sound_eqSym` and unlike
`sound_eqRefl`, no B13 typing pack is needed: the premise atoms only
hold where both sides already evaluate finitely, which is exactly the
evaluation evidence the conclusion atom needs. Core rather than
derived: the equality-from-inequalities inference has no catalogue
derivation — provably, since derivable equalities from eq-free contexts
have canonically-equal sides — so the rule must be primitive
(ReLU-Pos/Neg had been core solely as carriers of this inference). -/
```



```
theorem sound_antisym {H : Hypernet} {seq premiseLe premiseGe : Sequent}
  {tT tU : Term}
  (ihLe : premiseLe.semanticallyValid H)
  (ihGe : premiseGe.semanticallyValid H)
  (hpreLe : seq.pre = premiseLe.pre)
  (hpreGe : seq.pre = premiseGe.pre)
  (hpreLePost : premiseLe.post = .atom .le tT tU)
  (hpreGePost : premiseGe.post = .atom .ge tT tU)
  (hpost : seq.post = .atom .eq tT tU) :
  seq.semanticallyValid H
```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; sub-derivation premises as at **WEAKEN**.

- $t, u \mapsto tT \ tU : \text{Term}$ ; the two premise judgments  $\Phi \vdash t \leq u$  and  $\Phi \vdash t \geq u \mapsto ihLe, ihGe$  with the shape ties  $hpreLePost : \text{premiseLe.post} = .\text{atom} .\text{le} \ tT \ tU$  and  $hpreGePost : \text{premiseGe.post} = .\text{atom} .\text{ge} \ tT \ tU$  — note the  $\geq$  premise binds its own atom former at the *same* binder order ( $tT, tU$ ), not a flipped  $\leq$  atom.
- the shared context: the display writes one  $\Phi$  across all three sequents; the statement renders it as *two* equations,  $hpreLe : \text{seq.pre} = \text{premiseLe.pre}$  and  $hpreGe : \text{seq.pre} = \text{premiseGe.pre}$  — each premise tied to the conclusion's context, premise-to-premise equality following by transitivity. Same content, one equation more than the display suggests.
- conclusion  $\Phi \vdash t = u \mapsto$  the goal under  $hpost : \text{seq.post} = .\text{atom} .\text{eq} \ tT \ tU$ .
- no typing hypotheses appear on either side — principled, as the commentary explains (the premise atoms carry the evaluation evidence; contrast **EQREFL**'s `htyDef/hfin`).
- certified alongside as the semantic engine: **QTensor.reI2\_conj** — two elementwise-under-broadcast facts at the *same* tensor pair combine pointwise, because the broadcast branch taken is a function of the two shapes alone, so the quantified domains coincide. The soundness proof feeds it entrywise antisymmetry of the rational order.

Width note, both directions: every hypothesis is consumed, and the one place statement and display could be read apart is the conclusion atom's broadcast-sense semantics — the proven equality is the elementwise-under-broadcast lift (a scalar can stand equal to a vector whose every entry it matches), which is the equality atom's meaning everywhere in this calculus; the commentary's reading discipline names what that does *not* license.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **stale only**

hardened (safe\_verify (hardened screen)) **stale only**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_antisym, HOL4 development script bern\_tr\_antisymScript.sml script bound**Theorem qtRel2\_conj**

```

Theorem qtRel2_conj:
  ∀r s t a b.
    (∀x y. r x y ∧ s x y ⇒ t x y) ∧
    qtRel2 r a b ∧ qtRel2 s a b ⇒
    qtRel2 t a b

```

**Theorem sound\_antisym**

```

Theorem sound_antisym:
  ∀H sq premiseLe premiseGe tT tU.
    seqSemanticallyValid H premiseLe ∧
    seqSemanticallyValid H premiseGe ∧
    sq.pre = premiseLe.pre ∧
    sq.pre = premiseGe.pre ∧
    premiseLe.post = FAtom BowLe tT tU ∧
    premiseGe.post = FAtom BowGe tT tU ∧
    sq.post = FAtom BowEq tT tU ⇒
    seqSemanticallyValid H sq

```

## HOL verifier receipts

documentary Documentary citations, one register-honest caveat: this theory is the staged rule's whole standalone footprint (both theorems in one theory, deliberately, so the staged work moves as a unit), and — unlike the enacted siblings on this page — no isolated-build-host verification census is attached to it yet; the citation attests the script text, not a checked build artifact. Statement fidelity follows the ported-soundness register used throughout: the Lean statement's implicit arguments become leading universals and its curried named hypotheses a conjunction in source order; `.atom .le/.ge/.eq` renders as `FAtom BowLe/BowGe/BowEq`; the pointwise antisymmetry witness closes by the rational order's antisymmetry lemma. Because the rule has no checker arm, there is no family-check or certified-fragment claim to make here — the semantic stratum is the entire port, matching the statement-register staging the note describes.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1), staged — statement register only; see the section note. Setting as fixed in the EQELIM section.

**Claim** (`sound_antisym`). Two premise sequents conclude the atoms  $t \leq u$  and  $t \geq u$ , each over the premise formula  $\Phi$  shared with the conclusion sequent; the conclusion is the atom  $t = u$ . If both premises are semantically valid, so is  $\Phi \vdash t = u$ .

**Lemma R11.1** (elementwise lifts at a shared tensor pair combine pointwise). *Let  $r, s, w$  be binary relations on  $\mathbb{Q}$  with  $r(x, y) \wedge s(x, y) \Rightarrow w(x, y)$  for all  $x, y$ , and suppose tensors  $A, B$  stand in the elementwise-under-broadcast lift of  $r$  and also in the lift of  $s$ . Then they stand in the lift of  $w$ .*

*Proof.* The branch the lift takes is a function of the two shapes alone — not of the relation — so the two hypotheses and the goal all quantify over the *same* domain, and the pointwise implication closes each point. Spelled per branch: *equal shapes*: all three statements read the same zipped entry lists, pair by pair. *Rank-0 A against positive-rank B*: all three quantify  $A$ 's scalar against each entry of  $B$ ; mirror image for rank-0  $B$ . *Two unequal positive-rank shapes*: no lift holds there at all, so the hypothesis for  $r$  is already false and there is nothing to prove. (`QTensor.reL2_conj`.) □

**Theorem R11.2** (soundness of `ANTISYM`). *If  $\Phi \vdash t \leq u$  and  $\Phi \vdash t \geq u$  are semantically valid, so is  $\Phi \vdash t = u$ .*

*Proof.* Fix an admissible  $\sigma \models \Phi$ . The  $\leq$  premise yields evaluations  $A$  of  $t$  and  $B$  of  $u$  in the elementwise  $\leq$ -lift; the  $\geq$  premise yields evaluations  $A'$  of  $t$  and  $B'$  of  $u$  in the elementwise  $\geq$ -lift. Evaluation is a function of the valuation and the term, so  $A' = A$  and  $B' = B$ : both premises constrain the *same* evaluated pair, which is what makes Lemma R11.1 applicable at all. Apply it with  $r := \leq$ ,  $s := \geq$ ,  $w := =$  and the pointwise fact  $x \leq y \wedge x \geq y \Rightarrow x = y$  — antisymmetry of the order on  $\mathbb{Q}$  — and the equality lift closes the atom  $t = u$ . □

**Remark R11.3** (the negative space: no typing pack, and why both premises). No typing fields (`EQREFL`'s `tyDef/fin`): that rule must *manufacture* an evaluation for its premise-free conclusion, and pays for totality with static typing conditions; here each premise atom holds only where both sides already evaluate finitely at compatible shapes, and the proof above consumes exactly that evidence — nothing is manufactured (`EQSYM` makes the same economy for the same reason). Both orientations are load-bearing: dropping either premise leaves a one-sided bound, and any evaluation pair with  $A$  strictly below  $B$  entrywise satisfies the surviving premise while refuting the conclusion — the counterexample is immediate at scalar shape already.

*Remark R11.4* (what the broadcast-sense conclusion does not license). The concluded atom holds in the elementwise-under-broadcast sense: a scalar stands equal to a vector whose every entry it matches. That is the equality atom’s meaning everywhere in this calculus — sufficient for atom truth and for every consumer that reads the atom semantically — but it is not a license to interchange  $t$  and  $u$  inside an arbitrary formula. Substitution rides EQELIM, whose common-defined-shape typing condition remains in force unchanged: that section constructs a satisfiable-to-unsatisfiable flip from exactly the scalar-equals-vector equalities this rule can legitimately conclude.

*Remark R11.5* (why core, and what is staged). Core because the inference is not derivable: an induction over derivations shows every equality concluded from an equality-atom-free context equates terms with identical polynomial normal forms, so no chain of existing rules concludes  $t = u$  from the two bounds when the sides are genuinely different terms. Historically the calculus carried antisymmetry only inside two bespoke rules for the ReLU function; a core ANTISYM ends that monopoly and lets bespoke rules be judged on their own content. Staged: the theorem above is the statement register — no payload constructor, wire tag, or checker arm exists yet, so certificates cannot carry the rule today; the anchors table below will gain its “wired as core” line at enactment.

| proof step    | formal anchor                                                                              |
|---------------|--------------------------------------------------------------------------------------------|
| Lemma R11.1   | QTensor.rel2_conj                                                                          |
| Theorem R11.2 | sound_antisym; statement register only — not yet wired (no ProofObject constructor exists) |
| HOL twin      | theory bern_tr_antisym: qtRel2_conj, sound_antisym                                         |

## Rule 12. CONVELIM (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *ConvElim*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash x \in \mathcal{R}(\Delta, S)}{\Phi \vdash \bigwedge_{i=0}^n \text{canon}(\epsilon_i \lambda_i(x)) \leq 0 \quad \epsilon_i = +1 \text{ if } i \in S, \quad -1 \text{ if } i \in \bar{S}} \text{CONVELIM}$$

CONVELIM opens a region atom: from  $\Phi \vdash x \in \mathcal{R}(\Delta, S)$  it concludes the region's defining barycentric halfspace conjunction — one  $(\cdot) \leq 0$  atom per support vertex, in a pinned canonical spelling. It is the entry move of every argument that reasons *linearly* about a geometric premise: BERNCON consumes region atoms natively, but the combination rules (FARKAS, SYZGY) match ordinary comparison atoms by exact canonical polynomial, and this rule is what turns a region fact into atoms they can see.

What the checker recomputes: the *entire* conclusion, from  $(x, \Delta, S)$  alone — the two barycentric affine forms of the 1-D two-vertex support (weights are determined on a simplex, so nothing is chosen and nothing rides the certificate), the sign-folding that turns each core vertex's wanted  $\lambda \geq 0$  into a  $(-\lambda) \leq 0$  atom (the all- $\leq$  discipline the downstream matchers consume), and the strictly positive primitive-integer rescale that makes the spelling unique — then demands syntactic equality with the cited conclusion. No producer text is trusted; the side condition is one decidable equation. Outside the checkable fragment (ambient dimension 1, support exactly two vertices at distinct coordinates) the build refuses and the rule cannot fire — refusal is trivially sound, and the distinct-coordinates gate is precisely where a degenerate simplex's division by zero is made unrepresentable.

The rule needs no typing or positivity fields: definedness and shape of the region term are read out of the region atom's own semantics. Dropping the premise-form equation would sever the conclusion from any premise at all; dropping the canonical-spelling pin would not make the rule unsound but would strand its conclusions — equivalent spellings never match downstream — which is why the spelling is checked rather than suggested.

notes Geometric reading of extension sets, 1-D:  $S = \emptyset$  is the segment  $[V_0, V_1]$ ; naming one vertex stretches the region to the closed ray away from the other — membership with a *nonpositive* weight on the extended vertex. The region atom is thus the calculus's interface between simplex geometry and sign arithmetic, and this rule with CONVLNTR is that interface's two directions, exact inverses on the fragment.

### Lean statement

BernsteinLogic.Soundness . barycentricHalfspaceBuild\_spec

statement bound

```
/-- Structure of a successful barycentric-halfspace build (the 1-D case
`baryCoeffs1D` gates): the F4 all- $\leq$  canonical conjunction, one
`f4VertexAtom` per vertex of the filter pair. -/
theorem barycentricHalfspaceBuild_spec {t : Term} {simplex : Simplex}
  {S : ExtensionSet} {F : Formula}
  (h : barycentricHalfspaceBuild t simplex S = some F) :
  S.simplex = simplex  $\wedge$  simplex.complex.ambDim = 1  $\wedge$ 
   $\exists$  (a b : Fin simplex.complex.nVertices) (V0 V1 :  $\mathbb{Q}$ ),
    (List.finRange simplex.complex.nVertices).filter
      (fun v => simplex.contains v) = [a, b]  $\wedge$ 
    ( $\forall$  j, simplex.complex.coord a j = V0)  $\wedge$ 
    ( $\forall$  j, simplex.complex.coord b j = V1)  $\wedge$ 
    V0  $\neq$  V1  $\wedge$ 
    F = .and
      (f4VertexAtom (S.containsNat a.val)
        (V1 / (V1 - V0)) (-1 / (V1 - V0)) t)
      (f4VertexAtom (S.containsNat b.val)
        (-V0 / (V1 - V0)) (1 / (V1 - V0)) t)
```

BernsteinLogic.Soundness . f4VertexAtom\_holds\_iff

statement bound

```
/-- The F4 vertex atom holds iff the signed barycentric value is
nonpositive: ` $\lambda \leq 0$ ` for an extended vertex, ` $0 \leq \lambda$ ` for a core one
(the sign folded into the term, the rescale cancelled). -/
theorem f4VertexAtom_holds_iff {H : Hypernet} { $\sigma$  : H.Valuation} {t : Term}
  {q : QTensor} (hq : Term.evalQ  $\sigma$  t = some q) (hsq : SingletonShape q.shape)
```

```
(inS : Bool) (off slp : ℚ) :
Formula.holds σ (f4VertexAtom inS off slp t) ↔
  (if inS then off + slp * q.scalar ≤ 0
   else 0 ≤ off + slp * q.scalar)
```

BernsteinLogic.Soundness . **maskRegion\_pair\_elim**

statement bound

```
-- Extraction: a masked region membership over a two-vertex 1-D support
pins the barycentric weights to their rational closed forms, and exports
the sign of each weight according to its extension flag. -/
theorem maskRegion_pair_elim {C : SimpC} {mask ext : Fin C.nVertices → Bool}
  {x : Fin C.ambDim → ℚ} {a b : Fin C.nVertices} {V₀ V₁ : ℚ}
  (hfil : (List.finRange C.nVertices).filter (fun v => mask v) = [a, b])
  (hVa : ∀ j, C.coord a j = V₀) (hVb : ∀ j, C.coord b j = V₁)
  (hne : V₀ ≠ V₁) (j0 : Fin C.ambDim)
  (h : MaskRegion C mask ext x) :
  (ext a = true → (V₁ - x j0) / (V₁ - V₀) ≤ 0) ∧
  (ext a = false → 0 ≤ (V₁ - x j0) / (V₁ - V₀)) ∧
  (ext b = true → (x j0 - V₀) / (V₁ - V₀) ≤ 0) ∧
  (ext b = false → 0 ≤ (x j0 - V₀) / (V₁ - V₀))
```

BernsteinLogic.Soundness . **sound\_convElim**

statement bound

```
theorem sound_convElim {H : Hypernet} {seq premise : Sequent}
  {term : Term} {simplex : Simplex} {S : ExtensionSet}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hpremiseForm : premise.post = .inRegion term simplex S)
  (hhalf : barycentricHalfspaceForm term simplex S seq.post) :
  seq.semanticallyValid H
```

**The canonical form, meaning bound by display.** The conclusion formula is the value of the build displayed below — affine data (baryCoeffs1D), positive primitive-integer rescale (primAffine), atom layout (f4Lhs), per-vertex sign-folding (f4VertexAtom), assembly (barycentricHalfspaceBuild) — and the payload's side condition is the one-line equation barycentricHalfspaceForm. A skeptical reader should confirm by eye that every gate failure yields none (refusal), that the rescale factor is strictly positive on every reachable branch, and that no valuation is consulted anywhere in the chain.

BernsteinLogic.Core.Catalogue . **baryCoeffs1D**

definition bound

```
-- The (vertex, offset, slope) affine data of the two barycentric
coordinate functions of a 1-dimensional simplex: `λ(x) = offset + slope·x`.
Same gates as `baryLambdas1D` (which builds the cell rules' pinned Term
shape from the same coordinates). -/
def baryCoeffs1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool) :
  Option (List (Fin cx.nVertices × Rat × Rat)) :=
  if h : cx.ambDim = 1 then
    let j0 : Fin cx.ambDim := (0, by omega)
    match (List.finRange cx.nVertices).filter (fun v => dContains v) with
    | [a, b] =>
      let V₀ := cx.coord a j0
      let V₁ := cx.coord b j0
      if V₀ = V₁ then none
      else
        let den := V₁ - V₀
        some [(a, V₁ / den, -1 / den), (b, -V₀ / den, 1 / den)]
    | _ => none
  else none
```

BernsteinLogic.Core.Catalogue . **primAffine**

definition bound

```
-- Primitive-integer rescale of the affine pair (slope `s`, offset `c`):
the positive multiple `(k·s, k·c)` whose entries are the coprime-integer
vector sympy's `canon_sympy` produces for `s·x + c` (producer finding F4).
`c = 0` degenerates to sign normalization (single-entry primitive vector
is ±1); `(0, 0)` is fixed. Computed rationally — `k·s` value-equals the
integer `n₁/g` because `s.den | L` and `g | n₁`. -/
```

```

def primAffine (s c : Rat) : Rat × Rat :=
  if c = 0 then
    (if s < 0 then -1 else if s = 0 then 0 else 1, 0)
  else
    let L : Nat := Nat.lcm s.den c.den
    let g : Nat := Int.gcd (s.num * (L / s.den : Nat)) (c.num * (L / c.den : Nat))
    let k : Rat := (L : Rat) / (g : Rat)
    (k * s, k * c)

```

BernsteinLogic.Core.Catalogue . **f4Lhs**

definition bound

```

/-- The F4 atom body for the rescaled pair: degree-sorted monomial shape
`s'·t + c`, the constant dropped when zero (sympy `_term_ast`). -/
def f4Lhs (s' c' : Rat) (t : Term) : Term :=
  if c' = 0 then .mul (.const s') t
  else .add (.mul (.const s') t) (.const c')

```

BernsteinLogic.Core.Catalogue . **f4VertexAtom**

definition bound

```

/-- One vertex's F4 halfspace atom: sign-folded (complement vertex's λ
negated into the term), rescaled, `≤ 0`-shaped. -/
def f4VertexAtom (inS : Bool) (off slp : Rat) (t : Term) : Formula :=
  let sc := if inS then primAffine slp off else primAffine (-slp) (-off)
  .atom .le (f4Lhs sc.1 sc.2 t) (.const 0)

```

BernsteinLogic.Core.Catalogue . **barycentricHalfspaceBuild**

definition bound

```

def barycentricHalfspaceBuild (t : Term) (simplex : Simplex)
  (S : ExtensionSet) : Option Formula := do
  guard (S.simplex = simplex)
  let cs ← baryCoeffs1D simplex.complex simplex.contains
  match cs.map (fun (v, off, slp) =>
    f4VertexAtom (S.containsNat v.val) off slp t) with
  | [x, y] => some (.and x y)
  | _ => none

```

BernsteinLogic.Core.Catalogue . **barycentricHalfspaceForm**

definition bound

```

def barycentricHalfspaceForm (t : Term) (simplex : Simplex)
  (S : ExtensionSet) (F : Formula) : Prop :=
  barycentricHalfspaceBuild t simplex S = some F

```

BernsteinLogic.Core.Catalogue . **PConvElimData**

definition bound

```

inductive PConvElimData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (term : Term) (simplex : Simplex) (S : ExtensionSet)
  (preEq : seq.pre = premise.pre)
  (premiseForm : premise.post = .inRegion term simplex S)
  (halfspace : barycentricHalfspaceForm term simplex S seq.post)
  : PConvElimData H seq

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in **AX**; sub-derivation premise as at **WEAKEN**.

- $x \mapsto \text{term} : \text{Term}; \Delta \mapsto \text{simplex} : \text{Simplex}; S \mapsto S : \text{ExtensionSet}$ ; the premise judgment  $\Phi \vdash x \in \mathcal{R}(\Delta, S) \mapsto \text{ih}$  with the shape tie  $\text{hpremiseForm} : \text{premise.post} = \text{.inRegion term simplex S}; \Phi$  shared via  $\text{hpre}$ .

- the *entire* displayed conclusion — the flat conjunction, its vertex order, the sign pattern  $\epsilon_i$ , the affine spellings  $\lambda_i$  —  $\mapsto$  one hypothesis, `hhalf : barycentricHalfspaceForm term simplex S seq.post`: `seq.post` is literally the value of the verified canonical build. The display's  $\lambda_i$  and  $\epsilon_i$  are not statement binders but *outputs* of `barycentricHalfspaceBuild`, quoted in full below.
- conclusion judgment  $\mapsto$  the goal `seq.semanticallyValid H`; certified alongside, `barycentricHalfspaceBuild_spec` unpacks a successful build run into the geometric data the soundness proof consumes.

Width note, narrower and loud about it: a successful build run forces ambient dimension 1 with support exactly two vertices at distinct coordinates (the first conjuncts of `barycentricHalfspaceBuild_spec`), so the mechanized statement covers exactly the one-dimensional instance of the displayed general-dimension rule. Outside that fragment the build returns `none`, the hypothesis is unsatisfiable, and the rule cannot fire — refusal, never unverified acceptance; the displayed rule at higher dimension is design intent, not yet certified statement.

Spelling alignment, formerly a live discrepancy: the displayed conclusion and the certified build now agree — both state the flat sign-folded  $\text{all-}\leq$  conjunction, each atom  $\text{canon}(\epsilon_i \lambda_i(x)) \leq 0$  with the extension/core orientation carried by the sign  $\epsilon_i$  (`f4VertexAtom` negates the core vertex's affine form into the term and compares  $\leq 0$  — the commentary describes this discipline). Earlier printings displayed a mixed  $\leq/\geq$  per-vertex comparator pattern; the roster cell has since been reconciled to the realized form, and this certification binds the reconciled texts.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

`build` (lake build (elaboration)) current

`replay` (leanchecker (official kernel replay)) stale only

`hardened` (safe\_verify (hardened screen)) stale only

`decorrelated` (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory `bern_catalogue`, HOL4 development script `bern_catalogueScript.sml` script bound

### Definition `baryCoeffs1D_def`

```

Definition baryCoeffs1D_def:
  baryCoeffs1D cx dmask =
    if cx.ambDim = 1 then
      (case FILTER (λv. getD dmask v F) (rangeL cx.nVertices) of
       | [a; b] =>
         (let v0 = getD (getD cx.coord a []) 0 (0:rat) in

```



```

        let v1 = getD (getD cx.coord b []) 0 (0:rat) in
        let den = v1 - v0 in
        if den = 0 then NONE
        else SOME [(a, v1 / den, -(&l) / den);
                   (b, -v0 / den, &l / den)]
    | _ => NONE
  else NONE
End

```

**Definition primAffine\_def**

```

Definition primAffine_def:
  primAffine (s:rat) (c:rat) =
    if c = 0 then
      ((if s < 0 then -1 else if s = 0 then 0 else 1) : rat, 0:rat)
    else
      let l = lcm (RATD s) (RATD c) in
      let g = gcd (intAbsNum (RATN s * (&l DIV RATD s)))
                  (intAbsNum (RATN c * (&l DIV RATD c))) in
      (* g = 0 is unreachable (c ≠ 0 ⇒ RATN c ≠ 0 ⇒ the second gcd
         argument is nonzero); the guard is the translator's division
         precondition, spelled on the divisor (Lean's Rat division by
         zero yields 0, so the two sides agree even on the dead branch). *)
      let k = if &g = (0:rat) then 0 else &l / &g in
      (k * s, k * c)
End

```

**Definition f4Lhs\_def**

```

Definition f4Lhs_def:
  f4Lhs (s':rat) (c':rat) t =
    if c' = 0 then TmMul (TmConst s') t
    else TmAdd (TmMul (TmConst s') t) (TmConst c')
End

```

**Definition f4VertexAtom\_def**

```

Definition f4VertexAtom_def:
  f4VertexAtom inS (off:rat) (slp:rat) t =
    let sc = if inS then primAffine slp off else primAffine (-slp) (-off) in
    FAtom BowLe (f4Lhs (FST sc) (SND sc) t) (TmConst 0)
End

```

**Definition barycentricHalfspaceBuild\_def**

```

Definition barycentricHalfspaceBuild_def:
  barycentricHalfspaceBuild t sx es =
    if es.simplex = sx then
      (case baryCoeffs1D sx.complex sx.vmask of
       | NONE => NONE
       | SOME cs =>
         (case MAP (λ(v,off,slp). f4VertexAtom (extContainsNat es v) off slp t)
                  cs of
          | [x; y] => SOME (FAnd x y)
          | _ => NONE))
    else NONE
End

```

**Definition barycentricHalfspaceFormB\_def**

```

Definition barycentricHalfspaceFormB_def:
  barycentricHalfspaceFormB t sx es fm ⇔
    barycentricHalfspaceBuild t sx es = SOME fm
End

```

**HOL verifier receipts**

**documentary** hol-strict CLEAN census over the ported checker theory: bern\_catalogue, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here. The earlier loud caveat (the canonical-form revision of 2026-07-18 postdated the prior census run) is retired: this run attests exactly the quoted text. There is no signed HOL receipt store yet, and these citations are documentary. Semantic stratum, also ported: the soundness twin sound\_convElim (theory bern\_tr\_fragclose: hol-strict clean, 26 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (checkPf\_sound\_frag, theory bern\_tr\_splice, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1). Setting (valuations, partial evaluation, elementwise atom semantics) as fixed in the EQELIM section; this section additionally fixes the region-atom semantics and the canonical halfspace form, which the CONVINTRO section then cites.

**Region atoms, in the checkable fragment.** A region atom  $x \in \mathcal{R}(\Delta, S)$  holds at  $\sigma$  when the extension set  $S$  is tied to the simplex  $\Delta$ , the region term evaluates to a rank-1 tensor of extent equal to the ambient dimension, and the point it holds lies in the *masked region*: it is an affine combination of  $\Delta$ 's vertices with weights summing to 1, nonnegative on core vertices and *nonpositive* on the vertices named by  $S$ . The rules below operate in the fragment their checker accepts — ambient dimension 1, support exactly two vertices  $a, b$  at distinct coordinates  $V_0 \neq V_1$  — where the geometry is completely explicit: the two barycentric weights are *determined*, affine in the point,

$$\lambda_a(x) = \frac{V_1 - x}{V_1 - V_0}, \quad \lambda_b(x) = \frac{x - V_0}{V_1 - V_0}, \quad \lambda_a + \lambda_b \equiv 1,$$

and the region is: the segment  $[V_0, V_1]$  for  $S = \emptyset$ ; the closed ray from  $V_1$  away from  $V_0$  for  $S = \{a\}$  (weight of the extended vertex  $\leq 0$ ); the mirror ray for  $S = \{b\}$ . ( $S$  is a proper subset, so both-extended does not arise.) Determinacy of the weights is why elimination below is complete — no witness is chosen, none rides the certificate.

**The canonical halfspace form.** The conclusion is not “some conjunction equivalent to the sign facts” but one *pinned* formula, recomputed by the checker from  $(x, \Delta, S)$  alone: one atom per support vertex, in support order, each in  $(\cdot) \leq 0$  shape: for  $v \in S$  the atom  $\text{canon}(\lambda_v(x)) \leq 0$ , for  $v \notin S$  the atom  $\text{canon}(-\lambda_v(x)) \leq 0$  (the wanted  $\lambda_v \geq 0$  *sign-folded* into the term — the all- $\leq$  discipline), where  $\text{canon}$  rescales the affine pair (slope, offset) by a *strictly positive* rational to its primitive-integer representative and lays the atom body out in degree-sorted shape. If the build's gates fail (dimension  $\neq 1$ , support  $\neq 2$  vertices,  $V_0 = V_1$ , or the atom's simplex not the cited  $\Delta$ ), the build refuses, the payload is uninhabitable, and the rule cannot fire: refusal is trivially sound, and in the degenerate-simplex case it is also where a division by zero would otherwise live — made unrepresentable rather than checked-then-divided.

**Claim** (sound\_convElim). If  $\Phi \vdash x \in \mathcal{R}(\Delta, S)$  is semantically valid and the conclusion formula is the canonical halfspace form for  $(x, \Delta, S)$ , then  $\Phi \vdash$  (that conjunction) is semantically valid.

**Lemma R12.1** (structure of a successful build). *If the canonical build succeeds with output  $F$ , then the ambient dimension is 1,  $S$  is tied to  $\Delta$ , the support filter returns exactly  $[a, b]$  with constant vertex coordinates  $V_0 \neq V_1$ , and  $F$  is the conjunction of the two vertex atoms described above.*

*Proof.* Unfold the build: each claim is one of its gates, and the output shape is its final assembly. (barycentricHalfspaceBuild\_spec.)  $\square$

**Lemma R12.2** (canonical atom  $\leftrightarrow$  raw sign fact). *Let  $q$  be a defined, singleton-shaped evaluation of  $x$  at  $\sigma$ , and let  $(\text{off}, \text{slp})$  be an affine pair. The vertex atom for membership flag  $m$  holds at  $\sigma$  iff*

$$m = \text{true} : \quad \text{off} + \text{slp} \cdot q \leq 0, \quad m = \text{false} : \quad 0 \leq \text{off} + \text{slp} \cdot q.$$

*Proof.* Two steps. *Sign-folding*: for  $m = \text{false}$  the build negates the pair before canonicalizing, so the emitted atom asserts  $\text{canon}(-\lambda) \leq 0$ , i.e. the displayed  $\geq$ . *The rescale is sign-transparent*:  $\text{canon}$  multiplies the pair by  $k > 0$  — in the generic case  $k = L/g$  with  $L$  a positive lcm of denominators and  $g$  a positive gcd of the scaled numerators; in the zero-offset case the output slope is the sign  $\text{sgn}(\text{slp})$ , i.e. the pair scaled by  $1/|\text{slp}|$  — well-defined on every pair in play: a successful barycentric build has distinct vertices, so each emitted affine pair carries slope  $\pm 1/(V_1 - V_0) \neq 0$  (sign-folding only negates it), and in the zero-offset case that slope is the whole pair; for the degenerate all-zero pair, which no successful build emits, both sides of the displayed equivalence hold trivially and the statement is unaffected — and a strictly positive rescale of an affine function has the same sign locus. The atom's left side evaluates (constants and one already-defined  $q$ ), at singleton shape, so the elementwise reading is the scalar comparison of the displayed value against 0. (f4VertexAtom\_holds\_iff; the rescale is primAffine, the layout f4Lhs.)  $\square$

**Lemma R12.3** (region elimination, two-vertex case). *Suppose  $\sigma \models x \in \mathcal{R}(\Delta, S)$  in the fragment of Lemma R12.1, with  $q$  the evaluated coordinate. Then, conditionally on membership: if  $a \in S$  then  $\lambda_a(q) \leq 0$ , if  $a \notin S$  then  $\lambda_a(q) \geq 0$ ; likewise for  $b$ .*

*Proof.* The masked-region witness consists of weights on the support summing to 1 that reproduce  $q$  and satisfy the sign discipline. On a two-vertex support the reproduction and sum equations force the weights to be exactly  $\lambda_a(q), \lambda_b(q)$  (two linear equations, invertible since  $V_0 \neq V_1$ ), so the witness's sign discipline is the displayed conditional facts. (maskRegion\_pair\_elim.)  $\square$

**Theorem R12.4** (soundness of `CONVELIM`). *Under the payload’s checked hypotheses, if  $\Phi \vdash x \in \mathcal{R}(\Delta, S)$  is semantically valid, so is  $\Phi \vdash F$  for the built conclusion  $F$ .*

*Proof.* Fix an admissible  $\sigma \models \Phi$ . Premise validity gives the region atom at  $\sigma$ : in particular  $x$  *evaluates*, to a rank-1 extent-1 tensor (the atom’s own shape field — no typing pack is needed here; contrast [Remark R12.6](#)), whose scalar is  $q$ . Lemma [R12.3](#) delivers the four conditional sign facts; Lemma [R12.2](#) converts, per vertex, the applicable one into the built vertex atom; Lemma [R12.1](#) says the conclusion is exactly the conjunction of those two atoms.  $\square$

*Remark R12.5* (why a pinned canonical spelling). The combination rules downstream (`FARKAS`, `SYZGY`) match premise atoms *by exact canonical polynomial*; a conclusion emitted in any equivalent-but-different spelling would simply never match, stranding the geometric fact. Pinning the spelling in the checker — which recomputes the build from  $(x, \Delta, S)$  and demands syntactic equality with the cited conclusion, trusting no producer text — makes the emitted atoms the *same* tokens every producer and every downstream matcher agrees on. The all- $\leq$  polarity keeps each atom in the  $(\cdot) \leq 0$  normal form those rules consume, and the positive primitive-integer rescale makes the representative unique. Note the checked object is the *formula*, not a semantic condition: this rule’s side condition is one syntactic equation, decidable by construction.

*Remark R12.6* (what is *not* hypothesized). No shape or finiteness fields appear in this payload: elimination reads definedness and shape out of the region atom itself. The mirror rule cannot — its premise is the halfspace conjunction, whose atoms tolerate broadcast and are satisfied by a *scalar*  $x$  — and that asymmetry is exactly where `CONVINTRO`’s one extra typing field comes from (next section).

| proof step                                                                   | formal anchor                                                                                                                                                                                   |
|------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lemma <a href="#">R12.1</a><br>build, affine data, rescale, atom layout      | <code>barycentricHalfspaceBuild_spec</code><br><code>barycentricHalfspaceBuild</code> , <code>baryCoeffs1D</code> , <code>primAffine</code> , <code>f4Lhs</code> ,<br><code>f4VertexAtom</code> |
| Lemma <a href="#">R12.2</a>                                                  | <code>f4VertexAtom_holds_iff</code>                                                                                                                                                             |
| Lemma <a href="#">R12.3</a>                                                  | <code>maskRegion_pair_elim</code>                                                                                                                                                               |
| Theorem <a href="#">R12.4</a><br>form predicate (the checked side condition) | <code>sound_convElim</code> ; wired as <code>core RuleData.convElim</code> into <code>ProofObject.sound</code><br><code>barycentricHalfspaceForm</code> (payload field <code>halfspace</code> ) |

## Rule 13. CONVINTRO (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *ConvIntro*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash \bigwedge_{i=0}^n \text{canon}(\epsilon_i \lambda_i(x)) \leq 0 \quad \epsilon_i = +1 \text{ if } i \in S, -1 \text{ if } i \in \bar{S} \quad \text{bshape}_H(x) = [n]}{\Phi \vdash x \in \mathcal{R}(\Delta, S)} \text{ CONVINTRO}$$

CONVINTRO closes a region atom: from the canonical barycentric halfspace conjunction for  $(x, \Delta, S)$  — the same pinned formula CONVELIM emits, recomputed by the same build — it concludes  $\Phi \vdash x \in \mathcal{R}(\Delta, S)$ . A producer uses it to re-enter the geometric world after arithmetic: a subdivision branch or a combination argument establishes the two vertex atoms, and this rule converts them back into the region fact that BERNCONC or a further region rule consumes.

What the checker recomputes: the build (as in CONVELIM — one syntactic equation, producer text untrusted, refusal outside the 1-D two-vertex fragment), plus this rule's one extra typing field:  $\text{bshape}_H(x) = [d]$ , the region term is statically rank-1 of the ambient extent. The asymmetry with CONVELIM is principled: elimination reads the term's shape out of the region atom it consumes, while introduction is handed only halfspace atoms — which hold *elementwise under broadcast* and are perfectly satisfied by a scalar  $x$  — so the shape must arrive as a checked hypothesis. Drop it and a closed scalar inside the segment yields a satisfiable (indeed universally true) premise whose region conclusion is unsatisfiable — a rank-0 tensor never satisfies a region atom; the proof section builds the refutation concretely. It is the same broadcast liberality that breaks un-gated EQELIM, repaired by the same typing-field pattern.

**notes** On the fragment, CONVINTRO and CONVELIM are exact inverses — membership, the four weight-sign facts, and the canonical conjunction are three presentations of one determinate-weight geometry, so the pair is a lossless interface between region atoms and the arithmetic fragment.

### Lean statement

BernsteinLogic.Soundness . Term.evalQ\_bshape

statement bound

```
-- bshape soundness: a defined evaluation takes the broadcast shape. -/
theorem Term.evalQ_bshape {H : Hypernet} (σ : H.Valuation) :
  ∀ (t : Term) (s : Shape) (q : QTensor),
    t.bshape H = some s → Term.evalQ σ t = some q → q.shape = s
```

BernsteinLogic.Soundness . f4VertexAtom\_holds\_inv

statement bound

```
-- A held F4 vertex atom forces the core term to evaluate. -/
theorem f4VertexAtom_holds_inv {H : Hypernet} {σ : H.Valuation} {t : Term}
  {inS : Bool} {off slp : Q}
  (h : Formula.holds σ (f4VertexAtom inS off slp t)) :
  ∃ q, Term.evalQ σ t = some q
```

BernsteinLogic.Soundness . maskRegion\_pair\_intro

statement bound

```
-- Construction: rational weights of the right signs assemble a
`MaskRegion` witness over a two-vertex 1-D support. -/
theorem maskRegion_pair_intro {C : SimpC} {mask ext : Fin C.nVertices → Bool}
  {x : Fin C.ambDim → Q} {a b : Fin C.nVertices} {V₀ V₁ : Q}
  (hfil : (List.finRange C.nVertices).filter (fun v => mask v) = [a, b])
  (hVa : ∀ j, C.coord a j = V₀) (hVb : ∀ j, C.coord b j = V₁)
  (hne : V₀ ≠ V₁) (hdim : C.ambDim = 1) (j0 : Fin C.ambDim)
  (hw0t : ext a = true → (V₁ - x j0) / (V₁ - V₀) ≤ 0)
  (hw0f : ext a = false → 0 ≤ (V₁ - x j0) / (V₁ - V₀))
  (hw1t : ext b = true → (x j0 - V₀) / (V₁ - V₀) ≤ 0)
  (hw1f : ext b = false → 0 ≤ (x j0 - V₀) / (V₁ - V₀)) :
  MaskRegion C mask ext x
```

BernsteinLogic.Soundness . sound\_convIntro

statement bound

```

theorem sound_convIntro {H : Hypernet} {seq premise : Sequent}
  {term : Term} {simplex : Simplex} {S : ExtensionSet}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (htermShape : term.bshape H = some [simplex.complex.ambDim])
  (hhalf : barycentricHalfspaceForm term simplex S premise.post)
  (hpost : seq.post = .inRegion term simplex S) :
  seq.semanticallyValid H

```

**Shared machinery.** The build chain is displayed in full in the `CONVELIM` section; here the book re-binds the assembly and the form equation, plus this rule's payload with its one extra typing field (`termShape`).

BernsteinLogic.Core.Catalogue . **barycentricHalfspaceBuild**

definition bound

```

def barycentricHalfspaceBuild (t : Term) (simplex : Simplex)
  (S : ExtensionSet) : Option Formula := do
  guard (S.simplex = simplex)
  let cs ← baryCoeffs1D simplex.complex simplex.contains
  match cs.map (fun (v, off, slp) =>
    f4VertexAtom (S.containsNat v.val) off slp t) with
  | [x, y] => some (.and x y)
  | _ => none

```

BernsteinLogic.Core.Catalogue . **barycentricHalfspaceForm**

definition bound

```

def barycentricHalfspaceForm (t : Term) (simplex : Simplex)
  (S : ExtensionSet) (F : Formula) : Prop :=
  barycentricHalfspaceBuild t simplex S = some F

```

BernsteinLogic.Core.Catalogue . **PConvIntroData**

definition bound

```

inductive PConvIntroData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (term : Term) (simplex : Simplex) (S : ExtensionSet)
  (preEq : seq.pre = premise.pre)
  (termShape : term.bshape H = some [simplex.complex.ambDim])
  (halfspace : barycentricHalfspaceForm term simplex S premise.post)
  (postEq : seq.post = .inRegion term simplex S)
  : PConvIntroData H seq

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in `AX`; sub-derivation premise as at `WEAKEN`; binders as at `CONVELIM`, mirrored.

- the displayed premise conjunction  $\mapsto$  `hhalf : barycentricHalfspaceForm term simplex S premise.post` — the same verified build, its value now equated with the *premise's* post; the premise judgment  $\mapsto$  `ih`, context shared via `hpre`.
- the display's typing premise  $\text{bshape}_H(x) = [n] \mapsto \text{htermShape} : \text{term.bshape } H = \text{some } [\text{simplex.complex.ambDim}]$  — statically rank-1 at the ambient extent, the one hypothesis `CONVELIM` does not have; the commentary derives why introduction needs it and elimination does not.
- conclusion  $\Phi \vdash x \in \mathcal{R}(\Delta, S) \mapsto$  the goal under `hpost : seq.post = .inRegion term simplex S`.

Width note: the same one-dimensional fragment confinement as `CONVELIM`, forced by the same spec conjuncts — the mechanized statement is the displayed rule at ambient dimension 1; outside it the build refuses and nothing is accepted unverified. The `CONVELIM` dictionary's spelling alignment carries over unchanged: the displayed premise conjunction and the certified premise formula both state the build's sign-folded  $\text{all-}\leq$  shape, the extension/core orientation carried by the sign  $\epsilon_i$  — same halfspaces, same folded spelling, display and mechanization reconciled.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_catalogue, HOL4 development script bern\_catalogueScript.sml script bound

#### Definition barycentricHalfspaceBuild\_def

```

Definition barycentricHalfspaceBuild_def:
  barycentricHalfspaceBuild t sx es =
    if es.simplex = sx then
      (case baryCoeffs1D sx.complex sx.vmask of
       | NONE => NONE
       | SOME cs =>
         (case MAP (λ(v,off,slp). f4VertexAtom (extContainsNat es v) off slp t)
          cs of
          | [x; y] => SOME (FAnd x y)
          | _ => NONE))
    else NONE
End

```

#### Definition barycentricHalfspaceFormB\_def

```

Definition barycentricHalfspaceFormB_def:
  barycentricHalfspaceFormB t sx es fm ↔
  barycentricHalfspaceBuild t sx es = SOME fm
End

```

## HOL verifier receipts

documentary hol-strict CLEAN census over the ported checker theory: bern\_catalogue, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here — the re-attestation the CONVELIM cell's earlier caveat awaited (the canonical-form revision of 2026-07-18 postdated the prior census) is this run (documentary citations — no signed HOL

store yet). Scope note: the shape side condition (`termShape`) rides the un-ported hypernet typing island, deferred to the port's proof-object stage. Semantic stratum, also ported: the soundness twin `sound_convIntro` (theory `bern_tr_fragclose`: `hol-strict` clean, 26 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splce`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named `future work`.

## Proof

Genre: core rule (tier 1). Setting, region-atom semantics, the checkable fragment, and the canonical halfspace form as fixed in the `CONVELIM` section.

**Claim** (`sound_convIntro`). The payload supplies the region data  $(x, \Delta, S)$  and two checked conditions: the premise concludes the canonical halfspace form for  $(x, \Delta, S)$ , and (the shape condition)  $\text{bshape}_H(x) = [d]$  where  $d$  is the ambient dimension — the region term is statically a rank-1 tensor of the ambient extent. If the premise is semantically valid, then  $\Phi \vdash x \in \mathcal{R}(\Delta, S)$  is semantically valid.

**Lemma R13.1** (evaluation recovered from the premise). *Suppose  $\sigma$  satisfies the built conjunction. Then  $x$  evaluates at  $\sigma$ .*

*Proof.* The first vertex atom holds, so its left side — the canonicalized affine image of  $x$ , built from constants and  $x$  by product and sum — evaluates; evaluation of a compound is defined only if evaluation of every part is — the evaluator is strict, each node's value built from its children's values, so a defined parent forces defined children — and inverting it at the product node (and sum node, when the offset is present) yields a value for  $x$  itself. (`f4VertexAtom_holds_inv`)  $\square$

**Lemma R13.2** (shape pinned by the payload field). *With the shape condition, the evaluation of Lemma R13.1 is a rank-1 tensor of extent 1 (the fragment has  $d = 1$ ), hence singleton-shaped, with scalar  $q$ .*

*Proof.* Broadcast-shape soundness: a defined evaluation of a `bshape`-typed term lands at the declared shape (`Term.evalQ_bshape`), here  $[1]$  by the build's dimension gate (`CONVELIM`, Lemma R12.1).  $\square$

**Theorem R13.3** (soundness of `CONVINTRO`). *Under the payload's checked hypotheses, if  $\Phi \vdash F$  is semantically valid for the built conjunction  $F$ , so is  $\Phi \vdash x \in \mathcal{R}(\Delta, S)$ .*

*Proof.* Fix an admissible  $\sigma \models \Phi$ ; premise validity gives both vertex atoms at  $\sigma$ . Lemmas R13.1 and R13.2 produce the singleton evaluation  $q$  of  $x$ . Reading each held atom through the canonical-atom characterization (`CONVELIM`, Lemma R12.2) yields the four conditional sign facts on  $\lambda_a(q)$ ,  $\lambda_b(q)$ . The two-vertex introduction assembles the masked-region witness from exactly these: the weights  $\lambda_a(q)$ ,  $\lambda_b(q)$  sum to 1 and reproduce  $q$  by construction (affine identities, valid since  $V_0 \neq V_1$ ), and the sign discipline — nonnegative on core vertices, nonpositive on  $S$ -vertices — is the four facts (`maskRegion_pair_intro`). Together with the build's tie of  $S$  to  $\Delta$  and the shape of Lemma R13.2, this is the region atom's holding, term slot included.  $\square$

**Remark R13.4** (why the shape condition is not decoration). Drop it and the rule is *unsound*. Over  $\Delta = \text{conv}(1, 2)$ ,  $S = \emptyset$ , instantiate the region term at the *closed scalar*  $t := 3/2$ . The built premise is then a conjunction of two closed atoms asserting  $\lambda_a(3/2) \geq 0$  and  $\lambda_b(3/2) \geq 0$  — both true ( $\lambda_a = \lambda_b = 1/2$ ), so the conjunction holds at *every* valuation, and taking  $\Phi$  to be that very conjunction,  $\text{Ax}$  proves the premise. The un-gated introduction would conclude  $\Phi \vdash (\frac{3}{2} : \text{Term}) \in \mathcal{R}(\Delta, \emptyset)$  — unsatisfiable: a constant evaluates to a rank-0 tensor, and the region atom demands rank 1. Satisfiable premise, unsatisfiable conclusion. The mechanism is the same broadcast liberality that broke the un-gated `EQELIM`: the halfspace atoms constrain the *values* of the affine images and are perfectly happy with a scalar  $x$ ; only the region atom cares about the tensor's shape. The shape field restores, at the introduction rule, the well-formedness that region atoms carry in the source calculus's typing discipline — one repair family (the `EQELIM/NORM` typing pattern), one failure family.

**Remark R13.5** (elimination and introduction are exact inverses on the fragment). On singleton-shaped, well-shaped region terms, Lemmas R12.3 (elimination) and the introduction step above are mutually inverse readings of the same determinate-weight geometry: membership  $\leftrightarrow$  the four sign facts  $\leftrightarrow$  the canonical conjunction. The round trip loses nothing — which is why the pair can serve as the sole interface between region atoms and the arithmetic fragment, with no completeness caveat inside the checkable fragment.

| proof step             | formal anchor                                                                                                    |
|------------------------|------------------------------------------------------------------------------------------------------------------|
| Lemma R13.1            | <code>f4VertexAtom_holds_inv</code>                                                                              |
| Lemma R13.2            | <code>Term.evalQ_bshape</code> ; payload field <code>termShape</code>                                            |
| canonical-atom reading | <code>f4VertexAtom_holds_iff</code> ( <code>CONVELIM</code> section)                                             |
| witness assembly       | <code>maskRegion_pair_intro</code>                                                                               |
| Theorem R13.3          | <code>sound_convIntro</code> ; wired as core <code>RuleData.convIntro</code> into <code>ProofObject.sound</code> |
| Remark R13.4           | the <code>counterexample</code> inventory, <code>Soundness.lean</code> header                                    |



## Rule 14. ARITHATOM (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *ArithAtom*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash (t_1 - t_2) \bowtie 0}{\Phi \vdash t_1 \bowtie t_2} \text{ARITHATOM}$$

$$\frac{\Phi \vdash t_1 \bowtie t_2}{\Phi \vdash (t_1 - t_2) \bowtie 0} \text{ARITHATOM}^{-1}$$

with  $\bowtie \in \{<, \leq, =, \geq, >\}$ .

ARITHATOM toggles a comparison atom between its two spellings: the *direct form*  $t_1 \bowtie t_2$  and the *zero-normal form*  $(t_1 - t_2) \bowtie 0$ , in both directions (one payload constructor each), for  $\bowtie \in \{<, \leq, =, \geq, >\}$ . It is the adapter at the calculus's normal-form boundary: the combination rules (FARKAS, SYZGY) and the canonical conclusions of the geometry rules live in  $(\cdot) \leq 0$ -normal shapes, while human-facing specifications, box premises of the relaxation rules, and final theorem statements live in direct forms. A producer folds a zero-form conclusion into the literal bound a downstream rule wants, or unfolds a specification atom so its polynomial content can be matched.

What the checker recomputes: two syntactic form equations — the premise (respectively conclusion) must be *literally* the atom built from the same payload terms with the syntactic difference  $t_1 - t_2$  against the literal constant 0 — plus the comparator gate  $\bowtie \neq \neq$  and the shared-context equation. Deliberately severe: no term evaluating to zero, no algebraic rearrangement qualifies (a producer wanting those routes through NORM first); the severity is what keeps the side conditions decidable by construction and the trusted surface at the two form equations, with the comparator gate and the context equation beside them.

There is no droppable typing hypothesis, and that is the rule's quiet achievement: both forms mention the same wires, and whenever either form holds, both evaluations exist and transport — in the fold direction the difference's definedness arrives with the premise, and in the unfold direction it is manufactured from the fact that a *holding* elementwise comparison already certifies broadcast compatibility, on exactly which subtraction is total. The comparator gate is the one non-structural field: the rational semantics would accept  $\neq$  (the entrywise bridge is comparator-uniform), and the gate instead pins the source calculus's comparator set and keeps the development's auxiliary three-valued screen uniform across the atom-shape rules (see the proof section's closing remark — the screen itself would survive  $\neq$  here, unlike at the ground-fact rule).

**notes** The transported object is the per-entry rational comparison  $(p - q) \bowtie 0 \iff p \bowtie q$  — translation invariance of the order of  $\mathbb{Q}$  — lifted entrywise through the broadcast discipline by an invertible totality argument. Exact arithmetic end to end: no rounding, no tolerance, in either direction.

### Lean statement

BernsteinLogic.Soundness.QTensor.map2\_defined\_of\_rel2

statement bound

```
-- A holding rel2 forces broadcast-compatible shapes, so map2 succeeds. -/
theorem QTensor.map2_defined_of_rel2 {r : Q → Q → Prop} {a b : QTensor}
  (h : QTensor.rel2 r a b) (f : Q → Q → Q) :
  ∃ c, QTensor.map2 f a b = some c
```

BernsteinLogic.Soundness.QTensor.rel2\_sub\_zero\_iff

statement bound

```
-- The ArithAtom bridge, tensor-general: comparing a subtraction against
rank-0 zero is comparing the operands directly, in every broadcast case. -/
theorem QTensor.rel2_sub_zero_iff {b : BowTie} {al ar d : QTensor}
  (hd : QTensor.map2 (· - ·) al ar = some d) :
  QTensor.rel2 (BowTie.holdsQ b) d ([], [0], rfl)
  ↔ QTensor.rel2 (BowTie.holdsQ b) al ar
```

BernsteinLogic.Soundness.Term.evalQ\_sub\_inv

statement bound

```
theorem Term.evalQ_sub_inv {H : Hypernet} (σ : H.Valuation) {l r : Term}
  {q : QTensor} (h : Term.evalQ σ (.sub l r) = some q) :
  ∃ a b, Term.evalQ σ l = some a ∧ Term.evalQ σ r = some b ∧
  QTensor.map2 (· - ·) a b = some q
```

BernsteinLogic.Soundness . **sound\_arithAtom\_fold**

statement bound

```

theorem sound_arithAtom_fold {H : Hypernet} {seq premise : Sequent}
  {termL termR : Term} {b : BowTie}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hprem : atomicCompareZero b termL termR premise.post)
  (hconcl : atomicCompareForm b termL termR seq.post) :
  seq.semanticallyValid H

```

BernsteinLogic.Soundness . **sound\_arithAtom\_unfold**

statement bound

```

theorem sound_arithAtom_unfold {H : Hypernet} {seq premise : Sequent}
  {termL termR : Term} {b : BowTie}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hprem : atomicCompareForm b termL termR premise.post)
  (hconcl : atomicCompareZero b termL termR seq.post) :
  seq.semanticallyValid H

```

**The two form equations, in full.** Everything this rule checks is displayed below: two one-line syntactic equations and the payload's two constructors (one per direction) with the comparator gate notNE. No valuation is consulted; semantic truth enters only through the soundness lemmas.

BernsteinLogic.Core.Catalogue . **atomicCompareForm**

definition bound

```

def atomicCompareForm (b : BowTie) (l r : Term) (F : Formula) : Prop :=
  F = .atom b l r

```

BernsteinLogic.Core.Catalogue . **atomicCompareZero**

definition bound

```

def atomicCompareZero (b : BowTie) (l r : Term) (F : Formula) : Prop :=
  F = .atom b (.sub l r) (.const 0)

```

BernsteinLogic.Core.Catalogue . **PArithAtomData**

definition bound

```

inductive PArithAtomData : Hypernet → Sequent → Type where
| fold (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (termL termR : Term) (b : BowTie)
  (notNE : b ≠ .ne)
  (preEq : seq.pre = premise.pre)
  (premiseIsZero : atomicCompareZero b termL termR premise.post)
  (conclIsDirect : atomicCompareForm b termL termR seq.post)
  : PArithAtomData H seq
| unfold (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (termL termR : Term) (b : BowTie)
  (notNE : b ≠ .ne)
  (preEq : seq.pre = premise.pre)
  (premiseIsDirect : atomicCompareForm b termL termR premise.post)
  (conclIsZero : atomicCompareZero b termL termR seq.post)
  : PArithAtomData H seq

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in **AX**; sub-derivation premise as at **WEAKEN**.

- the display's two inference lines  $\mapsto$  the lemma pair, mirroring the payload's two constructors: **ARITHATOM** (zero-form premise, direct conclusion)  $\mapsto$  `sound_arithAtom_fold`; **ARITHATOM**<sup>-1</sup>  $\mapsto$  `sound_arithAtom_unfold`.

- $t_1, t_2 \mapsto \text{termL termR} : \text{Term}; \bowtie \mapsto b : \text{BowTie}$ ; the two spellings  $\mapsto$  the form predicates quoted below, each a one-line syntactic equation: `atomicCompareForm` (the atom is literally  $t_1 \bowtie t_2$ ) and `atomicCompareZero` (literally  $(t_1 - t_2) \bowtie 0$ , syntactic difference against the literal constant).
- $\Phi \mapsto$  the shared context via `hpre`; the premise judgment  $\mapsto \text{ih}$ ; the conclusion  $\mapsto$  the goal `seq.semanticallyValid H`. Certified alongside: `QTensor.map2_defined_of_rel2`, the totality bridge with which the unfold direction manufactures the difference's definedness from the holding premise.

Width note, deliberate and load-bearing (the `PConstExact` pattern): the display restricts  $\bowtie \in \{<, \leq, =, \geq, >\}$ , while both lemmas are proven for *all* constructors of `BowTie`, including  $\neq$  — no  $\bowtie \neq$  hypothesis appears in either statement. The exclusion is an admission-layer fact: the payload's `notNE` field refuses  $\neq$  certificates to protect the three-valued screen the commentary describes. The mechanized statements are therefore strictly wider than the displayed rule, and the displayed rule is exactly the admitted fragment of them.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory `bern_catalogue`, HOL4 development script `bern_catalogueScript.sml` script bound

### Definition `atomicCompareFormB_def`

```
Definition atomicCompareFormB_def:
  atomicCompareFormB b l r fm ⇔ fm = FAtom b l r
End
```

### Definition `atomicCompareZeroB_def`

```
Definition atomicCompareZeroB_def:
  atomicCompareZeroB b l r fm ⇔ fm = FAtom b (TmSub l r) (TmConst 0)
End
```

## HOL verifier receipts

**documentary** hol-strict CLEAN census over the ported checker theory: `bern_catalogue`, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: `DISK_THM`; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here — the earlier stale-run caveat (a 2026-07-18 script revision postdating the prior census) is retired by this run, which attests the current text. There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twins `sound_arithAtom_fold` (theory `bern_tr_ready8`: hol-strict clean, 8 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources) and `sound_arithAtom_unfold` (theory `bern_tr_fragclose`, 26 theorems, 0 offences, same run), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1). Setting as fixed in the EQELIM section. The rule is a pair: one payload constructor per direction, sharing the comparator gate; both directions are proven, and the two proofs are deliberately *not* mirror images — the asymmetry is the interesting content.

**Claim** (`sound_arithAtom_fold`, `sound_arithAtom_unfold`). The payload supplies terms  $t_1, t_2$  and a comparator  $\bowtie \in \{<, \leq, =, \geq, >\}$  (the gate  $\bowtie \neq \neq$  is a checked field). The two atom spellings in play, both recomputed syntactically by the checker from the same payload terms, are the *direct form*  $t_1 \bowtie t_2$  and the *zero-normal form*  $(t_1 - t_2) \bowtie 0$ , where the subtrahend side is the literal constant 0 and the left side the literal syntactic difference. *Fold*: premise concludes the zero-normal form, conclusion is the direct form. *Unfold*: the converse. In both directions, validity of the premise sequent gives validity of the conclusion sequent, over the shared  $\Phi$ .

**Lemma R14.1** (scalar bridge). *For  $p, q \in \mathbb{Q}$  and  $\bowtie$  any of the five comparators:  $(p - q) \bowtie 0 \iff p \bowtie q$ .*

*Proof.* Add  $q$  to both sides, respectively subtract it — the order of  $\mathbb{Q}$  is translation-invariant. The argument is uniform in  $\bowtie$ ; it would, for that matter, be equally valid for  $\neq$  — the comparator gate is *not* forced at this level (Remark R14.8).  $\square$

**Lemma R14.2** (tensor bridge). *Let  $A, B$  be tensors whose elementwise difference is defined, say  $D$ . Then  $D$  stands in the elementwise lift of  $\bowtie$  against the rank-0 tensor 0 iff  $A, B$  stand in the elementwise lift of  $\bowtie$ .*

*Proof.*  $D$ 's entries are the pointwise differences under the broadcast discipline: at equal shapes, entry  $i$  of  $D$  is  $A_i - B_i$ ; with rank-0  $A$ , entry  $i$  is  $A_{\text{scalar}} - B_i$ ; with rank-0  $B$ ,  $A_i - B_{\text{scalar}}$  — and  $D$  takes the positive-rank operand's shape (the broadcast join). The left side of the claimed equivalence reads every entry of  $D$  against the scalar 0; the right side reads the operands in the corresponding branch of the elementwise lift. In each of the three branches the two readings pair off the same scalar comparisons entry by entry, and Lemma R14.1 closes each pair. (`qTensor.rel2_sub_zero_iff`.)  $\square$

**Lemma R14.3** (definedness, fold direction). *If the zero-normal atom holds at  $\sigma$ , then both  $t_1$  and  $t_2$  evaluate at  $\sigma$ , and their elementwise difference is the evaluation of  $t_1 - t_2$ .*

*Proof.* The holding atom includes a defined evaluation of the syntactic difference; the evaluator is structural and strict (the Setting: a compound evaluates exactly when its parts evaluate and their shapes join), so inverting it at the subtraction node yields defined evaluations of both operands and the elementwise-difference equation. (`Term.evalQ_sub_inv`.)  $\square$

**Lemma R14.4** (definedness, unfold direction). *If the direct atom holds at  $\sigma$ , then the elementwise difference of the two (defined, by the atom's holding) evaluations is itself defined.*

*Proof.* A holding elementwise comparison forces broadcast-compatible shapes — the lift's fourth branch, unequal positive ranks, is empty — and elementwise subtraction is total on exactly the broadcast-compatible pairs. A holding comparison is thereby itself a compatibility certificate. (`qTensor.map2_defined_of_rel2`.)  $\square$

**Theorem R14.5** (soundness of ARITHATOM, both directions). *Over shared  $\Phi$ : validity of  $\Phi \vdash (t_1 - t_2) \bowtie 0$  gives validity of  $\Phi \vdash t_1 \bowtie t_2$ , and conversely.*

*Proof.* Fix an admissible  $\sigma \models \Phi$  in either direction. *Fold*: the premise's zero-normal atom holds; its constant side evaluates to the rank-0 tensor 0; Lemma R14.3 recovers evaluations  $A, B$  of  $t_1, t_2$  with difference  $D$ , and the premise's relation is  $D \bowtie 0$  elementwise; Lemma R14.2 converts it to  $A \bowtie B$  elementwise — the direct atom, on the operands' own evaluations. *Unfold*: the premise's direct atom holds with evaluations  $A, B$ ; Lemma R14.4 manufactures the difference  $D$ ; the evaluator assembles  $\text{ev}_\sigma(t_1 - t_2) = D$  from its operands' evaluations, the constant side is again rank-0 0, and Lemma R14.2, read right to left, closes the zero-normal atom.  $\square$

**Remark R14.6** (the direction asymmetry, and payload minimalism). The fold direction *consumes* definedness of the difference (the premise evaluated it); the unfold direction must *manufacture* it — and can, precisely because a holding comparison certifies broadcast compatibility (Lemma R14.4). The consequence is worth stating plainly: whenever either atom *holds*, both operands' evaluations exist and transport to the other form. That is why this payload carries *no* typing or positivity fields at all — compare EQREFL's typing pack, EQELIM's shape agreement, and the positive-size fields of the combination rules: every one of those exists

to manufacture or protect an evaluation this rule gets handed for free. (Definedness itself is not mirrored form to form: at broadcast-incompatible operand shapes the direct form’s sides still evaluate while the zero-normal difference does not — but there neither atom can hold, each failing in its own way, and holding is all the rule transports; at a compatible zero-size shape the two forms hold together, vacuously.) The “exact rational evaluation transported both directions” is exactly the pair (Lemma R14.1, Lemma R14.2): a translation-invariance fact of ordered  $\mathbb{Q}$ , lifted entrywise through an invertible totality argument.

*Remark R14.7* (the syntactic pins). Both form conditions are single syntactic equations recomputed from the payload — the zero-normal form demands the *literal* constant 0 and the *literal* difference term of the same  $t_1, t_2$ , not any term evaluating to zero or any algebraic rearrangement. A producer wanting  $t_1 \bowtie t_2$  from, say,  $(t_2 - t_1) \cdot (-1) \bowtie 0$  routes through `NORM` first; this rule does arithmetic on *atom shape* only. That severity is what makes the side conditions decidable by construction and keeps the rule’s checked surface at two form equations, the shared-context equation, and one comparator gate.

*Remark R14.8* (why  $\neq$  is excluded). Not for the reason one might guess — and not for its sibling’s reason either. Lemmas R14.1 and R14.2 are comparator-uniform, and the rational semantics of this section would accept  $\neq$  without change. Nor would the development’s auxiliary three-valued screen (the countermodel construction that polices vacuity properties, developed at the  $\vee$ E section) break here: both spellings of a  $\neq$  atom mention the same wires, and the screen’s disequality clause reads only designated-wire membership, so fold and unfold preserve its value — unlike at the premise-free ground-fact rule, where a wire-free  $\neq$  atom’s screen value genuinely breaks the invariant (the `PCONSTEXACT` section). The comparator set  $\{<, \leq, =, \geq, >\}$  is the source calculus’s own set for this rule, and the payload’s gate keeps it checked: the screen’s per-rule argument stays a one-clause conclusion-shape case (the gate vacates its  $\neq$  branch), and the two atom-shape rules stay aligned.  $\neq$  remains in the formula grammar — it is simply not this rule’s business to move it across the zero-normal boundary.

| proof step                      | formal anchor                                                                                                                                                |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lemma R14.1                     | inlined in <code>QTensor.rel2_sub_zero_iff</code> ’s entrywise closure                                                                                       |
| Lemma R14.2                     | <code>QTensor.rel2_sub_zero_iff</code>                                                                                                                       |
| Lemma R14.3                     | <code>Term.evalQ_sub_inv</code>                                                                                                                              |
| Lemma R14.4                     | <code>QTensor.map2_defined_of_rel2</code>                                                                                                                    |
| Theorem R14.5, fold             | <code>sound_arithAtom_fold</code>                                                                                                                            |
| Theorem R14.5, unfold           | <code>sound_arithAtom_unfold</code> ; both wired as core <code>RuleData.arithAtom</code> into <code>ProofObject.sound</code>                                 |
| form equations, comparator gate | <code>atomicCompareForm</code> , <code>atomicCompareZero</code> ; <code>payload</code> <code>field</code> <code>notNE</code> ( <code>ParithAtomData</code> ) |

## Rule 15. BERNCONe (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *BernCone*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} \Delta = \text{conv}(V_0, \dots, V_n) \subset \mathbb{R}^n \text{ an } n\text{-simplex}; \quad S \subsetneq \{0, 1, \dots, n\} \\ \Phi \vdash x \in \mathcal{R}(\Delta, S) \quad p: \mathbb{R}^n \rightarrow W \text{ polynomial map of degree } d, \text{ with coefficient tensor } (b_\alpha)_{|\alpha|=d} \\ \text{wires}(p) = \emptyset \vee (x = w \wedge \text{wires}(p) \subseteq \{w\}) \quad S \neq \emptyset \Rightarrow P \text{ conic}; \quad S = \emptyset \Rightarrow P \text{ convex suffices} \\ (p \text{ idx-free, with } p \text{ constant } \vee n = 1) \vee (p \text{ projection-indexed, indices } < n, S = \emptyset) \\ \text{for every } j \in \mathbb{N}^S \text{ with } |j| \leq d, \text{ every degree-}d \text{ Bernstein coefficient of } P_j \text{ on } F_S \text{ satisfies } P \end{array}}{\Phi \vdash P(p(x))} \text{ BERNCONe}$$

the disjunctive premise on  $p$  is the checker's scope guard: coefficient checks on coordinate polynomials over ambient dimension  $\neq 1$  are refused rather than attempted outside the proven kernels.

BERNCONe is the calculus's arithmetic-closure primitive: from a *geometric* fact (a wire value lies in a region) it derives an *analytic* one of unbounded degree (a polynomial of that value satisfies a predicate) — where the conversion rules exchange region atoms with their defining affine sign conditions, BERNCONe is the rule that concludes something new. From a proven region membership  $\Phi \vdash x \in \mathcal{R}(\Delta, S)$  and a degree- $d$  polynomial term  $p$ , it concludes  $\Phi \vdash P(p(x))$  for any predicate  $P$  of the accepted grammar, provided every certified Bernstein coefficient of  $p$  over the region satisfies  $P$ . A producer reaches for it wherever a Lyapunov decrease, an invariant bound, or an output enclosure must be established over a whole region rather than at sample points: choose a degree, optionally subdivide (SUBDIVIDE is the completeness partner — enclosures converge under degree elevation and subdivision), and emit the payload.

What the checker recomputes, given the payload (simplex, extension set, degree, polynomial term, predicate): that the extension set is tied to the premise's simplex and the simplex is full-dimensional; that the polynomial term translates into the region's ambient coordinates (two routes: an idx-free single-wire canonical form, or the projection-indexed coordinate form); and then — the heart — *every* forward-difference Bernstein coefficient of the translated polynomial over the region, by exact  $\mathbb{Q}$  arithmetic, testing each against the predicate. No coefficient tensors ride the wire: the certificate names the polynomial and the region, and the checker rebuilds the entire coefficient table itself (the recompute pattern), so a forged table is unrepresentable rather than detected.

Each hypothesis is load-bearing. Drop the wire tie (`polyTie`) and a certificate could prove its region fact about one wire while running the coefficient check on a polynomial over a different one — concluding a bound about a value the premise never constrained. Drop the predicate-shape clause and a convex-but-not-conic predicate falsely transfers along an extension ray (the proof section computes a two-line counterexample: certified coefficients  $\{0, -1\}$  both satisfy  $\square \geq -1$ , yet the polynomial dips to  $-2$  on the ray). Drop the coefficient recomputation and the conclusion is unrelated to the polynomial. Drop the simplex tie or full-dimensionality and the barycentric reading of the premise breaks. The checker's own scope guard (constant canonical forms at any dimension; coordinate polynomials only over 1-dimensional regions when the extension set is nonempty) is part of the soundness story: everything beyond the proven fragment is refused, never accepted on faith.

**notes** Predicates enter through a deliberately closed-world grammar: finite conjunctions of scalar hole-atoms  $s \cdot \square \bowtie c$  with  $s > 0$  and  $\bowtie \in \{\leq, \geq, =\}$ , classified for convexity (always) and conicity ( $c = 0$  throughout). Comparators outside the grammar are refused — conservatism that can lose completeness, never soundness. The elementwise atom semantics batch the conclusion: a tensor-shaped polynomial term gives per-entry predicate application for free.

### Lean statement

`BernsteinLogic.Soundness.sound_bernCone_empty`

statement bound

```
-- BernCone at S = ∅ — the kernel assembly, at full D2 scope (leg D3:
-- the former `hd2` corner guard is gone — a polynomial whose wires cancel
-- in canonical form over a higher-dimensional region variable is handled
-- by the per-entry evaluation lemma `Term.evalQ_brRead`: every entry of
-- its semantic tensor equals the closed canonical value, which the kernel
-- certifies). -/
theorem sound_bernCone_empty {H : Hypernet} {seq regionPremise : Sequent}
  {termX : Term} {simplex : Simplex} {S : ExtensionSet} {degree : Nat}
  {polynomial : Term} {predicate : Formula}
  (ih : regionPremise.semanticallyValid H)
  (hpre : seq.pre = regionPremise.pre)
```



```

(hregionPost : regionPremise.post = .inRegion termX simplex S)
(hpolyTie : polynomial.freeWires = [] v
  ∃ w, termX = Term.var w ∧ polynomial.freeWires ⊆ [w])
(hconvex : isExtensionSetEmpty S → isConvexPredicate predicate)
(hsempty : isExtensionSetEmpty S)
(hcoeff : allForwardDiffBernCoeffsSatisfyP simplex S degree polynomial
  predicate)
(hpostApply : applyPredicateToTerm predicate polynomial seq.post) :
seq.semanticallyValid H

```

BernsteinLogic.Soundness . **sound\_bernCone**

statement bound

```

/-- BernCone, all extension sets (leg D3). At S = ∅ this is the
convex-predicate kernel assembly above; at S ≠ ∅ the check's own D2
scope guard confines reachable instances to constant canonical
polynomials (any dimension – `fdBernstein_sound_const`) and single-wire
polynomials over 1-dimensional regions, where a proper nonempty
extension set is exactly one of the two vertices and the region is the
extension ray (`fdBernstein_sound_dim1`, the univariate (★) identity).
The conic predicate-shape clause supplies zero/scaling/convexity
closure. -/
theorem sound_bernCone {H : Hypernet} {seq regionPremise : Sequent}
  {termX : Term} {simplex : Simplex} {S : ExtensionSet} {degree : Nat}
  {polynomial : Term} {predicate : Formula}
  (ih : regionPremise.semanticallyValid H)
  (hpre : seq.pre = regionPremise.pre)
  (hregionPost : regionPremise.post = .inRegion termX simplex S)
  (hpolyTie : polynomial.freeWires = [] v
    ∃ w, termX = Term.var w ∧ polynomial.freeWires ⊆ [w])
  (hpredShape : (¬ isExtensionSetEmpty S → isConicPredicate predicate)
    ∧ (isExtensionSetEmpty S → isConvexPredicate predicate))
  (hcoeff : allForwardDiffBernCoeffsSatisfyP simplex S degree polynomial
    predicate)
  (hpostApply : applyPredicateToTerm predicate polynomial seq.post) :
seq.semanticallyValid H

```

**What the coefficient side-condition means — and what it does not.** The rule's main premise appears in the payload as the single name `allForwardDiffBernCoeffsSatisfyP`, so this book binds its meaning by displaying the definition chain in full, from the checker sources. Reading the chain: `satisfiesPredQ` substitutes a candidate rational for the designated predicate-hole wire and runs the *closed* formula evaluator — the predicate is judged at a concrete value, by exact  $\mathbb{Q}$  arithmetic, never at a wire. `Term.asCoordPoly?` translates the polynomial term into the region's ambient coordinates, by one of two routes: an `idx`-free term goes through its wire-keyed canonical polynomial (at most one wire, remapped to coordinate 0), and a projection-indexed term maps each projection  $w[j]$  to coordinate variable  $j$ . `bernConeCoeffCheck` then gates (simplex tie, full-dimensionality, and the scope guard: non-constant `idx`-free polynomials only over 1-dimensional regions; multivariate coordinate forms only at  $S = \emptyset$  with all projection indices in range), computes the full forward-difference Bernstein coefficient table of the translated polynomial over the masked vertex list, and sweeps every grid entry through the predicate.

What a skeptical reader should confirm by eye in the definitions below: the conclusion formula never enters the check — only the polynomial, the region data, and the predicate do (the conclusion is pinned separately, as the syntactic substitution of the polynomial into the predicate's hole); nothing in the chain reads a valuation, so the check cannot covertly assume anything about wire values; and every false branch of the guards is a refusal — the fragment boundary (dimension, extension set, index range) fails closed. Semantic truth is established only in `sound_bernCone`, from the region premise's validity.

BernsteinLogic.Core.Catalogue . **satisfiesPredQ**

definition bound

```

/-- Predicate satisfaction at a rational value. -/
def satisfiesPredQ (P : Formula) (c : Rat) : Bool :=
  (Formula.subst predicateHole (.const c) P).evalClosed = some true

```

BernsteinLogic.Core.Catalogue . **Term.asCoordPolyIdx?**

definition bound

```

/-- The G1 coordinate-form translation: a Term whose every wire mention
is an idx projection `idx (.var w) j` becomes a polynomial over the
ambient-coordinate variables j. Which wire is projected is deliberately
NOT recorded here – the BernCone payload's `polyTie` (B10/B7) pins every
free wire to the region variable. `none` on: a bare (vector-valued) wire
mention, and on idx of any non-var argument (v0.5 scope; einsum is the

```



```

planned v1+ extension). Index range is checked separately
(`Term.idxBounded`, conjoined by `bernConeCoeffCheck`). -/
def Term.asCoordPolyIdx? : Term → Option QArith.Poly
| .var _ => none
| .const c => some (QArith.Poly.const c)
| .add l r =>
  do pure (QArith.Poly.add (← l.asCoordPolyIdx?) (← r.asCoordPolyIdx?))
| .mul l r =>
  do pure (QArith.Poly.mul (← l.asCoordPolyIdx?) (← r.asCoordPolyIdx?))
| .sub l r =>
  do pure (QArith.Poly.sub (← l.asCoordPolyIdx?) (← r.asCoordPolyIdx?))
| .neg t => (Term.asCoordPolyIdx? t).map QArith.Poly.neg
| .idx (.var _) i => some (QArith.Poly.var i)
| .idx _ _ => none

```

BernsteinLogic.Core.Catalogue . **Term.asCoordPoly?**

definition bound

```

def Term.asCoordPoly? (t : Term) : Option QArith.Poly :=
  if t.idxFree then
    let p := t.toPoly
    match QArith.Poly.vars p with
    | [] => some p
    | [_] => some (p.map fun (m, c) => (m.map fun (_, e) => (0, e), c))
    | _ => none
  else
    t.asCoordPolyIdx?

```

BernsteinLogic.Core.Catalogue . **bernConeCoeffCheck**

definition bound

```

/-- The recomputation behind BernCone's main side-condition (rem:recompute-
vs-trust option (a), CORRESPONDENCE.md D-6; see original for the guard
rationale). -/
def bernConeCoeffCheck (simplex : Simplex) (S : ExtensionSet) (degree : Nat)
  (polynomial : Term) (predicate : Formula) : Bool :=
  if S.simplex = simplex then
    let cx := simplex.complex
    let verts := (List.finRange cx.nVertices).filter (fun v => simplex.contains v)
    -- full-dimensionality: an (ambDim)-simplex needs ambDim+1 vertices
    if verts.length = cx.ambDim + 1 then
      match Term.asCoordPoly? polynomial with
      | none => false
      | some p =>
        if polynomial.idxFree then
          -- legacy D2 scope: coordinate-referencing polynomials only
          -- over 1-D regions (any extension set)
          if QArith.Poly.vars p ≠ [] ∧ cx.ambDim ≠ 1 then false
        else
          let coords := verts.map fun v =>
            (List.finRange cx.ambDim).map fun j => cx.coord v j
          let smask := verts.map fun v => S.containsNat v.val
          match QArith.fdBernstein coords smask degree p with
          | some table => QArith.gridAll (satisfiesPredQ predicate) table
          | none => false
    else
      -- G1 idx coordinate form (multivariate): S = ∅ only – the
      -- general-S multivariate soundness is the v1 StarIdentity
      -- obligation – and every projection index below the ambient
      -- dimension (out-of-range reads make the conclusion
      -- unevaluable; see `Term.idxBounded`).
      if S.isEmptyB ∧ polynomial.idxBounded cx.ambDim then
        let coords := verts.map fun v =>
          (List.finRange cx.ambDim).map fun j => cx.coord v j
        let smask := verts.map fun v => S.containsNat v.val
        match QArith.fdBernstein coords smask degree p with
        | some table => QArith.gridAll (satisfiesPredQ predicate) table
        | none => false

```

```

      else false
    else false
  else false

```

BernsteinLogic.Core.Catalogue.allForwardDiffBernCoeffsSatisfyP

definition bound

```

def allForwardDiffBernCoeffsSatisfyP
  (simplex : Simplex) (S : ExtensionSet) (degree : Nat)
  (polynomial : Term) (predicate : Formula) : Prop :=
  bernConeCoeffCheck simplex S degree polynomial predicate = true

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; sub-derivation premise as at WEAKEN. The general statement is `sound_bernCone`; `sound_bernCone_empty` (certified alongside) is its  $S = \emptyset$  specialization, through which the general proof routes that case.

- $x \mapsto \text{termX} : \text{Term}; \Delta \mapsto \text{simplex}; S \mapsto S : \text{ExtensionSet}; d \mapsto \text{degree} : \text{Nat}; p \mapsto \text{polynomial} : \text{Term}; P \mapsto \text{predicate} : \text{Formula}$  (the hole-atom grammar of the note, its hole a designated wire).
- the region premise  $\Phi \vdash x \in \mathcal{R}(\Delta, S) \mapsto \text{ih}$  with `hregionPost : regionPremise.post = .inRegion termX simplex S;  $\Phi$`  shared via `hpre`.
- the wire-tie premise  $\text{wires}(p) = \emptyset \vee (x = w \wedge \text{wires}(p) \subseteq \{w\}) \mapsto \text{hpolyTie}$ , the same disjunction verbatim — note `termX = Term.var w`: the display's “ $x = w$ ” means the region term literally *is* that wire.
- the two-clause predicate premise  $\mapsto \text{hpredShape}$ : the conjunction of ( $S$  nonempty  $\rightarrow$  conic) and ( $S$  empty  $\rightarrow$  convex), by the decidable predicate classifiers.
- the displayed coefficient sweep  $\mapsto \text{hcoeff} : \text{allForwardDiffBernCoeffsSatisfyP simplex S degree polynomial predicate}$  — one name whose meaning this book binds by displaying its definition chain in full (the intro above); the display's first premise line (simplex tie, full-dimensionality) and its scope-guard premise have no Lean binder of their own because the checker's gates certify them: *absorbed*, as `PCONSTEXACT`'s closedness is.
- conclusion  $\Phi \vdash P(p(x)) \mapsto$  the goal under `hpostApply : applyPredicateToTerm predicate polynomial seq.post` — the conclusion is the syntactic substitution of the polynomial term into the predicate's hole; the “at  $x$ ” lives in the wire tie, not in a separate application node.

Width note, two-sided and honest: the statement is *proven* at every hypernet, sequent, and payload — and `hcoeff` is satisfiable only inside the displayed scope-guard premise (constant canonical forms at any dimension; non-constant `idx`-free forms over 1-dimensional regions; multivariate coordinate forms at  $S = \emptyset$  only), so the certified reach of the displayed rule is exactly its guard-passing fragment, by refusal outside it — never by unverified acceptance. Conversely nothing in either statement is wider than the display: every hypothesis is consumed, and the display's own two-clause predicate license is carried exactly.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

**comparator: fixture DemoBernCone** the three-implementation comparator harness (compiled machine code, kernel-reduction probes, and the compiler-IR interpreter, all executing the same elaborated checker functions) carries the worked example's certificate — an  $Ax \rightarrow \wedge E \rightarrow \text{BERNCONE}$  chain at  $S = \emptyset$  — as its fixture DemoBernCone. The suite passes only when all three paths return identical verdicts on it, and the kernel path probes two-sided (reporting its own verdict rather than confirming the compiled one), so a false green would require the same wrong answer from execution stacks that share nothing below the elaborated definitions.

## HOL4 statement

theory bern\_catalogue, HOL4 development script bern\_catalogueScript.sml script bound

#### Definition bernConeCoeffCheck\_def

```

Definition bernConeCoeffCheck_def:
  bernConeCoeffCheck sx es dg pl pr =
    if es.simplex = sx then
      (let cx = sx.complex in
        let verts = FILTER (λv. simplexContainsNat sx v) (rangeL cx.nVertices) in
        if LENGTH verts = cx.ambDim + 1 then
          (case termAsCoordPoly pl of
            | NONE => F
            | SOME p =>
              if termIdxFree pl then
                (if polyVars p ≠ [] ∧ cx.ambDim ≠ 1 then F
                 else
                  let coords = MAP (λv. MAP (λj. getD (getD cx.coord v []) j (0:rat))
                                         (rangeL cx.ambDim)) verts in
                  let smask = MAP (λv. extContainsNat es v) verts in
                  case fdBernstein coords smask dg p of
                    | SOME table => gridAll (satisfiesPredQ pr) table
                    | NONE => F)
              else
                (if extIsEmptyB es ∧ termIdxBounded cx.ambDim pl then
                  (let coords = MAP (λv. MAP (λj. getD (getD cx.coord v []) j (0:rat))
                                         (rangeL cx.ambDim)) verts in
                  let smask = MAP (λv. extContainsNat es v) verts in
                  case fdBernstein coords smask dg p of
                    | SOME table => gridAll (satisfiesPredQ pr) table
                    | NONE => F)
                else F))
          else F)
        else F
    End

```

#### Definition bernConeIntvCheck\_def

```

Definition bernConeIntvCheck_def:
  bernConeIntvCheck sx es dg pl pr dt =
    if es.simplex = sx then
      (let cx = sx.complex in
        let verts = FILTER (λv. simplexContainsNat sx v) (rangeL cx.nVertices) in
        if LENGTH verts = cx.ambDim + 1 then
          (case termAsCoordPoly pl of
            | NONE => F

```

```

| SOME p =>
  if ¬ termIdxFree pl then F
  else if polyVars p ≠ [] ∧ cx.ambDim ≠ 1 then F
  else
    let coords = MAP (λv. MAP (λj. getD (getD cx.coord v []) j (0:rat))
                        (rangeL cx.ambDim)) verts in
    let smask = MAP (λv. extContainsNat es v) verts in
    case fdBernstein coords smask dg p of
    | SOME table =>
      gridAll (λc. case dtypeRoundE dt Rdn c of
                    | Fin lo => (lo ≤ c ∧ satisfiesPredQ pr lo)
                    | _ => F) table
    | NONE => F)
  else F
End

```

## HOL verifier receipts

[documentary](#) hol-strict CLEAN over the ported checker chain: theory bern\_catalogue, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, scripts sha-matched verbatim to the sources cited here (git blob e41b9ed0...). The ported receipt table replays the BERNcone rows of the Lean development's own receipt battery: the worked-example accept (bernConeCoeffCheck at  $S = \emptyset$ , degree 1) plus four mutant rejections (tightened goal, wrong dynamics sign, degree lie, sub-dimensional simplex) and the nonconvex-predicate rejection. Downstream, the same chain carries the checker to a CakeML binary with the machine\_code\_sound tag (theory bernCertProofARM8). bernConeIntvCheck is the ported companion of a since-retired validated-numeric rule variant; its round-down-then-check strategy is destined to fold into the BernCone checker as an internal fast path (evaluation strategy, not proof content), and the ported definition with its three receipt rows (exact-dyadic accept, rounded-bound accept, non-representable-bound reject) is retained as that branch's reference. Semantic stratum, also ported: the soundness twins sound\_bernCone and its  $S = \emptyset$  specialization sound\_bernCone\_empty (theory bern\_tr\_berncone: hol-strict clean, 14 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (checkPf\_sound\_frag, theory bern\_tr\_splice, 20 theorems, 0 offences, same run) under the payload's decode-time gates: a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail. Every hypothesis of the rule is used below and remarked on, with the counterexample its removal admits where one exists; the formal-anchor table maps each step to the Lean declaration that carries it.

**Setting and claim.** Fix a hypernet  $H$ . A valuation  $\sigma$  assigns each wire a rational tensor of its declared shape; a sequent  $\Phi \vdash \Psi$  is semantically valid when every admissible  $\sigma$  satisfying  $\Phi$  satisfies  $\Psi$ , with comparison atoms read per-entry. A region atom  $x \in \mathcal{R}(\Delta, S)$ , for  $\Delta = \text{conv}(V_0, \dots, V_n)$  an  $n$ -simplex and  $S \subsetneq \{0, \dots, n\}$  a proper set of *extended* vertices, holds at  $\sigma$  when the evaluated point admits barycentric weights  $\lambda_v$  with  $\sum_v \lambda_v = 1$ ,  $\lambda_v \geq 0$  at core vertices  $v \notin S$ , and  $\lambda_v \leq 0$  at  $v \in S$  — the closed simplex when  $S = \emptyset$ , and the unbounded set extending beyond the faces opposite  $S$  otherwise. (Field bookkeeping, fixed once: the vertices  $V_0, \dots, V_n$  are exact rational data of the payload; the weights  $\lambda_v$  are quantified over  $\mathbb{R}$ ; and the evaluated point is rational, valuations assigning rational tensors — Step 2 of the theorem is where these three facts meet.) The rule's payload supplies the region premise  $\Phi \vdash x \in \mathcal{R}(\Delta, S)$ , a polynomial term  $p$  of degree  $\leq d$ , a predicate  $P$  from the accepted grammar (a finite conjunction of scalar hole-atoms  $s \cdot \square \bowtie c$ ,  $s > 0$ ,  $\bowtie \in \{\leq, \geq, =\}$ ), the checked wire tie polyTie, the predicate-shape clause ( $P$  conic when  $S \neq \emptyset$ ; convex suffices when  $S = \emptyset$ ), and the checked coefficient condition: every forward-difference Bernstein coefficient of  $p$  over  $\mathcal{R}(\Delta, S)$ , recomputed exactly, satisfies  $P$ . For  $S \neq \emptyset$  the checker's scope guard is part of the rule's acceptance: certificates are confined to constant canonical forms (any dimension) or single-wire polynomials over a 1-dimensional region (**Remark R15.8**) — “accepted” below always means guard-passing. The claim (sound\_bernCone): the composite sequent  $\Phi \vdash P(p(x))$  is then semantically valid.

**Lemma R15.1** (Bernstein form on the closed simplex). *Let  $p$  be a polynomial of degree  $\leq d$  on  $\mathbb{R}^n$  with degree- $d$  Bernstein coefficients  $(b_\alpha)_{|\alpha|=d}$  on  $\Delta$ , so that  $p(x) = \sum_{|\alpha|=d} b_\alpha B_\alpha^d(\lambda)$  with  $B_\alpha^d(\lambda) = \binom{d}{\alpha} \lambda^\alpha$  in the barycentric coordinates  $\lambda$  of  $x$ . For  $x \in \Delta$  (all  $\lambda_v \geq 0$ ),*

$$B_\alpha^d(\lambda) \geq 0 \quad \text{and} \quad \sum_{|\alpha|=d} B_\alpha^d(\lambda) = \left( \sum_v \lambda_v \right)^d = 1,$$

*so  $p(x)$  is a convex combination of the coefficients  $b_\alpha$ .*

*Proof.* Nonnegativity is termwise ( $\lambda^\alpha \geq 0$ ); the partition of unity is the multinomial theorem applied to  $(\lambda_0 + \dots + \lambda_n)^d = 1^d$ . The representation itself is the classical change to the Bernstein basis [18, 74]; the checker *computes* the  $b_\alpha$  from  $p$ 's canonical monomial form by exact forward differencing over  $\mathbb{Q}$ , so the representation is an identity of exact rational data, not an approximation. That Bernstein coefficients enclose the range of  $p$  on  $\Delta$  — the reading of this lemma used below — is the sound (easy) direction of [39].  $\square$

**Lemma R15.2** (accepted predicates are convex). *Let  $P$  be accepted by the predicate classifier. Then its satisfaction set  $\{v \in \mathbb{Q} : \text{satisfiesPredQ}(P, v)\}$  is convex.*

*Proof.* Each accepted atom orients (possibly by mirroring) to  $s \cdot v \bowtie c$  with  $s > 0$  and  $\bowtie \in \{\leq, \geq, =\}$ ; its satisfaction set is a closed ray or a point — an interval either way — and division by the positive  $s$  does not disturb this. A conjunction intersects finitely many intervals. Strict comparators and  $\neq$  are *refused* by the classifier — closed-world conservatism that can lose completeness but never soundness; the refusal is a checker verdict, not a soundness case.  $\square$

**Lemma R15.3** (the univariate ray identity). *Let  $d \geq 0$  and let  $b_0, \dots, b_d$  be the degree- $d$  Bernstein coefficients of  $p$  on the segment  $[V_a, V_b]$ , indexed by the multiplicity of  $V_b$  (labels chosen so that the extended vertex is  $V_a$  and the core vertex is  $V_b$ ; the mirrored case relabels — concretely, with the other vertex extended, apply the identity to the reversed sequence  $c_i := b_{d-i}$ , whose differences against the un-reversed one are the signed  $(\Delta^j c)_{d-j} = (-1)^j (\Delta^j b)_0$ ; the checked table is computed relative to the extension mask, so its rows are the  $(\Delta^j c)_{d-j}$  of the relabeled orientation). Let  $x$  lie on the extension ray beyond  $V_b$ : in barycentric coordinates  $x = \lambda_a V_a + \lambda_b V_b$  with  $\lambda_a = -u \leq 0$ ,  $\lambda_b = 1 + u$ ,  $u \geq 0$ . Then, with  $(\Delta b)_i := b_{i+1} - b_i$  the forward difference,*

$$p(x) = \sum_{j=0}^d \binom{d}{j} u^j (\Delta^j b)_{d-j}, \quad (*)$$

a combination of the certified table values  $(\Delta^j b)_{d-j}$  with nonnegative weights  $\binom{d}{j} u^j$ , the  $j = 0$  weight being 1.

*Proof.* Substitute  $\lambda_a = -u$ ,  $\lambda_b = 1 + u$  into the Bernstein form and expand  $(1 + u)^i$  binomially:

$$p(x) = \sum_{i=0}^d \binom{d}{i} b_i (-u)^{d-i} (1 + u)^i = \sum_{i=0}^d \sum_{k=0}^i \binom{d}{i} \binom{i}{k} (-1)^{d-i} b_i u^{d-i+k}.$$

Collect the coefficient of  $u^j$ : setting  $i = d - j + m$  with  $m = 0, \dots, j$ , it is  $\sum_{m=0}^j \binom{d}{d-j+m} \binom{d-j+m}{m} (-1)^{j-m} b_{d-j+m}$ . The product of binomials telescopes,

$$\binom{d}{d-j+m} \binom{d-j+m}{m} = \frac{d!}{(j-m)! m! (d-j)!} = \binom{d}{j} \binom{j}{m},$$

so the coefficient of  $u^j$  is  $\binom{d}{j} \sum_{m=0}^j \binom{j}{m} (-1)^{j-m} b_{d-j+m} = \binom{d}{j} (\Delta^j b)_{d-j}$ , which is  $(*)$ . (Degree-1 sanity check:  $p(x) = b_1 + u(b_1 - b_0)$ , matching  $(1 + u)b_1 - ub_0$  directly.) The values  $(\Delta^j b)_{d-j}$  are exactly the rows the checker's forward-difference table certifies on this fragment: the  $j$ -th difference of the coefficient sequence, anchored at the core vertex — in the rule display's notation, the degree- $d$  Bernstein coefficients of the difference polynomials  $P_j$  on the core face  $F_S$ , specialized to the one-dimensional and constant shapes the scope guard admits.  $\square$

**Lemma R15.4** (conic transfer). *Let  $P$  be accepted with the conic classification (every atom's constant is 0). Then its satisfaction set contains 0, and is closed under nonnegative scaling and under finite nonnegative combinations.*

*Proof.* Each conic atom reads  $s \cdot v \bowtie 0$ , satisfied at  $v = 0$  and preserved under  $v \mapsto tv$  for  $t \geq 0$ ; conjunction preserves both facts. Closure under addition follows from convexity (Lemma R15.2) plus scaling:  $v + w = 2 \cdot \frac{1}{2}(v + w)$ ; induction extends to finite nonnegative combinations.  $\square$

**Theorem R15.5** (soundness of BERNCONe). *Under the payload's checked hypotheses, if the region premise is semantically valid, then so is  $\Phi \vdash P(p(x))$ .*

*Proof.* Fix an admissible  $\sigma \models \Phi$ ; the region premise yields barycentric weights for the evaluated point of  $x$  as in the Setting. The conclusion formula is, by the checked `postApply`, the predicate  $P$  with the polynomial term substituted for its hole, so it suffices to establish `satisfiesPredQ`( $P, \cdot$ ) at the evaluated value(s) of  $p$  — per-entry, when  $p$  is tensor-shaped.

*Step 1 (the polynomial reads one point).* The wire tie `polyTie` says  $p$  is closed, or mentions exactly the region wire  $w$  with  $x = w$ . In the second case the region premise pins  $\sigma$ 's read of  $w$  to a point of  $\mathcal{R}(\Delta, S)$ ; in the first, no wire is read at all and  $p$ 's value is its closed constant.

*Step 2 (rational weights).* The model's barycentric weights are *real*; the kernel computes over  $\mathbb{Q}$ . The weights solve a linear system with rational data (the vertex coordinates and the evaluated point) under the region atom's sign pattern —  $\lambda_v \geq 0$  at core vertices,  $\lambda_v \leq 0$  at extended ones. Substituting  $\mu_v := -\lambda_v$  at the extended vertices turns the pattern into a nonnegative-orthant condition on a rational polyhedron, so a real solution can be traded for a rational one with the same signs, and the exact kernel consumes the rational weights. This step is invisible in informal treatments and load-bearing in the mechanization.

*Step 3 (case  $S = \emptyset$ : the convex kernel).* By Lemma R15.1,  $p$ 's value is a convex combination of the certified coefficients; every coefficient satisfies  $P$  (the checked table sweep); by Lemma R15.2 satisfaction transfers to the combination. On this case the

checker admits both translation routes — the idx-free single-wire form and the multivariate projection-indexed form with all indices in range — and the per-entry evaluation lemma carries the scalar argument to every entry of a tensor-shaped value.

*Step 4 (case  $S \neq \emptyset$ ).* The checker’s scope guard confines accepted certificates to two shapes. (4a: *constant canonical form, any dimension.*) If  $p$ ’s canonical polynomial has no variables, its value everywhere is the constant — the zero-order forward-difference row is that constant, certified directly. (4b: *single-wire polynomial over a 1-dimensional region.*) A proper nonempty extension set on a 1-simplex extends exactly one of the two vertices, and  $\mathcal{R}(\Delta, S)$  is the closed ray from the core vertex; Lemma R15.3 writes  $p$ ’s value as a nonnegative combination of the certified table values with the  $j = 0$  weight equal to 1, and Lemma R15.4 transfers satisfaction through the combination. Everything else is *refused* by the guard, so no further case exists for soundness to cover.

*Step 5 (read off).* In each case  $\text{satisfiesPredQ}(P, \cdot)$  holds at the evaluated value; the substitution reading turns this into  $\sigma \models P(p(x))$ .  $\square$

*Remark R15.6 (why polyTie is not decoration).* Nothing else ties the checked polynomial to the wire the premise constrains. Without the field, take the satisfiable premise  $w_0 \in \mathcal{R}(\text{conv}(1, 2), \emptyset)$  and the polynomial  $p = w_1$  over a *different* wire: the translation remaps the single-variable canonical form onto the region coordinate, the certified coefficients are  $\{1, 2\}$  — both satisfying  $\square \geq 1$  — and the rule would conclude  $\Phi \vdash w_1 \geq 1$ , false at any valuation reading  $\sigma(w_1) = 0$ . Closed polynomials need no tie: their value reads no wire.

*Remark R15.7 (why conic, not merely convex, on rays).* The predicate-shape clause is two-sided because the two region shapes transfer differently: convex combinations (Lemma R15.1) on the closed simplex, conic combinations with unbounded weights ( $\star$ ) on a ray. A convex-but-not-conic predicate fails on the ray, by hand: take  $d = 1$ ,  $\Delta = \text{conv}(0, 1)$  with the vertex at 0 extended (the region is the ray  $[1, \infty)$ ),  $p(t) = 1 - t$ , and  $P = (\square \geq -1)$ . The certified table is  $\{b_1, (\Delta b)_0\} = \{0, -1\}$ , every value satisfying  $P$ ; yet at  $u = 2$  (the point  $t = 3$ ), ( $\star$ ) gives  $p = b_1 + 2(b_1 - b_0) = -2 < -1$ . The classifier refuses exactly this: an atom with nonzero constant classifies non-conic, and the  $S \neq \emptyset$  clause then rejects the certificate. Conversely, at  $S = \emptyset$  demanding conic would be needless strength — the rule carries the license (any accepted convex predicate) in the premise itself rather than in surrounding prose.

*Remark R15.8 (the scope guard is part of the theorem).* The checker refuses non-constant idx-free polynomials over regions of dimension  $\neq 1$ , and multivariate projection-indexed polynomials whenever  $S \neq \emptyset$ . Soundness is stated and proven for every certificate the checker *accepts*; the guard makes the unproven territory unrepresentable rather than discouraged. The fully general multivariate ray identity (the arbitrary-shape analogue of Lemma R15.3) remains open in the development and becomes load-bearing only if the guard lifts — the two must move together.

*Remark R15.9 (relation to the literature).* The rule is the certificate form of Bernstein-coefficient range enclosure: coefficients bound the range on the simplex [39], enclosures converge under degree elevation and subdivision [85] — which is why SUBDIVIDE is this rule’s completeness partner — and the basis itself goes back to [18], surveyed in [74]. With polynomial atoms and positivity certificates the rule sits beside the Krivine–Handelman family [110, 144]: as everywhere in this calculus, only the trivial (sound) direction is claimed or needed — no Positivstellensatz completeness enters the kernel. The forward-difference computation of the coefficient table is the classical finite-difference reading of the Bernstein form; the subdivision algorithms of [40, 151] operate on the same data one rule over.

| proof step                                                                            | formal anchor                                                                                                                                                                                   |
|---------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Setting: region atoms as masked barycentric membership<br>Lemma R15.1 + Step 3 kernel | MaskRegion; Formula.holds (inRegion case)<br>the $S = \emptyset$ convex kernel consumed by sound_bernCone_empty; per-entry<br>carry Term.evalQ_brRead                                           |
| Step 2 rational weights<br>Lemma R15.2                                                | maskRegion_rat_weights; ConvexRat.exists_nonneg_rat_solution                                                                                                                                    |
| Lemma R15.3 (univariate ( $\star$ ))                                                  | classifyPred, satisfiesPredQ_convex                                                                                                                                                             |
| Step 4a constant corner<br>Lemma R15.4                                                | QArith.fdBernstein_sound_dim1                                                                                                                                                                   |
| coefficient recomputation (the checked table)                                         | QArith.fdBernstein_sound_const<br>satisfiesPredQ_conic_zero, satisfiesPredQ_conic_scale<br>bernConeCoeffCheck, allForwardDiffBernCoeffsSatisfyP,                                                |
| coordinate translation + index gate                                                   | QArith.fdBernstein, QArith.gridAll                                                                                                                                                              |
| Step 1 wire tie                                                                       | Term.asCoordPoly?, Term.asCoordPolyIdx?, Term.idxBounded                                                                                                                                        |
| Step 5 conclusion reading<br>Theorem R15.5                                            | payload field polyTie<br>applyPredicateToTerm, holds_substHole_of_satisfies<br>sound_bernCone (case split onto sound_bernCone_empty); wired as core<br>RuleData.bernCone into ProofObject.sound |
| open residue (Remark R15.8)                                                           | ConeLemma.StarIdentity (general shapes; not consulted by the current<br>guard)                                                                                                                  |



## 15.3 Admissible and Alias Rules

### Rule 16. NORM (admissible rule)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Norm*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{t_1, t_2 \text{ polynomial terms in the same variables with equal monomial canonical forms over } \mathbb{Q} \quad \text{bshape}_H(t_1) = \text{bshape}_H(t_2) \neq \perp \text{ (a shared defined broadcast shape)} \quad \text{wires}(t_1), \text{wires}(t_2) \text{ all finitely typed in } H}{\Phi \vdash t_1 = t_2} \text{ NORM}$$

NORM is the calculus's computation rule: it states, as a premise-free sequent, every equality of polynomial terms that holds by *canonical form* —  $w + w = 2 \cdot w$ , distributivity, collection of like monomials, exact  $\mathbb{Q}$  coefficient arithmetic. A producer uses it constantly and invisibly: whenever an emitted derivation needs two spellings of the same polynomial to be interchangeable (before an EQELIM, to massage a term into the pinned shape another rule's payload demands), a single NORM node discharges the rewriting that an equational-logic presentation would smear across many congruence steps.

What the checker recomputes: that both sides are idx-free (projection terms have no canonical form in the wire-keyed polynomial space — the comparison would not carry the rule's meaning), and that the two canonical polynomials are *identical* — coefficient-for-coefficient equality over  $\mathbb{Q}$ , never evaluation at test points, and never rescaling:  $2 \cdot w$  and  $w + w$  are equal as canonical forms;  $2 \cdot w$  and  $w$  are not, proportional though they are. The remaining payload fields are well-typedness, not computation: both sides share a defined broadcast shape and mention only finitely typed wires.

Each hypothesis is load-bearing. Drop the canonical-form identity and the rule asserts arbitrary equalities. Drop the shared-shape and finiteness fields and the rule is false *as staged*: at a wire of extended dtype, even  $\vdash t = t$  is invalid — the atom fails to hold at every valuation giving the wire an infinite entry, while holding at the rational ones — and since the premise is arbitrary, the staged rule concluded failing sequents from satisfiable premises; a cancelling sub-term  $w_2 - w_2$  over a wire whose shape does not join with the other side's leaves the same canonical form but an atom that never evaluates at any valuation — unsatisfiable outright. The typing fields are exactly what makes “equal canonical forms” imply “equal evaluated tensors” at every valuation.

**notes** The rule also names a convention the premise-free rules of this catalogue share (*the NORM convention*): the premise  $\Phi$  is arbitrary and never consulted — a fact that holds at every valuation holds under any premise. The catalogued stratum records an unpacking sketch through BERNONE and ARITHATOM; the shipped warrant is direct and stronger (one verified semantic lemma, all tensor shapes), with the well-typedness fields carrying what the sketch left implicit.

#### Lean statement

BernsteinLogic.Soundness . sound\_norm

statement bound

```
-- Norm, sound under the PROPOSED-B13-Norm fields: both sides evaluate
(bshape + finWires), at the same shape (`tyEq` + `evalQ_bshape`), with
per-entry values equal because the canonical polynomials are literally
equal. -/
theorem sound_norm {H : Hypernet} {seq : Sequent} {termL termR : Term}
  (hcanon : polynomialCanonicalEq termL termR)
  (htyEq : termL.bshape H = termR.bshape H)
  (htyDef : ∃ s, termL.bshape H = some s)
  (hfinL : termL.finWires H) (hfinR : termR.finWires H)
  (hpost : seq.post = .atom .eq termL termR) :
  seq.semanticallyValid H
```

**The whole computational content, displayed.** The check is two gates and one equality: both sides idx-free, and the canonical polynomials literally equal. Everything else the rule needs is well-typedness data carried in the payload and consumed by the soundness lemma, not by the checker.

BernsteinLogic.Core.Catalogue . Term.toPoly

statement bound



```

/-- Exact sparse-polynomial canonical form of a term, variables = wires.
The ONLY canonicalization in leg C (F7 discipline; see original). -/
def Term.toPoly : Term → QArith.Poly
| .var w => QArith.Poly.var w
| .const c => QArith.Poly.const c
| .add l r => QArith.Poly.add l.toPoly r.toPoly
| .mul l r => QArith.Poly.mul l.toPoly r.toPoly
| .sub l r => QArith.Poly.sub l.toPoly r.toPoly
| .neg t => QArith.Poly.neg t.toPoly
-- G1 INVARIANT (load-bearing): the wire-keyed variable space cannot
-- express per-entry projections, so idx has NO canonical form here --
-- the placeholder below is semantically inert because every consumer
-- of `toPoly` first requires `Term.idxFree` (the canonical-form
-- relations `polynomialCanonicalEq`/`farkasCheck` carry the guard).
-- idx terms enter polynomial land ONLY through `asCoordPolyIdx?`.
| .idx _ => QArith.Poly.const 0

/-- Ground-term evaluation: `none` iff a wire occurs – and on any idx
node: a well-typed idx (rank-1 argument) is never closed. -/
def Term.evalQ? : Term → Option Rat
| .var _ => none
| .const c => some c
| .add l r => do pure ((← l.evalQ?) + (← r.evalQ?))
| .mul l r => do pure ((← l.evalQ?) * (← r.evalQ?))
| .sub l r => do pure ((← l.evalQ?) - (← r.evalQ?))
| .neg t => do pure (-(← t.evalQ?))
| .idx _ => none

/-- Comparison evaluation over  $\mathbb{Q}$ . -/
def BowTie.evalQ : BowTie → Rat → Rat → Bool
| .lt, x, y => decide (x < y)
| .le, x, y => decide (x ≤ y)
| .eq, x, y => decide (x = y)
| .ge, x, y => decide (y ≤ x)
| .gt, x, y => decide (y < x)
| .ne, x, y => decide (¬ x = y)

/-- Closed-formula evaluation: `none` iff a wire or region atom occurs
(strict; see original for the closed-world reading). -/
def Formula.evalClosed : Formula → Option Bool
| .atom b l r => do pure (b.evalQ (← l.evalQ?) (← r.evalQ?))
| .inRegion _ _ => none
| .and l r => do pure ((← l.evalClosed) && (← r.evalClosed))
| .or l r => do pure ((← l.evalClosed) || (← r.evalClosed))
| .tt => some true
| .ff => some false

/-- The barycentric coordinate functions of a 1-dimensional simplex, as
canonical affine Terms in `t` (pinned shape `offset + slope·t`; see the
original for the full derivation notes and D2 scope). -/
def baryLambdas1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool)
(t : Term) : Option (List (Fin cx.nVertices × Term))

```

BernsteinLogic.Core.Catalogue . PNormData

statement bound

```

inductive PNormData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (termL termR : Term)
  (canonEq : polynomialCanonicalEq termL termR)
-- PROPOSED-B13-Norm (register B13 rider; realized provisionally
-- 2026-07-16 on the repair hint "asserting wires have finite
-- type" – the same field family that repaired EqElim/ConvIntro):
-- both sides are shape-stable at a common broadcast shape and
-- finitely typed. Without these, an  $\infty$ -valued ext wire makes even
--  $\vdash t = t$  unsatisfiable (pre is arbitrary), and a cancelling
-- wire of incompatible shape ( $w_2 - w_2$  at a shape unjoinable with

```

```

-- termL's) makes the conclusion atom never evaluate.
(tyEq   : termL.bshape H = termR.bshape H)
(tyDef  : ∃ s, termL.bshape H = some s)
(finL   : termL.finWires H)
(finR   : termR.finWires H)
(postEq : seq.post = .atom .eq termL termR)
: PNormData H seq

```

BernsteinLogic.Core.Catalogue . Term.idxFree

definition bound

```

/-- Structural absence of the idx constructor. The polynomial
canonical-form fragment (D2): `toPoly`'s wire-keyed variable space cannot
express per-entry projections, so every canonical-form consumer
(`polynomialCanonicalEq`, `farkasCheck`, the legacy `asCoordPoly?`
remap) requires this; idx coordinate polynomials take the
`asCoordPolyIdx?` route instead (G1). -/
def Term.idxFree : Term → Bool
| .var _ | .const _ => true
| .add l r | .mul l r | .sub l r => l.idxFree && r.idxFree
| .neg t => t.idxFree
| .idx _ _ => false

```

BernsteinLogic.Core.Catalogue . polynomialCanonicalEq

definition bound

```

def polynomialCanonicalEq (l r : Term) : Prop :=
  l.idxFree = true ∧ r.idxFree = true ∧ l.toPoly = r.toPoly

```

## Correspondence (Steele display ↔ Lean)

reviewer-certified against the displayed texts

The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax.

- $t_1, t_2 \mapsto \text{termL termR} : \text{Term}$ ; the displayed side condition (equal monomial canonical forms)  $\mapsto \text{hcanon} : \text{polynomialCanonicalEq termL termR}$ , quoted in full below — both sides idx-free and the canonical polynomials literally equal. The display's "same variables" and "polynomial terms" have no separate binders: canonical forms are wire-keyed (equality forces the same canonical support — a mentioned-but-cancelled wire is no constraint on the polynomial; evaluability polices it through the typing fields instead) and the term grammar is polynomial by construction.
- conclusion  $\Phi \vdash t_1 = t_2 \mapsto$  the goal under  $\text{hpost} : \text{seq.post} = \text{.atom .eq termL termR}$ ; the arbitrary  $\Phi$  is premise-freeness by silence — the precondition component of  $\text{seq} : \text{Sequent}$  is unconstrained by any hypothesis (the  $\text{PConstExact}$  representation).

Width note: the display carries the typing pack inline, matching the mechanized statement's four well-typedness hypotheses one for one —  $\text{htyEq}$  (both broadcast shapes equal),  $\text{htyDef}$  (and defined — the some),  $\text{hfinL}/\text{hfinR}$  (all free wires at finite dtypes). The bound  $\text{PNormData}$  payload carries the same conditions under its own field names —  $\text{hcanon} \leftrightarrow \text{canonEq}$ ,  $\text{htyEq} \leftrightarrow \text{tyEq}$ ,  $\text{htyDef} \leftrightarrow \text{tyDef}$ ,  $\text{hfinL}/\text{hfinR} \leftrightarrow \text{finL}/\text{finR}$ ,  $\text{hpost} \leftrightarrow \text{postEq}$  — and they are not optional: without them the bare canonical-form rule is refuted (an  $\infty$ -valued wire makes even  $t = t$  fail at every extended-valued assignment — the roster row's counterexample). The one-line canonical-form sentence is the rule's *intent*; the certified statement is that sentence plus exactly this typing pack.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: fixture NormOutsideFragment the three-implementation comparator harness (compiled machine code, kernel-reduction probes, and the compiler-IR interpreter, all executing the same elaborated checker functions) carries a well-typed single NORM node as its fixture NormOutsideFragment, expected verdicts check = true, v0frag = false. The fixture is the suite's false-verdict control: NORM sits outside the restricted fragment recognizer by design, so the fixture pins the reject lane end to end — a comparator that could only confirm acceptance would be blind to a path that wrongly accepts everything.

## HOL4 statement

theory bern\_catalogue, HOL4 development script bern\_catalogueScript.sml script bound

#### Definition polynomialCanonicalEqB\_def

```
Definition polynomialCanonicalEqB_def:
  polynomialCanonicalEqB l r ↔
    termIdxFree l ∧ termIdxFree r ∧ termToPoly l = termToPoly r
End
```

## HOL verifier receipts

documentary hol-strict CLEAN over the ported checker chain: theory bern\_catalogue, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, scripts sha-matched verbatim to the sources cited here (git blob e41b9ed0...). The ported receipt table pins the rule's exactness discipline in both directions: the canonical-form accept ( $w_0 + w_0 = 2 \cdot w_0$ ) and the scale-preserving reject ( $2 \cdot w_0 \neq w_0$  as canonical forms). Scope note, honest by the port's own seam inventory: the ported surface is the first-order check function; the payload's well-typedness fields (broadcast shapes, finite typing) live in the dependently-typed proof-object layer, which is a separately tracked porting stage. Semantic stratum, also ported: the soundness twin sound\_norm (theory bern\_tr\_fragclose: hol-strict clean, 26 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (checkPf\_sound\_frag, theory bern\_tr\_splce, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. The tier taxonomy defines tier 2 as *verified unpacking into core rules* (one node → many), with rules admissible by *reduction to verified machinery* as the recorded boundary case; NORM is of the latter kind, and this section — the book's first admissible rule — states the general template that the later recipes instantiate.

**Recipe R16.1** (the admissible template, general form). An admissible rule  $R$  with premises  $P_1, \dots, P_n$  and conclusion  $C$  is warranted by a *rewrite*: a function taking any proof-tree node

$$\frac{\mathcal{T}_1 \triangleright P_1 \quad \cdots \quad \mathcal{T}_n \triangleright P_n}{C} R \quad \rightsquigarrow \quad \mathcal{D}[\mathcal{T}_1, \dots, \mathcal{T}_n] \triangleright C$$

where  $\mathcal{D}$  is a derivation fragment over core (and previously-justified admissible) rules with the same conclusion, consuming the premise subtrees  $\mathcal{T}_i$  at its leaves. Where such a fragment exists, applying it leaf-first eliminates every  $R$ -node from a finite tree in

finitely many steps, so any judgment derivable with  $R$  is derivable without it — admissibility in the derivability sense. In the tier’s recorded boundary case — rules admissible *by reduction to verified machinery*, this one among them — no fragment is applied: where no catalogue derivation of  $C$  can exist, the node is instead discharged in place against a named verified theorem, no derivability claim is made, and the conserved property — weaker, and sufficient — is the calculus’s soundness theorem itself. Either way, no new kernel semantic content enters. The verified object shipped per admissible rule is a soundness lemma for  $R$  that factors through already-verified machinery: for a fragment-backed rule, provable by replaying  $\mathcal{D}$ ; for a boundary-case rule, the discharge theorem itself.

**Recipe R16.2** (the NORM instance — the degenerate rewrite). NORM has no proof premises ( $n = 0$ ): the node

$$\frac{\text{polynomialCanonicalEq}(t_1, t_2) \quad \text{both sides shape-stable and finitely typed}}{\Phi \vdash t_1 = t_2} \text{NORM} \rightsquigarrow \text{the checker replays the canonical forms; sound\_no}$$

The side condition is *computational*: both sides idx-free, and their canonical monomial polynomials *identical* over  $\mathbb{Q}$  — coefficient-for-coefficient, decidable, replayed by the checker. The catalogued stratum records an unpacking sketch through `BERNCONE` and `ARITHATOM` (an equality as a pair of zero-polynomial bounds); the shipped warrant is direct and stronger — one verified semantic lemma at all tensor shapes — with the payload’s well-typedness fields carrying exactly what the sketch left implicit.

**Proposition R16.3** (semantic warrant). *Suppose  $t_1, t_2$  are idx-free with  $t_1.\text{toPoly} = t_2.\text{toPoly}$ , the two sides carry the same defined broadcast shape in  $H$ , and every wire of either side is finitely typed. Then  $\Phi \vdash t_1 = t_2$  is semantically valid over  $H$  for arbitrary  $\Phi$ .*

*Proof.* Fix any admissible valuation  $\sigma$ ; the premise is not consulted — this is *the NORM convention*, shared by every premise-free rule of the catalogue: a fact holding at every valuation holds under any premise. The shape and finiteness fields give both sides defined evaluations, as tensors of the common broadcast shape  $s$ : a term’s broadcast shape is computed structurally, so its definedness already certifies that every subterm’s join exists; finitely typed wires always read; and idx-freedom leaves no projection to fail. If  $s$  has positive rank and zero total size, both sides evaluate to the unique empty tensor of shape  $s$  and the equality atom holds outright — there is no entry to compare. Otherwise fix an entry index of  $s$  (for  $s = []$ , the single scalar position) and read each wire at it: a rank-0 wire contributes its scalar, and a positive-rank wire — necessarily of shape  $s$  itself, the only positive rank that joins with  $s$  under this broadcast discipline — contributes its entry at the index. Evaluation at that read-out is a ring homomorphism into  $\mathbb{Q}$  from the polynomial ring over the union of the wires the two sides mention (a wire absent from one side, or cancelled from its canonical form, simply carries coefficient zero there); a syntactic identity of canonical forms is an identity of polynomials, and ring homomorphisms preserve polynomial identities — so the two evaluated tensors agree entry by entry, and their shapes agree by hypothesis. Hence the equality atom holds at  $\sigma$ . That is the whole proof; `sound_norm` packages it once, per-entry, at every tensor shape.  $\square$

**Remark R16.4** (what broke without the typing fields). As first staged — canonical-form equality alone — the rule was *false*, twice over, and the counterexamples are why the fields exist. (i) At a wire of extended dtype, the staged rule concluded *invalid* sequents: at any valuation giving the wire an infinite entry, even  $t = t$  fails to hold — evaluation is over exact rationals, and an infinite entry refuses — so  $\vdash t = t$  fails exactly at the extended-valued assignments while holding at the rational ones. (ii) A cancelling sub-term leaves the canonical form unchanged but not the evaluation:  $t_2 = t_1 + (w_2 - w_2)$  has  $t_2.\text{toPoly} = t_1.\text{toPoly}$ , yet if  $w_2$ ’s shape does not join with  $t_1$ ’s, the right-hand atom side never evaluates at *any* valuation — there the concluded atom is unsatisfiable outright. In either case a satisfiable (indeed arbitrary) premise concluded a failing sequent; the shared-shape and finiteness fields exclude both. A hypothesis whose necessity is semantic (evaluability), not algebraic, is exactly the kind a recipe must state, or the next porter re-learns it the hard way.

**Remark R16.5** (exactness discipline, and the idx-freedom gate). Canonical-form equality is never up-to-rescaling:  $2 \cdot w$  and  $w + w$  are equal as canonical forms;  $2 \cdot w$  and  $w$  are not, proportional though they are — the matching discipline every combination rule of this catalogue (`FARKAS`, `SZYGY`) inherits when it compares atom polynomials. The idx-freedom gate is the meaning-preservation condition of the translation: the wire-keyed polynomial space cannot express per-entry projections, so a projection-bearing term has no canonical form here and the comparison is refused rather than performed on the wrong object. (The combination rules’ widened replays translate projections into their own variable spaces; those translations belong to those rules, not to NORM.)

#### proof step

Recipe **R16.2** warrant  
the computational check  
idx-freedom gate (Remark **R16.5**)  
typing fields (Remark **R16.4**)  
premise-freedom ( $\Phi$  arbitrary)  
wire placement

#### formal anchor

`sound_norm` (per-entry evaluation at the common broadcast shape)  
`polynomialCanonicalEq` = idx-freedom gates + `Term.toPoly` equality  
`Term.idxFree`; `Term.toPoly`’s wire-keyed variable space  
payload fields `tyEq`, `tyDef`, `finL`, `finR` (`PNormData`)  
the NORM convention; pre unused in `sound_norm`  
`RuleTag.norm`

## Rule 17. FARKAS (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Farkas*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} \{\ell_j \bowtie_j 0\}_j \text{ a sub-conjunction of } \Phi \text{ with each } \ell_j \text{ linear, } \bowtie_j \in \{\leq, =\} \\ \{\lambda_j \in \mathbb{Q}_{\geq 0}\}_j \text{ nonneg coefficients, with } \lambda_j \text{ free when } \bowtie_j \text{ is } = \\ \text{wires}(\sum_j \lambda_j \ell_j) \text{ all finitely typed in } H \end{array}}{\Phi \vdash \sum_j \lambda_j \ell_j \leq 0} \text{ FARKAS}$$

FARKAS is the affine-combination rule over the premise: from atoms  $\{\ell_j \bowtie_j 0\}$  occurring as sub-conjuncts of  $\Phi$  and nonnegative multipliers  $\lambda_j$  (free-signed on equality atoms), it concludes  $\Phi \vdash \sum_j \lambda_j \ell_j \leq 0$ . It is the one-rule form of what SYZVGV does with sub-*proofs*: here every combined atom must already sit in the premise conjunction, so no component derivations enter. A producer finds the multipliers by whatever method it likes (an LP feasibility call, typically) and ships only the combination term; nothing about the search survives into the certificate.

What the checker recomputes, given the payload (combination term, conclusion term): decode the combination into weighted entries — a fixed canonical encoding (a right-nested sum of  $\lambda_j \cdot e_j$  products with a zero terminator), anything else rejected; require each weight nonnegative; require each entry's polynomial to occur among the  $(\cdot \leq 0)$ -normal polynomials of a premise conjunct — equality atoms expose *both* polarities to the matcher, which is exactly how a free-sign multiplier on an equality is realized as a nonnegative weight on the appropriate polarity; and require the weighted fold of the entry polynomials to equal the conclusion's polynomial *identically*, as canonical forms over  $\mathbb{Q}$ . The check is a disjunction of three branches of one replay algorithm, differing only in the variable space where polynomials are compared: the legacy idx-free canonical-form branch, a wire-aware coordinate branch for projection-indexed terms, and a mixed tagged-union branch for combinations mentioning both bare wires and projections.

Each hypothesis is load-bearing. Drop the sign condition and a producer multiplies a bound by  $-1$  and concludes its negation. Drop the atom-sourcing check and the combination smuggles in atoms nobody assumed. Drop the exact fold identity and the conclusion is unrelated to the premise. Drop the positive-size wire condition and the rule is unsound at degenerate shapes: comparison atoms over a zero-size (empty-tensor) wire hold *vacuously*, and a combination of two vacuous contradictory atoms —  $5 \leq w$  and  $w \leq 3$  at a wire of shape  $[0]$  — derives the false scalar conclusion  $\vdash 2 \leq 0$  from a satisfiable premise. Drop the conclusion's shape/finiteness fields and a conclusion mentioning an untyped wire has a correct canonical polynomial but never evaluates, so the concluded atom never holds. The proof section walks these in order.

**notes** The catalogued statement ranges over *linear*  $\ell_j$ ; the replay itself is degree-blind, and soundness never consults linearity — the restriction is the rule's published register (the shape producers emit and the wire format prices), not a checked side condition. SYZVGV lifts the same fold to component conclusions with degrees unrestricted; PBOUND re-spells that lift as the compact affine bound-node.

### Lean statement

BernsteinLogic.Soundness . **farkas\_fold\_eval\_nonpos**

statement bound

```
-- Evaluation of the Farkas combination fold: a nonnegative combination
of nonpositive values is nonpositive (valuation-general). -/
theorem farkas_fold_eval_nonpos (p : Nat → ℚ) :
  ∀ (entries : List (ℚ × Term)),
  (∀ e ∈ entries, 0 ≤ e.1 ∧ QArith.Poly.eval p e.2.toPoly ≤ 0) →
  QArith.Poly.eval p
    (entries.foldr
      (fun (le : ℚ × Term) acc =>
        QArith.Poly.add (QArith.Poly.scale le.1 le.2.toPoly) acc) []) ≤ 0
```

BernsteinLogic.Soundness . **farkas\_fold\_eval\_nonpos\_polys**

statement bound

```
-- The nonneg fold in the paired space (valuation-general; entries
carry their polynomials directly – the `farkasEntriesW`-translated
form). -/
theorem farkas_fold_eval_nonpos_polys (p : Nat → ℚ) :
  ∀ (pentries : List (ℚ × QArith.Poly)),
  (∀ e ∈ pentries, 0 ≤ e.1 ∧ QArith.Poly.eval p e.2 ≤ 0) →
```

```

QArith.Poly.eval p
  (pentries.foldr
    (fun (le : Q × QArith.Poly) acc =>
      QArith.Poly.add (QArith.Poly.scale le.1 le.2) acc) []) ≤ 0

```

BernsteinLogic.Soundness . **sound\_farkas**

statement bound

```

/-- Farkas, general tensor shapes (supersedes the v0 singleton form):
the conclusion's B13 shape/finiteness fields give it a defined
evaluation; the PROPOSED-B14 positivity field rules out vacuously-held
zero-size atoms; the per-entry claim runs through the CLAMPED valuation,
which serves heterogeneous premise-atom shapes at every conclusion
index. -/
theorem sound_farkas {H : Hypernet} {seq : Sequent}
  {combinationCoeffs conclusion : Term}
  (hshape : ∃ s, conclusion.bshape H = some s)
  (hfin : conclusion.finWires H)
  (hwpos : ∀ w ∈ seq.pre.freeWires ++ conclusion.freeWires,
    ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape)
  (hvalid : farkasCombinationValid combinationCoeffs seq.pre conclusion)
  (hpost : seq.post = .atom .le conclusion (.const 0)) :
  seq.semanticallyValid H

```

**What the combination check means — and what it does not.** The side condition appears in the payload as the single name `farkasCombinationValid`, so this book binds its meaning by displaying the decoding chain from the checker sources. `farkasEntries` decodes the combination term against a *pinned* canonical encoding — anything else is rejected, so the wire format cannot express a malformed combination. `atomLeZeroPolys` is the  $(\cdot \leq 0)$ -normalization of one premise atom, exact and never rescaled; note that equality atoms contribute both polarities, which is the entire realization of the classical free-sign multiplier discipline for equality constraints. `farkasCheckCore` strings the gates together in the `idx`-free canonical-form space; `farkasCheck` is the three-way branch disjunction — the two widened branches replay the same gates verbatim in the wire-aware coordinate space (projection terms  $w[j]$  keyed by an injective pairing of wire and coordinate, so distinct wires can never collide) and in the mixed tagged-union space (bare wires and projections as disjoint variable classes; the variant identifying a bare wire with its zeroth projection admits a false sequent and is deliberately unrepresentable).

What a skeptical reader should confirm by eye: the conclusion term enters only as the target of the exact fold identity; the step's posted formula appears nowhere in the check; and nothing in the chain evaluates or consults the truth of anything — the check manipulates polynomials, never valuations. Semantic truth is established only in `sound_farkas`, which case-splits on the accepted branch.

BernsteinLogic.Core.Catalogue . **atomLeZeroPolysW**

statement bound

```

/-- `atomLeZeroPolys` in the paired coordinate space: the
`(e ≤ 0)`-normal-form polynomials of one atom under the wire-aware idx
translation. An atom with an untranslatable side entails nothing (like
non-atoms – it can then never source a combination entry). -/
def atomLeZeroPolysW : Formula → List QArith.Poly
| .atom op l r =>
  match l.asCoordPolyIdxW? with
  | none => []
  | some pl =>
    match r.asCoordPolyIdxW? with
    | none => []
    | some pr =>
      match op with
      | .le | .lt => [QArith.Poly.sub pl pr]
      | .ge | .gt => [QArith.Poly.sub pr pl]
      | .eq => [QArith.Poly.sub pl pr, QArith.Poly.sub pr pl]
      | .ne => []
  | _ => []

/-- Translate every decoded entry's vehicle term; `none` if any vehicle
is outside the idx coordinate fragment. (Explicit recursion rather than
`mapM` so soundness inverts it by plain `cases`.) -/
def farkasEntriesW : List (Rat × Term) → Option (List (Rat × QArith.Poly))
| [] => some []
| (l, e) :: rest =>
  match e.asCoordPolyIdxW?, farkasEntriesW rest with

```

```

    | some pe, some prest => some ((l, pe) :: prest)
    | _, _ => none

```

```

/-- The idx-form branch of the Farkas check: every polynomial – entry
vehicles, matched premise atoms, and the conclusion – rides the
wire-aware coordinate translation; entry encoding, nonneg weights,
atom matching, and the exact combination fold mirror the idx-free
branch verbatim. -/
def farkasCheckW (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

BernsteinLogic.Core.Catalogue . **atomLeZeroPolysM**

statement bound

```

/-- `atomLeZeroPolys` in the mixed tagged space: the `(e ≤ 0)`-normal-
form polynomials of one atom under the mixed translation. An atom with
an untranslatable side entails nothing (the W-branch discipline). -/
def atomLeZeroPolysM : Formula → List QArith.Poly
| .atom op l r =>
  match l.asCoordPolyMixed? with
  | none => []
  | some pl =>
    match r.asCoordPolyMixed? with
    | none => []
    | some pr =>
      match op with
      | .le | .lt => [QArith.Poly.sub pl pr]
      | .ge | .gt => [QArith.Poly.sub pr pl]
      | .eq => [QArith.Poly.sub pl pr, QArith.Poly.sub pr pl]
      | .ne => []
| _ => []

/-- Translate every decoded entry's vehicle term into the mixed space;
`none` if any vehicle is outside the mixed fragment. -/
def farkasEntriesM : List (Rat × Term) → Option (List (Rat × QArith.Poly))
| [] => some []
| (l, e) :: rest =>
  match e.asCoordPolyMixed?, farkasEntriesM rest with
  | some pe, some prest => some ((l, pe) :: prest)
  | _, _ => none

/-- The mixed-variable branch of the Farkas check: every polynomial –
entry vehicles, matched premise atoms, and the conclusion – rides the
mixed tagged translation; entry encoding, nonneg weights, atom matching,
and the exact combination fold mirror the sibling branches verbatim. -/
def farkasCheckM (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

BernsteinLogic.Core.Catalogue . **farkasCheckW**

statement bound

```

/-- The idx-form branch of the Farkas check: every polynomial – entry
vehicles, matched premise atoms, and the conclusion – rides the
wire-aware coordinate translation; entry encoding, nonneg weights,
atom matching, and the exact combination fold mirror the idx-free
branch verbatim. -/
def farkasCheckW (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

BernsteinLogic.Core.Catalogue . **farkasCheckM**

statement bound

```

/-- The mixed-variable branch of the Farkas check: every polynomial –
entry vehicles, matched premise atoms, and the conclusion – rides the
mixed tagged translation; entry encoding, nonneg weights, atom matching,
and the exact combination fold mirror the sibling branches verbatim. -/
def farkasCheckM (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

BernsteinLogic.Core.Catalogue . **farkasCombinationValid**

statement bound

```

def farkasCombinationValid (cc : Term) (premise : Formula)
  (conclusion : Term) : Prop

```



BernsteinLogic.Core.Catalogue . **wireCoordKey**

statement bound

```

/-- Injective key for the (wire, coordinate) variable space of the
multi-wire idx coordinate translation below. This is `Nat.pair`'s
definition inlined (the core is mathlib-free; Soundness.lean proves the
`Nat.unpair` round-trip through the definitional equality). Distinct
wires MUST get distinct polynomial variables here: a Syzygy composite
mixes projections of several region wires in one combination, and
conflating `w0[j]` with `w2[j]` would let semantically different
polynomials collide (contrast `asCoordPolyIdx?`, whose BernCone scope
has a single region wire pinned by `polyTie`, so dropping the wire is
sound THERE and only there). -/
def wireCoordKey (w i : Nat) : Nat

```

BernsteinLogic.Core.Catalogue . **Term.asCoordPolyIdxW?**

statement bound

```

/-- The wire-aware idx coordinate translation: a Term whose every wire
mention is a projection `.idx (.var w) j` becomes a polynomial over the
paired variables `wireCoordKey w j`. `none` on bare wire mentions and
on idx of any non-var argument (v0.5 scope, matching
`asCoordPolyIdx?`). Constants translate, so idx-free closed terms ride
either route. -/
def Term.asCoordPolyIdxW? : Term → Option QArith.Poly
| .var _ => none
| .const c => some (QArith.Poly.const c)
| .add l r =>
  do pure (QArith.Poly.add (← l.asCoordPolyIdxW?) (← r.asCoordPolyIdxW?))
| .mul l r =>
  do pure (QArith.Poly.mul (← l.asCoordPolyIdxW?) (← r.asCoordPolyIdxW?))
| .sub l r =>
  do pure (QArith.Poly.sub (← l.asCoordPolyIdxW?) (← r.asCoordPolyIdxW?))
| .neg t => (Term.asCoordPolyIdxW? t).map QArith.Poly.neg
| .idx (.var w) i => some (QArith.Poly.var (wireCoordKey w i))
| .idx _ _ => none

/-- `atomLeZeroPolys` in the paired coordinate space: the
`(e ≤ 0)`-normal-form polynomials of one atom under the wire-aware idx
translation. An atom with an untranslatable side entails nothing (like
non-atoms – it can then never source a combination entry). -/
def atomLeZeroPolysW : Formula → List QArith.Poly
| .atom op l r =>
  match l.asCoordPolyIdxW? with
  | none => []
  | some pl =>
    match r.asCoordPolyIdxW? with
    | none => []
    | some pr =>
      match op with
      | .le | .lt => [QArith.Poly.sub pl pr]
      | .ge | .gt => [QArith.Poly.sub pr pl]
      | .eq => [QArith.Poly.sub pl pr, QArith.Poly.sub pr pl]
      | .ne => []
| _ => []

/-- Translate every decoded entry's vehicle term; `none` if any vehicle
is outside the idx coordinate fragment. (Explicit recursion rather than
`mapM` so soundness inverts it by plain `cases`.) -/
def farkasEntriesW : List (Rat × Term) → Option (List (Rat × QArith.Poly))
| [] => some []
| (l, e) :: rest =>
  match e.asCoordPolyIdxW?, farkasEntriesW rest with
  | some pe, some prest => some ((l, pe) :: prest)
  | _, _ => none

/-- The idx-form branch of the Farkas check: every polynomial – entry
vehicles, matched premise atoms, and the conclusion – rides the

```

```

wire-aware coordinate translation; entry encoding, nonneg weights,
atom matching, and the exact combination fold mirror the idx-free
branch verbatim. -/
def farkasCheckW (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

BernsteinLogic.Core.Catalogue . **Term.asCoordPolyMixed?**

statement bound

```

/-- The mixed-variable translation: bare wires and idx projections are
DISJOINT opaque variables of one combination algebra – bare `var w`
keyed even `(2·w)`, projections `idx (.var w) j` keyed odd
`(2·wireCoordKey w j + 1)` – so the two classes can never collide. In
particular bare `w` is NEVER identified with `w[0]`: the identification
variant admits a false sequent (a countermodel witnesses the failure,
alongside the REJECT specimen in EvalDemo). This matches the python reference
(`bl_checker` keys bare and projected mentions as distinct symbols) and
the geolog/fastcheck mixed evaluators. `none` only on idx-of-non-var
(outside every reading's fragment). -/
def Term.asCoordPolyMixed? : Term → Option QArith.Poly
| .var w => some (QArith.Poly.var (2 * w))
| .const c => some (QArith.Poly.const c)
| .add l r =>
  do pure (QArith.Poly.add (← l.asCoordPolyMixed?) (← r.asCoordPolyMixed?))
| .mul l r =>
  do pure (QArith.Poly.mul (← l.asCoordPolyMixed?) (← r.asCoordPolyMixed?))
| .sub l r =>
  do pure (QArith.Poly.sub (← l.asCoordPolyMixed?) (← r.asCoordPolyMixed?))
| .neg t => (Term.asCoordPolyMixed? t).map QArith.Poly.neg
| .idx (.var w) j => some (QArith.Poly.var (2 * wireCoordKey w j + 1))
| .idx _ _ => none

/-- `atomLeZeroPolys` in the mixed tagged space: the `(e ≤ 0)`-normal-
form polynomials of one atom under the mixed translation. An atom with
an untranslatable side entails nothing (the W-branch discipline). -/
def atomLeZeroPolysM : Formula → List QArith.Poly
| .atom op l r =>
  match l.asCoordPolyMixed? with
  | none => []
  | some pl =>
    match r.asCoordPolyMixed? with
    | none => []
    | some pr =>
      match op with
      | .le | .lt => [QArith.Poly.sub pl pr]
      | .ge | .gt => [QArith.Poly.sub pr pl]
      | .eq => [QArith.Poly.sub pl pr, QArith.Poly.sub pr pl]
      | .ne => []
| _ => []

/-- Translate every decoded entry's vehicle term into the mixed space;
`none` if any vehicle is outside the mixed fragment. -/
def farkasEntriesM : List (Rat × Term) → Option (List (Rat × QArith.Poly))
| [] => some []
| (l, e) :: rest =>
  match e.asCoordPolyMixed?, farkasEntriesM rest with
  | some pe, some prest => some ((l, pe) :: prest)
  | _, _ => none

/-- The mixed-variable branch of the Farkas check: every polynomial –
entry vehicles, matched premise atoms, and the conclusion – rides the
mixed tagged translation; entry encoding, nonneg weights, atom matching,
and the exact combination fold mirror the sibling branches verbatim. -/
def farkasCheckM (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

BernsteinLogic.Core.Catalogue . **PFarkasData**

statement bound

```

inductive PFarkasData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)

```

```

(combinationCoeffs : Term)
(conclusion : Term)
-- The per-witness-node denominator bound,  $\log_2$  – claim semantics
--  $\text{den}(q) \leq 2^{\text{denBoundLog2}}$  over this node's witness rationals.
-- ZERO proof-theoretic weight in v1 (no applicability axiom
-- consumes it). Wire: u64 den_bound_log2 on tag 14 (BINARY.md §4).
(denBoundLog2 : Nat)
(conclShape :  $\exists s, \text{conclusion.bshape } H = \text{some } s$ )
(conclFin : conclusion.finWires H)
-- PROPOSED-B14 (approval-gated; realized provisionally per the
-- B10/B13 pattern, 2026-07-16): every premise and conclusion wire
-- is typed at a positive-size shape. Comparison atoms over
-- zero-size (empty-tensor) wires hold vacuously, and a Farkas
-- combination of two vacuous contradictory atoms – e.g.  $5 \leq w$  and
--  $w \leq 3$  at  $w : \{[0], Q\}$  – derives the false scalar conclusion
--  $\vdash 2 \leq 0$  from a satisfiable premise. (The v0 singleton wire
-- discipline excluded this by accident of strength; this is the
-- honest general-shape guard.)
(wiresPos :  $\forall w \in \text{seq.pre.freeWires} ++ \text{conclusion.freeWires},$ 
 $\exists \tau, H.\text{wireType } w = \text{some } \tau \wedge$ 
 $0 < \text{Shape.totalSize } \tau.\text{shape}$ )
(valid : farkasCombinationValid combinationCoeffs seq.pre conclusion)
(postEq : seq.post = .atom .le conclusion (.const 0))
: PFarkasData H seq

```

#### BernsteinLogic.Core.Catalogue . Formula.conjunctList

definition bound

```

/-- Flatten a conjunction tree into its leaf list (left-to-right). -/
def Formula.conjunctList : Formula → List Formula
| .and l r => l.conjunctList ++ r.conjunctList
| f => [f]

```

#### BernsteinLogic.Core.Catalogue . atomLeZeroPolys

definition bound

```

/-- The  $(e \leq 0)$ -normal-form polynomials entailed by one premise atom
(exact polynomial forms – never rescaled; F7 discipline). -/
def atomLeZeroPolys : Formula → List QArith.Poly
| .atom .le l r => [QArith.Poly.sub l.toPoly r.toPoly]
| .atom .lt l r => [QArith.Poly.sub l.toPoly r.toPoly]
| .atom .ge l r => [QArith.Poly.sub r.toPoly l.toPoly]
| .atom .gt l r => [QArith.Poly.sub r.toPoly l.toPoly]
| .atom .eq l r => [QArith.Poly.sub l.toPoly r.toPoly,
QArith.Poly.sub r.toPoly l.toPoly]
| _ => []

```

#### BernsteinLogic.Core.Catalogue . farkasEntries

definition bound

```

/-- Decode the combination-coefficient Term (pinned canonical encoding:
right-nested  $\text{.add}$  chain of  $\text{.mul } (\text{.const } \lambda_j) e_j$ ,  $\text{.const } 0$  end). -/
def farkasEntries : Term → Option (List (Rat × Term))
| .const c => if c = 0 then some [] else none
| .add (.mul (.const l) e) rest =>
(farkasEntries rest).map ((l, e) :: ·)
| _ => none

```

#### BernsteinLogic.Core.Catalogue . farkasCheckCore

definition bound

```

def farkasCheckCore (cc : Term) (premise : Formula) (conclusion : Term) : Bool :=
match farkasEntries cc with
| none => false
| some entries =>
let atoms := premise.conjunctList
conclusion.idxFree &&
entries.all (fun (l, e) =>
decide (0 ≤ l) &&
atoms.any fun a =>

```

```

a.atomSidesIdxFree &&
(atomLeZeroPolys a).any fun pa => decide (pa = e.toPoly)) &&
decide ((entries.foldr
  (fun (l, e) acc => QArith.Poly.add (QArith.Poly.scale l e.toPoly) acc)
  []) = conclusion.toPoly)

```

BernsteinLogic.Core.Catalogue . **farkasCheck**

definition bound

```

/-- The Farkas check: the legacy idx-free branch (exact `toPoly`
replay), OR the wire-aware idx coordinate branch (`farkasCheckW`),
OR the mixed tagged branch (`farkasCheckM`). Soundness
case-splits on the disjuncts; each branch carries its own argument
(`sound_farkas`). -/
def farkasCheck (cc : Term) (premise : Formula) (conclusion : Term) : Bool :=
  farkasCheckCore cc premise conclusion || farkasCheckW cc premise conclusion ||
  farkasCheckM cc premise conclusion

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax. Distinctively, there is no sub-derivation hypothesis at all: the displayed premise row describes the syntactic conjunct structure of  $\Phi$  itself, so the statement consumes `seq.pre` directly and no `ih` exists.

- the combination  $\sum_j \lambda_j \ell_j \mapsto$  the binder conclusion : Term; the multipliers  $\lambda_j \mapsto$  encoded in `combinationCoeffs` : Term under the pinned canonical encoding the intro describes.
- the displayed premise row — sub-conjunction of  $\Phi$ , each  $\ell_j$  linear,  $\bowtie_j \in \{\leq, =\}$ , multipliers nonnegative and free exactly on equalities, and the identity of conclusion with the combination  $\mapsto$  one hypothesis, `hvalid` : `farkasCombinationValid combinationCoeffs seq.pre conclusion`, whose meaning this book binds by displaying the decoding chain in full (absorbed hypotheses, the PCONSTEXACT pattern: a successful check certifies the structure; the free-sign discipline is realized as both-polarity contribution of equality atoms). The displayed row is the rule's published register; the realized matcher is wider — see the width note below.
- the displayed typing premises  $\text{bshape}_H(\sum \lambda_j \ell_j) \neq \perp$  and finite typing  $\mapsto$  `hshape` (existence of a defined broadcast shape — the some) and `hfin` : `conclusion.finWires H`.
- conclusion  $\Phi \vdash \sum_j \lambda_j \ell_j \leq 0 \mapsto$  the goal under `hpost` : `seq.post = .atom .le conclusion (.const 0)`.

Width notes. On the side-condition surface the statement and the display agree: `hwpos` — every free wire of `seq.pre` and of the conclusion typed at a shape of strictly positive total size — appears inline as the display's positive-size row, and it is not optional: without it, zero-size wires make atoms vacuously true and a combination of two vacuous atoms derives a false scalar bound (the zero-size counterexample beside the warrant proof). One direction is genuinely *wider*: the matcher behind `hvalid` normalizes arbitrary two-sided comparison atoms over all five admitted comparators (strict forms contribute their non-strict normal polynomials; equality atoms both polarities) and the replay is degree-blind — the displayed linear zero-form  $\{\leq, =\}$  row is the rule's published register (the note above), not the checked boundary. The displayed cell is the rule's intent; the certified statement is that intent with the register widened exactly as stated.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_catalogue, HOL4 development script bern\_catalogueScript.sml script bound

#### Definition farkasEntries\_def

```

Definition farkasEntries_def:
  farkasEntries (TmConst c) = (if c = 0 then SOME [] else NONE) ∧
  farkasEntries (TmAdd (TmMul (TmConst l) e) rest) =
    (case farkasEntries rest of
     | NONE => NONE
     | SOME es => SOME ((l,e)::es)) ∧
  farkasEntries _ = NONE
End

```

#### Definition atomLeZeroPolys\_def

```

Definition atomLeZeroPolys_def:
  atomLeZeroPolys (FAtom BowLe l r) = [polySub (termToPoly l) (termToPoly r)] ∧
  atomLeZeroPolys (FAtom BowLt l r) = [polySub (termToPoly l) (termToPoly r)] ∧
  atomLeZeroPolys (FAtom BowGe l r) = [polySub (termToPoly r) (termToPoly l)] ∧
  atomLeZeroPolys (FAtom BowGt l r) = [polySub (termToPoly r) (termToPoly l)] ∧
  atomLeZeroPolys (FAtom BowEq l r) = [polySub (termToPoly l) (termToPoly r);
                                         polySub (termToPoly r) (termToPoly l)] ∧
  atomLeZeroPolys _ = []
End

```

#### Definition farkasCheckCore\_def

```

Definition farkasCheckCore_def:
  farkasCheckCore cc premise conclusion =
    case farkasEntries cc of
    | NONE => F
    | SOME entries =>
      (let atoms = conjunctList premise in
       termIdxFree conclusion ∧
       EVERY (λ(l,e).
         0 ≤ l ∧
         EXISTS (λa. atomSidesIdxFree a ∧
           EXISTS (λpa. pa = termToPoly e) (atomLeZeroPolys a))
         atoms) entries ∧
       (FOLDR (λ(l,e) acc. polyAdd (polyScale l (termToPoly e)) acc)
        [] entries =
        termToPoly conclusion))
End

```

**Definition farkasCheckW\_def**

```

Definition farkasCheckW_def:
  farkasCheckW cc premise conclusion =
    case farkasEntries cc of NONE => F | SOME entries =>
      case farkasEntriesW entries of NONE => F | SOME pentries =>
        case asCoordPolyIdxW conclusion of NONE => F | SOME pc =>
          (let atoms = conjunctList premise in
            EVERY (λ(l,pe).
              0 ≤ l ∧
              EXISTS (λa. EXISTS (λpa. pa = pe) (atomLeZeroPolysW a)) atoms)
              pentries ∧
            (FOLDR (λ(l,pe) acc. polyAdd (polyScale l pe) acc) [] pentries =
              pc))
  End

```

**Definition farkasCheck\_def**

```

Definition farkasCheck_def:
  farkasCheck cc premise conclusion ↔
    farkasCheckCore cc premise conclusion v
    farkasCheckW cc premise conclusion
  End

```

**Definition farkasCheckFull\_def**

```

Definition farkasCheckFull_def:
  farkasCheckFull cc premise conclusion ↔
    farkasCheckCore cc premise conclusion v
    farkasCheckW cc premise conclusion v
    farkasCheckM cc premise conclusion
  End

```

**HOL verifier receipts**

[documentary](#) hol-strict CLEAN over the ported checker chain: theory `bern_catalogue`, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: `DISK_THM`; the four `bool` axioms) — verification run of 2026-07-22 on an isolated build host, scripts sha-matched verbatim to the sources cited here (git blob `e41b9ed0...`). The ported receipt table replays the  $\lambda = 1$  re-canonicalization accept. Downstream, the same chain carries the checker to a CakeML binary with the `machine_code_sound` tag (theory `bernCertProofARM8`). Divergence repaired, dated: an earlier port run predated the Lean checker's third branch (the mixed tagged-union replay, added 2026-07-19) and carried the two-branch `farkasCheck` with a refusal-direction-only caveat; `farkasCheckFull` — displayed above — is now the honest three-branch disjunction, and the reference checker's Farkas arm consumes it verbatim (verified in the sources at the 2026-07-22 run cited here). Semantic stratum, also ported: the soundness twin `sound_farkas` (theory `bern_tr_ready8`: hol-strict clean, 8 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), stated at `farkasCheckFull` — matching the widened checker arm verbatim — and consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splICE`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

**Proof**

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe; the general admissible template is stated in the `NORM` section and instantiated here. FARKAS is admissible by reduction to verified machinery: the affine-combination replay is checked computation, and one verified lemma warrants every accepted node.

**Recipe R17.1** (the FARKAS instance). FARKAS has no proof premises — its inputs are sub-conjuncts of  $\Phi$  itself, not sub-derivations:

$$\frac{\{a_j : \ell_j \bowtie_j r_j\}_j \subseteq \Phi \text{ conjunct-wise, } p_j \text{ their } (\cdot) \leq 0\text{-normal forms} \quad \lambda_j \geq 0 \text{ (free-signed on equalities)} \\
 \sum_j \lambda_j p_j \equiv c \text{ (exact canonical fold)} \quad c \text{ shape-stable, finitely typed; wires}(\Phi) \cup \text{wires}(c) \text{ positive-size}}{\Phi \vdash c \leq 0} \quad \text{FARKAS} \rightsquigarrow \text{the checker replays th}$$

The rewrite target contains zero catalogue nodes: the entire content is the checked combination (decoded, sign-gated, atom-sourced, folded — the fragment display quoted above walks the chain) plus the semantic argument of Proposition R17.3. The matcher is wider than the catalogued cell's published register (linear zero-form  $\{\leq, =\}$  rows): it normalizes arbitrary two-sided comparison atoms of the five admitted comparators — strict forms contribute their non-strict normal forms, semantically

(Proposition R17.2) — and the replay is degree-blind (the book note). The catalogued stratum records an unpacking sketch through BERNCONE at degree 1; the shipped warrant is direct, at all tensor shapes, heterogeneous premise-atom shapes included.

**Proposition R17.2** (nonnegative combination, per entry). *Work in an ordered field. Suppose values  $a_j$  satisfy  $a_j \leq 0$ , and weights  $\lambda_j \geq 0$ . Then  $\sum_j \lambda_j a_j \leq 0$ . Moreover the  $(\cdot \leq 0)$ -normal reading of each premise atom delivers such values: an atom  $l \bowtie r$  with  $\bowtie \in \{\leq, <\}$  yields  $l - r \leq 0$ ; with  $\bowtie \in \{\geq, >\}$  it yields  $r - l \leq 0$ ; and an equality yields both  $l - r \leq 0$  and  $r - l \leq 0$  — so a free-sign multiplier  $\nu$  on an equality is realized as the nonnegative weight  $|\nu|$  on the polarity matching  $\nu$ 's sign.*

*Proof.* Each  $\lambda_j a_j \leq \lambda_j \cdot 0 = 0$  (multiplication by a nonnegative element preserves  $\leq$ ), and a finite sum of nonpositive elements is nonpositive, by induction. The polarity readings are the definitions of the normal forms; a holding strict atom *implies* its non-strict normal form — the matcher emits the same polynomial for  $<$  as for  $\leq$ , the weakening being a fact about holding, not a checker step. The equality discipline is the textbook sign convention for multipliers of equality constraints [216, §7.3], realized here syntactically: the matcher sees both polarities, so a negative multiplier is unrepresentable rather than checked for.  $\square$

**Proposition R17.3** (semantic warrant). *Suppose the payload's combination check accepts (any branch), the conclusion term  $c$  is shape-stable and finitely typed, and every wire of  $\Phi$  and  $c$  is typed at a positive-size shape. If  $\sigma \models \Phi$  for an admissible  $\sigma$ , then  $\sigma \models c \leq 0$ .*

*Proof.* The shape and finiteness fields give  $c$  a defined evaluation; fix a conclusion entry  $i$ . The accepted branch fixes a variable space and a per-entry valuation for it: the idx-free branch reads each bare wire *clamped* at entry  $i$  — a rank-0 wire reads its scalar; a positive-rank wire of  $m$  entries reads entry  $i$  when  $i < m$  and entry 0 otherwise, so the read is always an honest in-range entry of the same tensor, and is the plain positional read wherever  $i$  is in range (a read that serves heterogeneous premise atom shapes at every conclusion index); the wire-aware branch reads each projection  $w[j]$  at its fixed coordinate; the mixed branch does both at once, on disjoint variable classes. Two facts from the Setting's broadcast discipline make the clamped reads honest and exact. First, a holding atom never compares unequal positive ranks, and a term's defined broadcast shape forces every positive-rank wire in it to carry that very shape (only rank-0 joins across shapes) — so all positive-rank wires of one premise atom share one shape, and the clamp lands them at one *common* honest entry of that atom, rank-0 wires reading their scalar under any index; here positive size is load-bearing: an atom over positively-sized wires has at least one honest entry for the clamp to land on, so its per-entry normal form is genuinely nonpositive at the branch's valuation (Remark R17.4 shows what happens otherwise). Second, the same discipline pins every positive-rank wire of  $c$  at  $c$ 's own shape, so at conclusion entry  $i$  the clamp is exact and the fold's value at the branch's valuation is entry  $i$ . Thus each sourced entry polynomial is the normal form of a premise conjunct that holds at  $\sigma$ , read per-entry; Proposition R17.2 bounds the weighted fold by 0; the certified fold identity says the fold is  $c$ 's polynomial in the branch's variable space; evaluating at the branch's valuation gives entry  $i$  of  $c$ 's tensor  $\leq 0$ . Entries were arbitrary, so the atom  $c \leq 0$  holds. `sound_farkas` case-splits on the branch disjunct and runs exactly this argument three times.  $\square$

**Remark R17.4** (why positive-size wires are not decoration). Comparison atoms hold per-entry, and a zero-size wire has no entries: at  $w$  of shape  $[0]$ , the atoms  $5 \leq w$  and  $w \leq 3$  both hold *vacuously* — yet their  $(\cdot \leq 0)$ -normal forms  $5 - w$  and  $w - 3$  are perfectly good polynomials, and with  $\lambda = 1$  each they fold to the constant 2. Without the positive-size condition the rule derives  $\vdash 2 \leq 0$  from a satisfiable premise: the classic mismatch between empty-product semantics and nonempty algebra, and the reason the field exists. (A narrower wire discipline once masked the failure by accident of strength; the field is the honest general-shape guard.)

**Remark R17.5** (why the conclusion's shape fields are not decoration). A conclusion mentioning an untyped wire — say  $c' = 0 \cdot w_{999} + c$  with  $w_{999}$  untyped — has a correct canonical polynomial: the dead monomial vanishes in canonical form, so the fold identity holds exactly as it does for  $c$ . But evaluation is structural on the *term*, not on its canonical form — it still reads  $w_{999}$ , which is untyped — so the conclusion never evaluates, and the concluded atom never holds at any valuation. The shape-stability and finiteness fields make “the fold is  $c$ 's polynomial” imply “ $c$  evaluates”, closing the gap between syntax and semantics.

**Remark R17.6** (the three branches are one algorithm). The branches differ only in the variable space where polynomials are compared: canonical wire-keyed forms; projection coordinates keyed by an injective (wire, coordinate) pairing — distinct wires *must* get distinct variables, since a composite may mix projections of several wires and conflating  $w_0[j]$  with  $w_2[j]$  would let semantically different polynomials collide (a one-line witness: from the holding premise  $w_0[0] \leq 0$  at  $w_0[0] = -1$ ,  $w_2[0] = 1$ , the conflated space accepts the false conclusion  $w_2[0] \leq 0$ ); and the mixed tagged union, where bare wires and projections are disjoint classes — in particular a bare  $w$  is never identified with  $w[0]$ : the identification variant admits a false sequent (witness:  $w = (-1, 1)$  satisfies  $w[0] \leq 0$ , and the identified space would conclude the whole-wire bound  $w \leq 0$ , false at the second entry), so the tagging is a soundness fact, not a convenience. Entry encoding, sign gates, atom sourcing, and the exact fold are the same gates, replayed branch by branch in its own variable space (the branch definitions cited at the anchors), and each branch carries its own per-entry valuation in the soundness proof.

**Remark R17.7** (relation to the literature). The linear ancestor is Farkas' lemma [73]; the multiplier sign conventions ( $\lambda \geq 0$  on inequalities, free on equalities) are [216, §7]. As everywhere in this calculus, only the sound (trivial) direction is claimed: the classical lemma's content — that every valid affine consequence *has* such a combination — is a completeness statement about producers, never consulted by the kernel. Degree-blindness of the replay places the rule's polynomial generalizations in the Krivine–Handelman family [110, 144], reached in this catalogue through SYZGYV and BERNCONE rather than by widening this rule.

#### proof step

Recipe R17.1 warrant  
entry decoding (pinned encoding)

#### formal anchor

`sound_farkas` (branch case split); `farkasCombinationValid`  
`farkasEntries`



atom normal forms, both polarities on equality  
the three branches  
projection variable spaces (Remark R17.6)  
Proposition R17.2 fold  
per-entry valuations  
  
positive-size guard (Remark R17.4)  
conclusion evaluability (Remark R17.5)  
wire placement

atomLeZeroPolys (idx-free); atomLeZeroPolysW, atomLeZeroPolysM (widened branches)  
farkasCheckCore, farkasCheckW, farkasCheckM; disjunction farkasCheck  
wireCoordKey, Term.asCoordPolyIdxW?, Term.asCoordPolyMixed?  
farkas\_fold\_eval\_nonpos, farkas\_fold\_eval\_nonpos\_polys  
the clamped, coordinate, and mixed entry valuations consumed by  
sound\_farkas's three cases  
payload field wiresPos (PFarkasData)  
payload fields conclShape, conclFin  
RuleTag.farkas

## Rule 18. SUBDIVIDE (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Subdivide*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash x \in \mathcal{R}(\Delta, S) \quad \{(\Delta_k, S_k)\}_{k=1}^N \text{ finite, with cover side-condition } \mathcal{R}(\Delta, S) \subseteq \bigcup_k \mathcal{R}(\Delta_k, S_k) \quad \forall k. \Phi \wedge (x \in \mathcal{R}(\Delta_k, S_k)) \vdash \Psi}{\Phi \vdash \Psi} \text{ SUBDIVIDE}$$

SUBDIVIDE is the case-split rule of region reasoning: from a proven membership  $\Phi \vdash x \in \mathcal{R}(\Delta, S)$ , a finite family of branch regions covering the parent, and one sub-proof of the goal per branch (each under the strengthened premise  $\Phi \wedge x \in \mathcal{R}(\Delta_k, S_k)$ ), it concludes  $\Phi \vdash \Psi$  on the whole parent region. It is degree escalation's partner and load-bearing for every non-toy proof: where a single BERNCONE certificate at feasible degree is too coarse, a producer splits the region, certifies each piece at low degree, and lets SUBDIVIDE reassemble the global conclusion — the branch-and-bound shape that makes Bernstein enclosures converge.

What the checker recomputes, given the payload (parent region, branch family, per-branch sub-proofs): that the parent region reads as a closed interval — ambient dimension 1, support exactly two vertices at distinct coordinates, empty extension set; that every branch region reads as a closed interval and is tied to its own extension set; and that the branch intervals *chain* across the parent interval — a greedy reach, iterated to its fixpoint, that extends past any interval whose left endpoint has already been reached and must arrive at the parent's right endpoint. Everything beyond this fragment (extension-ray parents or branches, regions of dimension  $\geq 2$ , supports of more than two vertices) is refused, never accepted on faith. The repair for that narrowness is CONVINTRO:  $n$ -dimensional box grids arise by *nesting* per-coordinate splits across several rule applications, not inside one — each branch establishes its coordinate facts as order atoms, and CONVINTRO converts them back into the region atom the next coordinate's split consumes, so the one-dimensional rule reaches every box grid without widening its checker.

Each hypothesis is load-bearing. Drop the cover check and a producer omits the branch where the goal fails — a gap family such as  $[1, \frac{4}{3}] \cup [\frac{3}{2}, 2]$  over  $[1, 2]$  proves the goal everywhere except where it is false. Drop the branch-region tie and the chained intervals belong to different regions than the ones the sub-proofs assume. Drop the branch-premise shape and a sub-proof could be consulted outside the piece it was proven on. Drop the branch-conclusion tie and the branches may prove any formulas whatever —  $\Psi$  never enters their obligations while the rule concludes it anyway. Drop the parent region premise and a valuation satisfying  $\Phi$  with the point outside the parent falls through every branch: the cover says nothing about it, yet the rule still concludes  $\Psi$  there. The certificate slot for cover evidence rides the wire but is deliberately *not consulted*: the checker recomputes the cover from the branch regions themselves, so ignoring the slot can only make checking stricter, never unsound.

**notes** The region variable itself plays no role in the cover geometry: the soundness argument feeds the *same* evaluated point to the parent and to the selected branch, so no tie between the term and the cover is needed — the cover is a fact about intervals, not about wires. A future interpretation may read per-gap infeasibility certificates out of the wire slot; the catalogued stratum records the corresponding unpacking sketch (per-branch cuts discharged through FARKAS and  $\vee$ -elimination).

### Lean statement

BernsteinLogic.Soundness .sound\_subdivide

statement bound

```
/-- Soundness of Subdivide (the covering argument): if the region
premise puts the evaluated point in the parent region, the checker's
cover chains the branch intervals across the parent interval, and every
branch sequent is valid on its region, then the conclusion holds on the
parent. -/
theorem sound_subdivide {H : Hypernet} {seq regionPremise : Sequent}
  {nBranches : Nat} {branchPremise : Fin nBranches → Sequent}
  {termX : Term} {simplex : Simplex} {S : ExtensionSet}
  {branchSimplex : Fin nBranches → Simplex}
  {branchS : Fin nBranches → ExtensionSet}
  {coverCertificate : Term}
  (ihRegion : regionPremise.semanticallyValid H)
  (ihBranch : ∀ k, (branchPremise k).semanticallyValid H)
  (hpre : seq.pre = regionPremise.pre)
  (hregionPost : regionPremise.post = .inRegion termX simplex S)
  (hbranchPre : ∀ k, (branchPremise k).pre
    = .and regionPremise.pre
```

```

      (.inRegion termX (branchSimplex k) (branchS k)))
(hbranchPost : ∀ k, (branchPremise k).post = seq.post)
(hcover : coverInfeasibilityValid termX simplex S nBranches
      branchSimplex branchS coverCertificate) :
seq.semanticallyValid H

```

BernsteinLogic.Soundness . **intervalD\_elim**

statement bound

```

/-- Elim: a `MaskRegion` witness against a successful interval reading
lands the point's coordinate in `[lo, hi]`, and the interval is proper. -/
theorem intervalD_elim {S : ExtensionSet} {lo hi : ℚ}
  (hint : S.intervalD? = some (lo, hi))
  {x : Fin S.simplex.complex.ambDim → ℚ}
  (hmask : MaskRegion S.simplex.complex S.simplex.contains S.contains x)
  (j0 : Fin S.simplex.complex.ambDim) :
  lo ≤ x j0 ∧ x j0 ≤ hi ∧ lo < hi

```

BernsteinLogic.Soundness . **intervalD\_intro**

statement bound

```

/-- Intro: a coordinate inside a successful interval reading assembles a
`MaskRegion` witness (the extension set is empty in this fragment, so
both barycentric weights come out nonnegative). -/
theorem intervalD_intro {S : ExtensionSet} {lo hi : ℚ}
  (hint : S.intervalD? = some (lo, hi))
  {x : Fin S.simplex.complex.ambDim → ℚ}
  (j0 : Fin S.simplex.complex.ambDim)
  (hxlo : lo ≤ x j0) (hxhi : x j0 ≤ hi) :
  MaskRegion S.simplex.complex S.simplex.contains S.contains x

```

BernsteinLogic.Soundness . **ChainInv**

statement bound

```

/-- The greedy chain invariant: the reach never sits below `lo`, and
once it has moved past `lo`, every point of `[lo, reach]` lies in some
interval of the FULL list. -/
def ChainInv (ivs : List (ℚ × ℚ)) (lo r : ℚ) : Prop

```

BernsteinLogic.Soundness . **chainInv\_extendPass**

statement bound

```

/-- One pass preserves the invariant (stated over a sublist so the foldl
induction goes through; the covering intervals always come from the full
list). -/
theorem chainInv_extendPass (ivs : List (ℚ × ℚ)) (lo : ℚ) :
  ∀ (sub : List (ℚ × ℚ)), (∀ iv ∈ sub, iv ∈ ivs) →
  ∀ (r : ℚ), ChainInv ivs lo r →
  ChainInv ivs lo (coverExtendPass sub r)

```

BernsteinLogic.Soundness . **chainInv\_reach**

statement bound

```

theorem chainInv_reach (ivs : List (ℚ × ℚ)) (lo : ℚ) :
  ∀ (n : Nat) (r : ℚ), ChainInv ivs lo r →
  ChainInv ivs lo (coverReach n ivs r)

```

BernsteinLogic.Soundness . **coverChain\_covers**

statement bound

```

/-- Soundness of the greedy chain check: on success, every point of
`[lo, hi]` lies in some interval of the list. -/
theorem coverChain_covers {ivs : List (ℚ × ℚ)} {lo hi : ℚ} (hlt : lo < hi)
  (hcheck : coverChainCheck ivs lo hi = true) :
  ∀ x : ℚ, lo ≤ x → x ≤ hi → ∃ iv ∈ ivs, iv.1 ≤ x ∧ x ≤ iv.2

```

**What the cover check means — and what it does not.** The side condition appears in the payload as the single name `coverInfeasibilityValid`, so this book binds its meaning by displaying the definition chain in full, from the checker sources. Reading the chain: `ExtensionSet.intervalD?` is the closed-interval reading of a region — it succeeds only on the checkable fragment (dimension 1, two-vertex support at distinct coordinates, empty extension set), returning sorted endpoints, and everything else reads none, which the cover check turns into refusal. `coverExtendPass` is one greedy pass: extend the reach past any interval whose left endpoint has been reached; `coverReach` iterates it, and as many passes as there are intervals suffice for the fixpoint (a

productive pass strictly increases the reach onto a new right endpoint, of which there are only finitely many). `coverChainCheck` asks whether the reach from the parent's left endpoint arrives at its right endpoint; `coverCheck` assembles the whole condition, tying each branch's extension set to its simplex on the way.

What a skeptical reader should confirm by eye: the region term and the wire-borne cover certificate are *ignored* — the relation discards both arguments, and the cover is recomputed from the branch regions alone (the recompute pattern; a forged certificate cannot influence the verdict, and a missing one costs nothing); no valuation is consulted anywhere; and the branch sub-proofs enter soundness, not the cover computation — the checker decides pure interval geometry.

BernsteinLogic.Core.Catalogue . **ExtensionSet.intervalID?**

definition bound

```

/-- The closed-interval reading of a region `R(Δ, S)`: succeeds iff the
ambient dimension is 1, the simplex's support is exactly two vertices at
distinct coordinates, and the extension set is empty – returning the
region's endpoints, sorted (`lo < hi` by construction). Everything else
reads `none` (and the cover check then rejects). -/
def ExtensionSet.intervalID? (S : ExtensionSet) : Option (Rat × Rat) :=
  if h : S.simplex.complex.ambDim = 1 then
    if S.isEmptyB then
      match (List.finRange S.simplex.complex.nVertices).filter
        (fun v => S.simplex.contains v) with
      | [a, b] =>
        let j0 : Fin S.simplex.complex.ambDim := {0, by omega}
        let V₀ := S.simplex.complex.coord a j0
        let V₁ := S.simplex.complex.coord b j0
        if V₀ < V₁ then some (V₀, V₁)
        else if V₁ < V₀ then some (V₁, V₀)
        else none
      | _ => none
    else none
  else none

```

BernsteinLogic.Core.Catalogue . **coverExtendPass**

definition bound

```

/-- One greedy pass of the 1-D chain cover: extend the reach past any
interval whose left endpoint has already been reached. -/
def coverExtendPass (ivs : List (Rat × Rat)) (r : Rat) : Rat :=
  ivs.foldl (fun r iv => if iv.1 ≤ r ∧ r < iv.2 then iv.2 else r) r

```

BernsteinLogic.Core.Catalogue . **coverReach**

definition bound

```

/-- Iterated greedy reach. `ivs.length` passes suffice for the fixpoint:
a productive pass strictly increases the reach onto a new interval's
right endpoint (of which there are at most `ivs.length`), and an
unproductive pass is already the fixpoint. -/
def coverReach : Nat → List (Rat × Rat) → Rat → Rat
| 0, _, r => r
| n + 1, ivs, r => coverReach n ivs (coverExtendPass ivs r)

```

BernsteinLogic.Core.Catalogue . **coverChainCheck**

definition bound

```

/-- Greedy 1-D chain check: `[lo, hi] ⊆ ⋃ ivs`, all intervals closed.
Soundness: `coverChain_covers` (Soundness §55'). -/
def coverChainCheck (ivs : List (Rat × Rat)) (lo hi : Rat) : Bool :=
  decide (hi ≤ coverReach ivs.length ivs lo)

```

BernsteinLogic.Core.Catalogue . **coverCheck**

definition bound

```

/-- The Subdivide cover computation (fragment documented at the section
header): the parent region is an interval, every branch region is an
interval tied to its own extension set, and the branch intervals chain
across the parent's. -/
def coverCheck (simplex : Simplex) (S : ExtensionSet) (nBranches : Nat)
  (branchSimplex : Fin nBranches → Simplex)
  (branchS : Fin nBranches → ExtensionSet) : Bool :=
  match S.intervalID? with
  | none => false

```

```

| some (lo, hi) =>
  decide (S.simplex = simplex) &&
  ((List.finRange nBranches).all fun k =>
    decide ((branchS k).simplex = branchSimplex k) &&
    ((branchS k).interval1D?).isSome) &&
  coverChainCheck
  ((List.finRange nBranches).filterMap fun k => (branchS k).interval1D?)
  lo hi

```

BernsteinLogic.Core.Catalogue . **coverInfeasibilityValid**

definition bound

```

def coverInfeasibilityValid
  (_termX : Term) (simplex : Simplex) (S : ExtensionSet)
  (nBranches : Nat)
  (branchSimplex : Fin nBranches → Simplex)
  (branchS : Fin nBranches → ExtensionSet)
  (_coverCert : Term) : Prop :=
  coverCheck simplex S nBranches branchSimplex branchS = true

```

## Correspondence (Steele display ↔ Lean)

reviewer-certified against the displayed texts

The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in **Ax**; sub-derivation premises as at **WEAKEN**.

- $x \mapsto \text{termX} : \text{Term}$  (a free term — unlike **BERNCONE**, no wire tie is needed here: the note above explains why);  $(\Delta, S) \mapsto \text{simplex}, S$ ; the branch family  $\{(\Delta_k, S_k)\}_{k=1}^N \mapsto \text{branchSimplex}, \text{branchS} : \text{Fin } n\text{Branches} \rightarrow \_$ .
- region premise  $\Phi \vdash x \in \mathcal{R}(\Delta, S) \mapsto \text{ihRegion}$  with  $\text{hregionPost} : \text{regionPremise.post} = \text{.inRegion termX simplex S}; \Phi$  shared via  $\text{hpre}$ .
- the branch judgments  $\forall k. \Phi \wedge (x \in \mathcal{R}(\Delta_k, S_k)) \vdash \Psi \mapsto \text{ihBranch}$  with  $\text{hbranchPre}$  pinning each branch premise to the displayed conjunction in the displayed order ( $\Phi$  left) and  $\text{hbranchPost}$  tying every branch conclusion to  $\text{seq.post}$  —  $\Psi$  is whatever formula the conclusion sequent carries; nothing constrains its shape.
- the cover side-condition  $\mathcal{R}(\Delta, S) \subseteq \bigcup_k \mathcal{R}(\Delta_k, S_k) \mapsto \text{hcover} : \text{coverInfeasibilityValid}$ , i.e.  $\text{coverCheck} = \text{true}$ : the parent extension set must present as a 1-dimensional interval ( $\text{ExtensionSet.interval1D?}$  — outside that fragment the check refuses, so the certified reach of the displayed set-inclusion is the interval fragment, the **CONVELIM** class), each branch's simplex-tie is checked, and the greedy chain ( $\text{coverExtendPass}/\text{coverReach}/\text{coverChainCheck}$ , certified alongside) verifies the branch intervals chain across the parent interval — a decidable *sufficient* witness of the displayed inclusion, conservative like every refusal boundary.
- consumed by nothing: the payload's cover certificate term rides into  $\text{coverInfeasibilityValid}$  as an ignored argument (the binder is underscore-named in the displayed definition) — retained for the per-gap-infeasibility interpretation the note records; the realized checker recomputes the cover itself and reads no producer-supplied cover data.

Width note: within the interval fragment neither side is wider — every other hypothesis is consumed, and the displayed rule's “finite family” is carried by the  $\text{Fin } n\text{Branches}$  index type itself.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when

the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_catalogue, HOL4 development script bern\_catalogueScript.sml script bound

#### Definition extIntervalID\_def

```

Definition extIntervalID_def:
  extIntervalID es =
    if es.simplex.complex.ambDim = 1 then
      (if extIsEmptyB es then
        (case FILTER (λv. simplexContainsNat es.simplex v)
          (rangeL es.simplex.complex.nVertices) of
         | [a; b] =>
            (let v0 = getD (getD es.simplex.complex.coord a []) 0 (0:rat) in
             let v1 = getD (getD es.simplex.complex.coord b []) 0 (0:rat) in
              if v0 < v1 then SOME (v0, v1)
              else if v1 < v0 then SOME (v1, v0)
              else NONE)
         | _ => NONE)
      else NONE)
    else NONE
End

```

#### Definition coverExtendPass\_def

```

Definition coverExtendPass_def:
  coverExtendPass ivs (r:rat) =
    FOLDL (λr iv. if FST iv ≤ r ∧ r < SND iv then SND iv else r) r ivs
End

```

#### Definition coverReach\_def

```

Definition coverReach_def:
  coverReach 0 ivs (r:rat) = r ∧
  coverReach (SUC n) ivs r = coverReach n ivs (coverExtendPass ivs r)
End

```

#### Definition coverChainCheck\_def

```

Definition coverChainCheck_def:
  coverChainCheck ivs (lo:rat) hi = (hi ≤ coverReach (LENGTH ivs) ivs lo)
End

```

#### Definition coverCheck\_def

```

Definition coverCheck_def:
  coverCheck sx es branches =

```

```

case extIntervalID es of
| NONE => F
| SOME (lo, hi) =>
  (es.simplex = sx ∧
   EVERY (λ(bsx, bes). bes.simplex = bsx ∧ IS_SOME (extIntervalID bes))
    branches ∧
   coverChainCheck
    (FLAT (MAP (λ(bsx, bes). case extIntervalID bes of
      | NONE => []
      | SOME iv => [iv]) branches))
    lo hi)
End

```

## HOL verifier receipts

**documentary** hol-strict CLEAN over the ported checker chain: theory bern\_catalogue, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, scripts sha-matched verbatim to the sources cited here (git blob e41b9ed0...). The ported receipt table replays the cover battery: the positive chain  $([1, \frac{3}{2}] \cup [\frac{3}{2}, 2]$  over  $[1, 2]$ ) and four negative controls — the gap mutant, a point-supported branch, an extension-ray parent, and untied branch simplices. Representation seam, inventoried in the script: the indexed branch family becomes one list of (simplex, extension-set) pairs in index order, and the Lean relation's two inert arguments (region term, cover certificate) are not reproduced — both value-preserving deltas under the port's seam discipline.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe; the general admissible template is stated in the NORM section and instantiated here. SUBDIVIDE is the genuinely one-node-to-many member of this batch: its rewrite is a case analysis over the branch sub-proofs, and the checked cover condition is what makes the case analysis exhaustive.

**Recipe R18.1** (the SUBDIVIDE instance). The node carries a region premise and one sub-proof per branch:

$$\frac{\mathcal{T}_0 \triangleright (\Phi \vdash x \in \mathcal{R}(\Delta, S)) \quad \mathcal{T}_k \triangleright (\Phi \wedge (x \in \mathcal{R}(\Delta_k, S_k)) \vdash \Psi) \quad (k = 1, \dots, N) \quad \mathcal{R}(\Delta, S) \subseteq \bigcup_k \mathcal{R}(\Delta_k, S_k) \text{ (checked: the interval chain)}}{\Phi \vdash \Psi} \text{SUBDIVIDE} \rightsquigarrow \text{select the covering branch; consult its subtree.}$$

Conceptually the rewrite is an  $N$ -way case analysis: at any valuation satisfying  $\Phi$ , the premise puts the evaluated point in the parent region, the cover places it in some branch region  $k$ , and  $\mathcal{T}_k$ 's strengthened premise then holds — so  $\mathcal{T}_k$  delivers  $\Psi$ . The shipped warrant packages this selection argument as one verified lemma rather than as catalogue surgery (the catalogued stratum records the corresponding derivation sketch through FARKAS-discharged infeasibility cuts and  $\vee$ -elimination); the checked side condition is Proposition R18.2, the exhaustiveness of the case analysis.

**Proposition R18.2** (the greedy chain covers). *Let ivs be a finite list of closed intervals and  $\text{lo} < \text{hi}$  rationals. Define one greedy pass  $r \mapsto \text{extend}(r)$ : fold over the list, replacing  $r$  by an interval's right endpoint whenever that interval's left endpoint is  $\leq r$  and its right endpoint is  $> r$ . If after  $|\text{ivs}|$  passes from  $\text{lo}$  the reach is  $\geq \text{hi}$ , then  $[\text{lo}, \text{hi}] \subseteq \bigcup \text{ivs}$ .*

*Proof.* Maintain the invariant: the reach  $r$  satisfies  $\text{lo} \leq r$ , and either  $r \leq \text{lo}$  or every point of  $[\text{lo}, r]$  lies in some interval of the full list. It holds initially ( $r = \text{lo}$ , the first disjunct). One pass preserves it: when the fold takes an interval  $[a, b]$  with  $a \leq r < b$ , any point  $y \in [\text{lo}, b]$  is either  $\leq r$  — covered by the invariant, or by  $[a, b]$  itself in the  $r \leq \text{lo}$  base case since then  $a \leq \text{lo} \leq y$  — or lies in  $(r, b] \subseteq [a, b]$ ; and when the fold declines an interval the state is unchanged. As many passes as there are intervals reach the fixpoint: a productive pass strictly increases  $r$  onto a new interval's right endpoint, of which there are at most  $|\text{ivs}|$ , and an unproductive pass is already the fixpoint. If the final reach is  $\geq \text{hi} > \text{lo}$ , the invariant's second disjunct is in force and covers all of  $[\text{lo}, \text{hi}]$ .  $\square$

**Proposition R18.3** (semantic warrant). *Suppose the region premise and every branch premise-sequent are semantically valid, the branch premises have the checked shape ((branch  $k$ )'s premise is  $\Phi \wedge (x \in \mathcal{R}(\Delta_k, S_k))$  and its conclusion is  $\Psi$ ), and the cover check accepts. Then  $\Phi \vdash \Psi$  is semantically valid.*

*Proof.* Fix an admissible  $\sigma \models \Phi$ . The region premise puts the evaluated point  $q$  of  $x$  in  $\mathcal{R}(\Delta, S)$ ; the accepted cover check says the parent reads as a closed interval  $[\text{lo}, \text{hi}]$  (dimension 1, two-vertex support,  $S = \emptyset$ ), so  $q$ 's coordinate lies in  $[\text{lo}, \text{hi}]$  — eliminating the parent's barycentric weights to endpoint bounds. By Proposition R18.2 some branch interval contains the coordinate. Each branch interval was itself produced by running the closed-interval reading on that branch's own simplex and extension set (the cover check refuses any branch it cannot so read, and checks the tie between the two), so “the coordinate lies in branch  $k$ 's interval” converts back — re-introducing barycentric weights over branch  $k$ 's own support from the two endpoint bounds — into membership of



the *same* evaluated point in  $\mathcal{R}(\Delta_k, S_k)$ : this is where nothing about the region term is needed beyond its one evaluation, used twice. Hence  $\sigma$  satisfies branch  $k$ 's premise  $\Phi \wedge (x \in \mathcal{R}(\Delta_k, S_k))$ ; its validity delivers its conclusion, which is  $\Psi$  by the checked branch–conclusion shape.  $\square$

*Remark R18.4* (what breaks without the cover check). Everything: the rule without the cover is “prove  $\Psi$  wherever convenient.” The canonical mutant is the gap family  $\{[1, \frac{4}{3}], [\frac{3}{2}, 2]\}$  over the parent  $[1, 2]$  — each branch sub-proof can be perfectly honest, and the goal simply fails on  $(\frac{4}{3}, \frac{3}{2})$ , where no branch constrains it. The greedy reach from 1 arrives at  $\frac{4}{3}$  and stops: no remaining interval's left endpoint is  $\leq \frac{4}{3}$ , so the check refuses. The checker's replayed battery keeps this mutant, together with a point-supported branch (a degenerate interval that cannot read), an extension-ray parent, and untied branch simplices, as standing negative controls.

*Remark R18.5* (the fragment boundary, and why refusal is the right default). The checkable fragment is deliberate: 1-dimensional closed intervals, chained. Extension-ray parents or branches, higher-dimensional regions, and richer supports are refused — not because the rule is false there, but because no verified covering argument exists there yet, and the fail-toward-refusal posture makes the unproven territory unrepresentable. Box domains lose no expressive reach:  $n$ -dimensional grids are expressible by nesting per-coordinate splits across several applications, each within the fragment, the inner split's parent membership entering through the outer split's branch premise. The cover-certificate wire slot is likewise part of the posture: recomputing the cover from the branch regions means a forged certificate cannot influence the verdict — ignoring the slot cannot affect it either — and a future interpretation that additionally *required* per-gap infeasibility certificates from the slot would only tighten the check (one that instead accepted new covers on certificate evidence would be a new checking route, owing its own soundness argument).

*Remark R18.6* (relation to the literature). The rule is the reassembly half of branch-and-bound over Bernstein enclosures: subdivision sharpens coefficient bounds quadratically in the subdivision width and drives them to the range in the limit [85], with the subdivision algorithms themselves classical [40, 151]. In that literature the splitting is a completeness device — *some* refinement certifies any strict global fact; here only the soundness direction enters the kernel: whatever refinement the producer chose, the chain check makes its case analysis exhaustive. None of the producer-side effectivity data (piece counts, degrees, tolerances) is consulted.

| proof step                                  | formal anchor                                                                                                                          |
|---------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| Recipe R18.1 warrant                        | sound_subdivide                                                                                                                        |
| interval reading of the parent and branches | ExtensionSet.intervalID?; elim/intro pair intervalID_elim, intervalID_intro                                                            |
| Proposition R18.2                           | coverExtendPass, coverReach, coverChainCheck; invariant ChainInv, chainInv_extendPass, chainInv_reach; covering fact coverChain_covers |
| the assembled check (ties included)         | coverCheck, coverInfeasibilityValid                                                                                                    |
| branch premise/conclusion shape             | payload fields branchPre, branchPost (PSubdivideData)                                                                                  |
| same-point re-entry (Prop. R18.3)           | the region read used by both parent elimination and branch introduction in sound_subdivide's body                                      |
| wire placement                              | RuleTag.subdivide; the unconsulted certificate slot coverCertificate                                                                   |

## Rule 19. LiRPA (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *LiRPA*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{ccccc} \Phi \vdash a \leq h & \Phi \vdash h \leq b & \text{relu}_{\{h\} \rightarrow \{y\}} \in H & a < 0 & 0 < b \\ \alpha a + \beta \leq 0 & \beta \leq 0 & \alpha b + \beta \leq b & & \\ 0 \leq \gamma a + \delta & 0 \leq \delta & b \leq \gamma b + \delta & & \end{array}}{\Phi \vdash (\alpha h + \beta) \leq y \wedge y \leq (\gamma h + \delta)} \quad \text{LiRPA}$$

LiRPA is the calculus's neural-network bound-propagation rule — the certificate form of the linear relaxation of the ReLU activation that bound-propagation verifiers compute (the CROWN/LiRPA family, the triangle relaxation, DeepPoly). Given already-proven box bounds  $\Phi \vdash a \leq h$  and  $\Phi \vdash h \leq b$  on the preactivation wire of a cited ReLU application, with  $a < 0 < b$  (the box straddles the kink — the *unstable* case, in the verification literature's vocabulary), it concludes the two-sided affine envelope  $\alpha h + \beta \leq y \wedge y \leq \gamma h + \delta$  on the application's output wire  $y$ . The producer finds the envelope however it likes — CROWN's adaptive slopes, the triangle relaxation's chord, any heuristic — and ships only the box endpoints and the four coefficients; no relaxation machinery is re-run or trusted.

What the checker recomputes, given the payload (two premise proofs, the two wires, six rationals): the *linkage* — the hypernet really contains a packaged ReLU application at exactly these ports, of positive total size (a one-line decidable port lookup); that the premises are literally the atoms  $a \leq h$  and  $h \leq b$  over one shared premise formula; the sign conditions  $a < 0$  and  $0 < b$ ; the six envelope inequalities, exactly over  $\mathbb{Q}$  — the lower line under the function at the three checkpoints  $a, 0, b$  ( $\alpha a + \beta \leq 0, \beta \leq 0, \alpha b + \beta \leq b$ ) and the upper line above it at the same three ( $0 \leq \gamma a + \delta, 0 \leq \delta, b \leq \gamma b + \delta$ ); and that the conclusion is *exactly* the pinned conjunction of the two affine comparison atoms, built by the checker from the coefficients — the producer proposes no conclusion shape at all.

Each hypothesis but one is load-bearing. Drop the linkage and  $y$  is an arbitrary wire with no semantic tie to  $h$  (the rule as first staged had exactly this gap and was kept vacuous — rejecting every certificate — until the port-lookup premise landed). Drop the lower kink check ( $\beta \leq 0$ ) and the chord through the box endpoints, which passes both endpoint checks, would certify as a lower envelope while lying strictly above the function at  $h = 0$ . (The upper kink check  $0 \leq \delta$  is the one redundancy: under the strict straddle the two upper endpoint checks already force  $\delta \geq -ab/(b-a) > 0$  — evaluate the affine interpolation of the endpoint bounds at 0 — so it rides along because the certificate's check set is the uniform six.) Drop an endpoint check and a steep line passing the remaining two checkpoints escapes the function at the dropped end (the proof section constructs both failures). The box premises themselves are what make an affine *upper* envelope possible at all — no affine function dominates  $\max(0, h)$  on all of  $\mathbb{Q}$  — and what license lower-line slopes outside  $[0, 1]$ .

**notes** The family's shared machinery — the packaged-node linkage, the elementwise two-case semantic contract, and the conservativity warrant behind it — is stated once, in this section's recipe preamble, and referenced by the four sibling rules (ReLU-POS, ReLU-NEG, ReLU-NN, ReLU-ABOVE). One envelope serves the whole node: the rule is elementwise over the cited application's tensor shape, with a single  $(a, b, \alpha, \beta, \gamma, \delta)$  for every entry — per-entry envelopes are expressed with singleton-shaped applications. The payload additionally declares a denominator bound on its witness rationals (a size claim for resource accounting; it carries no proof-theoretic weight in the soundness argument). The rule's name is the literature's: linear-relaxation-based perturbation analysis [243].

### Lean statement

**The linkage and the pinned conclusion, from the checker sources.** `reluLinked` is the whole of the family's side condition: the hypernet's application list contains a packaged ReLU application at exactly the cited preactivation/output ports, of positive total size — one decidable lookup (`PrimApp.isReluAt`), no pattern-matching over composites. The positive-size gate refuses vacuous surface at the source: a size-zero node's comparison atoms hold vacuously, and a fact no valuation can falsify must not be citable (the same lesson the FARKAS rule's zero-size counterexample taught that payload). `affineWireTerm` and `liRpaConclusionForm` pin the conclusion: the rule concludes exactly the conjunction of the two affine comparison atoms, built from the payload coefficients — there is no producer-supplied conclusion shape to audit.

BernsteinLogic.Core.Catalogue . **PrimApp.isReluAt**

definition bound

```
-- Port-lookup matcher for the ReLU linkage: a packaged ReLU
-- application at exactly these ports, of positive total size. The size
-- gate is the shape-aware rider (conservative-extension memo register 2,
-- the `PFarkasData.wiresPos` lesson generalized): a shape-[0] node's
-- atoms hold vacuously, and vacuous surface must not be citable — the
```

```

rules refuse it at the linkage, before any soundness question arises. -/
def PrimApp.isReluAt (preact post : Wire) : PrimApp → Bool
| .relu s i o =>
  decide (i = preact) && decide (o = post) &&
  decide (0 < Shape.totalSize s)
| _ => false

```

BernsteinLogic.Core.Catalogue . **reluLinked**

definition bound

```

def reluLinked (H : Hypernet) (preact post : Wire) : Prop :=
  H.applications.any (PrimApp.isReluAt preact post) = true

```

BernsteinLogic.Core.Catalogue . **affineWireTerm**

definition bound

```

def affineWireTerm (offset slope : Rat) (w : Wire) : Term :=
  .add (.const offset) (.mul (.const slope) (.var w))

```

BernsteinLogic.Core.Catalogue . **lirpaConclusionForm**

definition bound

```

def lirpaConclusionForm
  (preact post : Wire) (α β γ δ : Rat) (F : Formula) : Prop :=
  F = .and (.atom .le (affineWireTerm β α preact) (.var post))
    (.atom .le (.var post) (affineWireTerm δ γ preact))

```

**The verified chain.** `reluLinked_facts` turns the linkage into semantics: under any admissible valuation the two cited wires carry tensors of one shared shape and positive entry count, related entrywise by the two-case ReLU relation. The four interpolation lemmas are the checkpoint arithmetic — an affine function suitably bounded at an interval's endpoints is bounded on the interval, one lemma per envelope side per sign branch — and `sound_lirpa` assembles them entrywise into the rule's warrant.

BernsteinLogic.Soundness . **reluLinked\_facts**

statement bound

```

/-- Extraction for the realized B12 linkage: under any admissible
valuation, a cited ReLU application yields both wire tensors – one
shared shape, positive entry count – and the elementwise two-case
relation. Typing rides `H.typeConsistent` at the cited application (no
payload typing fields needed – the wire format is untouched). -/
theorem reluLinked_facts {H : Hypernet} {preact post : Wire}
  (hlink : reluLinked H preact post) (σ : H.Valuation)
  (hadm : σ.Admissible) :
  ∃ qp qy : QTensor,
    σ.wireQ preact = some qp ∧ σ.wireQ post = some qy ∧
    qp.shape = qy.shape ∧ 0 < qp.entries.length ∧
    qp.entries.length = qy.entries.length ∧
    ∀ i, i < qp.entries.length →
      (qp.entries.getD i 0 ≤ 0 ∧ qy.entries.getD i 0 = 0) ∨
      (0 ≤ qp.entries.getD i 0 ∧
       qp.entries.getD i 0 = qp.entries.getD i 0)

```

BernsteinLogic.Soundness . **lirpa\_lower\_neg**

statement bound

```

theorem lirpa_lower_neg {a α β p : ℚ} (ha : a < 0)
  (h1 : α * a + β ≤ 0) (h2 : β ≤ 0) (hap : a ≤ p) (hp0 : p ≤ 0) :
  α * p + β ≤ 0

```

BernsteinLogic.Soundness . **lirpa\_lower\_pos**

statement bound

```

theorem lirpa_lower_pos {b α β p : ℚ} (hb : 0 < b)
  (h2 : β ≤ 0) (h3 : α * b + β ≤ b) (hp0 : 0 ≤ p) (hpb : p ≤ b) :
  α * p + β ≤ p

```

BernsteinLogic.Soundness . **lirpa\_upper\_neg**

statement bound

```

theorem lirpa_upper_neg {a γ δ p : ℚ} (ha : a < 0)
  (h1 : 0 ≤ γ * a + δ) (h2 : 0 ≤ δ) (hap : a ≤ p) (hp0 : p ≤ 0) :
  0 ≤ γ * p + δ

```

BernsteinLogic.Soundness . **lirpa\_upper\_pos**

statement bound

```
theorem lirpa_upper_pos {b γ δ p : ℚ} (hb : 0 < b)
  (h2 : 0 ≤ δ) (h3 : b ≤ γ * b + δ) (hp0 : 0 ≤ p) (hpb : p ≤ b) :
  p ≤ γ * p + δ
```

BernsteinLogic.Soundness . **sound\_lirpa**

statement bound

```
/-- LiRPA: the packaged degree-1 relaxation – from box premises
`a ≤ h ≤ b` and the eight endpoint side conditions, the affine sandwich
`αh + β ≤ γ ∧ γ ≤ γh + δ` holds entrywise, by the four interpolation
lemmas applied per two-case branch. -/
theorem sound_lirpa {H : Hypernet}
  {seq lowerPremise upperPremise : Sequent}
  {preact post : Wire} {a b α β γ δ : ℚ}
  (ihl : lowerPremise.semanticallyValid H)
  (ihu : upperPremise.semanticallyValid H)
  (hlink : reluLinked H preact post)
  (hAneg : a < 0) (hBpos : 0 < b)
  (hLower1 : α * a + β ≤ 0) (hLower2 : β ≤ 0) (hLower3 : α * b + β ≤ b)
  (hUpper1 : 0 ≤ γ * a + δ) (hUpper2 : 0 ≤ δ) (hUpper3 : b ≤ γ * b + δ)
  (hpre : seq.pre = lowerPremise.pre)
  (hpreMatch : lowerPremise.pre = upperPremise.pre)
  (hlowerPost : lowerPremise.post = .atom .le (.const a) (.var preact))
  (hupperPost : upperPremise.post = .atom .le (.var preact) (.const b))
  (hpostForm : lirpaConclusionForm preact post α β γ δ seq.post) :
  seq.semanticallyValid H
```

**Why consuming the packaged two-case fact is not new kernel content.** The two-case relation is the packaged symbol's defined semantics, consumed by the family as model fact at every positive size; what this pair of declarations adds is the conservativity warrant — that the definition mints no genuinely new primitive: `reluExpH` is the definitional expansion (a copy, a sign-split cell complex, an indicator cast, and an exact rank-0 contraction), and `relu_projection` shows — both directions — that the two-case relation is exactly the expansion's model-level facts existentially projected onto the external ports, proven at the canonical single-entry instance. The boundary point  $h = 0$  rides the one-hot cell semantics: both cells admit it, and every admissible choice forces  $y = 0$ . Applications of total size other than one currently have no core expansion (a recorded packing gap in the development's notes): at those sizes the node stands as a genuine primitive with the stated semantics — the rules remain sound there, while the not-a-new-primitive claim is certified only at size one.

BernsteinLogic.ReluExpansion . **relu\_projection**

statement bound

```
/-- **Obligation P (the projection lemma), both directions**: the
packaged ReLU's derived sp-fact – the two-case relation – EQUALS the
expansion's sp-facts existentially projected onto the external ports.
The internal wires (the copies, the indicators, the cast image, the
rank-0 legs) are exactly the existentially-quantified content of the
valuation; the boundary case  $p = 0$  is covered because both one-hot
choices are admissible there and both force  $y = 0$ . -/
theorem relu_projection (p y : ℚ) :
  (∃ σ : reluExpH.Valuation, σ.Admissible ∧
    σ.entryFlat 0 0 = some (.fin p) ∧
    σ.entryFlat 8 0 = some (.fin y)) ↔
  ((p ≤ 0 ∧ y = 0) ∨ (0 ≤ p ∧ y = p))
```

BernsteinLogic.ReluExpansion . **reluExpH**

definition bound

```
/-- The expansion: copy;(id ⊗ Cell[Σ±]);Contract, with the typing
plumbing (Cast  $\mathbb{B} \rightarrow \mathbb{Q}$  on the indicator, reshapes on the rank-1 ports). -/
def reluExpH : Hypernet where
  wireType := expTy
  applications :=
    [.copy 2 0 [1, 2],
     .cell sigmaPM 2 [3, 4],
     .cast Dtype.B Dtype.Q .exact [] 4 5,
     .reshape [] 1 6,
     .contract Dtype.Q .exact [6, 5] 7 [0, 0] 0 pat0,
     .reshape [1] 7 8]
  linearity := by decide
  acyclic := by decide
  typeConsistent := by decide +kernel
```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; sub-derivation premises as at WEAKEN. This dictionary doubles as the family preamble: its linkage row is shared verbatim by the four sibling rules.

- $h \mapsto \text{preact} : \text{Wire}; y \mapsto \text{post} : \text{Wire}; a, b, \alpha, \beta, \gamma, \delta \mapsto$  the six rational binders of the same names in `sound_lirpa`.
- the linkage premise  $\text{relu}_{\{h\} \rightarrow \{y\}} \in H \mapsto \text{hlink} : \text{reluLinked } H \text{ preact post}$  — membership *plus* the positive-size gate: unfolding (the definition above), the certified premise demands a packaged ReLU application at exactly these ports *of positive total size*. The display's bare  $\in$  shows no size condition; the certified linkage is strictly narrower, by design (vacuous surface refused at the source — the intro above), and the narrowing is family-wide.
- box premises  $\Phi \vdash a \leq h$  and  $\Phi \vdash h \leq b \mapsto \text{ihl}, \text{ihu}$  with posts pinned by `hlowerPost : lowerPremise.post = .atom .le (.const a) (.var preact)` and `hupperPost : upperPremise.post = .atom .le (.var preact) (.const b)` — displayed comparators and operand orders carried exactly;  $\Phi$  shared three ways via `hpre` and `hpreMatch`.
- the eight arithmetic side conditions  $\mapsto \text{hAneg}, \text{hBpos}, \text{hLower1} \text{--} \text{hLower3}, \text{hUpper1} \text{--} \text{hUpper3}$ , verbatim over  $\mathbb{Q}$  — scalar hypotheses on the payload rationals, not judgments.
- conclusion  $\Phi \vdash (\alpha h + \beta) \leq y \wedge y \leq (\gamma h + \delta) \mapsto$  the goal under `hpostForm : lirpaConclusionForm preact post` applied to the four coefficients and `seq.post` to the conjunction of the two affine atoms in the displayed order (lower envelope left). Spelling drift, disclosed: the pinned affine term is `affineWireTerm`'s offset-first  $\beta + \alpha h$ , where the display writes  $\alpha h + \beta$  — semantically identical, but formulas are ordered trees, so the certified rule accepts exactly the offset-first spelling (the `CONVELIM/CONVINTRO` build-output class).

Width notes beyond the linkage gate: the certified atoms are elementwise over the cited application's tensor shape — one envelope for every entry, per the note above — where the display reads scalar; that is the batch semantics of the atom grammar, not a strengthening. `reluLinked_facts`, the four interpolation lemmas, `relu_projection`, and `reluExpH` are certified alongside as machinery: the first extracts the elementwise two-case relation the family consumes, the last two ground the packaged symbol in the core expansion — they are *about* the linkage, in the satellite sense of ORE's `ctx`-rule. No hypothesis of `sound_lirpa` is unused, and nothing certified is wider than the display.

## Lean verifier receipts

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.ReluExpansion`; bound build `68395e3f44af...` (bound from receipt `b1a5795a8e57ce30`, 2026-07-18T00:07:06Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 7d0d53fd422f509e | 2026-07-17T20:41:28Z | build+replay+hardened+decorrelated | current |
| 3219a0974f9b52a3 | 2026-07-17T21:00:42Z | build+replay+hardened+decorrelated | current |
| b1a5795a8e57ce30 | 2026-07-18T00:07:06Z | build+decorrelated                 | current |

4 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_catdata2, HOL4 development script bern\_tr\_catdata2Script.sml script bound

#### Theorem plirpdata\_receipt

```
Theorem plirpdata_receipt:
  ruledata0k [] cd2Hrelu
  <| pre := FAnd (FAtom BowLe (TmConst (-&1)) (TmVar 3))
    (FAtom BowLe (TmVar 3) (TmConst (&1))) ;
  post := FAnd (FAtom BowLe (affineWireTerm 0 0 3) (TmVar 4))
    (FAtom BowLe (TmVar 4) (affineWireTerm 1 0 3)) |>
  (RdLirpa (PLirpaD
    <| pre := FAnd (FAtom BowLe (TmConst (-&1)) (TmVar 3))
      (FAtom BowLe (TmVar 3) (TmConst (&1))) ;
    post := FAtom BowLe (TmConst (-&1)) (TmVar 3) |>
    (ProofObj (RdAndE (PAndEDL
      <| pre := FAnd (FAtom BowLe (TmConst (-&1)) (TmVar 3))
        (FAtom BowLe (TmVar 3) (TmConst (&1))) ;
      post := FAnd (FAtom BowLe (TmConst (-&1)) (TmVar 3))
        (FAtom BowLe (TmVar 3) (TmConst (&1))) |>
      (ProofObj RdAx)
      (FAtom BowLe (TmVar 3) (TmConst (&1))))))
    <| pre := FAnd (FAtom BowLe (TmConst (-&1)) (TmVar 3))
      (FAtom BowLe (TmVar 3) (TmConst (&1))) ;
    post := FAtom BowLe (TmVar 3) (TmConst (&1)) |>
    (ProofObj (RdAndE (PAndEDR
      <| pre := FAnd (FAtom BowLe (TmConst (-&1)) (TmVar 3))
        (FAtom BowLe (TmVar 3) (TmConst (&1))) ;
      post := FAnd (FAtom BowLe (TmConst (-&1)) (TmVar 3))
        (FAtom BowLe (TmVar 3) (TmConst (&1))) |>
      (ProofObj RdAx)
      (FAtom BowLe (TmConst (-&1)) (TmVar 3))))))
    3 4 (-&1) (&1) 0 0 0 (&1) 0)) ^
  ~ ruledata0k [] hnEmpty
  <| pre := FAnd (FAtom BowLe (TmConst (-&1)) (TmVar 3))
    (FAtom BowLe (TmVar 3) (TmConst (&1))) ;
  post := FAnd (FAtom BowLe (affineWireTerm 0 0 3) (TmVar 4))
    (FAtom BowLe (TmVar 4) (affineWireTerm 1 0 3)) |>
  (RdLirpa (PLirpaD
    <| pre := FTt ; post := FTt |> (ProofObj RdAx)
    <| pre := FTt ; post := FTt |> (ProofObj RdAx)
    3 4 (-&1) (&1) 0 0 0 (&1) 0))
```

## HOL verifier receipts

documentary hol-strict CLEAN: theory bern\_tr\_catdata2, 272 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of



the exact sources cited here (git blob cd3c1e95...). The proof-object stratum is ported as one mutual datatype family with a mutual wellformedness predicate, exactly as the Lean mutual inductive block; this rule's arm carries the two premise sub-proofs, the six sandwich inequalities verbatim, the application-list linkage (`reluLinkedB`, theory `bern_tr_catdata`, 252 theorems, 0 offences, same run — the lookup the earlier port excluded wholesale now has a ported realization), and the conclusion form (`lirpaConclusionFormB`, theory `bern_tr_catforms`, 23 theorems, 0 offences). The receipt theorem cited here is the kernel-evaluated accept/reject pair over a hypernet carrying the packaged ReLU application. The four bound lemmas behind the sandwich soundness are ported as well (`lirpa_lower_neg` / `lirpa_lower_pos` / `lirpa_upper_neg` / `lirpa_upper_pos`, theory `bern_tr_soundmid`, 16 theorems, 0 offences, run of 2026-07-22). Honest residue: the apex semantic theorem assembling them (the Lean development's `sound_lirpa`) is not yet ported; its ingredients above are.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. This section additionally carries the *family template* for the five ReLU rules (`LIRPA`, `RELU-POS`, `RELU-NEG`, `RELU-NN`, `RELU-ABOVE`): the packaged-node machinery they share is stated once here, and the four sibling sections instantiate it by reference.

**Recipe R19.1** (the ReLU family template). The general admissible template — one rule node rewrites to a derivation fragment over core and previously-justified rules with the same conclusion — is stated in full in the `PCONSTEXACT` section; the five ReLU rules are its instances over one shared semantic fact. The realized term grammar has no `relu` symbol: `relu` is a *packaged hypernet primitive*, an application  $\text{relu}_{\{h\} \rightarrow \{y\}} \in H$  with preactivation (input) wire  $h$  and output wire  $y$ . Every family rule cites it through one side condition, the *linkage*  $\text{reluLinked}(H, h, y)$ : the hypernet's application list contains a packaged ReLU at exactly these ports, of positive total size — a single decidable port lookup, not a pattern-match over composites. The packaged node's semantic contract is the elementwise two-case fact: under every admissible valuation, at every entry  $i$  of the node's (shared) tensor shape,

$$(h_i \leq 0 \wedge y_i = 0) \vee (0 \leq h_i \wedge y_i = h_i).$$

Three remarks fix what the template means and what it does not.

(i) *The rewrite shape.* Each family node rewrites to a case analysis over this disjunction: the sign-split fact enters the branch contexts (context-carrying  $\vee E$  — the disjunction arrives as a sub-conjunct of the premise; see the  $\vee E$  section's note), equality transport (`EqELIM`) carries the branch equality  $y = 0$  or  $y = h$  through the goal, and `ARITHATOM/NORM` plumbing normalizes atom orientations. Where a branch discharges fully into catalogue nodes the instance recipe displays the derivation (`RELU-NN` and `RELU-ABOVE`); where it does not — the boundary branches of `RELU-POS/RELU-NEG` need an antisymmetry step no catalogue rule provides — the instance says so and the rule is admissible by *reduction to verified machinery*, the honest tier note the `PCONSTEXACT` section established.

(ii) *The semantic contract and the conservativity warrant, kept distinct.* The two-case relation is the packaged symbol's *defined* semantics — part of what an admissible valuation of a hypernet containing the node means, at every entry of every positive-size application — and the family's soundness lemmas consume it as model fact at any size (that is (iii)'s extraction lemma). What the conservativity warrant adds is that the definition mints no genuinely new primitive: a definitional expansion into core primitives (a copy, the sign-split cell complex  $\Sigma^\pm$ , an indicator cast, and an exact rank-0 contraction — `reluExpH`), a typing rule derivable from the expansion's, and the projection theorem `relu_projection`: *both directions*, the two-case relation is exactly the expansion's model-level facts existentially projected onto the external ports — proven at the canonical single-entry instance. The boundary point  $h = 0$  rides the one-hot cell semantics: both cells of  $\Sigma^\pm$  admit it, and every admissible choice forces  $y = 0$ . Scope, stated loudly: applications of total size other than one currently have *no* core expansion (a recorded packing gap), so at those sizes the packaged node stands as a genuine primitive with the stated semantics — the five rules remain sound there, consuming the defined relation, while the not-a-new-primitive claim is certified only at size one.

(iii) *The shipped verified objects.* Per the catalogue's convention, each family rule ships a direct soundness lemma factoring through already-verified machinery rather than a derivation object: `reluLinked_facts` converts the linkage, under any admissible valuation, into (a) tensors at both ports, (b) one shared shape of positive entry count, and (c) the per-entry two-case relation; each rule's lemma is then an entrywise case analysis. The positive-size gate inside the linkage is deliberate hygiene: a size-zero node's comparison atoms hold vacuously, and vacuous surface must not be citable — the general lesson of the `FARKAS` rule's zero-size counterexample, enforced here at the source.

**Recipe R19.2** (the `LIRPA` instance — subdivide at the kink). The node, with its premise subtrees and side conditions:

$$\frac{\begin{array}{ccccccc} \mathcal{T}_a \triangleright \Phi \vdash a \leq h & \mathcal{T}_b \triangleright \Phi \vdash h \leq b & \text{relu}_{\{h\} \rightarrow \{y\}} \in H & a < 0 & 0 < b \\ \alpha a + \beta \leq 0 & \beta \leq 0 & \alpha b + \beta \leq b & 0 \leq \gamma a + \delta & 0 \leq \delta & b \leq \gamma b + \delta \\ \hline \Phi \vdash (\alpha h + \beta) \leq y \wedge y \leq (\gamma h + \delta) \end{array}}{\text{LIRPA}}$$

rewrites to a two-branch case analysis at the kink,

$$\frac{\mathcal{D}_- \triangleright \Phi \wedge (h \leq 0) \vdash (\alpha h + \beta) \leq y \wedge y \leq (\gamma h + \delta) \quad \mathcal{D}_+ \triangleright \Phi \wedge (0 \leq h) \vdash (\alpha h + \beta) \leq y \wedge y \leq (\gamma h + \delta)}{\Phi \vdash (\alpha h + \beta) \leq y \wedge y \leq (\gamma h + \delta)} \text{CASES}$$

where the case split is the box subdivision  $[a, b] = [a, 0] \cup [0, b]$  (region atoms enter from the order atoms via `CONVINTRO`; the cover is the one `SUBDIVIDE` checks), and the branches discharge as follows. In  $\mathcal{D}_-$ , the branch context proves  $h \leq 0$ , so `RELU-NEG`



yields  $y = 0$  (straight from the packaged contract: on  $h \leq 0$  both disjuncts force  $y = 0$ ); the two envelope atoms then reduce, through EQELIM along  $y = 0$ , to degree-1 bound facts on  $[a, 0] — \alpha h + \beta \leq 0$  and  $0 \leq \gamma h + \delta —$  whose degree-1 Bernstein certificates on that cell are *exactly* the checkpoint conditions at  $a$  and at  $0$  (BERNCONE at  $d = 1$ ; on a 1-simplex the degree-1 Bernstein coefficients of an affine gap polynomial are its two endpoint values). Dually in  $\mathcal{D}_+$ : RELU-Pos yields  $y = h$  (from the contract's active disjunct; at the boundary  $h = 0$  the inactive disjunct gives the same  $y = 0 = h$ ), and the envelope atoms reduce to degree-1 facts on  $[0, b] — \alpha h + \beta \leq h$  and  $h \leq \gamma h + \delta —$  whose Bernstein certificates are the checkpoint conditions at  $0$  and at  $b$ . The six side conditions are thus the six distinct endpoint values of the four affine gap polynomials, the kink checkpoints  $\beta \leq 0$  and  $0 \leq \delta$  each shared between their line's two cells. ARITHATOM, NORM, and CUT supply the atom-normalization and stitching plumbing throughout (the RELU-ABOVE section walks a chain of this kind in full). Scope of the rewrite, matching template remark (ii): this two-branch derivation is the canonical *single-entry* story — at larger tensor shapes the sign split is per-entry, so the displayed branch contexts  $h \leq 0$  and  $0 \leq h$  do not jointly cover a mixed-sign tensor and the derivation does not lift; the rule's warrant at every positive size is the direct semantic lemma (Proposition R19.3), per the family convention of remark (iii).

**Proposition R19.3** (semantic warrant). *Suppose the linkage holds, the two premise sequents are semantically valid with a common premise formula, and the eight arithmetic side conditions hold (the strict straddle pair  $a < 0, 0 < b$ , plus the six envelope checkpoints — “the six side conditions” elsewhere in this section always means the envelope six). Then the conclusion sequent — the pinned conjunction  $(\alpha h + \beta) \leq y \wedge y \leq (\gamma h + \delta)$  over the same premise — is semantically valid.*

*Proof.* Fix an admissible valuation  $\sigma$  satisfying  $\Phi$ . The premises, instantiated at  $\sigma$ , eliminate to entrywise bounds: every entry  $p$  of the preactivation tensor satisfies  $a \leq p \leq b$  (a rank-0 constant against a tensor compares against every entry). The linkage extraction (reluLinked\_facts) gives the output tensor at the same shape and, per entry, either  $p \leq 0 \wedge y = 0$  or  $0 \leq p \wedge y = p$ . Fix an entry and take the cases.

*Inactive case* ( $p \leq 0, y = 0$ ; here  $p \in [a, 0]$ ). Lower line:  $\alpha p + \beta \leq 0 = y$  by two-endpoint interpolation on  $[a, 0]$  — the affine function  $\alpha(\cdot) + \beta$  is  $\leq 0$  at  $a$  (checkpoint  $\alpha a + \beta \leq 0$ ) and at  $0$  (checkpoint  $\beta \leq 0$ ), hence on all of  $[a, 0]$  (lirpa\_lower\_neg; the verified proof is an explicit product certificate riding the identity  $(-a)(\alpha p + \beta) + (-p)(-(\alpha a + \beta)) + (p - a)(-\beta) = 0$ : on  $[a, 0]$  the last two products are nonnegative, so the assumed violation  $\alpha p + \beta > 0$  — which, with  $-a > 0$ , makes the first term strictly positive — is impossible in a zero sum). Upper line:  $y = 0 \leq \gamma p + \delta$  likewise from the checkpoints at  $a$  and  $0$  (lirpa\_upper\_neg). *Active case* ( $0 \leq p, y = p$ ; here  $p \in [0, b]$ ). Lower line:  $\alpha p + \beta \leq p = y$  by interpolation on  $[0, b]$  from the checkpoints  $\beta \leq 0$  and  $\alpha b + \beta \leq b$  (lirpa\_lower\_pos). Upper line:  $y = p \leq \gamma p + \delta$  from  $0 \leq \delta$  and  $b \leq \gamma b + \delta$  (lirpa\_upper\_pos).

The conclusion's two atoms evaluate entrywise at the shared shape (the affine terms evaluate to  $\beta + \alpha p$  and  $\delta + \gamma p$  per entry — affineWireTerm\_evalQ), so the four cases assemble into the conjunction. That is sound\_lirpa, whose structure the paragraph mirrors step for step.  $\square$

**Remark R19.4** (three checkpoints suffice). Because relu is affine on each side of  $0$  and  $a < 0 < b$ , an affine line bounds it on  $[a, b]$  iff it bounds it at the three points  $a, 0, b$  — the six side conditions are precisely these six facts (two lines  $\times$  three checkpoints), and equivalently the degree-1 Bernstein certificates of Recipe R19.2. A producer can therefore emit the tightest certifiable envelope for its chosen slopes by solving three scalar inequalities per line — nothing about the rule rewards loose coefficients.

**Remark R19.5** (what the strict straddle buys). With  $a < 0 < b$  the checkpoint conditions read, verbatim, “the line is on the correct side of relu at  $a, 0$ , and  $b$ ” (relu( $a$ ) =  $0$  needs  $a \leq 0$ ; relu( $b$ ) =  $b$  needs  $0 \leq b$ ). Semantic soundness would in fact survive relaxing the strictness: whatever the signs of  $a \leq b$ , the six conditions pin the lower line beneath relu on  $[\min(a, 0), \max(b, 0)] \supseteq [a, b]$  — at  $a$  because  $\alpha a + \beta \leq 0 \leq \text{relu}(a)$  holds unconditionally, and on each of  $[\min(a, 0), 0], [0, \max(b, 0)]$  by the same endpoint interpolation; for the upper line the interval containment does the work (the checks at  $0$  and  $b$  dominate relu on  $[0, \max(b, 0)]$ , the checks at  $a$  and  $0$  dominate it on  $[\min(a, 0), 0]$ ). The rule as adopted nevertheless demands the strict straddle, and reasonably: it pins LIRPA to the genuinely unstable case — a stable neuron rides RELU-Pos or RELU-Neg *exactly*, losing nothing to a relaxation — it keeps both interpolation cells nondegenerate, and it is consumed as stated by the verified interpolation lemmas, whose contradiction products multiply by  $-a$  and  $b$ .

**Remark R19.6** (the conclusion is built, not audited). The conclusion formula is pinned syntactically (lirpaConclusionForm): it is the conjunction of the two comparison atoms over the affine terms affineWireTerm  $\beta \alpha h$  and affineWireTerm  $\delta \gamma h$ , constructed from the payload coefficients — the producer supplies no conclusion shape. A payload field that the checker rebuilds rather than reads is one the payload cannot mis-declare (the PCONSTEXACT section's closedness observation, in conclusion-shape form): no “equivalent” respelling can smuggle in a different wire, comparator, or shape.

| proof step                           | formal anchor                                                   |
|--------------------------------------|-----------------------------------------------------------------|
| family linkage extraction            | reluLinked_facts                                                |
| checkpoint interpolation, lower line | lirpa_lower_neg, lirpa_lower_pos                                |
| checkpoint interpolation, upper line | lirpa_upper_neg, lirpa_upper_pos                                |
| affine conclusion evaluation         | affineWireTerm_evalQ                                            |
| rule warrant (Proposition R19.3)     | sound_lirpa                                                     |
| linkage side condition               | reluLinked; PrimApp.isReluAt (the positive-size gate)           |
| pinned conclusion                    | lirpaConclusionForm; affineWireTerm                             |
| packaged-node conservativity         | relu_projection over reluExpH (canonical single-entry instance) |

Literature: the two-sided affine relaxation of ReLU is the verification community's workhorse — the triangle relaxation [66], its dual/convex-outer form [242], CROWN's adaptive slope choice [246], DeepPoly's abstract-domain rendering [220], and the LIRPA framework that named the family [243]. This rule is the checker-side certificate form: the cited works *find* envelopes; the rule only ever *checks* one, at six rational inequalities.

## Rule 20. ReLU-Pos (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *ReLU-Pos*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash h \geq 0 \quad \text{relu}_{\{h\} \rightarrow \{y\}} \in H}{\Phi \vdash y = h} \text{ReLU-Pos}$$

ReLU-Pos is the exactness rule for the active side of a ReLU: once  $\Phi \vdash h \geq 0$  is proven for the preactivation wire of a cited ReLU application, the output wire is not merely bounded but *equal* —  $\Phi \vdash y = h$ . A producer reaches for it when earlier bound propagation has shown a neuron stably active (a proven lower box bound at or above zero): the lossy affine envelope of LiRPA is replaced by an identity, and the equality feeds EQELIM, so every fact known about  $h$  transports to  $y$  verbatim.

What the checker recomputes, given the payload (premise proof, the two wires): the linkage — the hypernet really contains a packaged ReLU application at exactly these ports, of positive total size (the family's port-lookup side condition, displayed in the LiRPA section); that the premise's conclusion is literally the atom  $h \geq 0$  over the same premise formula; and that the conclusion is literally  $y = h$ . Both atoms are elementwise over the node's tensor shape.

Drop the premise and the rule is false at any entry with  $h < 0$  (there  $y = 0 \neq h$ ). Drop the linkage and  $y$  is an arbitrary wire with no semantic tie to  $h$  — the linkage-free rule is unsound over unlinked wires, and a realization without the port lookup could only be kept sound by rejecting every certificate. The interesting hypothesis is the one that is *not* there: no side condition excludes the boundary  $h = 0$ , because the node's two semantic cases agree at the kink — the proof section makes this the centre of the argument.

**notes** TIER NOTE: the unpacking by case analysis over the sign-split fact discharges the active branch cleanly, but its boundary branch needs the antisymmetry step  $h \leq 0 \wedge h \geq 0 \Rightarrow h = 0$ , and no catalogue rule concludes an equality from a pair of opposite inequalities — so, like PCONSTEXACT, this rule is admissible by *reduction to verified machinery* rather than by a complete catalogue derivation: the shipped warrant is the direct lemma `sound_reluPos`, whose boundary case is one antisymmetry step in the model. The proof section states the enumeration behind that claim.

### Lean statement

The warrant consumes the family linkage extraction (`reluLinked_facts`, displayed in the LiRPA section) and the premise, eliminated entrywise (the elimination lemma is displayed alongside); the payload structure is `PreLUPosData`, quoted in full below.

BernsteinLogic.Soundness . **sound\_reluPos**

statement bound

```
/-- ReLU-Pos: from `Φ ⊢ h ≥ 0` conclude `Φ ⊢ y = h` — on each entry the
premise forces the nonpositive case's input to 0, where both cases
agree. -/
theorem sound_reluPos {H : Hypernet} {seq premise : Sequent}
  {preact post : Wire}
  (ih : premise.semanticallyValid H)
  (hlink : reluLinked H preact post)
  (hpre : seq.pre = premise.pre)
  (hprePost : premise.post = .atom .ge (.var preact) (.const 0))
  (hpost : seq.post = .atom .eq (.var post) (.var preact)) :
  seq.semanticallyValid H
```

BernsteinLogic.Soundness . **QTensor.rel2\_const\_right\_elim**

statement bound

```
/-- rel2 against a rank-0 constant on the right, eliminated to the
all-entries relation. -/
theorem QTensor.rel2_const_right_elim {r : Q → Q → Prop} {q : QTensor}
  {c : Q} (h : QTensor.rel2 r q {[], [c], rfl}) :
  ∀ x ∈ q.entries, r x c
```

BernsteinLogic.Core.Catalogue . **PreLUPosData**

definition bound

```

inductive PReLUPosData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (preact post : Wire)
  (link : reluLinked H preact post)
  (preEq      : seq.pre = premise.pre)
  (premisePost : premise.post = .atom .ge (.var preact) (.const 0))
  (conclPost  : seq.post  = .atom .eq (.var post) (.var preact))
  : PReLUPosData H seq

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; sub-derivation premise as at **WEAKEN**; linkage premise — including the positive-size narrowing hidden under the displayed  $\in$  — exactly as in the **LiRPA** dictionary, the family preamble.

- $h \mapsto \text{preact}; y \mapsto \text{post}; \text{relu}_{\{h\} \rightarrow \{y\}} \in H \mapsto \text{hlink} : \text{reluLinked } H \text{ preact post}$  (payload alias: `PReLUPosData.link`, same proposition).
- premise  $\Phi \vdash h \geq 0 \mapsto \text{ih}$  with `hprempost : premise.post = .atom .ge (.var preact) (.const 0)` — displayed comparator and operand order carried exactly, with no sign-folding (in contrast to the **CONVELIM** family's  $\text{all-}\leq$  build output);  $\Phi$  shared via `hpre`. Payload aliases, same propositions:  $\text{ih} \leftrightarrow \text{premiseProof}$  (over the shared premise),  $\text{hpre} \leftrightarrow \text{preEq}$ ,  $\text{hprempost} \leftrightarrow \text{premisePost}$ .
- conclusion  $\Phi \vdash y = h \mapsto$  the goal under `hpost : seq.post = .atom .eq (.var post) (.var preact)` (payload alias: `conclPost`, same equation) — output wire left, preactivation right, as displayed.

Width note: the certified atoms are elementwise over the cited application's tensor shape (batch semantics, as at **LiRPA**); the boundary  $h = 0$  needs no side condition — the node's two semantic cases agree at the kink, by one antisymmetry step in the model (the commentary and tier note). Beyond the family's linkage gate, neither side is wider: every hypothesis of `sound_reluPos` is consumed.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_catdata2, HOL4 development script bern\_tr\_catdata2Script.sml script bound

### Theorem preluposdata\_receipt

```
Theorem preluposdata_receipt:
  ruledata0k [] cd2Hrelu
    <| pre := FAtom BowGe (TmVar 3) (TmConst 0) ;
      post := FAtom BowEq (TmVar 4) (TmVar 3) |>
    (RdReluPos (PReluPosD
      <| pre := FAtom BowGe (TmVar 3) (TmConst 0) ;
        post := FAtom BowGe (TmVar 3) (TmConst 0) |> (ProofObj RdAx)
      3 4)) ∧
  ¬ ruledata0k [] hnEmpty
    <| pre := FAtom BowGe (TmVar 3) (TmConst 0) ;
      post := FAtom BowEq (TmVar 4) (TmVar 3) |>
    (RdReluPos (PReluPosD
      <| pre := FAtom BowGe (TmVar 3) (TmConst 0) ;
        post := FAtom BowGe (TmVar 3) (TmConst 0) |> (ProofObj RdAx)
      3 4))
```

## HOL verifier receipts

documentary hol-strict CLEAN: theory bern\_tr\_catdata2, 272 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob cd3c1e95...). The proof-object stratum is ported as one mutual datatype family with a mutual wellformedness predicate, exactly as the Lean mutual inductive block; this rule's arm states its entire checkable content — the premise post-form, the family linkage, and the conclusion equation — and the linkage lookup into the hypernet's application list has a ported realization (reluLinkedB, theory bern\_tr\_catdata, 252 theorems, 0 offences, same run). The receipt theorem cited here is the kernel-evaluated pair: an accept over a hypernet carrying the packaged ReLU application, and a refusal over one without it — linkage doing exactly its job. Honest residue: the semantic soundness twin (the Lean development's sound\_reluPos) is not yet ported; the payload/checker layer above is.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2). The family machinery — the packaged node, the linkage, the two-case semantic fact, and its conservativity warrant — is stated in the LiRPA section's family template and is not restated here.

**Recipe R20.1** (the ReLU-Pos instance). The node, and its intended rewrite over the family's sign-split fact:

$$\frac{\mathcal{T} \triangleright \Phi \vdash h \geq 0 \quad \text{relu}_{\{h\} \rightarrow \{y\}} \in H}{\Phi \vdash y = h} \text{ReLU-Pos} \rightsquigarrow \text{case analysis on } (h \leq 0 \wedge y = 0) \vee (0 \leq h \wedge y = h).$$

The *active* branch discharges cleanly: its context carries the conjunct  $y = h$  (the case disjunct enters the branch premise as one ordered subtree, so two  $\wedge E$  steps reach it), which  $Ax$  and  $\wedge E$  extract — the goal verbatim. The *boundary* branch does not: its context carries  $h \leq 0$  and  $y = 0$ , the premise supplies  $h \geq 0$ , and the goal  $y = h$  now needs the antisymmetry fact  $h = 0$  — which, as **Remark R20.3** shows, no catalogue derivation that withholds this rule itself provides. The rewrite therefore does *not* fully discharge into catalogue nodes; like PCONSTEXACT, the rule is admissible by reduction to verified machinery: the shipped warrant is the direct lemma sound\_reluPos, in which the boundary case is a single antisymmetry step in the model ( $p \leq 0$  from the case,  $0 \leq p$  from the premise, so  $p = 0$  and the two cases agree).

**Proposition R20.2** (semantic warrant). *Suppose the linkage holds and the premise sequent — same premise formula, conclusion the atom  $h \geq 0$  — is semantically valid. Then the sequent concluding  $y = h$  is semantically valid.*

*Proof.* Fix an admissible valuation  $\sigma$  satisfying  $\Phi$ . The premise eliminates entrywise: every entry  $p$  of the preactivation tensor satisfies  $0 \leq p$ . The linkage extraction (reluLinked\_facts) gives the output tensor at the same shape and, per entry, either  $p \leq 0 \wedge y = 0$  or  $0 \leq p \wedge y = p$ . In the second case  $y = p$  is the goal entry. In the first,  $p \leq 0$  meets the premise's  $0 \leq p$ , so  $p = 0$  (antisymmetry — the model step the catalogue lacks) and  $y = 0 = p$ . The conclusion atom is the entrywise equality of the two tensors at their shared shape. That is sound\_reluPos, step for step.  $\square$

*Remark R20.3* (the kink case has no catalogue derivation). The claim in Recipe [R20.1](#) is an enumeration over the catalogue’s equality-capable conclusions. `NORM` and `EQREFL` conclude only canonical-form identities ( $h$  and  $0$  are not canonically equal); `EQSYM`, `EQELIM`, and `ARITHATOM` at the `=` comparator only *transport* equalities already proven; this family and the cell rules conclude equalities only over their packaged applications’ ports; the combination rules (`FARKAS`, `SYZGY`, `PBOUND`, `BERNCONE` with an order predicate) conclude order atoms, never `=`; and `BERNCONE`’s degenerate conic-predicate route would need a region atom pinning  $h$  to the single point  $0$ , which no rule derives — region atoms denote regions of positive-dimensional simplices. The catalogue at large *does* reach the fact, and the route is instructive: over the two-wire hypernet carrying only the linked ReLU application, `ReLU-NEG` (premise  $h \leq 0$ , an `AX` extraction) concludes  $y = 0$ , this rule (premise  $h \geq 0$ ) concludes  $y = h$ , and one `EQELIM` along  $y = h$  rewrites  $y = 0$  into  $h = 0$  — the linked application is itself the antisymmetry oracle, its output forced to agree with both branches at the kink. But the elaboration needs the fact *without the rule being eliminated* — the leaf-first species of the admissible template consumes every rule except that one — and with either envelope rule withheld the supply dries up. Write  $d(t = u)$  for the canonical form of  $t - u$ , and consider a smallest derivation, withholding one envelope rule, that concludes an equality atom with  $d$  not a polynomial multiple of the surviving port difference ( $y$  if `ReLU-Pos` is withheld with `ReLU-NEG`’s  $y = 0$  surviving;  $y - h$  mirrored). Its final step cannot be the surviving port conclusion or a canonical-form identity ( $d$  a multiple of the port difference, respectively  $0$ ); not `EQSYM`, the zero-form respelling, or `EQELIM` transport (each changes  $d$  by a sign or by polynomial multiples of a previously concluded equality’s own  $d$  — a smaller derivation, hence a permitted multiple); not a closed ground fact ( $d = 0$ , the constants being exactly evaluated); not a combination or region rule (order and region conclusions only); and not a structural or case-analysis step, which only re-exposes an equality already concluded, or sitting in a premise set — the root premises ( $h \leq 0 \wedge h \geq 0$ , at either spelling) contain no equality, premise sets grow only by `CUT` formulas and case disjuncts concluded or contained in smaller derivations, and a case split whose smuggled disjunct carries the target is idle: its sibling branch must conclude the same target without it, a smaller offender. The target  $d(h = 0) = h$  is a multiple of neither  $y$  nor  $y - h$ , so  $h \leq 0 \wedge h \geq 0 \vdash h = 0$  has no `ReLU-Pos`-free derivation (nor, mirrored, a `ReLU-NEG`-free one), and the boundary branch cannot discharge into nodes. This is not a defect to repair but a fact about where the calculus keeps its arithmetic: an elaboration can source order antisymmetry only from the verified model step, one line of `sound_reluPos`, rather than from a rule every certificate would pay wire format for.

*Remark R20.4* (what the premise is for). Without  $\Phi \vdash h \geq 0$  the rule is false at any entry with  $h < 0$ : there the node’s semantics forces  $y = 0 \neq h$ . The premise is consumed *entrywise* — a tensor-shaped preactivation must be proven nonnegative in every entry, which is what a rank- $0$  constant against a tensor atom asserts. No side condition excludes the boundary  $h = 0$ , and none is needed: the two semantic cases agree there, which is exactly the antisymmetry step above.

| proof step                           | formal anchor                                                       |
|--------------------------------------|---------------------------------------------------------------------|
| Recipe <a href="#">R20.1</a> warrant | <code>sound_reluPos</code>                                          |
| family linkage extraction            | <code>reluLinked_facts</code> (see the <code>LiRPA</code> section)  |
| entrywise premise elimination        | <code>QTensor.relu_const_right_elim</code>                          |
| the kink antisymmetry                | the <code>le_antisymm</code> step inside <code>sound_reluPos</code> |
| payload structure                    | <code>PReLUPosData</code> (premise proof, two wires, linkage)       |

## Rule 21. ReLU-NEG (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *ReLU-Neg*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash h \leq 0 \quad \text{relu}_{\{h\} \rightarrow \{y\}} \in H}{\Phi \vdash y = 0} \text{ReLU-NEG}$$

ReLU-NEG is the exactness rule for the inactive side of a ReLU, ReLU-Pos's mirror: once  $\Phi \vdash h \leq 0$  is proven for the preactivation wire of a cited ReLU application, the output is pinned to a constant —  $\Phi \vdash y = 0$ . A producer reaches for it when bound propagation has shown a neuron stably inactive (a proven upper box bound at or below zero): the neuron is dead on the region of interest, and the equality lets EQELIM rewrite  $y$  to the literal constant 0 downstream — dead-neuron elimination in certificate form, at the rank-0 instances the constant-sided transport gate admits (the family's shape-scope remark).

What the checker recomputes, given the payload (premise proof, the two wires): the linkage — a packaged ReLU application at exactly these ports, of positive total size (the family's port-lookup side condition, displayed in the LiRPA section); that the premise's conclusion is literally the atom  $h \leq 0$  over the same premise formula; and that the conclusion is literally  $y = 0$ . Both atoms are elementwise over the node's tensor shape.

Drop the premise and the rule is false at any entry with  $h > 0$  (there  $y = h \neq 0$ ). Drop the linkage and an arbitrary wire is asserted zero. As with ReLU-Pos, the boundary  $h = 0$  needs no exclusion: both semantic cases of the node force  $y = 0$  there.

**notes** TIER NOTE: as for ReLU-Pos, and for the mirrored reason — the unpacking's active branch would need  $h = 0$  from  $h \geq 0 \wedge h \leq 0$ , an antisymmetry step no catalogue rule provides — the rule is admissible by reduction to verified machinery: the shipped warrant is the direct lemma `sound_reluNeg`. The enumeration argument is stated once, in the ReLU-Pos proof section.

### Lean statement

The warrant consumes the family linkage extraction (`reluLinked_facts`, displayed in the LiRPA section) and the premise, eliminated entrywise (the elimination lemma is displayed alongside); the payload structure is `PreLUNegData`, quoted in full below.

BernsteinLogic.Soundness . **sound\_reluNeg**

statement bound

```
/-- ReLU-Neg: from `Φ ⊢ h ≤ 0` conclude `Φ ⊢ y = 0` — on each entry the
premise forces the nonnegative case's input to 0, where both cases
agree. -/
theorem sound_reluNeg {H : Hypernet} {seq premise : Sequent}
  {preact post : Wire}
  (ih : premise.semanticallyValid H)
  (hlink : reluLinked H preact post)
  (hpre : seq.pre = premise.pre)
  (hprePost : premise.post = .atom .le (.var preact) (.const 0))
  (hpost : seq.post = .atom .eq (.var post) (.const 0)) :
  seq.semanticallyValid H
```

BernsteinLogic.Soundness . **QTensor.rel2\_const\_right\_elim**

statement bound

```
/-- rel2 against a rank-0 constant on the right, eliminated to the
all-entries relation. -/
theorem QTensor.rel2_const_right_elim {r : Q → Q → Prop} {q : QTensor}
  {c : Q} (h : QTensor.rel2 r q ([], [c], rfl)) :
  ∀ x ∈ q.entries, r x c
```

BernsteinLogic.Core.Catalogue . **PreLUNegData**

definition bound

```
inductive PreLUNegData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (preact post : Wire)
  (link : reluLinked H preact post)
```



```



```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; sub-derivation premise as at WEAKEN; linkage premise — including the positive-size narrowing hidden under the displayed  $\in$  — exactly as in the LiRPA dictionary, the family preamble.

- $h \mapsto \text{preact}; y \mapsto \text{post}; \text{relu}_{\{h\} \rightarrow \{y\}} \in H \mapsto \text{hlink} : \text{reluLinked } H \text{ preact post}$  (payload alias: `PReLUegData.link`, same proposition).
- premise  $\Phi \vdash h \leq 0 \mapsto \text{ih}$  with `hprempost : premise.post = .atom .le (.var preact) (.const 0)` — displayed comparator and operand order carried exactly;  $\Phi$  shared via `hpre`. Payload aliases, same propositions:  $\text{ih} \leftrightarrow \text{premiseProof}$  (over the shared premise),  $\text{hpre} \leftrightarrow \text{preEq}$ ,  $\text{hprempost} \leftrightarrow \text{premisePost}$ .
- conclusion  $\Phi \vdash y = 0 \mapsto$  the goal under `hpost : seq.post = .atom .eq (.var post) (.const 0)` (payload alias: `conclPost`, same equation) — output wire left, constant right, as displayed.

Width note: certified atoms elementwise over the cited application's tensor shape (batch semantics, as at LiRPA); the boundary  $h = 0$  is again condition-free — the mirrored antisymmetry step of the tier note. Beyond the family's linkage gate, neither side is wider: every hypothesis of `sound_reluNeg` is consumed.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **stale only**

hardened (safe\_verify (hardened screen)) **stale only**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement



theory bern\_tr\_catdata2, HOL4 development script bern\_tr\_catdata2Script.sml script bound**Theorem prelunegdata\_receipt**

```

Theorem prelunegdata_receipt:
  ruledata0k [] cd2Hrelu
    <| pre := FAtom BowLe (TmVar 3) (TmConst 0) ;
      post := FAtom BowEq (TmVar 4) (TmConst 0) |>
    (RdReluNeg (PReLUNegD
      <| pre := FAtom BowLe (TmVar 3) (TmConst 0) ;
        post := FAtom BowLe (TmVar 3) (TmConst 0) |> (ProofObj RdAx)
      3 4)) ∧
  ¬ ruledata0k [] cd2Hrelu
    <| pre := FAtom BowLe (TmVar 3) (TmConst 0) ;
      post := FAtom BowEq (TmVar 4) (TmVar 3) |>
    (RdReluNeg (PReLUNegD
      <| pre := FAtom BowLe (TmVar 3) (TmConst 0) ;
        post := FAtom BowLe (TmVar 3) (TmConst 0) |> (ProofObj RdAx)
      3 4))

```

**HOL verifier receipts**

documentary hol-strict CLEAN: theory bern\_tr\_catdata2, 272 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob cd3c1e95...). The proof-object stratum is ported as one mutual datatype family with a mutual wellformedness predicate, exactly as the Lean mutual inductive block; this rule's arm states its entire checkable content — the premise post-form, the family linkage, and the zero-clamp conclusion equation — and the linkage lookup into the hypernet's application list has a ported realization (reluLinkedB, theory bern\_tr\_catdata, 252 theorems, 0 offences, same run). The receipt theorem cited here is the kernel-evaluated pair: an accept over a hypernet carrying the packaged ReLU application, and a wrong-conclusion refusal. Honest residue: the semantic soundness twin (the Lean development's sound\_reluNeg) is not yet ported; the payload/checker layer above is.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

**Proof**

Genre: admissible rule (tier 2). The family machinery is stated in the LiRPA section's family template; ReLU-Pos is this rule's mirror, and the shared boundary argument is stated there.

**Recipe R21.1** (the ReLU-NEG instance). The node, and its intended rewrite over the family's sign-split fact:

$$\frac{\mathcal{T} \triangleright \Phi \vdash h \leq 0 \quad \text{relu}_{\{h\} \rightarrow \{y\}} \in H}{\Phi \vdash y = 0} \text{ReLU-NEG} \rightsquigarrow \text{case analysis on } (h \leq 0 \wedge y = 0) \vee (0 \leq h \wedge y = h).$$

The *inactive* branch discharges cleanly: its context carries the conjunct  $y = 0$  (one ordered subtree of the branch premise; two  $\wedge E$  steps), extracted by Ax and  $\wedge E$  — the goal verbatim. The *boundary* branch is the mirror of ReLU-Pos's: its context carries  $0 \leq h$  and  $y = h$ , the premise supplies  $h \leq 0$ , and the goal  $y = 0$  needs  $h = 0$  — the antisymmetry fact that no catalogue derivation withholding this rule itself provides (the enumeration argument, covering both withheld orientations, is stated once, as **Remark R20.3** in the ReLU-Pos section). The rule is therefore admissible by reduction to verified machinery: the shipped warrant is the direct lemma sound\_reluNeg, whose boundary case is the same one-line antisymmetry step in the model.

**Proposition R21.2** (semantic warrant). *Suppose the linkage holds and the premise sequent — same premise formula, conclusion the atom  $h \leq 0$  — is semantically valid. Then the sequent concluding  $y = 0$  is semantically valid.*

*Proof.* Fix an admissible valuation  $\sigma$  satisfying  $\Phi$ . The premise eliminates entrywise: every entry  $p$  of the preactivation tensor satisfies  $p \leq 0$ . The linkage extraction (reluLinked\_facts) gives, per entry, either  $p \leq 0 \wedge y = 0$  — the goal entry directly — or  $0 \leq p \wedge y = p$ , in which case  $p = 0$  by antisymmetry against the premise, so  $y = p = 0$ . The conclusion atom compares the output tensor entrywise against the rank-0 constant 0. That is sound\_reluNeg, step for step.  $\square$

**Remark R21.3** (what the premise is for, and what the rule feeds). Without  $\Phi \vdash h \leq 0$  the rule is false at any entry with  $h > 0$  (there  $y = h \neq 0$ ). Downstream, the conclusion differs from ReLU-Pos's in kind:  $y = 0$  pins the output to a *literal constant*, so EqELIM can rewrite  $y$  away — dead-neuron elimination inside a certificate — at the rank-0 instances the transport gate admits (a constant-sided equality rides EqELIM's shared-broadcast-shape condition; the family's shape-scope remark, ReLU-NN section), after which ground arithmetic on the constant rides PCONSTEXACT.

| proof step                           | formal anchor                                    |
|--------------------------------------|--------------------------------------------------|
| Recipe <a href="#">R21.1</a> warrant | sound_reluNeg                                    |
| family linkage extraction            | reluLinked_facts (see the LiRPA section)         |
| entrywise premise elimination        | QTensor.rel2_const_right_elim                    |
| the kink antisymmetry                | the le_antisymm step inside sound_reluNeg        |
| payload structure                    | PReLUNegData (premise proof, two wires, linkage) |

## Rule 22. ReLU-NN (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *ReLU-NN*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{relu}_{\{h\} \rightarrow \{y\}} \in H}{\Phi \vdash y \geq 0} \text{ReLU-NN}$$

ReLU-NN is the range fact: the output wire of a cited ReLU application is nonnegative,  $\Phi \vdash y \geq 0$ , with no proof premises at all —  $\Phi$  is arbitrary (the premise-free convention shared with NORM and PCONSTEXACT). It is the first thing a producer knows about every ReLU output: the lower half of every output box starts here, before any information about the preactivation exists, and together with ReLU-ABOVE it yields the exact pointwise lower envelope of the activation ( $y \geq \max(0, h)$ , as two atoms).

What the checker recomputes is nearly everything, because there is nearly nothing: the linkage — the hypernet really contains a packaged ReLU application at exactly these ports, of positive total size (the family's port-lookup side condition, displayed in the LiRPA section) — and that the conclusion is literally the atom  $y \geq 0$ , elementwise over the node's tensor shape.

The linkage is the only hypothesis, and dropping it is the whole failure mode: an arbitrary wire asserted nonnegative, against an arbitrary premise — the linkage-free rule is unsound over unlinked wires, and a realization without the port lookup could only be kept sound by rejecting every certificate. The positive-size gate inside the linkage keeps even the vacuous-surface corner unciteable (a size-zero node satisfies every elementwise atom trivially; the linkage refuses it at the source).

**notes** This is the family's cleanest unpacking: the case analysis over the sign-split fact discharges *both* branches into catalogue nodes — the inactive branch's leaf is the ground fact  $0 \geq 0$ , which rides PCONSTEXACT (the one rule that can state a closed ground fact). The proof section displays the full intended derivation (its case-closing disjunction elimination is the context-carrying form — a satellite theorem today, so the display is an elaboration blueprint rather than an emittable certificate), and states its shape scope honestly: the constant-sided equality transport type-checks at rank-0 instances, and the shape-general warrant is the direct lemma.

### Lean statement

The warrant consumes the family linkage extraction (`reluLinked_facts`, displayed in the LiRPA section): each output entry is either 0 or a copy of a nonnegative input entry. The payload structure, `PreLUNNData` — quoted in full below — carries no premise field.

BernsteinLogic.Soundness . **sound\_reluNN**

statement bound

```
/-- ReLU-NN: `Φ ⊢ y ≥ 0`, premise-free — every entry of the packaged
output is 0 or a nonnegative copy of the input entry. (A review finding
exhibited the rule as unsound over unbound wires; the linkage is the repair.) -/
theorem sound_reluNN {H : Hypernet} {seq : Sequent} {preact post : Wire}
  (hlink : reluLinked H preact post)
  (hpost : seq.post = .atom .ge (.var post) (.const 0)) :
  seq.semanticallyValid H
```

BernsteinLogic.Core.Catalogue . **PreLUNNData**

definition bound

```
inductive PreLUNNData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (preact post : Wire)
  (link : reluLinked H preact post)
  (conclPost : seq.post = .atom .ge (.var post) (.const 0))
  : PreLUNNData H seq
```

### Correspondence (Steele display ↔ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the

three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; linkage premise — including the positive-size narrowing hidden under the displayed  $\in$  — exactly as in the LiRPA dictionary, the family preamble.

- $h \mapsto \text{preact}; y \mapsto \text{post}; \text{relu}_{\{h\} \rightarrow \{y\}} \in H \mapsto \text{hlink} : \text{reluLinked } H \text{ preact post}$  (payload alias: `PreLUNNData.link`, same proposition). The display's universality over the preactivation — no premise constrains  $h$  — is carried by the linkage alone: `preact` occurs in no other hypothesis (the roster note records that the appendix payload named only the output wire — the certified linkage names both ports).
- premise-freeness  $\mapsto$  silence, the `PConstExact` representation: the precondition component of `seq : Sequent` is unconstrained by any hypothesis, so the statement holds under arbitrary premises  $\Phi$ .
- conclusion  $\Phi \vdash y \geq 0 \mapsto$  the goal under `hpost : seq.post = .atom .ge (.var post) (.const 0)` (payload alias: `conclPost`, same equation) — displayed comparator and operand order carried exactly.

Width note: certified atoms elementwise over the cited application's tensor shape (batch semantics, as at LiRPA). Beyond the family's linkage gate, neither side is wider: both hypotheses of `sound_reluNN` are consumed, and nothing else is stated.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory `bern_tr_catdata2`, HOL4 development script `bern_tr_catdata2Script.sml` script bound

### Theorem `prelunndata_receipt`

```
Theorem prelunndata_receipt:
  ruledata0k [] cd2Hrelu
    <| pre := FTt ; post := FAtom BowGe (TmVar 4) (TmConst 0) |>
    (RdReluNN (PReluNND 3 4)) ^
  ~ ruledata0k [] hnEmpty
    <| pre := FTt ; post := FAtom BowGe (TmVar 4) (TmConst 0) |>
    (RdReluNN (PReluNND 3 4))
```

## HOL verifier receipts

**documentary** hol-strict CLEAN: theory bern\_tr\_catdata2, 272 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob cd3c1e95...). The proof-object stratum is ported as one mutual datatype family with a mutual wellformedness predicate, exactly as the Lean mutual inductive block; this premise-free rule's arm states its entire checkable content — the family linkage plus the nonnegativity conclusion equation — and the linkage lookup into the hypernet's application list has a ported realization (reluLinkedB, theory bern\_tr\_catdata, 252 theorems, 0 offences, same run). The receipt theorem cited here is the kernel-evaluated accept/refusal pair for the arm. Honest residue: the semantic soundness twin (the Lean development's sound\_reluNN) is not yet ported; the payload/checker layer above is.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2). The family machinery is stated in the LiRPA section's family template. This rule is premise-free:  $\Phi$  is arbitrary and never consulted (the NORM convention).

**Recipe R22.1** (the ReLU-NN instance — the full unpacking). The node:

$$\frac{\text{relu}_{\{h\} \rightarrow \{y\}} \in H}{\Phi \vdash y \geq 0} \text{ReLU-NN} \quad \rightsquigarrow \quad \text{case analysis on } (h \leq 0 \wedge y = 0) \vee (0 \leq h \wedge y = h),$$

and both branches discharge into catalogue nodes — the family's cleanest unpacking. *Inactive branch* (context carries  $y = 0$ ):

1.  $\vdash 0 \geq 0$  — PCONSTEXACT, the closed exact ground fact (the leaf that rule exists to state: no other catalogue rule concludes a closed ground fact);
2.  $\Phi_- \vdash y = 0$  —  $\text{Ax} + \wedge\text{E}$  from the branch context (the case disjunct enters as one ordered subtree; two  $\wedge\text{E}$  steps); then  $\Phi_- \vdash 0 = y$  by EQSYM;
3.  $\Phi_- \vdash y \geq 0$  — EQELIM along  $0 = y$ , transporting  $\Psi[a := 0] = (0 \geq 0)$  (step 1, weakened) to  $\Psi[a := y]$ , where  $\Psi = (a \geq 0)$ .

*Active branch* (context carries  $0 \leq h$  and  $y = h$ ): the context's sign atom is extracted ( $\text{Ax} + \wedge\text{E}$ ) and re-oriented to  $h \geq 0$  by the bound node's polarity-flip reading (PBOUND, reading (G): the single component  $0 \leq h$  with weight 1, canonical polynomial  $c = -h$ , read as  $e \geq 0$  with  $e = h$  — one node of the toolkit whose longer chains the ReLU-ABOVE section walks); EQSYM flips the context's  $y = h$  to  $h = y$ , and EQELIM along  $h = y$  (an equality between two wires of the node's one shared shape) transports it to  $y \geq 0$ .  $\vee\text{E}$  in its context-carrying form — today a satellite theorem, not a core constructor (the split rule's section prices exactly this), so what follows is the intended elaboration, emittable when that constructor lands — closes the case analysis, and CUT stitches the sign-split fact into place.

**Remark R22.2** (the shape scope of the displayed derivation). The realized EQELIM carries a broadcast-shape side condition (its section documents the counterexample that forced it): both sides of the transported equality must share a broadcast shape. Step 3's equality  $0 = y$  has a rank-0 constant on one side, so the displayed transport type-checks where the output wire is rank-0 — at tensor-shaped instances that step is stopped by the gate, and the family's realized warrant does not route through the derivation at all: it is the direct lemma sound\_reluNN, shape-general, via the linkage extraction. (Wire-to-wire transports such as the active branch's  $h = y$  are unaffected: the linkage forces one shared shape.) The derivation above is thus the intended admissibility story at scalar instances — intended, since its case-closing step awaits the constructor named at its step — and the reduction-to-verified-machinery story, the PCONSTEXACT pattern stated in the family template, is the standing warrant at every shape either way.

**Proposition R22.3** (semantic warrant). *If the linkage holds, the sequent concluding  $y \geq 0$  is semantically valid over any premise formula  $\Phi$ .*

*Proof.* Fix an admissible valuation  $\sigma$  ( $\sigma \models \Phi$  is taken but never consulted — the rule is premise-free). The linkage extraction (reluLinked\_facts) gives the output tensor and, per entry, either  $y = 0$  or  $y = p$  with  $0 \leq p$ ; in both cases  $0 \leq y$ . The conclusion atom compares the output tensor entrywise against the rank-0 constant 0. That is sound\_reluNN, step for step.  $\square$

**Remark R22.4** (the only hypothesis). The linkage is the rule's sole side condition — the conclusion-shape check beside it is a reconstruction pin, not a droppable hypothesis: the checker rebuilds the atom  $y \geq 0$  rather than trusting it, so “dropping” the pin merely re-addresses the rule at a conclusion it never concluded. Dropping the linkage is the whole failure mode: an arbitrary wire asserted nonnegative against an arbitrary premise. Its positive-size gate additionally keeps the vacuous corner unciteable — a size-zero node satisfies every elementwise atom trivially, and a fact no valuation can falsify must not enter a derivation (the family template's hygiene remark).

### proof step

Recipe R22.1 warrant  
family linkage extraction  
ground-fact leaf (step 1)  
payload structure

### formal anchor

sound\_reluNN  
reluLinked\_facts (see the LiRPA section)  
sound\_pConstExact (the PCONSTEXACT section)  
PRELUNNData (two wires, linkage — no premise field)

## Rule 23. ReLU-ABOVE (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *ReLU-Above*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{relu}_{\{h\} \rightarrow \{y\}} \in H}{\Phi \vdash y \geq h} \text{ReLU-ABOVE}$$

ReLU-ABOVE is the identity minorant: the output wire of a cited ReLU application dominates its preactivation,  $\Phi \vdash y \geq h$ , with no proof premises —  $\Phi$  arbitrary, the premise-free convention. Together with ReLU-NN it gives the exact pointwise lower envelope of the activation as two always-available atoms ( $y \geq 0$  and  $y \geq h$ ); a producer needs no box knowledge at all to emit either. Upper bounds are different in kind — no affine upper bound on the activation exists without a box, which is why the upper side of the family lives in LiRPA behind two box premises.

What the checker recomputes: the linkage — the hypernet really contains a packaged ReLU application at exactly these ports, of positive total size (the family's port-lookup side condition, displayed in the LiRPA section) — and that the conclusion is literally the atom  $y \geq h$ , elementwise over the node's tensor shape.

The linkage is the only hypothesis; without it, two arbitrary wires are asserted ordered. Entrywise the fact is immediate from the node's two-case semantics: on the inactive branch  $y = 0 \geq h$  (there  $h \leq 0$ ), on the active branch  $y = h$ .

**notes** The proof section uses this rule's derivation as the family's worked example of atom-orientation plumbing: its inactive branch discharges into catalogue nodes through a chain of a bound-node polarity flip, a NORM identity, an ARITHATOM fold, and two EQELIM transports — a faithful specimen of what “one node unpacks to many” means in practice.

### Lean statement

The warrant consumes the family linkage extraction (`reluLinked_facts`, displayed in the LiRPA section): entrywise,  $y = 0 \geq h$  on the inactive case and  $y = h$  on the active one. The payload structure, `PRELUAboveData` — quoted in full below — carries no premise field.

`BernsteinLogic.Soundness . sound_reluAbove`

statement bound

```
-- ReLU-Above: `Φ ⊢ y ≥ h`, premise-free — entrywise, `y = 0 ≥ h` on
the nonpositive case and `y = h` on the nonnegative one. -/
theorem sound_reluAbove {H : Hypernet} {seq : Sequent} {preact post : Wire}
  (hlink : reluLinked H preact post)
  (hpost : seq.post = .atom .ge (.var post) (.var preact)) :
  seq.semanticallyValid H
```

`BernsteinLogic.Core.Catalogue . PRELUAboveData`

definition bound

```
inductive PRELUAboveData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (preact post : Wire)
  (link : reluLinked H preact post)
  (conclPost : seq.post = .atom .ge (.var post) (.var preact))
  : PRELUAboveData H seq
```

### Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; linkage premise — including the positive-size narrowing hidden under the displayed  $\in$  — exactly as in the LiRPA dictionary, the family preamble.

- $h \mapsto \text{preact}; y \mapsto \text{post}; \text{relu}_{\{h\} \rightarrow \{y\}} \in H \mapsto \text{hlink} : \text{reluLinked } H \text{ preact post}$  (payload alias: `PRELUAboveData.Link`, same proposition).
- premise-freeness  $\mapsto$  silence, the `PConstExact` representation: the precondition component of `seq : Sequent` is unconstrained by any hypothesis, so the statement holds under arbitrary premises  $\Phi$  — the display's universality over the preactivation rides the linkage, as at `ReLU-NN`.
- conclusion  $\Phi \vdash y \geq h \mapsto$  the goal under `hpost : seq.post = .atom .ge (.var post) (.var preact)` (payload alias: `conclPost`, same equation) — output wire left, preactivation right, as displayed; the only two-wire atom among the family's premise-free conclusions, so this dictionary is where the shared shape of the two tensors (from the linkage extraction) is load-bearing rather than convenient.

Width note: certified atoms elementwise over the cited application's tensor shape (batch semantics, as at `LiRPA`). Beyond the family's linkage gate, neither side is wider: both hypotheses of `sound_reluAbove` are consumed, and nothing else is stated.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

`build` (lake build (elaboration)) current

`replay` (leanchecker (official kernel replay)) stale only

`hardened` (safe\_verify (hardened screen)) stale only

`decorrelated` (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory `bern_tr_catdata2`, HOL4 development script `bern_tr_catdata2Script.sml` script bound

### Theorem `preluabovedata_receipt`

```
Theorem preluabovedata_receipt:
  ruledata0k [] cd2Hrelu
    <| pre := FTt ; post := FAtom BowGe (TmVar 4) (TmVar 3) |>
    (RdReluAbove (PREluAboveD 3 4)) ^
  ~ ruledata0k [] cd2Hrelu
    <| pre := FTt ; post := FAtom BowGe (TmVar 3) (TmVar 4) |>
    (RdReluAbove (PREluAboveD 3 4))
```

## HOL verifier receipts



**documentary** hol-strict CLEAN: theory bern\_tr\_catdata2, 272 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob cd3c1e95...). The proof-object stratum is ported as one mutual datatype family with a mutual wellformedness predicate, exactly as the Lean mutual inductive block; this premise-free rule's arm states its entire checkable content — the family linkage plus the output-dominates-preactivation conclusion equation — and the linkage lookup into the hypernet's application list has a ported realization (reluLinkedB, theory bern\_tr\_catdata, 252 theorems, 0 offences, same run). The receipt theorem cited here is the kernel-evaluated accept/refusal pair for the arm. Honest residue: the semantic soundness twin (the Lean development's sound\_reluAbove) is not yet ported; the payload/checker layer above is.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2). The family machinery is stated in the LiRPA section's family template. This rule is premise-free:  $\Phi$  is arbitrary and never consulted.

**Recipe R23.1** (the ReLU-ABOVE instance — worked plumbing). The node:

$$\frac{\text{relu}_{\{h\} \rightarrow \{y\}} \in H}{\Phi \vdash y \geq h} \quad \text{ReLU-ABOVE} \quad \rightsquigarrow \quad \text{case analysis on } (h \leq 0 \wedge y = 0) \vee (0 \leq h \wedge y = h).$$

We walk the *inactive* branch in full — the family's honest specimen of what “one node unpacks to many” means in catalogue practice. Context  $\Phi_-$  carries  $h \leq 0$  and  $y = 0$ ; the goal is  $y \geq h$ .

1.  $\Phi_- \vdash h \leq 0$  —  $\text{Ax} + \wedge\text{E}$  (the case disjunct enters as one ordered subtree; two  $\wedge\text{E}$  steps).
2.  $\Phi_- \vdash -h \geq 0$  — the bound node's polarity-flip reading (PBOUND, reading (G)): the combination is the single component  $h \leq 0$  with weight 1, canonical polynomial  $c = h$ , read as  $e \geq 0$  with  $e = -h$ .
3.  $\Phi_- \vdash (-h) = (0 - h)$  — NORM: equal monomial canonical forms over  $\mathbb{Q}$ , same wires, shared broadcast shape.
4.  $\Phi_- \vdash (0 - h) \geq 0$  — EQELIM along step 3's equality (both sides shaped alike — the transport gate is met), rewriting step 2's atom.
5.  $\Phi_- \vdash 0 \geq h$  — ARITHATOM (fold): from  $(t_1 - t_2) \bowtie 0$  to  $t_1 \bowtie t_2$  at  $t_1 = 0, t_2 = h, \bowtie = \geq$ .
6.  $\Phi_- \vdash 0 = y$  — EQSYM on the context's  $y = 0$ ; then  $\Phi_- \vdash y \geq h$  — EQELIM along  $0 = y$  at  $\Psi = (a \geq h)$ .

The *active* branch (context  $0 \leq h, y = h$ ) derives  $h \geq h$  by the same pattern, spelled at the same grain: NORM gives  $(h - h) = 0$ , whence  $0 = (h - h)$  by EQSYM; PCONSTEXACT supplies the ground fact  $0 \geq 0$ , weakened into the branch context; EQELIM along  $0 = (h - h)$  transports it to  $(h - h) \geq 0$ , and ARITHATOM folds that to  $h \geq h$  — then EQSYM on the context's  $y = h$  and EQELIM along  $h = y$  (wire-to-wire, one shared shape) close the goal. Six arithmetic-plumbing nodes in the inactive branch — PBOUND, NORM, EQELIM, ARITHATOM, EQSYM, EQELIM; the extraction's Ax and  $\wedge\text{E}$  steps aside — to move one minus sign across a comparator: this is why the family rules exist as single nodes, and why the wire format prices them.

**Remark R23.2** (shape scope). Step 6 of the inactive branch and the active branch's route through  $(h - h) = 0$  transport along equalities with a rank-0 constant side, so the displayed derivation type-checks at rank-0 instances (step 4's transport along  $(-h) = (0 - h)$  is wire-shaped on both sides — the constant 0 sits *inside* the term  $0 - h$  — and is unaffected); the shape-general warrant is the direct lemma, exactly as **Remark R22.2** in the ReLU-NN section states for the whole family.

**Proposition R23.3** (semantic warrant). *If the linkage holds, the sequent concluding  $y \geq h$  is semantically valid over any premise formula  $\Phi$ .*

*Proof.* Fix an admissible valuation  $\sigma$ . The linkage extraction (reluLinked\_facts) gives both tensors at one shared shape and, per entry, either  $p \leq 0 \wedge y = 0$  — whence  $y = 0 \geq p$  — or  $y = p$  — whence  $y \geq p$  reflexively. The conclusion atom compares the two tensors entrywise at their shared shape. That is sound\_reluAbove, step for step.  $\square$

**Remark R23.4** (the exact lower envelope, premise-free). Together with ReLU-NN this rule states the exact pointwise lower envelope of the activation,  $y \geq \max(0, h)$ , as two atoms a producer can emit with *no* box knowledge whatsoever — exact in that the bound is attained:  $\max(0, h)$  is the activation's value, so no sound lower bound can improve on the pair. The asymmetry with the upper side is structural, not an artefact: no affine function dominates  $\max(0, h)$  on all of  $\mathbb{Q}$  (a candidate with positive slope dives below 0 as  $h \rightarrow -\infty$ ; one with slope  $\leq 0$  falls below  $h$  as  $h \rightarrow +\infty$ ) — so upper bounds require a box and live in LiRPA.

### proof step

Recipe R23.1 warrant  
family linkage extraction  
polarity-flip reading (step 2)  
ground-fact leaf (active branch)  
payload structure

### formal anchor

sound\_reluAbove  
reluLinked\_facts (see the LiRPA section)  
sound\_pBound (the PBOUND section)  
sound\_pConstExact (the PCONSTEXACT section)  
PReLUAboveData (two wires, linkage — no premise field)

## Rule 24. CELL-IN-INTERIOR (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Cell-In-Interior*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash \bigwedge_v \lambda_v^{C_i}(x) \bowtie_v^\circ 0 \text{ (strict barycentric interior atoms, flat vertex order; } < \text{ for extended } v, > \text{ for core } v) \quad \text{cell}[\Sigma]_{\{x\} \rightarrow \{b_1, \dots, b_N\}} \in H}{\Phi \vdash b_i = 1 \wedge \bigwedge_{j \neq i} b_j = 0} \quad \text{CELL-IN-INTERIOR}$$

CELL-IN-INTERIOR is one of the two rules that read the cell primitive — the calculus's only source of disjunction. A cell application  $\text{cell}[\Sigma]_{\{x\} \rightarrow \{b_1, \dots, b_N\}}$  tabulates which cell of a simplicial decomposition its input point lies in, as one-hot indicator outputs; piecewise definitions (a piecewise-linear activation, a case-defined dynamics) enter the hypernet through it. This rule discharges the case a proof is actually in: from a premise placing  $x$  *strictly inside* cell  $C_i$  — a conjunction of strict barycentric inequalities in the pinned flat vertex order — it concludes the full one-hot vector,  $b_i = 1$  and  $b_j = 0$  for every  $j \neq i$ . A producer reaches for it after region reasoning (the region introduction rules, SUBDIVIDE) has confined the input to one cell's interior, and needs the indicator values as arithmetic facts downstream — typically to substitute a branch's closed form for the primitive's output.

What the checker recomputes, given the payload (schema, cell index, cited application, premise proof): that the cited cell application is present in the hypernet with the input wire typed at shape `[ambDim]` and every output at total size one; that the premise formula is byte-for-byte the schema-derived strict-interior build for cell  $i$  (barycentric atoms recomputed from the vertex coordinates — no formula text trusted from the wire); and that the conclusion is the recomputed one-hot form. Soundness then rides the model: admissibility of the cell application forces some cell choice  $k$  with weak membership and the one-hot outputs at  $k$ ; the strict premise pins  $k = i$  through the schema's interior-disjointness well-formedness.

Each hypothesis is load-bearing. Weaken strictness to weak membership and a boundary point of two cells admits a valuation one-hotting the *other* cell — the conclusion fails. Drop interior-disjointness (distinct cells of the enumeration have disjoint interiors — which the complex's simplicial conformity does NOT imply: conformity never sees the extension masks and permits duplicate cell entries) and the model may one-hot an overlapping cell. Drop the output typing and the model's entry-level one-hot fact decouples from the whole-wire equality atoms (a zero-size output satisfies every atom vacuously; a multi-entry output can falsify the atom away from entry zero). Drop the input-shape tie and the premise's constraint on the input tensor no longer speaks about the model's evaluation point. The proof section walks these in order.

**notes** The checkable fragment is 1-dimensional (two-vertex cell supports on distinct coordinates) — the same fragment as the region machinery; everything beyond is refused, never approximated. The intended catalogue elaboration (case analysis over the cell axiom's disjunction) awaits the context-carrying case-analysis constructor; the shipped warrant is a direct semantic lemma, the degenerate-rewrite shape the PCONSTEXACT section defines.

### Lean statement

BernsteinLogic.Soundness . **sound\_cellInInterior**

statement bound

```
-- Cell-In-Interior: the strict-interior premise pins the model's cell
-- choice to whichI (via `cellIntDisjoint`), and the sp-fact's one-hot
-- valuation IS the conclusion. -/
theorem sound_cellInInterior {H : Hypernet} {seq premise : Sequent}
  {schema : SimpC} {input : Wire} {outputs : List Wire}
  {whichI : Fin schema.cells.length}
  (ih : premise.semanticallyValid H)
  (happ : PrimApp.cell schema input outputs ∈ H.applications)
  (hinshape : ∃ τ, H.wireType input = some τ ∧
    τ.shape = [schema.ambDim])
  (houts : ∀ o ∈ outputs, ∃ τ, H.wireType o = some τ ∧
    Shape.totalSize τ.shape = 1)
  (hpre : seq.pre = premise.pre)
  (hint : cellStrictInteriorForm schema (.var input) whichI premise.post)
  (hpost : cellOneHotForm schema outputs whichI seq.post) :
  seq.semanticallyValid H
```

## BernsteinLogic.Soundness .cellStrictInteriorBuild\_spec

statement bound

```

/-- Extraction for the strict-interior premise: the build succeeds only
over a 1-D complex with a two-vertex cell mask on distinct coordinates,
and the formula is the pinned pair of strict barycentric atoms. -/
theorem cellStrictInteriorBuild_spec {schema : SimpC} {t : Term}
  {k : Fin schema.cells.length} {F : Formula}
  (h : cellStrictInteriorForm schema t k F) :
  schema.ambDim = 1 ∧
  ∃ (a b : Fin schema.nVertices) (V₀ V₁ : ℚ),
  (List.finRange schema.nVertices).filter
    (fun v => (schema.cells.get k).1 v) = [a, b] ∧
  (∀ j, schema.coord a j = V₀) ∧
  (∀ j, schema.coord b j = V₁) ∧
  V₀ ≠ V₁ ∧
  F = .and
    (.atom (if (schema.cells.get k).2 a then .lt else .gt)
      (.add (.const (V₁ / (V₁ - V₀))) (.mul (.const (-1 / (V₁ - V₀))) t))
      (.const 0))
    (.atom (if (schema.cells.get k).2 b then .lt else .gt)
      (.add (.const (-V₀ / (V₁ - V₀))) (.mul (.const (1 / (V₁ - V₀))) t))
      (.const 0))

```

## BernsteinLogic.Soundness .maskRegionStrict\_pair\_intro

statement bound

```

/-- Construction: rational weights of strict signs assemble a
`MaskRegionStrict` witness over a two-vertex 1-D support (the strict
twin of `maskRegion_pair_intro`). Needs `hscope : ext ⊆ mask` – the
cellsWF invariant – because the strict region demands  $\lambda_m < 0$  at every
ext vertex, and support vertices are the only ones with nonzero  $\lambda_m$ 
(the weak twin's  $0 \leq \lambda_m$  escape is unavailable at strict signs). -/
theorem maskRegionStrict_pair_intro {C : SimpC}
  {mask ext : Fin C.nVertices → Bool}
  {x : Fin C.ambDim → ℚ} {a b : Fin C.nVertices} {V₀ V₁ : ℚ}
  (hfil : (List.finRange C.nVertices).filter (fun v => mask v) = [a, b])
  (hscope : ∀ v, ext v = true → mask v = true)
  (hVa : ∀ j, C.coord a j = V₀) (hVb : ∀ j, C.coord b j = V₁)
  (hne : V₀ ≠ V₁) (hdim : C.ambDim = 1) (j0 : Fin C.ambDim)
  (hw0t : ext a = true → (V₁ - x j0) / (V₁ - V₀) < 0)
  (hw0f : ext a = false → 0 < (V₁ - x j0) / (V₁ - V₀))
  (hw1t : ext b = true → (x j0 - V₀) / (V₁ - V₀) < 0)
  (hw1f : ext b = false → 0 < (x j0 - V₀) / (V₁ - V₀)) :
  MaskRegionStrict C mask ext x

```

## BernsteinLogic.Soundness .holds\_cellOneHot

statement bound

```

/-- The one-hot conclusion holds at any valuation whose output entries
realize the one-hot pattern at k (unit-size output typing carries the
entry facts to full wire reads). -/
theorem holds_cellOneHot {H : Hypernet} (σ : H.Valuation) {schema : SimpC}
  {outputs : List Wire} {k : Fin schema.cells.length} {F : Formula}
  (hbuild : cellOneHotForm schema outputs k F)
  (houts : ∀ o ∈ outputs, ∃ τ, H.wireType o = some τ ∧
    Shape.totalSize τ.shape = 1)
  (hsem : ∀ j : Fin outputs.length,
    σ.entryFlat (outputs.get j) 0
    = some (.fin (if j.val = k.val then 1 else 0))) :
  Formula.holds σ F

```

**The recomputed forms and the payload.** The premise and conclusion formulas are never read off the certificate as text: the checker rebuilds both from the schema — the strict-interior conjunction from the 1-D barycentric coordinate terms (`baryLambdas1D`, offset-first affine normal forms over the two-vertex support), the conclusion from the one-hot pattern — and the payload's form fields assert equality with the rebuilt formulas. A skeptical reader should confirm by eye: the builds succeed only over a 1-dimensional complex with exactly two support vertices at distinct coordinates (everything else reads none and the rule refuses); the strict comparators are  $<$  at extended vertices and  $>$  at core vertices; and the payload carries the port-typing obligations (input at `[ambDim]`, outputs at total size one) explicitly.

BernsteinLogic.Core.Catalogue . **baryLambdas1D**

definition bound

```

/-- The barycentric coordinate functions of a 1-dimensional simplex, as
canonical affine Terms in `t` (pinned shape `offset + slope·t`; see the
original for the full derivation notes and D2 scope). -/
def baryLambdas1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool)
  (t : Term) : Option (List (Fin cx.nVertices × Term)) :=
  if h : cx.ambDim = 1 then
    let j0 : Fin cx.ambDim := (0, by omega)
    match (List.finRange cx.nVertices).filter (fun v => dContains v) with
    | [a, b] =>
      let V0 := cx.coord a j0
      let V1 := cx.coord b j0
      if V0 = V1 then none
      else
        let den := V1 - V0
        let lam := fun (c s : Rat) => Term.add (.const c) (.mul (.const s) t)
        some [(a, lam (V1 / den) (-1 / den)), (b, lam (-V0 / den) (1 / den))]
    | _ => none
  else none

```

BernsteinLogic.Core.Catalogue . **cellStrictInteriorBuild**

definition bound

```

/-- Strict-interior conjunction (flat vertex-order convention, strict
bowties). -/
def cellStrictInteriorBuild (schema : SimpC) (t : Term)
  (k : Fin schema.cells.length) : Option Formula := do
  let c := schema.cells[k.val]'(k.isLt)
  let lams ← baryLambdas1D schema c.1 t
  match lams.map (fun (v, lt) =>
    Formula.atom (if c.2 v then .lt else .gt) lt (.const 0)) with
  | [x, y] => some (.and x y)
  | _ => none

```

BernsteinLogic.Core.Catalogue . **cellStrictInteriorForm**

definition bound

```

def cellStrictInteriorForm (schema : SimpC) (t : Term)
  (k : Fin schema.cells.length) (F : Formula) : Prop :=
  cellStrictInteriorBuild schema t k = some F

```

BernsteinLogic.Core.Catalogue . **cellOneHotBuild**

definition bound

```

/-- One-hot conclusion: `(b_k = 1) ∧ chainAnd [(b_j = 0) | j ≠ k]`. -/
def cellOneHotBuild (schema : SimpC) (outputs : List Wire)
  (k : Fin schema.cells.length) : Option Formula :=
  if outputs.length = schema.cells.length then
    let bk := Formula.atom .eq (.var (outputs.getD k.val 0)) (.const 1)
    let others := ((List.range outputs.length).filter (· ≠ k.val)).map
      fun j => Formula.atom .eq (.var (outputs.getD j 0)) (.const 0)
    some (.and bk (Formula.chainAnd others))
  else none

```

BernsteinLogic.Core.Catalogue . **cellOneHotForm**

definition bound

```

def cellOneHotForm (schema : SimpC) (outputs : List Wire)
  (k : Fin schema.cells.length) (F : Formula) : Prop :=
  cellOneHotBuild schema outputs k = some F

```

BernsteinLogic.Core.Catalogue . **PCellInIntData**

definition bound

```

inductive PCellInIntData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (schema : SimpC) (input : Wire) (outputs : List Wire)
  (whichI : Fin schema.cells.length)
  (appInH : PrimApp.cell schema input outputs ∈ H.applications)

```

```

(inputShape :  $\exists \tau, H.\text{wireType input} = \text{some } \tau \wedge$ 
   $\tau.\text{shape} = [\text{schema.ambDim}]$ )
(outputsScalar :  $\forall o \in \text{outputs}, \exists \tau, H.\text{wireType o} = \text{some } \tau \wedge$ 
   $\text{Shape.totalSize } \tau.\text{shape} = 1$ )
(preEq : seq.pre = premise.pre)
(interiorPremise : cellStrictInteriorForm schema (.var input) whichI
  premise.post)
(postOneHot : cellOneHotForm schema outputs whichI seq.post)
: PCellInIntData H seq

```

**The semantic side: strict regions and interior disjointness.** MaskRegionStrict is the strict-sign twin of the weak region membership the model's cell fact carries; SimpC.CellIntDisjoint is the well-formedness the pinning argument consumes, stated over cell *indices* (so duplicate entries are excluded) at rational points (the model's cell inputs are rational). It is consumed as a conjunct of the cell application's model-level fact: an application over a non-disjoint enumeration admits no valuation — hypothesis-side vacuity, sound but weak — rather than an unsound conclusion. The predicate is a proposed schema amendment realized provisionally; the roster page records its status.

BernsteinLogic.Catalogue . MaskRegionStrict

definition bound

```

/-- Strict-interior membership: the same weights with STRICT signs on
every mask vertex (< 0 extended, > 0 core) – the region the
Cell-In-Interior premise certifies. -/
def MaskRegionStrict (C : SimpC) (mask ext : Fin C.nVertices → Bool)
  (x : Fin C.ambDim →  $\mathbb{Q}$ ) : Prop :=
   $\exists \text{ lam} : \text{Fin C.nVertices} \rightarrow \mathbb{R},$ 
  ( $\forall v, \text{mask } v = \text{false} \rightarrow \text{lam } v = 0$ )  $\wedge$ 
   $\text{finSum C.nVertices lam} = 1 \wedge$ 
  ( $\forall j : \text{Fin C.ambDim},$ 
     $\text{finSum C.nVertices } (\text{fun } v \Rightarrow \text{lam } v * (\text{C.coord } v \text{ j} : \mathbb{R})) = (\text{x j} : \mathbb{R})) \wedge$ 
  ( $\forall v, \text{ext } v = \text{true} \rightarrow \text{lam } v < 0$ )  $\wedge$ 
  ( $\forall v, \text{mask } v = \text{true} \rightarrow \text{ext } v = \text{false} \rightarrow 0 < \text{lam } v$ )

```

BernsteinLogic.Catalogue . SimpC.CellIntDisjoint

definition bound

```

/-- PROPOSED-B2-DISJOINT (provisional): distinct cells of a schema's
enumeration have disjoint interiors – no rational point lies strictly
inside cell i and in the CLOSED region of a different cell k. This is
what the cell rules' admissibility derivation cites
("conformity: distinct cells have disjoint interiors"), and the
schema's `conformity` field does NOT deliver it: that field constrains
simplex-mask intersections only – it never sees the ( $\Delta, S$ ) extension
masks, and it permits literally duplicate cell entries, either of which
makes Cell-In-Interior UNSOUND (the model may one-hot a different
overlapping cell). Stated over INDICES so duplicates are excluded, at
 $\mathbb{Q}$  points because the model's Cell inputs are rational.  $\Sigma \pm$  satisfies it
(the two rays share only the origin, strictly inside neither).

```

Lives here (not as a `SimpC` field) because the quantification is over REAL weights – mathlib-side of the core/mathlib module split – and is consumed as a

conjunct of the Cell sp-fact below: a cell application over a non-disjoint schema admits NO valuation, the same "soundness errors are excluded from every value set" reading that rejects NaN executions (a hypothesis-side vacuity: sound, merely weak). -/

```

def SimpC.CellIntDisjoint (C : SimpC) : Prop :=
   $\forall i \text{ k} : \text{Fin C.cells.length}, i \neq k \rightarrow$ 
   $\forall x : \text{Fin C.ambDim} \rightarrow \mathbb{Q},$ 
  MaskRegionStrict C (C.cells.get i).1 (C.cells.get i).2 x  $\rightarrow$ 
   $\neg \text{MaskRegion C (C.cells.get k).1 (C.cells.get k).2 x}$ 

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as

a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in **AX**; sub-derivation premise as at **WEAKEN**.

- $\Sigma \mapsto \text{schema} : \text{SimpC}; i \mapsto \text{whichI} : \text{Fin } \text{schema.cells.length}; x \mapsto \text{the term } .\text{var } \text{input}$  — the region term literally is the input wire, as at **BERNCONE**;  $b_1, \dots, b_N \mapsto \text{outputs} : \text{List Wire}$ .
- the linkage  $\text{cell}[\Sigma]_{\{x\} \rightarrow \{b_1, \dots, b_N\}} \in H \mapsto \text{happ}$ , plain application-list membership — contrast the **LiRPA** family's size-gated lookup: here the port discipline rides two explicit typing hypotheses instead, **hinshape** (the input wire's type has shape  $[\text{schema.ambDim}]$ ) and **houtS** (every output wire singleton-sized). Neither has a displayed row; the payload carries both verbatim (**PCellInIntData**'s port-typing fields, displayed above); they are load-bearing twice: the model's sp-fact pins entry 0 of each output while the conclusion atom evaluates the whole wire, and the premise atoms constrain the input tensor, which ties to the model's rational point only at the declared shape. The same undisplayed typing pack as at **NORM**.
- premise  $\Phi \vdash \bigwedge_v \lambda_v^{C_i}(x) \bowtie_v^0 0 \mapsto \text{ih}$  with **hint** : **cellStrictInteriorForm** **schema** (.var **input**) **whichI** **premise.post** — the premise formula is pinned as the *build's* output: the two-atom conjunction in flat vertex order, comparators  $<$  on extended vertices and  $>$  on core ones exactly as displayed, each atom comparing the barycentric entry to .const 0. The build succeeds only at **schema.ambDim** = 1, so the certified reach of the displayed rule is the 1-dimensional fragment, by refusal (the **CONVELIM/CONVINTRO** class; the note above says it, this triple certifies it).  $\Phi$  shared via **hpre**.
- conclusion  $\Phi \vdash b_i = 1 \wedge \bigwedge_{j \neq i} b_j = 0 \mapsto \text{the goal under hpost} : \text{cellOneHotForm } \text{schema} \text{ outputs } \text{whichI } \text{seq.post}$  — pinned as the one-hot build's output: the  $b_i = 1$  atom first, then the ascending- $j$  chain of zeros; the displayed big-wedge has exactly one accepted spelling.
- nowhere in the certificate: the interior-disjointness well-formedness. It is consumed from the *model side* — admissible valuations carry **SimpC.CellIntDisjoint** in the cell sp-fact (certified alongside), and the proof pins the model's cell choice with it — so the display is right to show no such premise, and the rule's soundness is honestly conditional on that provisional schema conjunct (the commentary's **PROPOSED-B2-DISJOINT** story, visible in the statement chain).

Width note: within the 1-dimensional fragment, and modulo the undisplayed port-typing pack flagged above, nothing certified is narrower or wider than the display — every hypothesis is consumed, and the two builds pin both formulas to single spellings.

## Lean verifier receipts

module **BernsteinLogic.Soundness**; bound build **a20d90e66c13...** (bound from receipt **a9c0e34d3e3c9c8c**, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                 | minted               | layers             | status  |
|-------------------------|----------------------|--------------------|---------|
| <b>a9c0e34d3e3c9c8c</b> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module **BernsteinLogic.Core.Catalogue**; bound build **cbc92e976a1d...** (bound from receipt **5abdb3589707b5df**, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                 | minted               | layers             | status  |
|-------------------------|----------------------|--------------------|---------|
| <b>24804e66edba0bc9</b> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <b>5abdb3589707b5df</b> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module **BernsteinLogic.Catalogue**; bound build **c477b6015133...** (bound from receipt **d87d31fb92f86564**, 2026-07-20T07:43:40Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                 | minted               | layers             | status  |
|-------------------------|----------------------|--------------------|---------|
| <b>d87d31fb92f86564</b> | 2026-07-20T07:43:40Z | build+decorrelated | current |
| <b>85a439e3f753d74e</b> | 2026-07-20T20:44:30Z | build+decorrelated | current |

|                  |                      |                                    |         |
|------------------|----------------------|------------------------------------|---------|
| 1e680b8d5ca91dae | 2026-07-20T21:00:59Z | build+replay+hardened+decorrelated | current |
| c707aa2f41b0f58f | 2026-07-21T05:28:24Z | build+replay+hardened+decorrelated | current |

4 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_catdata2, HOL4 development script bern\_tr\_catdata2Script.sml script bound

#### Theorem pcellinintdata\_receipt

```

Theorem pcellinintdata_receipt:
  ruledata0k cd2wsCell cd2Hcell
  <| pre := FAnd (FAtom BowLt (TmAdd (TmConst (&1))
                                     (TmMul (TmConst (-&1)) (TmVar 9)))
                (TmConst 0))
    (FAtom BowGt (TmAdd (TmConst (&0))
                        (TmMul (TmConst (&1)) (TmVar 9)))
    (TmConst 0)) ;
  post := FAnd (FAtom BowEq (TmVar 5) (TmConst 1))
    (FAtom BowEq (TmVar 7) (TmConst 0)) |>
  (RdCellInInterior (PCellInIntD
    <| pre := FAnd (FAtom BowLt (TmAdd (TmConst (&1))
                                     (TmMul (TmConst (-&1))
                                             (TmVar 9)))
                (TmConst 0))
    (FAtom BowGt (TmAdd (TmConst (&0))
                        (TmMul (TmConst (&1))
                              (TmVar 9)))
    (TmConst 0)) ;
    post := FAnd (FAtom BowLt (TmAdd (TmConst (&1))
                                     (TmMul (TmConst (-&1))
                                             (TmVar 9)))
                (TmConst 0))
    (FAtom BowGt (TmAdd (TmConst (&0))
                        (TmMul (TmConst (&1))
                              (TmVar 9)))
    (TmConst 0)) |> (ProofObj RdAx)
  checkfnDemoSchema 9 [5; 7] 0)) ^
  ~ ruledata0k cd2wsCell hnEmpty
  <| pre := FTt ; post := FTt |>
  (RdCellInInterior (PCellInIntD
    <| pre := FTt ; post := FTt |> (ProofObj RdAx)
  checkfnDemoSchema 9 [5; 7] 0))

```

## HOL verifier receipts

documentary hol-strict CLEAN: theory bern\_tr\_catdata2, 272 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob cd3c1e95...). The formerly dropped cell-payload island is ported: this rule's arm of the mutual wellformedness predicate states the application membership, the input-shape gate, the strict-interior premise and the one-hot conclusion (the form faces as boolean Build-equations, the characterization convention), and the form characterizations themselves are ported in theory bern\_tr\_catforms (cellStrictInteriorForm\_char / cellOneHotForm\_char, 23 theorems, 0 offences, same run). The receipt theorem cited above is the arm's kernel-evaluated accept/refusal pair; the shared barycentric ingredient (baryCoeffs1D) was already ported and receipted. Honest residue: the cell soundness argument (the Lean development's sound\_cellInInterior) is not yet ported; the payload/builder layer above is.



HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. The intended elaboration is a case analysis over the cell primitive's one-hot disjunction (the  $\text{sp}[\text{cell}[\Sigma]]$  axiom — the calculus's only source of disjunction); the *realized* warrant is the degenerate rewrite in the  $\text{PCONSTEXACT}$  sense, discharging against a named verified semantic lemma. Both are presented, and the honest reason the first is not yet a checkable catalogue derivation is stated.

**Recipe R24.1** (the intended rewrite — case analysis over the cell disjunction). A cell application  $\text{cell}[\Sigma]_{\{x\} \rightarrow \{b_1, \dots, b_N\}}$  contributes to the ambient premise its one-hot disjunction

$$\text{sp}[\text{cell}[\Sigma]] = \bigvee_{k=1}^N \left( x \in \overline{C_k} \wedge b_k = 1 \wedge \bigwedge_{j \neq k} b_j = 0 \right).$$

The intended unpacking of one  $\text{CELL-IN-INTERIOR}$  node is a case split on  $k$ : in branch  $k = i$  the conclusion  $b_i = 1 \wedge \bigwedge_{j \neq i} b_j = 0$  is reached from the case hypothesis by conjunction plumbing ( $\wedge E$  and  $\wedge I$  under whatever association the axiom's serialization fixes); in a branch  $k \neq i$  the case hypothesis  $x \in \overline{C_k}$  contradicts the rule's strict-interior premise  $x \in \overset{\circ}{C}_i$  — *provided no point lies strictly inside one cell of the enumeration and weakly inside another*, which is exactly the schema well-formedness the soundness theorem consumes ([Remark R24.5](#)). Two honest gaps keep this a template rather than a shipped derivation: (i) the disjunction sits *inside* the premise conjunction, and the calculus's disjunction elimination as currently realized acts only on a disjunction at the root of the premise — the context-carrying form that reaches it is proven as a satellite theorem but its core constructor swap is still scheduled; (ii) the contradiction in the off-branches is semantic (disjoint interiors), not yet a formula-level refutation in the negation-free grammar.

**Recipe R24.2** (the realized rewrite — discharge against the semantic lemma). As shipped, the rule is admissible by *reduction to verified machinery* (the  $\text{PCONSTEXACT}$  pattern): the node

$$\frac{\mathcal{T} \triangleright \Phi \vdash \bigwedge_v \lambda_v^{C_i}(x) \bowtie_v^\circ 0}{\Phi \vdash b_i = 1 \wedge \bigwedge_{j \neq i} b_j = 0} \text{CELL-IN-INTERIOR} \rightsquigarrow \text{discharged by sound\_cellInInterior},$$

where the premise is the *strict* barycentric-interior conjunction for cell  $C_i$  (one strict atom per support vertex,  $<$  for extended vertices,  $>$  for core vertices, in the pinned flat vertex order) and the checker recomputes both formula shapes from the schema — the premise via  $\text{cellStrictInteriorBuild}$ , the one-hot conclusion via  $\text{cellOneHotBuild}$  — so no formula text is trusted from the certificate.

**Proposition R24.3** (semantic warrant). *Suppose the payload carries a proof of the strict-interior premise, the cited cell application is in the hypernet, the input wire is typed at shape  $[\text{ambDim}]$ , every output wire at total size 1, and the premise and conclusion formulas match the recomputed builds. Then the conclusion sequent is semantically valid.*

*Proof sketch (the lemma's actual route).* Fix an admissible valuation  $\sigma$  satisfying  $\Phi$ . Admissibility of the cited cell application yields the model-level fact: the schema's cells have disjoint interiors, the input wire holds a rational point  $x$ , and there is a cell choice  $k$  with  $x \in \mathcal{R}(\Delta_k, S_k)$  (weak membership) whose one-hot valuation the outputs realize — entry 0 of output  $j$  reads 1 if  $j = k$ , else 0. The premise, transported through the input wire's declared shape (the tensor-to-point tie), gives *strict* barycentric signs at  $x$  for cell  $i$ , i.e.  $x$  lies in the strict interior of  $C_i$  ( $\text{maskRegionStrict\_pair\_intro}$  assembles the region witness from the two strict atoms). If  $k \neq i$ , the interior-exclusion well-formedness is contradicted:  $x$  is strictly inside  $C_i$  and weakly inside  $C_k$ , the exact configuration [Remark R24.5](#)'s predicate forbids (strict-versus-weak exclusion — deliberately stronger than bare disjointness of open interiors, which a strictly-interior-to- $C_i$  point on  $C_k$ 's boundary would survive). So  $k = i$ , and the one-hot output pattern at  $k$  is the conclusion: each conjunct is an equality between a singleton-shaped output wire and the constant its entry holds ( $\text{holds\_cellOneHot}$ ).  $\square$

*Remark R24.4* (why the premise must be strict). At a shared boundary face the one-hot semantics deliberately admits *either* adjacent cell as the model's choice: a boundary point has the one-hot set of every cell whose closure contains it. A weak-interior premise therefore cannot pin  $k$  — for  $x$  on the face between  $C_i$  and  $C_k$  the valuation one-hotting  $C_k$  satisfies the weak premise and falsifies  $b_i = 1$ . Strictness excludes exactly the boundary, where the pinning genuinely fails; the companion rule  $\text{CELL-OUT}$  covers boundary points from the other side.

*Remark R24.5* (the disjointness hypothesis, and why the complex's conformity does not deliver it). The step  $k = i$  uses: no rational point lies strictly inside cell  $i$  and weakly inside a *different* cell  $k$  ( $\text{SimpC.CellIntDisjoint}$ , stated over cell *indices* so duplicate entries are excluded). The schema's simplicial conformity field does not imply this: conformity constrains simplex-mask intersections only, never sees the extension masks that carve cells out of simplices, and permits literally duplicate cell entries — either defect, were the predicate not consumed, would let the model one-hot an overlapping cell and falsify the conclusion. The predicate is consumed as a conjunct of the cell application's model-level fact: an application over a non-disjoint enumeration admits no valuation at all (hypothesis-side vacuity — sound, merely weak), the same reading that excludes NaN executions. It is a schema amendment adopted provisionally, pending ratification.

*Remark* R24.6 (the port-typing hypotheses are load-bearing twice). The model’s one-hot fact pins *entry* 0 of each output wire, while the conclusion’s equality atoms evaluate the *whole* wire; the total-size-1 typing is what makes the two coincide (a multi-entry output could satisfy the entry fact while its remaining entries falsify the atom; a zero-size output would satisfy every atom vacuously). Dually, the premise’s barycentric atoms constrain the input *tensor*, and the declared shape [ambDim] (with ambDim = 1 in the checkable fragment) ties that tensor to the model’s rational point.

*Remark* R24.7 (scope of the checkable fragment). `cellStrictInteriorBuild` succeeds only over a 1-dimensional complex with a two-vertex cell support on distinct coordinates — the same fragment as the region machinery elsewhere in the catalogue; everything beyond it is refused, not approximated (fail-toward-refusal). Higher-dimensional cell complexes await their own barycentric builders; the rule’s statement is dimension-general and only the builder gates narrow it.

| proof step                        | formal anchor                                                                              |
|-----------------------------------|--------------------------------------------------------------------------------------------|
| Recipe R24.2 warrant              | <code>sound_cellInInterior (Soundness)</code>                                              |
| premise shape extraction          | <code>cellStrictInteriorBuild_spec; builder cellStrictInteriorBuild / baryLambdas1D</code> |
| strict-region assembly            | <code>maskRegionStrict_pair_intro; MaskRegionStrict</code>                                 |
| the disjointness pin (Rem. R24.5) | <code>SimpC.CellIntDisjoint; the cell case of PrimApp.Sem0k</code>                         |
| one-hot conclusion                | <code>cellOneHotBuild; holds_cellOneHot</code>                                             |
| port typing ties                  | <code>PCellInIntData fields inputShape, outputsScalar</code>                               |

## Rule 25. CELL-OUT (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Cell-Out*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \vdash \lambda_w^{C_j}(x) \bowtie^\circ 0 \text{ (a single strict separating atom at a named witness vertex } w \text{ of } C_j; > \text{ for extended } w, < \text{ for core } w) \quad \text{cell}[\Sigma]_{\{x\} \rightarrow \{b_1, \dots, b_N\}} \in H}{\Phi \vdash b_j = 0} \quad \text{CELL-OUT}$$

applies even at boundary points.

CELL-OUT is CELL-IN-INTERIOR's refutation companion: from a premise strictly separating the input point from cell  $C_j$ , it zeroes that cell's indicator —  $\Phi \vdash b_j = 0$  — and it does so *even at boundary points of other cells*, where the pinning rule cannot fire. The paper states the premise as  $x \notin \overline{C_j}$ , a disjunction of strict halfspace violations; the negation-free formula grammar has no rendering of that, so the realized rule serializes a witness instead: a single strict barycentric atom at a *named witness vertex*  $w$  of  $C_j$ , with the polarity that contradicts membership ( $>$  at an extended vertex,  $<$  at a core vertex). One atom replaces the disjunction — no loss for producers, who know which halfspace their point violates, and a strict gain for the checker, which does no disjunctive reasoning. A producer uses the rule to kill off-branches of a piecewise definition: once every indicator but one is zero, downstream arithmetic (a weighted combination over branches, a substitution) simplifies to the live branch.

What the checker recomputes: presence of the cited cell application with the same port typing as the companion rule (input at [ambDim], outputs at total size one); that the witness vertex lies in cell  $j$ 's support mask (the build refuses otherwise); that the premise is byte-for-byte the recomputed separating atom for  $(j, w)$ ; and that the conclusion is the recomputed  $b_j = 0$  atom.

Each hypothesis is load-bearing. The support-mask guard: an atom at a vertex outside  $C_j$ 's support constrains a weight the membership predicate never mentions — no contradiction, no soundness. Strictness of the atom: the model's membership fact delivers weak signs, and only a strict opposite sign contradicts them (a weak atom is compatible with a boundary weight of zero, where the model may still choose cell  $j$ ). The port typing: same double duty as in the companion rule — the tensor-to-point tie on the input, and the entry-to-whole-wire lift on the output. Notably ABSENT here: the interior-disjointness well-formedness — the refutation is by weight uniqueness on the two-vertex support against the strict witness atom, within cell  $j$  alone, so this rule stands even where only the companion needs the provisional schema amendment. The proof section walks these in order.

### Lean statement

BernsteinLogic.Soundness . **sound\_cellOut**

statement bound

```
-- Cell-Out: the strict separating witness refutes the model choosing
-- cell whichJ (weight uniqueness on the two-vertex support), so the
-- one-hot sp-fact zeroes its indicator. -/
theorem sound_cellOut {H : Hypernet} {seq premise : Sequent}
  {schema : SimpC} {input : Wire} {outputs : List Wire}
  {whichJ : Fin schema.cells.length} {wv : Fin schema.nVertices}
  (ih : premise.semanticallyValid H)
  (happ : PrimApp.cell schema input outputs ∈ H.applications)
  (hinshape : ∃ τ, H.wireType input = some τ ∧
    τ.shape = [schema.ambDim])
  (houts : ∀ o ∈ outputs, ∃ τ, H.wireType o = some τ ∧
    Shape.totalSize τ.shape = 1)
  (hpre : seq.pre = premise.pre)
  (hwit : cellOutWitnessForm schema (.var input) whichJ wv premise.post)
  (hpost : cellOutZeroForm schema outputs whichJ seq.post) :
  seq.semanticallyValid H
```

BernsteinLogic.Soundness . **cellOutWitnessBuild\_spec**

statement bound

```
-- Extraction for the Cell-Out witness premise: additionally pins the
-- witness vertex to the cell's mask and identifies it with one of the two
-- support vertices, selecting the corresponding strict atom. -/
theorem cellOutWitnessBuild_spec {schema : SimpC} {t : Term}
  {k : Fin schema.cells.length} {w : Fin schema.nVertices} {F : Formula}
```

```

(h : cellOutWitnessForm schema t k w F) :
schema.ambDim = 1 ∧
(schema.cells.get k).1 w = true ∧
∃ (a b : Fin schema.nVertices) (V₀ V₁ : ℚ),
(List.finRange schema.nVertices).filter
  (fun v => (schema.cells.get k).1 v) = [a, b] ∧
(∀ j, schema.coord a j = V₀) ∧
(∀ j, schema.coord b j = V₁) ∧
V₀ ≠ V₁ ∧
((w = a ∧ F = .atom (if (schema.cells.get k).2 a then .gt else .lt)
  (.add (.const (V₁ / (V₁ - V₀))) (.mul (.const (-1 / (V₁ - V₀))) t))
  (.const 0)) v
(w = b ∧ F = .atom (if (schema.cells.get k).2 b then .gt else .lt)
  (.add (.const (-V₀ / (V₁ - V₀))) (.mul (.const (1 / (V₁ - V₀))) t))
  (.const 0)))

```

BernsteinLogic.Soundness .maskRegion\_pair\_elim

statement bound

```

/-- Extraction: a masked region membership over a two-vertex 1-D support
pins the barycentric weights to their rational closed forms, and exports
the sign of each weight according to its extension flag. -/
theorem maskRegion_pair_elim {C : SimpC} {mask ext : Fin C.nVertices → Bool}
{x : Fin C.ambDim → ℚ} {a b : Fin C.nVertices} {V₀ V₁ : ℚ}
(hfil : (List.finRange C.nVertices).filter (fun v => mask v) = [a, b])
(hVa : ∀ j, C.coord a j = V₀) (hVb : ∀ j, C.coord b j = V₁)
(hne : V₀ ≠ V₁) (j₀ : Fin C.ambDim)
(h : MaskRegion C mask ext x) :
(ext a = true → (V₁ - x j₀) / (V₁ - V₀) ≤ 0) ∧
(ext a = false → 0 ≤ (V₁ - x j₀) / (V₁ - V₀)) ∧
(ext b = true → (x j₀ - V₀) / (V₁ - V₀) ≤ 0) ∧
(ext b = false → 0 ≤ (x j₀ - V₀) / (V₁ - V₀))

```

**The recomputed forms and the payload.** As for the companion rule, no formula text is trusted from the certificate: the witness atom is rebuilt from the schema's barycentric data at the named vertex (note the guard — the build refuses a witness vertex outside the cell's support mask), the conclusion atom from the output list, and the payload's form fields assert equality with the rebuilt formulas. The same two-vertex 1-D fragment gates apply (`baryLambdas1D`, displayed in the `CELL-IN-INTERIOR` section); everything beyond the fragment is refused.

BernsteinLogic.Core.Catalogue . cellOutWitnessBuild

definition bound

```

/-- Single strict separating atom at witness vertex w. -/
def cellOutWitnessBuild (schema : SimpC) (t : Term)
(k : Fin schema.cells.length) (w : Fin schema.nVertices) :
Option Formula := do
let c := schema.cells[k.val]'(k.isLt)
guard (c.1 w = true)
let lams ← baryLambdas1D schema c.1 t
let entry ← lams.find? fun p => decide (p.1 = w)
some (.atom (if c.2 w then .gt else .lt) entry.2 (.const 0))

```

BernsteinLogic.Core.Catalogue . cellOutWitnessForm

definition bound

```

def cellOutWitnessForm (schema : SimpC) (t : Term)
(k : Fin schema.cells.length) (w : Fin schema.nVertices)
(F : Formula) : Prop :=
cellOutWitnessBuild schema t k w = some F

```

BernsteinLogic.Core.Catalogue . cellOutZeroBuild

definition bound

```

def cellOutZeroBuild (schema : SimpC) (outputs : List Wire)
(k : Fin schema.cells.length) : Option Formula :=
if outputs.length = schema.cells.length then
some (.atom .eq (.var (outputs.getD k.val 0)) (.const 0))
else none

```

BernsteinLogic.Core.Catalogue . cellOutZeroForm

definition bound

```
def cellOutZeroForm (schema : SimpC) (outputs : List Wire)
  (k : Fin schema.cells.length) (F : Formula) : Prop :=
  cellOutZeroBuild schema outputs k = some F
```

BernsteinLogic.Core.Catalogue . PCellOutData

definition bound

```
inductive PCellOutData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (premise : Sequent) (premiseProof : ProofObject H premise)
  (schema : SimpC) (input : Wire) (outputs : List Wire)
  (whichJ : Fin schema.cells.length)
  (witnessVertex : Fin schema.nVertices)
  (appInH : PrimApp.cell schema input outputs ∈ H.applications)
  (inputShape : ∃ τ, H.wireType input = some τ ∧
    τ.shape = [schema.ambDim])
  (outputsScalar : ∀ o ∈ outputs, ∃ τ, H.wireType o = some τ ∧
    Shape.totalSize τ.shape = 1)
  (preEq : seq.pre = premise.pre)
  (outWitness : cellOutWitnessForm schema (.var input) whichJ
    witnessVertex premise.post)
  (postZero : cellOutZeroForm schema outputs whichJ seq.post)
  : PCellOutData H seq
```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; sub-derivation premise as at WEAKEN; binders, linkage row, and the undisplayed port-typing pack (happ, hinshape, hout, — payload fields PCellOutData) exactly as in the CELL-IN-INTERIOR dictionary, with  $j \mapsto$  whichJ and the witness vertex  $w \mapsto$  ww : Fin schema.nVertices.

- premise  $\Phi \vdash \lambda_w^{C_j}(x) \bowtie^\circ 0 \mapsto$  ih with hwit : cellOutWitnessForm schema (.var input) whichJ ww premise.post — pinned as the witness build's output: a single strict atom at the flat-order entry of the named vertex, comparator  $>$  on an extended witness and  $<$  on a core one, exactly the display's polarity (note the deliberate flip against CELL-IN-INTERIOR's interior comparators); the build's guard demands the witness vertex lie in  $C_j$ 's support — the display's “of  $C_j$ ” — and, as at the companion, succeeds only at schema.ambDim = 1: the certified reach is the 1-dimensional fragment, by refusal.  $\Phi$  shared via hpre.
- conclusion  $\Phi \vdash b_j = 0 \mapsto$  the goal under hpost : cellOutZeroForm schema outputs whichJ seq.post — the single equality atom, indicator wire left, .const 0 right.
- the disjointness conjunct of the model's sp-fact is *discarded* by this rule's proof (the statement chain consumes the one-hot fact only) — certifying the commentary's claim that CELL-OUT stands independent of the provisional schema amendment its companion needs.

Width note: “applies even at boundary points” is carried structurally — the witness atom is strict and nothing demands interiority of  $x$ ; within the 1-dimensional fragment, and modulo the undisplayed port-typing pack inherited from the companion dictionary, neither side is wider, every hypothesis consumed.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_catdata2, HOL4 development script bern\_tr\_catdata2Script.sml script bound

#### Definition cellOutWitnessBuild\_def

```

Definition cellOutWitnessBuild_def:
  cellOutWitnessBuild (schema:simpc) t (k:num) (w:num) =
    if k < LENGTH schema.cells then
      (let c = EL k schema.cells in
        if getD (FST c) w F then
          (case baryLambdas1D schema (FST c) t of
            | NONE => NONE
            | SOME lams =>
              (case FILTER (λp. FST p = w) lams of
                | [] => NONE
                | (entry::rest) =>
                  SOME (FAtom (if getD (SND c) w F then BowGt else BowLt)
                    (SND entry) (TmConst 0))))
          else NONE)
        else NONE
    End

```

#### Definition cellOutWitnessFormB\_def

```

Definition cellOutWitnessFormB_def:
  cellOutWitnessFormB schema t k w fm ⇔
    cellOutWitnessBuild schema t k w = SOME fm
End

```

#### Theorem pcelloutdata\_receipt

```

Theorem pcelloutdata_receipt:
  ruledata0k cd2wsCell cd2Hcell
  <| pre := FAtom BowGt (TmAdd (TmConst (&1))
    (TmMul (TmConst (-&1)) (TmVar 9)))
    (TmConst 0) ;
    post := FAtom BowEq (TmVar 5) (TmConst 0) |>
  (RdCellOut (PCellOutD
    <| pre := FAtom BowGt (TmAdd (TmConst (&1))
      (TmMul (TmConst (-&1)) (TmVar 9)))
      (TmConst 0) ;
      post := FAtom BowGt (TmAdd (TmConst (&1))
        (TmMul (TmConst (-&1)) (TmVar 9)))

```

```

(TmConst 0) |> (ProofObj RdAx)
checkfnDemoSchema 9 [5; 7] 0 0)) ^
¬ ruledata0k cd2wsCell hnEmpty
<| pre := FTt ; post := FTt |>
(RdCellOut (PCellOutD
  <| pre := FTt ; post := FTt |> (ProofObj RdAx)
checkfnDemoSchema 9 [5; 7] 0 0))

```

## HOL verifier receipts

**documentary** hol-strict CLEAN: theory bern\_tr\_catdata2, 272 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob cd3c1e95...). The formerly dropped cell-builder island is ported: the witness build/form pair displayed here verbatim (the Form face as a boolean Build-equation, the characterization convention), with kernel-evaluated receipts (cellOutWitnessBuild\_receipt / cellOutWitnessFormB\_receipt), the payload arm of the mutual wellformedness predicate with its own accept/refusal receipt (cited above), the zero-build twin in theory bern\_tr\_checkfn (cellOutZeroBuild, 8 theorems, 0 offences, same run), and its form characterization in theory bern\_tr\_catforms (cellOutZeroForm\_char, 23 theorems, 0 offences). The shared barycentric ingredient (baryCoeffs1D) was already ported and receipted. Honest residue: the cell soundness argument (the Lean development's sound\_cellOut) is not yet ported; the payload/builder layer above is.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. Same family as CELL-IN-INTERIOR: the intended elaboration is a case analysis over the cell primitive's one-hot disjunction; the realized warrant is the degenerate rewrite discharging against a verified semantic lemma, for the same two reasons stated in that rule's first recipe (the context-carrying case analysis is not yet a core constructor, and the off-branch contradiction is semantic). Only the branch-closing argument differs — and it is *lighter*: no disjointness hypothesis is consumed.

**Recipe R25.1** (the CELL-OUT instance — the realized rewrite).

$$\frac{\mathcal{T} \triangleright \Phi \vdash \lambda_w^{C_j}(x) \bowtie^\circ 0}{\Phi \vdash b_j = 0} \text{ CELL-OUT} \quad \rightsquigarrow \quad \text{discharged by sound\_cellOut,}$$

where the premise is a *single strict separating atom* at a named witness vertex  $w$  of cell  $C_j$  ( $>$  if  $w$  is an extended vertex,  $<$  if a core vertex — the sign that contradicts membership), and the checker recomputes premise and conclusion shapes from the schema (cellOutWitnessBuild, cellOutZeroBuild). The payload names  $w$  explicitly; the build refuses any  $w$  outside  $C_j$ 's support mask.

**Proposition R25.2** (semantic warrant). *Under the payload hypotheses (premise proof; cited cell application in the hypernet; input typed at shape [ambDim]; outputs at total size 1; premise and conclusion matching the recomputed builds), the conclusion sequent is semantically valid.*

*Proof sketch (the lemma's actual route).* Fix an admissible valuation  $\sigma$  satisfying  $\Phi$ . The cell application's model-level fact yields the rational input point  $x$ , a cell choice  $k$  with  $x \in \mathcal{R}(\Delta_k, S_k)$ , and the one-hot output valuation at  $k$ . Suppose  $k = j$ . On the two-vertex support the barycentric weights of  $x$  are *unique* — they are determined affinely by the point — so weak membership in cell  $j$  pins their signs:  $\leq 0$  at extended vertices,  $\geq 0$  at core vertices (the pair-elimination lemma). The premise delivers the *strict opposite* sign for the weight at the witness vertex  $w$  (transported to the model's point through the input wire's declared shape), a direct contradiction. Hence  $k \neq j$ , and the one-hot fact reads entry 0 of output  $j$  as 0; the total-size-1 typing lifts that entry to the whole-wire equality  $b_j = 0$ .  $\square$

**Remark R25.3** (why a witness vertex replaces the paper's set complement). The source statement's premise is  $x \notin \overline{C_j}$  — a *disjunction* of strict halfspace violations, one per bounding halfspace. The formula grammar is negation-free and the premise of a rule application must be a formula the calculus can derive, so the amendment serializes a choice of disjunct instead: *which* halfspace is strictly violated, named by *which* vertex carries the violated barycentric sign. One atom replaces the disjunction with no loss for producers — within the fragment the builders admit, where the cell's closure is exactly the conjunction of its per-vertex sign constraints, a point outside the closure violates at least one of them, and the producer knows which — and a strict gain for the checker (no disjunctive reasoning at check time).

**Remark R25.4** (the witness must be a support vertex of  $C_j$ ). The build guards  $w \in \text{supp}(C_j)$ . Without the guard the premise atom would constrain a barycentric coordinate that cell  $j$ 's membership predicate never mentions — the sign contradiction in Proposition R25.2 needs the weight at  $w$  to be one of the two the membership fact actually constrains. (On the two-vertex support this is also what makes the atom well-defined: the builder computes  $\lambda_w^{C_j}$  from the support pair's coordinates.)

**Remark R25.5** (boundary points, and the division of labor with CELL-IN-INTERIOR). The rule applies even at boundary points of *other* cells: the contradiction is between a strict violated sign and weak membership of the *same* cell  $j$ , so no interior-disjointness hypothesis is consumed — a point on the shared face of  $C_i$  and  $C_k$  can still be strictly separated from a third cell  $C_j$ , and the rule concludes  $b_j = 0$  regardless of which of the two adjacent cells the model one-hots. This is the refutation companion of



CELL-IN-INTERIOR, whose pinning argument fails precisely on boundaries and needs disjointness; between them, interior points get the full one-hot vector, and a boundary point gets  $b_j = 0$  for every cell  $j$  whose closure excludes it — the cells actually sharing the face stay unpinned, necessarily: the model may one-hot either, so no sound rule could zero theirs.

*Remark* R25.6 (scope of the checkable fragment). As for CELL-IN-INTERIOR: the builders succeed over 1-dimensional complexes with two-vertex cell supports on distinct coordinates, and refuse everything else. The statement is dimension-general; only the builder gates narrow it.

| proof step                            | formal anchor                                                         |
|---------------------------------------|-----------------------------------------------------------------------|
| Recipe R25.1 warrant                  | sound_cellOut (Soundness)                                             |
| witness-atom extraction               | cellOutWitnessBuild_spec; builder cellOutWitnessBuild / baryLambdasID |
| weight uniqueness on the support pair | the pair-elimination lemma cited inside sound_cellOut                 |
|                                       | (maskRegion_pair_elim)                                                |
| conclusion form                       | cellOutZeroBuild                                                      |
| support-mask guard                    | the guard in cellOutWitnessBuild; payload field witnessVertex         |
| port typing ties                      | PCellOutData fields inputShape, outputsScalar                         |

## Rule 26. PConstExact (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *PConstExact*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{e_1, e_2 \text{ wire-free (closed) terms} \quad \bowtie \in \{<, \leq, =, \geq, >\} \quad e_1 \bowtie e_2 \text{ holds under exact evaluation in } \mathbb{Q}}{\Phi \vdash e_1 \bowtie e_2} \text{PConstExact}$$

the third premise is decided, not assumed: both terms are closed, so the comparison evaluates in exact rational arithmetic (the checker's realization is the verified closed evaluator, which returns no verdict on any term containing a wire — closedness is thereby implied, never a trusted annotation).

PConstExact is the calculus's ground-fact rule: it lets a derivation state an exact closed arithmetic fact  $\vdash -3/4 \leq 0$ , or  $\vdash (2/3)^2 = 4/9$  — with no premises. Before it, ground facts were nearly out of reach: NORM, premise-free, already states closed equalities of identical canonical forms ( $\vdash (2/3)^2 = 4/9$  falls to it), but no rule stated a ground *order* comparison  $\vdash -3/4 \leq 0$  had no derivation. A producer uses it wherever a numeric side condition enters a derivation — a constant bound, an exact constant comparison feeding a later combination step.

What the checker recomputes is the entire side condition: `evalClosed`, the verified closed-term evaluator, exact over  $\mathbb{Q}$ , must return `true` on the comparison. There is no separate “closedness” hypothesis to check: the evaluator returns `none` the moment it meets a wire, so a `true` result certifies closedness as a byproduct — a hypothesis the payload cannot mis-declare. One comparator,  $\neq$ , is excluded from the admissible set even though the underlying semantic lemma covers it: the calculus maintains a second, three-valued soundness screen, and a premise-free  $\neq$  atom would break it (the proof section unpacks why). Without the exclusion, the rule would silently invalidate an invariant the rest of the calculus is entitled to assume.

**notes** The rule generalizes the earlier scalar-constant-fact rule, retained as the corner case  $e_2 = 0$ . Its tier is the taxonomy's recorded boundary case: admissible by *reduction to verified machinery* — the soundness lemma is one step to the evaluator-correctness theorem — rather than by unpacking into a catalogue derivation, because for its order conclusions no such derivation exists; that impossibility is the rule's point. It is not core because it adds no new kernel semantic content.

### Lean statement

BernsteinLogic.Soundness . Formula.holds\_of\_evalClosed\_true

statement bound

```
/-- Closed formulas that evaluate true hold at every valuation. -/
theorem Formula.holds_of_evalClosed_true {H : Hypernet} (σ : H.Valuation) :
  ∀ (F : Formula), F.evalClosed = some true → F.holds σ
```

BernsteinLogic.Soundness . sound\_pConstExact

statement bound

```
/-- PConstExact soundness: a closed exact comparison whose closed evaluator
`Formula.evalClosed` decides `some true` holds at every valuation. One step to
the verified `Formula.holds_of_evalClosed_true` — closedness of `e1`, `e2` is
IMPLIED by `evalTrue` succeeding (`evalClosed` is `none` on any wire), and the
comparator is decided by exact  $\mathbb{Q}$  evaluation. `pre` is arbitrary (premise-free). -/
theorem sound_pConstExact {H : Hypernet} {seq : Sequent} {e1 e2 : Term}
  {b : BowTie}
  (hEval : (Formula.atom b e1 e2).evalClosed = some true)
  (hpost : seq.post = .atom b e1 e2) :
  seq.semanticallyValid H
```

BernsteinLogic.Core.Catalogue . Formula.evalClosed

statement bound

```
/-- Closed-formula evaluation: `none` iff a wire or region atom occurs
(strict; see original for the closed-world reading). -/
def Formula.evalClosed : Formula → Option Bool
| .atom b l r => do pure (b.evalQ (← l.evalQ?) (← r.evalQ?))
| .inRegion _ _ => none
| .and l r => do pure ((← l.evalClosed) && (← r.evalClosed))
| .or l r => do pure ((← l.evalClosed) || (← r.evalClosed))
```

```

| .tt => some true
| .ff => some false

```

```

/-- The barycentric coordinate functions of a 1-dimensional simplex, as
canonical affine Terms in `t` (pinned shape `offset + slope·t`; see the
original for the full derivation notes and D2 scope). -/
def baryLambdas1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool)
  (t : Term) : Option (List (Fin cx.nVertices × Term))

```

BernsteinLogic.Core.Catalogue . PConstExactData

statement bound

```

inductive PConstExactData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (e1 e2 : Term) (b : BowTie)
  (notNE : b ≠ .ne)
  (evalTrue : (Formula.atom b e1 e2).evalClosed = some true)
  (postEq : seq.post = .atom b e1 e2)
  : PConstExactData H seq

```

BernsteinLogic.Admissible . ProofObject.m3sound

statement bound

```

/-- Monotone replacement at a conjunct path: substituting a
lattice-smaller formula at any conjunct address shrinks the M3 value.
Meet monotonicity only – no distributivity – which is why Promise,
unlike vE-ctx, is lattice-sound and core-eligible. -/
theorem mval_conjReplace_mono :
  ∀ {π : ConjPath} {Γ Γ' x q : Formula},
  Γ.conjGet? π = some q → Γ.conjReplace? π x = some Γ' →
  M3.le (mval x) (mval q) = true →
  M3.le (mval Γ') (mval Γ) = true := by
intro π
induction π with
| here =>
  intro Γ Γ' x q hget hrep hle
  simp only [Formula.conjGet?] at hget
  simp only [Formula.conjReplace?] at hrep
  injection hget with h1
  injection hrep with h2
  rw [← h2, h1]
  exact hle
| left p ih =>
  intro Γ Γ' x q hget hrep hle
  cases Γ with
  | and l r =>
    simp only [Formula.conjGet?] at hget
    simp only [Formula.conjReplace?] at hrep
    cases hmap : l.conjReplace? p x with
    | none => rw [hmap] at hrep; nomatch hrep
    | some l' =>
      rw [hmap] at hrep
      injection hrep with h2
      rw [← h2]
      show M3.le ((mval l').inf (mval r)) ((mval l).inf (mval r)) = true
      exact M3.inf_mono_l _ _ _ (ih hget hmap hle)
  | atom b tl tr => nomatch hrep
  | inRegion t Δ S => nomatch hrep
  | or l r => nomatch hrep
  | tt => nomatch hrep
  | ff => nomatch hrep
| right p ih =>
  intro Γ Γ' x q hget hrep hle
  cases Γ with
  | and l r =>
    simp only [Formula.conjGet?] at hget
    simp only [Formula.conjReplace?] at hrep
    cases hmap : r.conjReplace? p x with

```

```

| none => rw [hmap] at hrep; nomatch hrep
| some r' =>
  rw [hmap] at hrep
  injection hrep with h2
  rw [← h2]
  show M3.le ((mval l).inf (mval r')) ((mval l).inf (mval r)) = true
  exact M3.inf_mono_r _ _ _ (ih hget hmap hle)
| atom b t1 tr => nomatch hrep
| inRegion t Δ S => nomatch hrep
| or l r => nomatch hrep
| tt => nomatch hrep
| ff => nomatch hrep

/-! ## §A9 The metatheorem: derivability is lattice-sound

Over a hypernet with no typed wires, every derivable sequent satisfies
[[pre]] ≤ [[post]] in M3. One case per rule of the catalogue; the
propositional rules are the lattice axioms, equality elimination is the
substitution invariance (its typing field forces both terms closed over
an untyped hypernet), the region/arithmetic-closure rules are handled
by their conclusion shapes or payload absurdities, and the
empty-payload rules are vacuous. -/
theorem ProofObject.m3sound : ∀ {H : Hypernet} {seq : Sequent},
  (∀ w, H.wireType w = none) → ProofObject H seq →
  M3.le (mval seq.pre) (mval seq.post) = true
| _, _, _, .mk _ _ _ (.ax _ _ (.mk _ _ hpp)) => by
  rw [hpp]; exact M3.le_refl _
| _, _, hE, .mk _ _ _ (.weaken _ _ (.mk _ _ _ pproof _ hpre hpost)) => by
  rw [hpre, hpost]
  exact M3.weaken_kit _ _ _ (pproof.m3sound hE)
| _, _, hE, .mk _ _ _ (.cut _ _
  (.mk _ _ _ _ lproof _ rproof hpre hpost hcl hcr)) => by
  have hl

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

- $e_1, e_2 \mapsto e_1 \bowtie e_2 : \text{Term}$ ;  $\bowtie \mapsto b : \text{BowTie}$ ; the judgment context  $\Phi$  and the concluded sequent  $\mapsto \text{seq} : \text{Sequent}$  with the conclusion tie  $\text{hpost} : \text{seq.post} = \text{atom } b \ e_1 \ e_2$ . Premise-freeness is represented by silence: no hypothesis mentions  $\text{seq.pre}$ , so the statement holds under arbitrary premises.
- the displayed premise ( $e_1 \bowtie e_2$  holds under exact evaluation in  $\mathbb{Q}$ )  $\mapsto \text{hEval} : (\text{Formula.atom } b \ e_1 \ e_2).\text{evalClosed} = \text{some true}$  — note the some: the verified evaluator is partial and returns none on any wire, so the display's separate “wire-free (closed) terms” hypothesis is *absorbed*: it has no Lean binder of its own because a successful evaluation certifies it.
- conclusion  $\Phi \vdash e_1 \bowtie e_2 \mapsto$  the goal  $\text{seq.semanticallyValid } H$ , validity at every valuation of the ambient hypernet.

Width note, deliberate and load-bearing: the display restricts  $\bowtie \in \{<, \leq, =, \geq, >\}$ , while the Lean statement is proven for *all* constructors of `BowTie`, including  $\neq$ . The comparator exclusion is an admission-layer fact — the checker refuses a premise-free  $\neq$  atom to protect the three-valued screen, as the commentary explains — not a semantic one. The mechanized statement is therefore strictly wider than the displayed rule, and the displayed rule is exactly the admitted fragment of it.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Admissible; bound build 52e5099ba522... (bound from receipt c17abf2e103fe745, 2026-07-21T05:28:22Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 7f3043c54f717472 | 2026-07-20T07:43:38Z | build+decorrelated                 | current |
| acaf0e8871731343 | 2026-07-20T20:44:28Z | build+decorrelated                 | current |
| 62974466ea131d89 | 2026-07-20T21:00:57Z | build+replay+hardened+decorrelated | current |
| c17abf2e103fe745 | 2026-07-21T05:28:22Z | build+replay+hardened+decorrelated | current |

6 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_catdata, HOL4 development script bern\_tr\_catdataScript.sml script bound

#### Definition pconstexactdata0k\_def

```

Definition pconstexactdata0k_def:
  pconstexactdata0k (sq:sequent) (d:pconstexactdata) ↔
    d.ceBow ≠ BowNe ∧
    formulaEvalClosed (FAtom d.ceBow d.ceL d.ceR) = SOME T ∧
    sq.post = FAtom d.ceBow d.ceL d.ceR
End

```

#### Theorem pconstexactdata0k\_receipt

```

Theorem pconstexactdata0k_receipt:
  pconstexactdata0k
    <| pre := FTt ; post := FAtom BowGt (TmConst 1) (TmConst 0) |>
    <| ceBow := BowGt ; ceL := TmConst 1 ; ceR := TmConst 0 |> ∧
  ¬ pconstexactdata0k
    <| pre := FTt ; post := FAtom BowNe (TmConst 1) (TmConst 0) |>
    <| ceBow := BowNe ; ceL := TmConst 1 ; ceR := TmConst 0 |>

```

## HOL verifier receipts

**documentary** hol-strict CLEAN: theory bern\_tr\_catdata, 252 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob f8fceac5...). The closed-comparison payload predicate displayed above is the rule's entire checkable content — comparator confined to the admissible five, both terms evaluated by the closed exact- $\mathbb{Q}$  evaluator, conclusion pinned to the payload — with a kernel-evaluated accept/refusal receipt. The proof-object stratum consumes it verbatim as this rule's arm of the mutual wellformedness predicate (theory bern\_tr\_catdata2, 272 theorems, 0 offences, same run), and the rule-tag roster carries the rule. As on the Lean side, the rule rides the proof-object family, not the first-order wire format: the wire tag remains deferred on both sides (the emit path refuses until allocated), so there is deliberately no column-side check to mirror. Honest residue: the semantic reduction (the Lean development's sound\_pConstExact, one step to the verified closed evaluator) is not yet ported.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. The tier taxonomy defines tier 2 as *verified unpacking into core rules* (one node  $\rightarrow$  many); this rule is the taxonomy's recorded boundary case — admissible by *reduction to verified machinery*. The general template, both species included, is stated in the NORM section; the recipe below typesets the honest degenerate instance.

**Recipe R26.1** (the PConstExact instance — the degenerate rewrite). PConstExact has *no proof premises* ( $n = 0$ ), and its rewrite target contains *zero catalogue nodes*: no catalogue derivation states a ground *order* comparison — closed equalities of identical canonical forms fall to NORM already, but  $\vdash -3/4 \leq 0$  had no derivation before this rule — which is the rule's entire point. The rewrite therefore discharges the node against a named *verified theorem* rather than against catalogue structure:

$$\frac{e_1 \bowtie e_2 \text{ holds under exact evaluation in } \mathbb{Q}}{\Phi \vdash e_1 \bowtie e_2} \text{PConstExact} \rightsquigarrow \text{the checker replays evalClosed; holds\_of\_evalClosed\_true stands as warrant.}$$

The side condition is *computational* (exact  $\mathbb{Q}$  evaluation of a closed comparison — decidable, replayed by the checker), and the warrant theorem is kernel-verified once, globally: soundness of the rule is literally “the exact evaluator is correct.”

**Proposition R26.2** (semantic warrant). *If  $(\text{atom} \bowtie e_1 e_2).\text{evalClosed} = \text{some true}$  and the sequent's conclusion is that atom, the sequent is semantically valid over any hypernet, with arbitrary premise  $\Phi$ .*

*Proof.* Fix any valuation  $\sigma$  (admissibility and  $\sigma \models \Phi$  are not consulted — the rule is premise-free, the NORM convention). By the verified evaluator theorem `Formula.holds_of_evalClosed_true`, a formula whose closed evaluation returns true holds at *every* valuation: a closed term's evaluation consults no wire, so  $\sigma$  is never read, and the comparison was decided by exact arithmetic over  $\mathbb{Q}$ . That is the whole proof — `sound_pConstExact` is one exact step.  $\square$

**Remark R26.3** (closedness is implied, not declared). There is no “ $e_1, e_2$  closed” payload field to check: `evalClosed` returns none on any wire occurrence, so *some true* certifies closedness as a byproduct. A well-formedness hypothesis that the evaluator itself enforces is one the payload cannot mis-declare — unrepresentable, not discouraged.

**Remark R26.4** (why  $\neq$  is excluded — the  $M_3$  countermodel). The comparator set is confined to  $\{<, \leq, =, \geq, >\}$ : five of the six. The semantic lemma covers all six, but the calculus also maintains a three-valued screen (`ProofObject.m3sound`), whose disequality clause reads designated free-wire membership — so a wire-free  $\neq$  atom, its wire list empty, interprets to  $\perp$  under the  $M_3$  countermodel. Admitting  $\neq$  against arbitrary  $\Phi$  would then let a premise-free rule conclude screen value  $\perp$  under a premise valued  $\top$  ( $\vdash 1 \neq 2$  is true over  $\mathbb{Q}$ , and breaks the screen's premise- $\leq$ -conclusion invariant), breaking `m3sound`. The `notNE` payload field is shared with `ARITHATOM`, where the same gate carries different weight: both of that rule's spellings mention the same wires, so its screen value is preserved and the gate serves alignment (that section's closing remark); here the breakage is real. A hypothesis whose necessity comes not from  $\mathbb{Q}$ -semantics but from a *second* soundness invariant is exactly the kind a recipe must state, or the next porter re-learns it the hard way.

**Remark R26.5** (the subsumed corner, and coverage). `PRATFACT` (the scalar constant fact  $c \bowtie 0$ ) is the corner  $e_2 = 0$  of this rule and was subsumed when the roster was consolidated. Coverage today: the Term grammar makes every closed term rank-0, so the rule states every closed *scalar* exact comparison; the matrix corner (exact closed matrix identities, MIX's use case) needs closed-tensor constructors and a `QTENSOR`-valued closed evaluator — the named next leg, deliberately not hand-faked here.

| proof step                          | formal anchor                                                                                                   |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Recipe R26.1 warrant                | <code>sound_pConstExact</code> (one step to <code>Formula.holds_of_evalClosed_true</code> )                     |
| closedness-is-implied               | <code>Formula.evalClosed</code> (none on any wire)                                                              |
| the $\neq$ exclusion                | <code>notNE</code> payload field; <code>ProofObject.m3sound</code>                                              |
| premise-freedom ( $\Phi$ arbitrary) | the NORM convention; pre unused in <code>sound_pConstExact</code>                                               |
| wire placement                      | <code>RuleTag.pConstExact</code> ; wire tag not yet allocated — the emit path refuses the rule until allocation |

## Rule 27. PBOUND (alias for SYZGY)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *PBound*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} \{\Phi_i \vdash e_i \leq 0\}_{i=1}^n, \quad \Phi \supseteq \Phi_i \text{ conjunct-wise (checked)} \quad \{g_k = 0\}_k \text{ equality sub-conjuncts of } \Phi \quad \lambda_i \in \mathbb{Q}_{\geq 0} \\ \nu_k \in \mathbb{Q} \quad c \equiv \sum_i \lambda_i e_i + \sum_k \nu_k g_k \text{ (exact canonical polynomial)} \quad \text{bshape}_H(c) \neq \perp, \quad \text{wires}(c) \text{ all finitely typed in } H \\ \text{wires}(\Phi) \cup \text{wires}(c) \cup \bigcup_i \text{wires}(e_i) \text{ all typed at positive-size shapes in } H \\ \Psi \in \{c \leq 0, \quad e \geq 0 \text{ where } c = -e, \quad l \leq r \text{ where } c = l - r\} \end{array}}{\Phi \vdash \Psi} \quad \text{PBOUND}$$

PBOUND is the workhorse bound-node: a compact spelling of a SYZGY application whose entries the register prices as affine, with the conclusion exposed through the three-way reading selector — the raw zero-form  $c \leq 0$ , the sign-flipped  $e \geq 0$  where  $c = -e$ , or the literal atom  $l \leq r$  where  $c = l - r$ . It is the taxonomy's canonical alias: the payload is SYZGY's field-for-field, the unpack is the identity transport, and the wire format prices the affine register compactly. A producer emits it constantly: it is how an affine consequence of proven bounds becomes a first-class node whose conclusion can feed the rules that demand a particular atom shape — the  $\geq$ -reading makes a nonnegativity premise reachable from bound facts without a detour, and the literal-atom reading feeds interval-shaped consumers with no separate re-arrangement step.

What the checker recomputes is exactly SYZGY's obligation set — entry decoding against the pinned combination encoding, nonnegative weights, entry-sourcing from component conclusions and premise equality sub-conjuncts (both polarities on equalities), and the exact canonical-polynomial fold identity — plus the same bookkeeping (conjunct-wise premise containment, conclusion shape and finiteness, positive-size wires on premise, conclusion, and component conclusions). That section's commentary on what breaks without each hypothesis applies verbatim here, through the identity transport.

The affine restriction is the rule's published register (the roster note's pricing vocabulary — wire-format compactness for coefficient rows); it appears in neither the displayed premises nor the statement, and it is not a checked side condition: the replay is degree-blind and soundness never consults linearity. The three readings are one exact  $\mathbb{Q}$  identity re-presented under a sign flip; their semantic transport is proven once, in the target rule's section, and inherited here as a pure re-spelling.

**notes** The wire tag for this rule is deferred to a payload-window allocation; the emit path refuses the rule until the tag is allocated. The compact-register pricing is the reason the alias exists as a distinct spelling at all — semantically it adds nothing to its target, which is the alias genre working as intended.

### Lean statement

BernsteinLogic.Soundness . holds\_of\_syzygyConclusionForm

statement bound

```
-- The ge-bridge ring lemma (the both-polarity core change): every
reading admitted by
`syzygyConclusionForm conclusion` holds whenever the zero-normal atom
`conclusion ≤ 0` holds — the readings are the same exact  $\mathbb{Q}$  identity
re-presented under a sign flip. The zero-form is the identity transport,
the ge-polarity flip is ` $-e \leq 0 \rightarrow e \geq 0$ ', and the literal atom is
` $l - r \leq 0 \rightarrow l \leq r$ ` (ArithAtom's tensor-general sub-zero bridge).
Applied once inside `sound_syzygy`, so PSyzygy (tier-1), PBound's
compact spelling (tier-3), and any future  $\leq 0$ -shaped fact rule inherit
all readings with no separate polarity justification. -/
theorem holds_of_syzygyConclusionForm {H : Hypernet} {σ : H.Valuation}
  {conclusion : Term} {F : Formula}
  (hle : Formula.holds σ (.atom .le conclusion (.const 0)))
  (hform : syzygyConclusionForm conclusion F) :
  Formula.holds σ F
```

BernsteinLogic.Soundness . sound\_syzygy

statement bound

```
-- Soundness of Syzygy (= Lemma B, dialect D): valid components give a
valid composite. The argument is `sound_farkas`'s, with the per-entry
source discharge generalized: a component-sourced entry's atom holds by
component validity (through `premiseContains`), a wiring-sourced entry's
```



```

atom holds as a premise sub-conjunct; either way
`atomLeZeroPolys_sound_entryC` delivers per-entry nonpositivity at every
clamped per-entry valuation, and the nonnegative fold closes. The
zero-normal bound then reaches `seq.post` through whichever
`szygyConclusionForm` reading the payload declared (both-polarity core
change, 2026-07-18). -/
theorem sound_szygy {H : Hypernet} (st : SzygyStep H)
  (hcomps : ∀ c ∈ st.components, c.semanticallyValid H) :
  st.seq.semanticallyValid H

```

#### BernsteinLogic.Soundness . sound\_pBound

statement bound

```

/-- PBound soundness: INHERITED from `sound_szygy`, literally – the payload's
fields are `PSzygyData`'s verbatim (tier-3 compact spelling, 1 node → 1 node),
so the proof is the same `SzygyStep` handed to the same theorem. Polarity is
not justified here: since the both-polarity core change,
`sound_szygy` itself carries every
`szygyConclusionForm` reading through `holds_of_szygyConclusionForm`, and
PBound inherits them as a pure re-spelling. -/
theorem sound_pBound {H : Hypernet} {seq : Sequent} {components : List Sequent}
  {combination conclusion : Term}
  (hcomps : ∀ c ∈ components, c.semanticallyValid H)
  (hpreC : ∀ c ∈ components, premiseContains seq.pre c.pre)
  (hshape : ∃ s, conclusion.bshape H = some s)
  (hfin : conclusion.finWires H)
  (hwpos : ∀ w ∈ seq.pre.freeWires ++ conclusion.freeWires,
    ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape)
  (hcompPos : ∀ c ∈ components, ∀ w ∈ c.post.freeWires,
    ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape)
  (hvalid : szygyCombinationValid combination components seq.pre conclusion)
  (hform : szygyConclusionForm conclusion seq.post) :
  seq.semanticallyValid H

```

#### BernsteinLogic.Core.Catalogue . PBoundData

statement bound

```

inductive PBoundData : Hypernet → Sequent → Type where
| mk (H : Hypernet) (seq : Sequent)
  (components : List Sequent)
  (componentProofs : ProofSeqList H components)
  (combination : Term)
  (conclusion : Term)
  (denBoundLog2 : Nat)
  (preContains : ∀ c ∈ components, premiseContains seq.pre c.pre)
  (conclShape : ∃ s, conclusion.bshape H = some s)
  (conclFin : conclusion.finWires H)
  (wiresPos : ∀ w ∈ seq.pre.freeWires ++ conclusion.freeWires,
    ∃ τ, H.wireType w = some τ ∧
    0 < Shape.totalSize τ.shape)
  (compPostWiresPos : ∀ c ∈ components, ∀ w ∈ c.post.freeWires,
    ∃ τ, H.wireType w = some τ ∧
    0 < Shape.totalSize τ.shape)
  (valid : szygyCombinationValid combination components seq.pre conclusion)
  (postForm : szygyConclusionForm conclusion seq.post) :
  PBoundData H seq

```

#### BernsteinLogic.Core.Catalogue . szygyConclusionForm

definition bound

```

def szygyConclusionForm (conclusion : Term) (F : Formula) : Prop :=
  F = .atom .le conclusion (.const 0)
  ∨ (∃ e, conclusion = .neg e ∧ F = .atom .ge e (.const 0))
  ∨ (∃ l r, conclusion = .sub l r ∧ F = .atom .le l r)

```

#### BernsteinLogic.Core.Catalogue . szygyCombinationValid

definition bound

```

def szygyCombinationValid (cc : Term) (components : List Sequent)
  (premise : Formula) (conclusion : Term) : Prop :=
  szygyCheck cc components premise conclusion = true

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Verbatim the SYZGY dictionary, unbundled: where SYZGY's statement packs its obligations into a SyzgyStep, sound\_pBound takes the same eleven fields as flat binders — hpreC, hshape, hfin, hwpos, hcompPos, hvalid, hform, over the same components/combination/conclusion data with hcomps the component validities — and its entire proof is one application of sound\_syzgy to the repacked Step, visible in the displayed statement. The alias relation is thereby itself certified:  $1 \rightarrow 1$  at statement level, and (since the both-polarity core change)  $1 \rightarrow 1$  at reading level too — the displayed  $\Psi$  selector is the same syzgyConclusionForm the target consumes. This cell displays the evaluability and positive-size rows inline; their mapping is the SYZGY dictionary's, unchanged. No hypothesis differs, none is unused, and neither side is wider — the affine restriction the roster note mentions is wire-format compactness, absent from both the display's premises and the statement.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **stale only**

hardened (safe\_verify (hardened screen)) **stale only**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_catalogue, HOL4 development script bern\_catalogueScript.sml **script bound**

### Definition syzgyCheck\_def

```

Definition syzgyCheck_def:
  syzgyCheck cc components premise conclusion  $\Leftarrow$ 
    syzgyCheckCore cc components premise conclusion v
    syzgyCheckW cc components premise conclusion
End

```

## HOL verifier receipts

**documentary** hol-strict CLEAN over the ported checker chain: theory bern\_catalogue, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, scripts sha-matched verbatim to the sources cited here (git blob e41b9ed0...). Scope, stated honestly: the alias's entire combination check is the target rule's, so the ported syzygyCheck (with its accept and two reject receipt rows, quoted in the target rule's section) covers this rule's computational content; the conclusion-form reading selector and the payload layer ride the dependently-typed proof-object surface, which is a separately tracked porting stage. As on the FARKAS page: the Lean check has since grown a third (mixed tagged-union) branch, added 2026-07-19 after this port run — the divergence is refusal-direction only, and its port is named residue. Semantic stratum, also ported: the soundness twin sound\_pBound (theory bern\_tr\_soundsyz: hol-strict clean, 7 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources); the certified-fragment keystone (checkPf\_sound\_frag, theory bern\_tr\_splICE, 20 theorems, 0 offences, same run) consumes it at the SYZGY wire arm, this rule's target relation.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: alias rule (tier 3) — a respelling template. An alias owes the book five artefacts: the verified  $1 \rightarrow 1$  unpack, the filled-in application of the target rule, the payload-transport table, the exact algebraic identity that makes the respelling correct, and the soundness corollary through the target rule's lemma. All five follow — and this alias is the tier taxonomy's *canonical* case, the one the tier was named for: the unpack is the identity.

**Recipe R27.1** (the alias template). An alias rule  $A$  is warranted by a verified function  $\text{unpack} : \text{Step}_A \rightarrow \text{Step}_T$  into the payload of a *single* target node  $T$  (core or admissible), such that the  $A$ -node and its image conclude the same sequent from the same premises: one node  $\rightarrow$  one node, payload transported, nothing derived. Soundness is then a corollary,  $\text{sound}_A = \text{sound}_T \circ \text{unpack}$ . The wire format may carry  $A$  as a compact register of its own or never carry it at all (producers elaborating before emission, as the SMALLGAIN section records); either way the checker's obligations are the target's, reached through the unpack. This rule takes the second branch in v0: no producer emits the once-contemplated affine coefficient-row register, so no tag is allocated — and the identity unpack makes a register pure wire compactness, purchasable later without semantic motion.

**Recipe R27.2** (the verified unpack — the identity transport). PBOUND's payload carries components, combination term, conclusion term, and the bookkeeping fields *verbatim* as the target's payload does, including the conclusion-form selector. The unpack is therefore not a computation but a re-labelling: the verified object is the direct specialization

$\text{sound}_{\text{PBOUND}} = \text{sound}_{\text{SYZGY}}$  applied to the same fields, re-assembled as the target's step

— in the sources, the body of  $\text{sound\_pBound}$  is one application of  $\text{sound\_syzygy}$  to the step built from PBOUND's own fields. Where SMALLGAIN's unpack computes a weighting, this one computes nothing; that is the tier-3 boundary case made flesh, and it is why the alias relation here is  $1 \rightarrow 1$  at the level of *readings* too (Recipe R27.4).

**Recipe R27.3** (the filled-in application). Under the identity transport the PBOUND node is the following application of the target rule (compare the rule display in the SYZGY section, field for field):

$$\frac{\begin{array}{l} \{\Phi_i \vdash e_i \leq 0\}_{i=1}^n, \quad \Phi \supseteq \Phi_i \text{ conjunct-wise (checked)} \quad \{g_k = 0\}_k \text{ sub-conjuncts of } \Phi \text{ in wire variables} \\ \lambda_i \in \mathbb{Q}_{\geq 0} \quad \nu_k \in \mathbb{Q} \quad c \equiv \sum_i \lambda_i e_i + \sum_k \nu_k g_k \text{ (exact canonical polynomial)} \\ \Psi \in \{c \leq 0, \quad e \geq 0 \text{ where } c = -e, \quad l \leq r \text{ where } c = l - r\} \end{array}}{\Phi \vdash \Psi} \quad \text{SYZGY}$$

where  $\Psi$  is whichever reading of  $c \leq 0$  the shared selector declares:  $(Z \leq) c \leq 0$  itself;  $(G) e \geq 0$  where  $c = -e$ ;  $(L \leq) l \leq r$  where  $c = l - r$ . Discharging the target's obligations at this instance is immediate — they are *the same obligations on the same data*: entry decoding, nonnegative weights, entry-sourcing (component conclusions and premise equality sub-conjuncts, both polarities on equalities), the exact fold identity, premise containment, conclusion shape/finiteness, and the positive-size wire conditions on premise, conclusion, and component conclusions — the last three families displayed inline in PBOUND's own cell, while in SYZGY's cell the shape/finiteness and positive-size conditions ride the payload (its display carries containment as a trailing line); the obligations are identical either way, only the display convention differs. PBOUND's published register prices the  $e_i$  as *affine* entries (the roster's register note); the displayed premises carry no such restriction, and nothing above consults one (Remark R27.8).

**Recipe R27.4** (payload transport — field for field).

| PBOUND payload             | target (SYZGY) payload                                                   |
|----------------------------|--------------------------------------------------------------------------|
| components                 | components (verbatim)                                                    |
| combination                | combination (verbatim)                                                   |
| conclusion                 | conclusion (verbatim)                                                    |
| denBoundLog2               | denBoundLog2 (verbatim; witness-side claim, zero proof-theoretic weight) |
| preContains                | preContains (verbatim)                                                   |
| conclShape, conclFin       | conclShape, conclFin (verbatim)                                          |
| wiresPos, compPostWiresPos | wiresPos, compPostWiresPos (verbatim)                                    |
| valid                      | valid (the same exact-combination proposition)                           |
| postForm                   | postForm (the same three-reading selector)                               |

Every row is the identity — including the last: since the target rule itself carries the three-reading conclusion selector, the alias inherits all readings with no residue, and the transport is  $1 \rightarrow 1$  at reading level, not merely at node level.

**Proposition R27.5** (the exactness identity). *The respelling is exact definitionally: the unpacked node’s combination identity is the same exact canonical-polynomial identity  $c \equiv \sum_i \lambda_i e_i + \sum_k \nu_k g_k$ , unchanged. What remains to make the three readings correct is the fact that they are one exact  $\mathbb{Q}$  identity re-presented under a sign flip — per entry,  $-e(\sigma) \leq 0 \iff e(\sigma) \geq 0$  and  $l(\sigma) - r(\sigma) \leq 0 \iff l(\sigma) \leq r(\sigma)$  — which is the target rule’s reading-transport lemma, proven once in the SYZYGY section and not re-justified here.*

*Proof.* The first claim is reflexivity of the transported field. The second is the cited lemma (the ordered-field axioms negate-and-flip and add-to-both-sides, packaged as the ge-bridge ring lemma); the alias contributes no polarity argument of its own, which is what keeps the  $1 \rightarrow 1$  relation clean.  $\square$

**Proposition R27.6** (soundness corollary). *If every component sequent is semantically valid, then under the payload’s checked hypotheses the sequent  $\Phi \vdash \Psi$  is semantically valid, for every reading  $\Psi$  the selector admits.*

*Proof.* `sound_pBound` = `sound_syzygy` at the transported (identical) payload data, fed the same component validities. There is no other content — the alias genre working as intended.  $\square$

**Remark R27.7** (why the rule exists at all). Semantically the alias adds nothing to its target; its existence is pragmatic, twice over. *Register:* affine entries admit a compact coefficient-row wire spelling, and bound nodes are the bulk commodity of produced certificates — the compact register is worth a tag. *Readings at the seams:* the (G) reading makes a nonnegativity premise (as `ReLU-Pos` demands,  $h \geq 0$ ) reachable directly from bound facts — the once-contemplated standalone  $\geq$ -bridge rule dissolved into the conclusion selector instead — and the (L $\leq$ ) reading hands interval-shaped consumers (`LiRPA`’s endpoint premises) their literal atom with no `ARITHATOM` re-arrangement node in between. Producers get the convenient conclusion shape; the kernel keeps exactly one combination argument.

**Remark R27.8** (the affine restriction is register, not soundness). The register’s “affine entries” — the roster note’s pricing vocabulary, appearing in neither the displayed premises nor the statement — names the wire spelling the rule is priced for, and is not a checked side condition: the combination replay is degree-blind, and the soundness argument (Proposition R27.6) never consults linearity. A higher-degree entry simply belongs to the target rule’s own spelling — what would refuse it is the published register’s compact coefficient-row spelling, an encoding fact of that format (asserted at its register, not displayed here), never the checker. Publishing the restriction beside the rule, while the checker enforces only what soundness needs, mirrors the `FARKAS` page’s linearity note — the register tells producers what the format prices; the checker tells them what truth requires.

| proof step                               | formal anchor                                                                                                                                                                                      |
|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Recipe R27.2 unpack (identity transport) | <code>sound_pBound</code> ’s body: one application of <code>sound_syzygy</code> to the step assembled from <code>PBOUND</code> ’s fields ( <code>PBoundData</code> = the target’s fields verbatim) |
| Recipe R27.3 obligations                 | <code>syzygyCombinationValid</code> and the target’s bookkeeping fields, unchanged                                                                                                                 |
| Recipe R27.4 reading selector            | <code>syzygyConclusionForm</code> (shared, not duplicated)                                                                                                                                         |
| Proposition R27.5 reading transport      | <code>holds_of_syzygyConclusionForm</code> (proven in the SYZYGY section)                                                                                                                          |
| Proposition R27.6                        | <code>sound_pBound</code> (a <code>sound_syzygy</code> specialization)                                                                                                                             |
| wire placement                           | <code>RuleTag.pBound</code> ; no wire tag allocated in <code>v0</code> — producers elaborate before emission (the template’s second branch), and the emit path refuses the rule accordingly        |

## Rule 28. PSDCERTTRI (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *PsdCertTri*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} A \in \mathbb{Q}^{n \times n} \text{ symmetric} \\ Z \text{ upper-triangular, } Z_{ii} > 0 \quad A \cdot Z \text{ lower-triangular, } (A \cdot Z)_{ii} > 0 \text{ (recomputed exactly)} \quad p_1, \dots, p_n \text{ payload terms} \\ c \equiv -\sum_{i,j} A_{ij} p_i p_j \text{ (exact canonical polynomial)} \quad \Psi \in \{c \leq 0, Q \geq 0 \text{ where } c = -Q, l \leq r \text{ where } c = l - r\} \end{array}}{\Phi \vdash \Psi} \text{PSDCERTTRI}$$

PSDCERTTRI is the exact tier of the positive-definiteness certificate family: a premise-free rule concluding nonnegativity of the quadratic form  $\sum_{i,j} A_{ij} p_i p_j$  of a symmetric rational matrix  $A$  at a payload term vector, warranted by an untrusted witness  $Z$  passing three decidable syntactic checks —  $A$  symmetric,  $Z$  upper-triangular with positive diagonal, and the product  $A \cdot Z$  (recomputed exactly, one sparse multiply) lower-triangular with positive diagonal. Those premises are a theorem-strength certificate of  $A \succ 0$ : the congruence  $Z^T(AZ)$  collapses to a positive diagonal, and positivity transports back through the invertible  $Z$ . The matrix-witness check is division-free,  $O(\alpha n^2)$  multiplications ( $\alpha$  average nonzeros per row), and runs in  $\mathbb{Z}$  on integer inputs.

A producer reaches for it wherever a proof obligation is “this explicit matrix is positive (semi)definite” — Lyapunov and storage-function certificates, energy quadratic forms, elimination-free replacements for exact rational LDL factorization, whose  $O(n^3)$  divisions-with-fill-in it displaces. The witness is produced off-stage (a float Cholesky, exact elimination, anything); soundness never consults its accuracy — a forged  $Z$  can only refuse. The concluded fact enters the logic through the same conclusion-form selector as SYZGY (raw zero-form, the flipped  $Q \geq 0$  reading, or a literal  $l \leq r$  split), so it slots directly into SYZGY/FARKAS combinations as a component conclusion; strictness is deliberately dropped at the formula boundary (a premise-free atom carries no  $p \neq 0$ ), so the in-logic fact is  $\succ$ -shaped even though the matrix-side theorem is strict.

What the checker recomputes: the three matrix checks entrywise over exact  $\mathbb{Q}$ , and — the recompute pattern — that the payload conclusion term's canonical polynomial IS the negated quadratic form  $-\sum_{i,j} A_{ij} p_i p_j$  built from the payload terms (in either the wire-keyed or the wire-coordinate polynomial space). What breaks without each hypothesis: without symmetry the congruence argument dies and the premises say nothing about the quadratic form; without the  $Z$  diagonal signs the product argument loses invertibility or positivity (the two sign families are determinant-linked — for an indefinite  $A$ , flipping a column to repair one violates the other); the  $AZ$  pattern and sign checks are the content itself, and for an indefinite  $A$  no witness whatsoever passes; without the conclusion identity the posted atom could be unrelated to  $A$ ; the shape/finiteness/positive-size fields on the conclusion are the standard evaluability-and-vacuity discipline (a zero-size wire satisfies atoms vacuously, and the positive-size field refuses that vacuous surface). The proof section carries the family preamble and walks these in order.

**notes** Provenance: the triangular-inverse method is due to David “davidad” Dalrymple; Geoffrey Irving derived and published the rounding-tolerant integer analysis and formalized that variant at scale (a certified 65536-node spectral expander) — see the proof section's citations. The kernel development here is an independent proof of the exact tier. Wire tag: deferred to the payload-window allocation; the emit path refuses the rule until allocated.

### Lean statement

BernsteinLogic.Soundness . **sound\_psdCertOf**

statement bound

```
-- The shared PsdCert semantic lemma: from the abstract rational-points
quadratic-form fact (supplied per-variant by the PsdCert bridge
theorems), through the checked conclusion identity, to sequent validity
at every reading of `syzygyConclusionForm`. -/
theorem sound_psdCertOf {H : Hypernet} {seq : Sequent} {a : Nat}
  {M : Fin a → Fin a → ℚ} {ts : List Term} {conclusion : Term}
  (hpsd : ∀ x : Fin a → ℚ, 0 ≤ PsdCert.finQSum a fun i =>
    PsdCert.finQSum a fun j => M i j * x i * x j)
  (hconcl : PsdCert.conclCheck a M ts conclusion = true)
  (hshape : ∃ s, conclusion.bshape H = some s)
  (hfin : conclusion.finWires H)
  (hwpos : ∀ w ∈ conclusion.freeWires,
    ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape)
  (hform : syzygyConclusionForm conclusion seq.post) :
  seq.semanticallyValid H
```

## BernsteinLogic.Soundness . sound\_psdCertTri

statement bound

```

/-- PsdCertTri soundness: the /T witness warrants  $A > 0$  over  $\mathbb{R}$ 
(`PsdCert.checkTri_posDef`, davidad's triangular-inverse theorem), whose
quadratic form is nonnegative at every rational point. -/
theorem sound_psdCertTri {H : Hypernet} {seq : Sequent} {a : Nat}
  {A Z : Fin a → Fin a →  $\mathbb{Q}$ } {ts : List Term} {conclusion : Term}
  (hwit : PsdCert.checkTri a A Z = true)
  (hconcl : PsdCert.conclCheck a A ts conclusion = true)
  (hshape :  $\exists s$ , conclusion.bshape H = some s)
  (hfin : conclusion.finWires H)
  (hwpos :  $\forall w \in \text{conclusion.freeWires}$ ,
     $\exists \tau$ , H.wireType w = some  $\tau \wedge 0 < \text{Shape.totalSize } \tau.\text{shape}$ )
  (hform : syzygyConclusionForm conclusion seq.post) :
  seq.semanticallyValid H

```

**The matrix theorems (mathlib side).** The chain from witness check to consumable fact: the congruence collapse and transport lemmas prove the triangular-witness theorem; checkTri\_posDef bridges the executable Bool check to positive definiteness of the  $\mathbb{R}$ -cast; and the rational-points corollary is the exact shape the rule soundness lemma consumes. Soundness never consults witness accuracy — every hypothesis of every theorem below is a checked premise, so a forged witness can only fail a decidable check.

## BernsteinLogic.PsdCert . congruence\_eq\_diagonal

statement bound

```

/-- The congruence  $Z^H * (A * Z)$  collapses to a diagonal matrix when `Z` is
upper-triangular and `A * Z` is lower-triangular, and its diagonal entries
are the products of the two matrices' diagonals. -/
theorem congruence_eq_diagonal {A Z : Matrix n n  $\mathbb{R}$ }
  (hA :  $\forall i j$ , A i j = A j i)
  (hZu :  $\forall \{i j\}$ , j < i → Z i j = 0)
  (hML :  $\forall \{i j\}$ , i < j → (A * Z) i j = 0) :
  ZH * (A * Z) = diagonal fun i => Z i i * (A * Z) i i

```

## BernsteinLogic.PsdCert . posDef\_of\_congruence\_posDef

statement bound

```

/-- Congruence transport (factored from the /T proof for reuse by /DD): if
the congruence  $Z^H * (A * Z)$  is positive definite and `Z` is
upper-triangular with positive diagonal (hence invertible), then `A` is
positive definite. -/
theorem posDef_of_congruence_posDef {A Z : Matrix n n  $\mathbb{R}$ }
  (hZu :  $\forall \{i j\}$ , j < i → Z i j = 0)
  (hZd :  $\forall i$ , 0 < Z i i)
  (hBd : (ZH * (A * Z)).PosDef) :
  A.PosDef

```

## BernsteinLogic.PsdCert . posDef\_of\_triangular\_witness

statement bound

```

/-- **PsdCertTri soundness** (davidad's theorem). A symmetric matrix with an
upper-triangular positive-diagonal witness `Z` such that `A * Z` is
lower-triangular with positive diagonal is positive definite. -/
theorem posDef_of_triangular_witness {A Z : Matrix n n  $\mathbb{R}$ }
  (hA :  $\forall i j$ , A i j = A j i)
  (hZu :  $\forall \{i j\}$ , j < i → Z i j = 0)
  (hZd :  $\forall i$ , 0 < Z i i)
  (hML :  $\forall \{i j\}$ , i < j → (A * Z) i j = 0)
  (hMd :  $\forall i$ , 0 < (A * Z) i i) :
  A.PosDef

```

## BernsteinLogic.PsdCert . checkTri\_posDef

statement bound

```

/-- `checkTri` warrants positive definiteness of the  $\mathbb{R}$ -cast. -/
theorem checkTri_posDef {a : Nat} {A Z : Fin a → Fin a →  $\mathbb{Q}$ }
  (h : checkTri a A Z = true) : (matR A).PosDef

```

## BernsteinLogic.PsdCert . quadForm\_nonneg\_of\_posSemidef

statement bound

```

/-- Nonnegativity of the  $\mathbb{Q}$  quadratic form at every rational point, from
positive semidefiniteness of the  $\mathbb{R}$ -cast — the fact the rule soundness

```

```

lemmas consume. -/
theorem quadForm_nonneg_of_posSemidef {a : Nat} {M : Fin a → Fin a → ℚ}
  (h : (matR M).PosSemidef) (x : Fin a → ℚ) :
  0 ≤ finQSum a (fun i => finQSum a (fun j => M i j * x i * x j))

```

BernsteinLogic.PsdCert . **checkTri\_quadForm\_nonneg**

statement bound

```

/-- /T's consumable corollary. -/
theorem checkTri_quadForm_nonneg {a : Nat} {A Z : Fin a → Fin a → ℚ}
  (h : checkTri a A Z = true) (x : Fin a → ℚ) :
  0 ≤ finQSum a (fun i => finQSum a (fun j => A i j * x i * x j))

```

**The executable check (zero-mathlib closure).** The witness checks are List-based exact rational computations on the executable checker's import closure — displayed in full so the reader can confirm by eye that they are sign, zero-pattern, and entrywise-equality inspections over recomputed products, with no divisions and no trusted certificate data beyond the witness itself; and that `conclCheck` pins the payload conclusion's canonical polynomial to the negated quadratic form (both polynomial branches mirror the `Syzygy` check's split). These definitions are shared by all three family members; the `/DD` and `/K` sections display only their own additional entries.

BernsteinLogic.Core.Catalogue . **qlsum**

definition bound

```

/-- List sum over Rat (zero-mathlib; `List.sum`'s instance path avoided
deliberately on the core closure). -/
def qlsum (l : List Rat) : Rat := l.foldr (· + ·) 0

```

BernsteinLogic.Core.Catalogue . **qabs**

definition bound

```

/-- `|q|` without lattice instances. -/
def qabs (q : Rat) : Rat := if q < 0 then -q else q

```

BernsteinLogic.Core.Catalogue . **finQSum**

definition bound

```

/--  $\sum_{i < n} f i$ , List-based. -/
def finQSum (n : Nat) (f : Fin n → Rat) : Rat := qlsum ((List.finRange n).map f)

```

BernsteinLogic.Core.Catalogue . **mulEntry**

definition bound

```

/-- Entry (i,j) of the product A·B, recomputed exactly. -/
def mulEntry (n : Nat) (A B : Fin n → Fin n → Rat) (i j : Fin n) : Rat :=
  finQSum n (fun k => A i k * B k j)

```

BernsteinLogic.Core.Catalogue . **symB**

definition bound

```

/-- Symmetry, checked entrywise. -/
def symB (n : Nat) (A : Fin n → Fin n → Rat) : Bool :=
  (List.finRange n).all fun i => (List.finRange n).all fun j =>
    decide (A i j = A j i)

```

BernsteinLogic.Core.Catalogue . **upperTriPosDiagB**

definition bound

```

/-- Upper-triangular with positive diagonal, checked entrywise. -/
def upperTriPosDiagB (n : Nat) (Z : Fin n → Fin n → Rat) : Bool :=
  ((List.finRange n).all fun i => (List.finRange n).all fun j =>
    decide (j < i → Z i j = 0)) &&
  ((List.finRange n).all fun i => decide (0 < Z i i))

```

BernsteinLogic.Core.Catalogue . **checkTri**

definition bound

```

/-- PsdCertTri witness check (davidad's theorem, all-syntactic premises):
A symmetric; Z upper-triangular with positive diagonal; A·Z
lower-triangular with positive diagonal. Concludes A > 0 (matrix side);
division-free,  $O(n^2)$  multiplications. -/
def checkTri (n : Nat) (A Z : Fin n → Fin n → Rat) : Bool :=
  symB n A && upperTriPosDiagB n Z &&
  ((List.finRange n).all fun i => (List.finRange n).all fun j =>
    decide (i < j → mulEntry n A Z i j = 0)) &&
  ((List.finRange n).all fun i => decide (0 < mulEntry n A Z i i))

```



BernsteinLogic.Core.Catalogue . **negQuadFormPoly**

definition bound

```

/-- The negated quadratic-form canonical polynomial
-Σi,j Mi,j·pi·pj (zero-normal orientation: the rule's primitive fact is
`conclusion ≤ 0` with conclusion ~ -Q, exposed through
`szygyConclusionForm` – a producer typically emits the ge-flip reading
`Q ≥ 0` with the conclusion term literally `neg Q`). -/
def negQuadFormPoly (n : Nat) (M : Fin n → Fin n → Rat)
  (ps : List QArith.Poly) : QArith.Poly :=
  QArith.Poly.neg ((List.finRange n).foldr (fun i acc =>
    (List.finRange n).foldr (fun j acc2 =>
      QArith.Poly.add (QArith.Poly.scale (M i j)
        (QArith.Poly.mul (ps.getD i.val []) (ps.getD j.val []))) acc2) acc) [])

```

BernsteinLogic.Core.Catalogue . **conclCheckCore**

definition bound

```

/-- Conclusion check, idx-free branch (szygyCheckCore's discipline):
the payload conclusion's canonical polynomial IS the negated quadratic
form of M at the payload terms' canonical polynomials – the recompute
pattern, no coefficient tensors trusted. -/
def conclCheckCore (n : Nat) (M : Fin n → Fin n → Rat) (ts : List Term)
  (conclusion : Term) : Bool :=
  decide (ts.length = n) &&
  conclusion.idxFree && ts.all Term.idxFree &&
  decide (conclusion.toPoly = negQuadFormPoly n M (ts.map Term.toPoly))

```

BernsteinLogic.Core.Catalogue . **conclCheckW**

definition bound

```

/-- Conclusion check, wire-coordinate branch (szygyCheckW's G1 route:
multi-wire idx atoms – the vector-wire quadratic forms the E-4a /
Lyapunov customers emit). -/
def conclCheckW (n : Nat) (M : Fin n → Fin n → Rat) (ts : List Term)
  (conclusion : Term) : Bool :=
  decide (ts.length = n) &&
  (match conclusion.asCoordPolyIdxW?, ts.mapM Term.asCoordPolyIdxW? with
  | some pc, some pts => decide (pc = negQuadFormPoly n M pts)
  | _, _ => false)

```

BernsteinLogic.Core.Catalogue . **conclCheck**

definition bound

```

/-- Either conclusion branch (mirrors `szygyCheck`'s split). -/
def conclCheck (n : Nat) (M : Fin n → Fin n → Rat) (ts : List Term)
  (conclusion : Term) : Bool :=
  conclCheckCore n M ts conclusion || conclCheckW n M ts conclusion

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax. This dictionary is the trio's template; /DD and /K differ in the witness row alone.

- $n \mapsto a : \text{Nat}; A \mapsto A; Z \mapsto Z$  (both  $\text{Fin } a \rightarrow \text{Fin } a \rightarrow \mathbb{Q}$ );  $p_1, \dots, p_n \mapsto ts : \text{List Term}; c \mapsto \text{conclusion} : \text{Term}$ .
- premise-freeness  $\mapsto$  silence, the PCONSTEXACT representation: no hypothesis mentions seq.pre, so  $\Phi$  is arbitrary — the NORM convention of the roster note, carried structurally.
- the three displayed matrix rows (symmetry; upper-triangular  $Z$  with positive diagonal; lower-triangular  $A \cdot Z$  with positive diagonal, recomputed exactly)  $\mapsto$  the single Boolean witness  $\text{hwit} : \text{PsdCert.checkTri } a \ A \ Z = \text{true}$ , whose definition (displayed above) recomputes all three. The mathematics enters through the certified-alongside bridge:  $\text{checkTri\_posDef}$  warrants  $A \succ 0$  over  $\mathbb{R}$ , and  $\text{sound\_psdCertTri}$  consumes only its rational-points quadratic-form corollary  $\text{checkTri\_quadForm\_nonneg}$  — the displayed strictness drop at the formula boundary, visible in the statement's type.

- the canonical-polynomial row  $c \equiv -\sum_{i,j} A_{ij}p_i p_j \mapsto \text{hconcl} : \text{conclCheck } a \text{ A ts conclusion} = \text{true}$  — the identity is checked on canonical forms by either of two certified routes (the idx-free toPoly replay of `conclCheckCore`, or the coordinate-indexed `conclCheckW`), and `sound_psdCertOf` discharges both.
- $\Psi \in \{c \leq 0; Q \geq 0 \text{ where } c = -Q; l \leq r \text{ where } c = l - r\} \mapsto \text{hform} : \text{syzygyConclusionForm conclusion seq.post}$  — the certified selector is literally that three-membered disjunction, reading the conclusion term's head syntactically (shared with `SYZGY` and `PBOUND`).
- with no displayed row: the evaluability pack — `hshape` (the conclusion has a broadcast shape in  $\mathbb{H}$ ), `hfin` (its wires are finite-typed), and `hwpos` (every free wire positively sized). The display omits all three; the certified statement demands them, and `hwpos` is the same vacuous-surface refusal the `FARKAS` display carries inline and the `LiRPA` linkage builds in.

Width note: elementwise atom semantics batch the conclusion (the roster note), so the display's scalar reading is the singleton case of the certified one; conversely the certified statement never uses witness accuracy beyond the Boolean check — a forged  $Z$  refuses, never falsely accepts — and no hypothesis is unused.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.PsdCert`; bound build `be59faf6aff0...` (bound from receipt `872a60504989acef`, 2026-07-29T07:57:01Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers                             | status  |
|-------------------------------|----------------------|------------------------------------|---------|
| <code>13403ea3a4e42f77</code> | 2026-07-28T23:44:21Z | build+decorrelated                 | current |
| <code>6ad2a620d75bd04e</code> | 2026-07-28T23:56:32Z | build+decorrelated                 | current |
| <code>d37ce088bc4caf69</code> | 2026-07-29T00:55:46Z | build+replay+hardened+decorrelated | current |
| <code>872a60504989acef</code> | 2026-07-29T07:57:01Z | build+replay+hardened+decorrelated | current |

5 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_soundpsd, HOL4 development script bern\_tr\_soundpsdScript.sml script bound**Theorem sound\_psdCert0f**

```

Theorem sound_psdCert0f:
  ∀H sq n M ts conclusion.
    (∀x. 0 ≤ quadForm n M x) ∧
    psdConclCheck n M ts conclusion ∧
    (∃s. tmBshape H conclusion = SOME s) ∧
    tmFinWires H conclusion ∧
    (∀w. MEM w (termFreeWires conclusion) ⇒
      ∃τ. H.wireType w = SOME τ ∧ 0 < shapeTotalSize τ.shape) ∧
    syzygyConclusionFormB conclusion sq.post ⇒
    seqSemanticallyValid H sq

```

**Theorem sound\_psdCertTri**

```

Theorem sound_psdCertTri:
  ∀H sq n A Z ts conclusion.
    psdCheckT n A Z ∧
    psdConclCheck n A ts conclusion ∧
    (∃s. tmBshape H conclusion = SOME s) ∧
    tmFinWires H conclusion ∧
    (∀w. MEM w (termFreeWires conclusion) ⇒
      ∃τ. H.wireType w = SOME τ ∧ 0 < shapeTotalSize τ.shape) ∧
    syzygyConclusionFormB conclusion sq.post ⇒
    seqSemanticallyValid H sq

```

**HOL verifier receipts**

documentary hol-strict CLEAN: theory bern\_tr\_soundpsd, 4 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms), 249-theory ancestry — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob c01e0b0c...). The twins mirror the Lean statements clause-for-clause per the script's statement-fidelity inventory: the shared semantic spine (sound\_psdCert0f) runs from the abstract rational-points quadratic-form fact through the checked conclusion identity to sequent validity at every reading of the shared conclusion-form selector, and the per-shape corollary discharges that fact via the ported triangular-witness bridge (psdCheckT\_quadForm, theory bern\_psd, where the witness check itself — psdCheckT — is ported with kernel-EVAL receipts). The payload record and its wellformedness arm ride the ported proof-object stratum (theory bern\_tr\_catdata2, 272 theorems, 0 offences, same run).

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

**Proof**

Genre: admissible rule (tier 2) — a recipe by *reduction to verified machinery* (the PCONSTEXACT pattern): no catalogue derivation of an exact matrix-positivity fact exists, which is the trio's point; each rule discharges against a named kernel-verified matrix theorem. This section also states the certificate-family preamble the PSDCERTDIAG and PSDCERTKERNEL sections reference.

**Recipe R28.1** (the PSD-certificate family — shared shape). All three rules certify positivity of a symmetric matrix  $A \in \mathbb{Q}^{n \times n}$  from an *untrusted witness*, by decidable syntactic checks in exact rational arithmetic — no divisions, no pivoting, no floating point on the trusted side. The shared discipline:

- (i) *Soundness never consults the witness's accuracy.* The conclusion is a theorem of the checked premises; a forged or degraded witness can only cause a false REJECT, never a false ACCEPT. Producers may therefore run arbitrary float factorizations off-stage.
- (ii) *The in-logic conclusion is the quadratic form, not the matrix.* The formula language has no matrix atoms, so the rule's conclusion is nonnegativity of  $Q(p) = \sum_{i,j} A_{ij} p_i p_j$  at a payload term vector  $p_1, \dots, p_n$ : the payload conclusion term  $c$  is checked — by exact canonical-polynomial identity, the recompute pattern — to  $be - \sum_{i,j} A_{ij} p_i p_j$ , and the sequent's posted formula is one of three readings of  $c \leq 0$  (the raw zero-form; the flipped  $Q \geq 0$ ; the literal  $l \leq r$  where  $c = l - r$ ) — the same conclusion-form selector displayed in the SYZGY section, so the fact slots directly into SYZGY/FARKAS combinations as a component conclusion.
- (iii) *Strictness is deliberately dropped at the formula boundary.* The matrix theorems certify  $A \succ 0$  (for /T and /DD), but the premise-free atom ships the  $\succ$ -shaped fact: strict positivity of  $Q(p)$  would need  $p \neq 0$ , which no premise-free atom carries.
- (iv) *The rules are premise-free;*  $\Phi$  is arbitrary (a fact holding in the empty context holds under any premise). Elementwise atom semantics batch the conclusion: at tensor-shaped payload terms, entry  $e$  of the conclusion is the quadratic form at the terms' entry- $e$  values, so one node states a family of instances.

The rewrite, per rule, is degenerate in the PCONSTEXACT sense: one node, zero catalogue premises, discharged by the rule's matrix theorem composed with the shared transport lemma (sound\_psdCert0f) that carries “PSD over  $\mathbb{R}$ ” through “nonnegative at every rational point” and the checked polynomial identity to the posted formula.

**Recipe R28.2** (the  $\text{PSDCERTTRI}$  instance). Witness:  $Z \in \mathbb{Q}^{n \times n}$ . Checked premises (all decidable, all exact):  $A$  symmetric;  $Z$  upper-triangular with positive diagonal;  $M := AZ$  (recomputed exactly, one sparse multiply) lower-triangular with positive diagonal. Then

$$\frac{\text{checkTri}(A, Z) = \text{true} \quad c \equiv -\sum_{i,j} A_{ij} p_i p_j}{\Phi \vdash \Psi} \quad \text{PSDCERTTRI} \xrightarrow{\sim} \text{discharged by sound\_psdCertTri},$$

with  $\Psi$  any reading of  $c \leq 0$  per the preamble. The matrix-witness check costs  $O(\alpha n^2)$  multiplications ( $\alpha$  = average nonzeros per row of  $A$ ) plus  $O(n^2)$  sign/zero inspections, division-free; on integer inputs it runs entirely in  $\mathbb{Z}$  with  $\{*, +, <\}$ . (The conclusion-identity recompute is additional, and its cost tracks the payload terms' polynomial sizes, not  $n$  alone.)

**Proposition R28.3** (matrix warrant). *If  $A$  is symmetric,  $Z$  upper-triangular with  $Z_{ii} > 0$ , and  $AZ$  lower-triangular with  $(AZ)_{ii} > 0$ , then  $A \succ 0$  over  $\mathbb{R}$ .*

*Proof.* Let  $B := Z^T(AZ)$ . As a product of two lower-triangular matrices ( $Z^T$  and  $AZ$ ) it is lower-triangular; since  $Z^T AZ$  is a congruence of the symmetric  $A$  it is also symmetric; hence  $B$  is *diagonal*, and the sum  $B_{ii} = \sum_k Z_{ki}(AZ)_{ki}$  collapses to the single term  $Z_{ii}(AZ)_{ii} > 0$ . So  $B \succ 0$ . Moreover  $\det Z = \prod_i Z_{ii} > 0$  makes  $Z$  invertible, and  $A = Z^{-T} B Z^{-1} \succ 0$  by congruence. (This is the route the kernel proof takes — `congruence_eq_diagonal` then `posDef_of_congruence_posDef` — avoiding leading principal minors; the classical alternative reads  $A = MZ^{-1}$  as an LU factorization with positive-diagonal factors and applies Sylvester's criterion [115].)  $\square$

*Remark R28.4* (each hypothesis, and what removing it admits). *Symmetry of  $A$* : without it  $Z^T(AZ)$  need not be symmetric, the diagonal-collapse argument dies, and the quadratic form reads only the symmetric part of  $A$  — the premises would say nothing about it. *Upper-triangularity and positive diagonal of  $Z$* : triangularity is the cheapest syntactic invertibility certificate ( $\det Z = \prod Z_{ii}$ ), and the diagonal signs feed  $B_{ii} > 0$ ; the two sign families are determinant-linked: for an indefinite  $A$  — a genuinely negative pivot — “repairing” the  $AZ$  diagonal sign by flipping the offending column of  $Z$  only moves the violation into  $Z$ 's own diagonal check ( $A = \text{diag}(1, -1)$ ,  $Z = \text{diag}(1, z)$ : making  $-z$  positive forces  $z < 0$ ; when  $A$  is in fact positive definite, a flip can repair both checks at once — acceptance working as intended, not a leak). *The pattern and sign checks on  $AZ$* : this is the entire content — the diagonal of  $AZ$  carries the pivot signs of  $A$ , and for an indefinite  $A$  no witness whatsoever passes (by Proposition R28.3 an accepting witness would prove  $A \succ 0$ ). *The conclusion identity*: without the recomputed polynomial check the posted atom could be an arbitrary formula unrelated to  $A$ .

*Remark R28.5* ( $/T$  needs an exact witness — the honest split with  $/DD$ ). The natural producer emits  $Z \approx$  an inverse transposed Cholesky factor. A *rounded* such  $Z$  makes the strictly-upper entries of  $AZ$  small but nonzero, and  $/T$ 's pattern check rightly refuses. An exact witness always exists when  $A \succ 0$ , and rationally:  $\text{LDL}^T$  elimination gives  $A = \text{LDL}^T$  with unit lower-triangular rational  $L$  and positive rational diagonal  $D$ , so  $Z := L^{-T}$  is rational upper-triangular with positive diagonal and  $AZ = LD$  is lower-triangular with positive diagonal — clear denominators columnwise if integer entries are wanted (each column is determined up to positive scale) — but the entries grow like scaled minors, i.e.  $\text{LDL}$ -style bit growth. (The Cholesky factor itself is generally irrational; the  $\text{LDL}$  route is the rational construction.) At sizes where that growth bites, the rounding-tolerant companion  $\text{PSDCERTDIAG}$  is the production tier; at moderate sizes  $/T$ 's  $O(\alpha n^2)$  check wins the clock. Neither tier ever risks soundness in the trade — only completeness margin moves.

*Remark R28.6* (provenance). The triangular-inverse method is due to David “davidad” Dalrymple [56]; Geoffrey Irving derived the rounding-tolerant integer variant and published the certificate analysis [123], then formalized that variant in Lean at scale (a certified 65536-node spectral expander) in the course of a sorting-network formalization [122]. The kernel development cited here is an independent in-house proof of the exact tier, grown to the  $/DD$  and  $/K$  companions.

| proof step                                           | formal anchor                                                                                                                                                                                                                                          |
|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Recipe R28.2 warrant<br>matrix theorem (Prop. R28.3) | <code>sound_psdCertTri</code> (Soundness); shared transport <code>sound_psdCertOf</code><br><code>PsdCert.posDef_of_triangular_witness</code> via <code>congruence_eq_diagonal</code> ,<br><code>posDef_of_congruence_posDef</code>                    |
| check-to-theorem bridge                              | <code>PsdCert.checkTri_posDef</code> ; <code>rational-points</code> corollary                                                                                                                                                                          |
| the executable check                                 | <code>checkTri_quadForm_nonneg</code> , <code>quadForm_nonneg_of_posSemidef</code><br><code>PsdCert.checkTri</code> (with <code>symB</code> , <code>upperTriPosDiagB</code> , <code>mulEntry</code> ) — <code>zero-mathlib</code> , exact $\mathbb{Q}$ |
| conclusion identity                                  | <code>PsdCert.conclCheck</code> against <code>negQuadFormPoly</code> ( <code>idx-free</code> and <code>wire-coordinate</code> branches)                                                                                                                |
| conclusion readings                                  | the <code>Svzvgv</code> section's conclusion-form selector and its transport lemma                                                                                                                                                                     |

## Rule 29. PSDCERTDIAG (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *PsdCertDiag*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} A \in \mathbb{Q}^{n \times n} \text{ symmetric} \quad Z \text{ upper-triangular, } Z_{ii} > 0 \\ B := Z^T A Z \text{ strictly row-diagonally dominant (recomputed exactly)} \quad p_1, \dots, p_n \text{ payload terms} \\ c \equiv -\sum_{i,j} A_{ij} p_i p_j \text{ (exact canonical polynomial)} \quad \Psi \in \{ c \leq 0, \ Q \geq 0 \text{ where } c = -Q, \ l \leq r \text{ where } c = l - r \} \end{array}}{\Phi \vdash \Psi} \text{PSDCERTDIAG}$$

PSDCERTDIAG is the rounding-tolerant tier of the positive-definiteness certificate family (the family discipline — untrusted witness, quadratic-form conclusion through SYZGY's conclusion-form selector, strictness dropped at the formula boundary, premise-freedom — is stated once in the PSDCERTTRI section). Here the checked premises are:  $A$  symmetric;  $Z$  upper-triangular with positive diagonal; and the congruence  $B := Z^T A Z$ , recomputed exactly by the checker, strictly row-diagonally dominant (positive diagonal implied, since the off-row absolute sum is nonnegative). A symmetric strictly dominant matrix is positive definite — the Gershgorin-circle reading; the kernel proof is a self-contained arithmetic-geometric-mean regrouping of the quadratic form — and positivity transports back through the invertible  $Z$ .

This is the tier a real producer feeds:  $Z := \lceil 2^s R^{-1} \rceil$  rounded from an off-stage float Cholesky  $A \approx R^T R$  works directly, with the required scale  $s$  growing only logarithmically in dimension and input scale rather than tracking any accuracy target — witness entries encode in  $s + O(\log^+ \|R^{-1}\|_{\max})$  bits (scale exponent plus the factorization's own magnitude), dimension entering only through the logarithm, where the exact  $/T$  witness inherits minor-scale bit growth. What a too-coarse witness costs is only completeness margin: the check fails closed and the producer retries finer, with soundness never in play. The price relative to  $/T$  is the  $O(n^3)$  congruence product — on bounded-width integers, embarrassingly parallel, still division-free.

What breaks without each hypothesis: without symmetry of  $A$  the regrouping step (which folds the dominance envelope using  $B_{ij} = B_{ji}$ ) fails and row dominance alone bounds nothing about the form; without strictness the matrix warrant  $A \succ 0$  would overclaim on singular matrices (a path-graph Laplacian is weakly dominant with the constants in its kernel — the  $\succ$ -shaped posted atom would survive, but the strict theorem  $/K$  consumes on its inner block would not); without exact recomputation of  $B$  a wire-supplied congruence could be dominant while unrelated to  $A$ ; the  $Z$  checks are, as in  $/T$ , the cheapest syntactic invertibility certificate. The conclusion identity and evaluability fields carry the same discipline as  $/T$ . The proof section walks these in order.

### Lean statement

BernsteinLogic.Soundness . sound\_psdCertDiag

statement bound

```
/-- PsdCertDiag soundness: the /DD witness warrants  $A > 0$  over  $\mathbb{R}$ 
(`PsdCert.checkDiag_posDef`, the diagonal-dominance congruence). -/
theorem sound_psdCertDiag {H : Hypernet} {seq : Sequent} {a : Nat}
  {A Z : Fin a → Fin a → ℚ} {ts : List Term} {conclusion : Term}
  (hwit : PsdCert.checkDiag a A Z = true)
  (hconcl : PsdCert.conclCheck a A ts conclusion = true)
  (hshape : ∃ s, conclusion.bshape H = some s)
  (hfin : conclusion.finWires H)
  (hwpos : ∀ w ∈ conclusion.freeWires,
    ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape)
  (hform : syzygyConclusionForm conclusion seq.post) :
  seq.semanticallyValid H
```

**The dominance core.** `posDef_of_dominant` is the self-contained finite-sum argument (no spectral machinery): off-diagonal mass retreats into its arithmetic-geometric-mean envelope and symmetry folds the envelope onto the diagonal, leaving  $\sum_i (B_{ii} - \sum_{j \neq i} |B_{ij}|) x_i^2 > 0$ . `posDef_of_dd_witness` composes it with the congruence transport displayed in the PSDCERTTRI section; `checkDiag_posDef` bridges the executable check, and the rational-points corollary feeds the rule lemma.

BernsteinLogic.PsdCert . posDef\_of\_dominant

statement bound

```
/-- **The /DD dominance core**: a symmetric strictly row-diagonally
dominant real matrix is positive definite. Self-contained Finset
algebra — the AM–GM regrouping: off-diagonal mass retreats into
```

```

`Σi (Σ{j≠i} |Bij|)·xi2` by symmetry, leaving `Σi (Bii - Σ{j≠i}|Bij|)·xi2`,
positive at any `x ≠ 0`. (Positivity of the diagonal is implied:
`Bii > Σ{j≠i}|Bij| ≥ 0`.) -/
theorem posDef_of_dominant [DecidableEq n] {B : Matrix n n ℝ}
  (hsym : ∀ i j, B i j = B j i)
  (hdom : ∀ i, (Σ j ∈ Finset.univ.erase i, |B i j|) < B i i) :
  B.PosDef

```

BernsteinLogic.PsdCert . posDef\_of\_dd\_witness

statement bound

```

/-- **PsdCertDiag soundness** (Irving's formalized shape): a symmetric
matrix whose congruence `ZH(AZ)` against an upper-triangular
positive-diagonal witness is strictly row-diagonally dominant is
positive definite. -/
theorem posDef_of_dd_witness [DecidableEq n] {A Z : Matrix n n ℝ}
  (hA : ∀ i j, A i j = A j i)
  (hZu : ∀ {i j}, j < i → Z i j = 0)
  (hZd : ∀ i, 0 < Z i i)
  (hdom : ∀ i, (Σ j ∈ Finset.univ.erase i, |(ZH * (A * Z)) i j|)
    < (ZH * (A * Z)) i i) :
  A.PosDef

```

BernsteinLogic.PsdCert . checkDiag\_posDef

statement bound

```

/-- `checkDiag` warrants positive definiteness of the ℝ-cast. -/
theorem checkDiag_posDef {a : Nat} {A Z : Fin a → Fin a → ℚ}
  (h : checkDiag a A Z = true) : (matR A).PosDef

```

BernsteinLogic.PsdCert . checkDiag\_quadForm\_nonneg

statement bound

```

/-- /DD's consumable corollary. -/
theorem checkDiag_quadForm_nonneg {a : Nat} {A Z : Fin a → Fin a → ℚ}
  (h : checkDiag a A Z = true) (x : Fin a → ℚ) :
  0 ≤ finQSum a (fun i => finQSum a (fun j => A i j * x i * x j))

```

**The executable check — what /DD adds.** Only the exact congruence entries and the dominance inspection are new; the shared helpers (finQSum, qabs, upperTriPosDiagB, the conclusion check) are displayed in the PsdCERTTri section.

BernsteinLogic.Core.Catalogue . congEntry

definition bound

```

/-- Entry (i,j) of the congruence B := ZT·(A·Z), recomputed exactly. -/
def congEntry (n : Nat) (A Z : Fin n → Fin n → Rat) (i j : Fin n) : Rat :=
  finQSum n (fun k => Z k i * mulEntry n A Z k j)

```

BernsteinLogic.Core.Catalogue . checkDiag

definition bound

```

/-- PsdCertDiag witness check (Irving's formalized shape – the tier a
float-LDL ROUNDED witness actually feeds): A symmetric; Z
upper-triangular with positive diagonal; B := ZTAZ strictly
row-diagonally dominant (whose positive diagonal is implied: the
off-row sum is ≥ 0). Concludes A > 0. -/
def checkDiag (n : Nat) (A Z : Fin n → Fin n → Rat) : Bool :=
  symB n A && upperTriPosDiagB n Z &&
  ((List.finRange n).all fun i =>
    decide (finQSum n (fun j => if j = i then 0 else qabs (congEntry n A Z i j))
      < congEntry n A Z i i))

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.



Verbatim the `PsDCertTri` dictionary — premise-free silence, the same `ts/conclusion` binders, the same `conclCheck` and `syzygyConclusionForm` rows, the same undisplayed evaluability pack (`hshape`, `hfin`, `hwpos`) — except the witness row: the displayed congruence  $B := Z^T A Z$  strictly row-diagonally dominant  $\mapsto$  `hwit : PsdCert.checkDiag a A Z = true`, entering through `checkDiag_posDef` and consumed as its rational-points corollary `checkDiag_quadForm_nonneg`. The shared assembly is one certified lemma, `sound_psdCertOf`, visible above — the dictionary's claim that the trio differs only in the witness is itself a fact of the statements, not editorial.

## Lean verifier receipts

module `BernsteinLogic.Soundness`; bound build `a20d90e66c13...` (bound from receipt `a9c0e34d3e3c9c8c`, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>a9c0e34d3e3c9c8c</code> | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.PsdCert`; bound build `be59faf6aff0...` (bound from receipt `872a60504989acef`, 2026-07-29T07:57:01Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers                             | status  |
|-------------------------------|----------------------|------------------------------------|---------|
| <code>13403ea3a4e42f77</code> | 2026-07-28T23:44:21Z | build+decorrelated                 | current |
| <code>6ad2a620d75bd04e</code> | 2026-07-28T23:56:32Z | build+decorrelated                 | current |
| <code>d37ce088bc4caf69</code> | 2026-07-29T00:55:46Z | build+replay+hardened+decorrelated | current |
| <code>872a60504989acef</code> | 2026-07-29T07:57:01Z | build+replay+hardened+decorrelated | current |

5 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.Core.Catalogue`; bound build `cbc92e976a1d...` (bound from receipt `5abdb3589707b5df`, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers             | status  |
|-------------------------------|----------------------|--------------------|---------|
| <code>24804e66edba0bc9</code> | 2026-08-03T20:25:09Z | build+decorrelated | current |
| <code>5abdb3589707b5df</code> | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

- build (lake build (elaboration)) current
- replay (leanchecker (official kernel replay)) current
- hardened (safe\_verify (hardened screen)) current
- decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory `bern_tr_soundpsd`, HOL4 development script `bern_tr_soundpsdScript.sml` script bound

### Theorem `sound_psdCertDiag`

Theorem `sound_psdCertDiag`:  
 $\forall h \, sq \, n \, A \, Z \, ts \, conclusion.$   
 $psdCheckDD \, n \, A \, Z \, \wedge$   
 $psdConclCheck \, n \, A \, ts \, conclusion \, \wedge$



```
(∃s. tmBshape H conclusion = SOME s) ∧
tmFinWires H conclusion ∧
(∀w. MEM w (termFreeWires conclusion) ⇒
  ∃τ. H.wireType w = SOME τ ∧ 0 < shapeTotalSize τ.shape) ∧
syzygyConclusionFormB conclusion sq.post ⇒
seqSemanticallyValid H sq
```

## HOL verifier receipts

**documentary** hol-strict CLEAN: theory bern\_tr\_soundpsd, 4 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms), 249-theory ancestry — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob c01e0b0c...). The shared semantic spine (sound\_psdCert0f) is displayed in the /Tri section per the family-preamble convention; this corollary discharges the quadratic-form fact via the ported congruence-plus-row-dominance bridge (psdCheckDD\_quadForm, theory bern\_psd, where the witness check itself — psdCheckDD — is ported with kernel-EVAL receipts). The payload record and its wellformedness arm ride the ported proof-object stratum (theory bern\_tr\_catdata2, 272 theorems, 0 offences, same run).

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — reduction to verified machinery. The certificate-family preamble — untrusted witness, quadratic-form conclusion through the SYZGYV conclusion-form selector, strictness dropped at the formula boundary, premise-freedom, elementwise batching — is stated in the PSDCERTTRI section (its first recipe) and applies verbatim; only the witness check differs here.

**Recipe R29.1** (the PSDCERTDIAG instance). **Witness:**  $Z \in \mathbb{Q}^{n \times n}$ . **Checked premises:**  $A$  symmetric;  $Z$  upper-triangular with positive diagonal; and the congruence  $B := Z^T A Z$  — *recomputed exactly by the checker* — strictly row-diagonally dominant:  $\sum_{j \neq i} |B_{ij}| < B_{ii}$  for every  $i$ . (Positivity of the diagonal is implied: the off-row sum is  $\geq 0$ .) Then

$$\frac{\text{checkDiag}(A, Z) = \text{true} \quad c \equiv -\sum_{i,j} A_{ij} p_i p_j}{\Phi \vdash \Psi} \text{PSDCERTDIAG} \rightsquigarrow \text{discharged by sound\_psdCertDiag.}$$

The matrix-witness check costs  $O(\alpha n^2 + n^3)$  multiplications ( $\alpha$  the average nonzeros per row of  $A$ , as at /T) with the product  $AZ$  materialized once (the  $Z^T(AZ)$  congruence is the  $n^3$  term; the reference check displayed below inlines the product entrywise for readability — it is the specification, not the cost model) on integers of *bounded* width — input bits plus twice the witness bits plus  $O(\log n)$  for the nested sums — embarrassingly parallel, still division-free.

**Proposition R29.2** (matrix warrant). *A symmetric real matrix  $B$  with  $\sum_{j \neq i} |B_{ij}| < B_{ii}$  for all  $i$  is positive definite; hence if additionally  $Z$  is upper-triangular with positive diagonal and  $B = Z^T A Z$  for symmetric  $A$ , then  $A \succ 0$ .*

*Proof.* For  $x \neq 0$ , split the quadratic form and bound each off-diagonal term by its arithmetic–geometric–mean envelope,  $x_i B_{ij} x_j \geq -|B_{ij}|(x_i^2 + x_j^2)/2$ ; folding the envelope's two symmetric halves by  $B_{ij} = B_{ji}$  gives

$$x^T B x \geq \sum_i \left( B_{ii} - \sum_{j \neq i} |B_{ij}| \right) x_i^2 > 0.$$

(The kernel proof is exactly this self-contained finite-sum argument — posDef\_of\_dominant — with no spectral machinery. The classical reading: strict dominance with positive diagonal places every Gershgorin disc in the open right half-plane, and symmetry makes the spectrum real, hence positive [88, 115, 230].) Transport through the invertible  $Z$  as in the /T proof (posDef\_of\_congruence\_posDef) yields  $A \succ 0$ .  $\square$

**Remark R29.3** (what each hypothesis is doing). *Symmetry of  $A$*  makes  $B$  symmetric by construction, which the envelope-folding step consumes ( $B_{ij} = B_{ji}$ ) — row dominance of a nonsymmetric matrix bounds neither the symmetric part's rows nor the form. *Triangularity + positive diagonal of  $Z$*  is, as in /T, only the cheapest syntactic invertibility certificate — invertibility is all the transport step needs, and triangularity is what the natural producer emits anyway. *Strict dominance* is the content of the strict tier: weakening it to non-strict admits singular  $B$  (e.g. the graph Laplacian of a path: weakly dominant, kernel = constants), and the matrix warrant  $A \succ 0$  would overclaim — the posted  $\succ$ -shaped atom would in fact survive, but the strict theorem is what PSDCERTKERNEL consumes on its inner block to confine kernels, and that consumer admits no slack. *Exact recomputation of  $B$* : the congruence is never taken from the certificate — a wire-supplied  $B$  could be dominant while unrelated to  $A$ .

**Remark R29.4** (rounding tolerance — why this is the production tier). Feed  $Z := \lceil 2^s R^{-1} \rceil$  from an off-stage float Cholesky  $A \approx R^T R$ : writing  $Z = 2^s R^{-1} + E$  with rounding  $|E_{ij}| \leq \frac{1}{2}$ , exact expansion (against an exact factorization) gives  $B = Z^T A Z = 4^s I + 2^s (RE + (RE)^T) + E^T A E$  — entrywise contamination  $O(2^s n \|R\|_{\max} + n^2 \|A\|_{\max})$ , off-row dominance sums one factor of  $n$  larger — plus the image of the float residual  $\Delta := A - R^T R$  — the term  $Z^T \Delta Z$ , of normalized leading size  $R^{-T} \Delta R^{-1}$  — whose control is a mantissa requirement of order  $\log_2 \kappa(A) + O(\log n)$  bits on the factorization (the dimension guard bits of the standard backward-error bound), not on the witness. Strict dominance therefore holds once  $2^s$  clears a polynomial in

$n$  times the input scales: a scale exponent  $s = O(\log n + \log \|R\|_{\max})$  suffices (sharp constants in the cited certificate analysis), so the witness *scale* grows only *logarithmically* in dimension and scale and never tracks an accuracy target; the witness *entries* encode in  $s + O(\log^+ \|R^{-1}\|_{\max})$  bits — the second term is the factorization’s own magnitude ( $Z$  rounds  $2^s R^{-1}$ ), present at any scale, and no accuracy target either. Contrast /T, whose exact witness inherits minor-scale bit growth. What a too-coarse  $Z$  costs is only completeness margin: the check fails closed, and the producer retries at a finer scale with soundness never in play. This rounded-integer-witness shape is the one Irving analysed and formalized at scale [122, 123]; the underlying tolerance phenomenon is the factor-width/scaled-diagonal-dominance circle of ideas [24] — context, not a characterization: that theorem equates factor-width two with scaled diagonal dominance of  $A$  *itself* (diagonal scaling only), while /DD checks dominance of the congruence target  $B = Z^T A Z$  for arbitrary triangular  $Z$ , and so accepts matrices far outside the SDD class — with an exact LDL witness the congruence target is diagonal, so /DD’s exact-witness reach is *every* rational SPD matrix, while e.g. the  $3 \times 3$  equicorrelation matrix with off-diagonal  $\rho \in (\frac{1}{2}, 1)$  is SPD but not SDD under any positive diagonal scaling.

*Remark R29.5* (the /T—/DD complementarity, stated once). Same witness idea, dual regimes: /T is exact-witness,  $O(\alpha n^2)$ , and wins at moderate sizes and exact-by-construction witnesses; /DD tolerates any rounding at  $O(n^3)$  bounded-width parallel arithmetic and wins at scale. A producer holding a float factorization reaches for /DD by default and /T when it can clear denominators cheaply. Both refuse indefinite  $A$  against every witness.

| proof step                     | formal anchor                                                                     |
|--------------------------------|-----------------------------------------------------------------------------------|
| Recipe R29.1 warrant           | sound_psdCertDiag (Soundness); shared transport sound_psdCert0f (see Ps-DCERTTri) |
| dominance core (Prop. R29.2)   | PsdCert.posDef_of_dominant; assembled in posDef_of_dd_witness                     |
| check-to-theorem bridge        | PsdCert.checkDiag_posDef; rational-points corollary<br>checkDiag_quadForm_nonneg  |
| the executable check           | PsdCert.checkDiag with the exact congruence entries congEntry                     |
| conclusion identity + readings | PsdCert.conclCheck; the SYZYGY section’s conclusion-form selector                 |

## Rule 30. PSDCERTKERNEL (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — `PsdCertKernel`. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} A \in \mathbb{Q}^{(m+k) \times (m+k)}, \text{ kernel coordinates last} \quad L := (I \mid N), N \in \mathbb{Q}^{m \times k} \\ A = L^\top B L \text{ entrywise exactly} \quad B \in \mathbb{Q}^{m \times m} \text{ with a /T-or-/DD inner witness } Z \quad p_1, \dots, p_{m+k} \text{ payload terms} \\ c \equiv -\sum_{i,j} A_{ij} p_i p_j \text{ (exact canonical polynomial)} \quad \Psi \in \{c \leq 0, Q \geq 0 \text{ where } c = -Q, l \leq r \text{ where } c = l - r\} \end{array}}{\Phi \vdash \Psi} \text{PSDCERTKERNEL}$$

PSDCERTKERNEL is the semidefinite member of the certificate family (family discipline in the PSDCERTTRI section): it certifies  $A \succcurlyeq 0$  with an exactly-checked kernel, for the boundary cases the two strict tiers cannot touch — energy forms with conservation laws, graph Laplacians with their all-ones kernel, sharp certificates whose slack vanishes identically. Data:  $A$  over  $\mathbb{Q}^{(m+k) \times (m+k)}$  with the  $k$  kernel coordinates last (the producer applies its permutation before emission — conjugation is index renaming, so the identity-permutation form loses no generality); a rational kernel description  $N$ ; and a leading block  $B$  with a /T-or-/DD inner witness. The checked premise is a single entrywise-exact factorization:  $A = L^\top B L$  for the kernel-completion rows  $L = (I_m \mid N)$  — equivalent, given the inner check's own symmetry verification of  $B$ , to  $(A \text{ symmetric} \wedge A \cdot K = 0 \text{ exactly for } K = (-N; I_k) \wedge \text{leading block} = B)$ , the cross and trailing blocks being forced either way, so a forged  $N$  or  $B$  refuses — plus the strict inner check on  $B$ . Rectangular congruence then gives  $A \succcurlyeq 0$  with the form vanishing exactly on the span of  $K$ : the certificate fixes kernel and rank, not just semidefiniteness.

The kernel data must be exact by necessity, not by preference: the semidefinite-with-kernel matrices form a boundary set with empty interior, so no rounding-tolerant witness for kernel directions can exist — any check tolerant to kernel perturbation also accepts slightly-indefinite matrices. Exact kernel plus rounding-tolerant bulk (the /DD inner witness) is therefore the forced split — tolerance can live nowhere else — and it matches the producers: kernels are semantic (theorems supply  $N$ ), floats only ever touch the bulk witness  $Z$ . Corners are structural:  $k = 0$  degenerates to the inner strict tier;  $m = 0$  forces  $A = 0$ , where the factorization premise is the entire content.

What breaks without each hypothesis: an approximate factorization certifies nothing (the boundary argument above); a merely semidefinite inner block would hide undeclared kernel directions inside  $B$ ; the kernel-coordinates-last convention is pure bookkeeping. What reaches the formula layer is the quadratic-form conclusion — honest for this rule, since it is  $\succcurlyeq$ -shaped — while the checked kernel/rank content becomes formula-expressible only with closed tensor terms (the extension the PCONSTEXACT section records); it is certificate-level assurance meanwhile, deliberately not hand-faked into the formula grammar. The proof section walks these in order.

### Lean statement

BernsteinLogic.Soundness . `sound_psdCertKernel`

statement bound

```
-- PsdCertKernel soundness: the /K witness warrants A ≥ 0 over ℝ
(`PsdCert.checkKernel_posSemidef`, the kernel-completion congruence). -/
theorem sound_psdCertKernel {H : Hypernet} {seq : Sequent} {m k : Nat}
  {A : Fin (m + k) → Fin (m + k) → ℚ} {N : Fin m → Fin k → ℚ}
  {B : Fin m → Fin m → ℚ} {innerT : Bool} {Z : Fin m → Fin m → ℚ}
  {ts : List Term} {conclusion : Term}
  (hwit : PsdCert.checkKernel m k A N B innerT Z = true)
  (hconcl : PsdCert.conclCheck (m + k) A ts conclusion = true)
  (hshape : ∃ s, conclusion.bshape H = some s)
  (hfin : conclusion.finWires H)
  (hwpos : ∀ w ∈ conclusion.freeWires,
    ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape)
  (hform : syzygyConclusionForm conclusion seq.post) :
  seq.semanticallyValid H
```

**The rectangular congruence.** `posSemidef_of_factor` is one line — semidefiniteness transports through ANY rectangular congruence, no invertibility anywhere — which is exactly why the factorization premise can carry the whole rule. `checkKernel_posSemidef` bridges the executable check (selecting the /T or /DD theorem for the inner block), and the rational-points corollary feeds the rule lemma.

BernsteinLogic.PsdCert . `posSemidef_of_factor`

statement bound

```

/-- **PsdCertKernel soundness core**: a matrix exhibited as the two-sided
congruence `L^HBL` of a positive-semidefinite block is positive
semidefinite (rectangular congruence; no invertibility anywhere). -/
theorem posSemidef_of_factor {m' n' : Type*} [Fintype m'] [Fintype n']
  {A : Matrix n' n' ℝ} {B : Matrix m' m' ℝ} {L : Matrix m' n' ℝ}
  (hB : B.PosSemidef) (hA : A = L^H * B * L) : A.PosSemidef

```

BernsteinLogic.PsdCert . **checkKernel\_posSemidef**

statement bound

```

/-- `checkKernel` warrants positive semidefiniteness of the ℝ-cast. -/
theorem checkKernel_posSemidef {m k : Nat} {A : Fin (m + k) → Fin (m + k) → ℚ}
  {N : Fin m → Fin k → ℚ} {B : Fin m → Fin m → ℚ} {innerT : Bool}
  {Z : Fin m → Fin m → ℚ}
  (h : checkKernel m k A N B innerT Z = true) : (matR A).PosSemidef

```

BernsteinLogic.PsdCert . **checkKernel\_quadForm\_nonneg**

statement bound

```

/-- /K's consumable corollary. -/
theorem checkKernel_quadForm_nonneg {m k : Nat} {A : Fin (m + k) → Fin (m + k) → ℚ}
  {N : Fin m → Fin k → ℚ} {B : Fin m → Fin m → ℚ} {innerT : Bool}
  {Z : Fin m → Fin m → ℚ}
  (h : checkKernel m k A N B innerT Z = true) (x : Fin (m + k) → ℚ) :
  0 ≤ finQSum (m + k) (fun i => finQSum (m + k) (fun j => A i j * x i * x j))

```

**The executable check — what /K adds.** The kernel-completion rows and the entrywise factorization check; the inner witness reuses checkTri/checkDiag and the conclusion check is shared (both displayed in the PsdCERTTri section).

BernsteinLogic.Core.Catalogue . **kernelExt**

definition bound

```

/-- The kernel-completion row matrix L = (I | N) : m × (m+k). -/
def kernelExt (m k : Nat) (N : Fin m → Fin k → Rat) (a : Fin m)
  (j : Fin (m + k)) : Rat :=
  if h : j.val < m then (if a.val = j.val then 1 else 0)
  else N a (j.val - m, by have := j.isLt; omega)

```

BernsteinLogic.Core.Catalogue . **checkKernel**

definition bound

```

/-- PsdCertKernel witness check (PSD-with-kernel companion): A – over
`Fin (m+k)`, the kernel coordinates LAST – IS the two-sided
kernel-completion congruence L^HBL of its leading block B, entrywise
exactly, plus a T-or-DD inner witness for B. Equivalent to the note's §1
premises (A symmetric ∧ A·K = 0 exactly for K = (-N; I_k) ∧ leading
block = B): C = BN and D = N^HBN are forced either way, and a forged N
or B refuses. The producer applies its permutation P before emission –
P-conjugation is index renaming on matrix and term vector alike, so the
catalogue rule is the P = id core. Corners are structural: k = 0 gives
L = I (A = B); m = 0 forces A = 0 (empty B, no inner witness content). -/
def checkKernel (m k : Nat) (A : Fin (m + k) → Fin (m + k) → Rat)
  (N : Fin m → Fin k → Rat) (B : Fin m → Fin m → Rat)
  (innerT : Bool) (Z : Fin m → Fin m → Rat) : Bool :=
  ((List.finRange (m + k)).all fun i => (List.finRange (m + k)).all fun j =>
    decide (A i j = qlsum ((List.finRange m).map fun a =>
      qlsum ((List.finRange m).map fun b =>
        kernelExt m k N a i * B a b * kernelExt m k N b j)))) &&
    (if innerT then checkTri m B Z else checkDiag m B Z))

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

As at PsdCERTTri — premise-free silence, the shared conclCheck row at size  $m + k$ , the same syzygyConclusionForm selector and undisplayed evaluability pack — with the witness row exchanged for the kernel-completion data:

- the block sizes  $\mapsto m\ k : \text{Nat}$  with  $A : \text{Fin } (m + k) \rightarrow \text{Fin } (m + k) \rightarrow \mathbb{Q}$  (kernel coordinates last, as displayed — the producer's permutation is applied before emission, so the certified rule is the  $P = \text{id}$  core of the roster note);  $N \mapsto N : \text{Fin } m \rightarrow \text{Fin } k \rightarrow \mathbb{Q}$ ;  $B \mapsto B$ ; the “/T-or-/DD inner witness”  $\mapsto$  the pair  $\text{innerT} : \text{Bool}$  (which inner check) and  $Z : \text{Fin } m \rightarrow \text{Fin } m \rightarrow \mathbb{Q}$  (its witness).
- the displayed factorization row  $A = L^T B L$  entrywise exactly,  $L = (I \mid N)$ , together with the inner witness  $\mapsto$  the single Boolean  $\text{hwit} : \text{PsdCert.checkKernel } m\ k\ A\ N\ B\ \text{innerT}\ Z = \text{true}$ , entering through  $\text{checkKernel\_posSemidef}$  ( $A \succcurlyeq 0$  — the trio's one genuinely  $\succcurlyeq$ -shaped member, so here the formula boundary loses nothing) and consumed as  $\text{checkKernel\_quadForm\_nonneg}$ .

Width note, beyond /T's: the checked kernel and rank content (the roster note's §1 equivalents) is consumed by the matrix-side bridge but never surfaces in the sequent — the certified formula-level claim is exactly the quadratic-form conclusion, neither more nor less, which is the honest reading the commentary states and this triple pins.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.PsdCert; bound build be59faf6aff0... (bound from receipt 872a60504989acef, 2026-07-29T07:57:01Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 13403ea3a4e42f77 | 2026-07-28T23:44:21Z | build+decorrelated                 | current |
| 6ad2a620d75bd04e | 2026-07-28T23:56:32Z | build+decorrelated                 | current |
| d37ce088bc4caf69 | 2026-07-29T00:55:46Z | build+replay+hardened+decorrelated | current |
| 872a60504989acef | 2026-07-29T07:57:01Z | build+replay+hardened+decorrelated | current |

5 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

- build (lake build (elaboration)) current
- replay (leanchecker (official kernel replay)) current
- hardened (safe\_verify (hardened screen)) current
- decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_soundpsd, HOL4 development script bern\_tr\_soundpsdScript.sml script bound

**Theorem** sound\_psdCertKernel

```

Theorem sound_psdCertKernel:
  ∀H sq m k A N B innerT Z ts conclusion.
  psdCheckK m k A N B innerT Z ∧
  psdConclCheck (m + k) A ts conclusion ∧
  (∃s. tmBshape H conclusion = SOME s) ∧
  tmFinWires H conclusion ∧
  (∀w. MEM w (termFreeWires conclusion) ⇒
    ∃τ. H.wireType w = SOME τ ∧ 0 < shapeTotalSize τ.shape) ∧
  syzygyConclusionFormB conclusion sq.post ⇒
  seqSemanticallyValid H sq

```

## HOL verifier receipts

**documentary** hol-strict CLEAN: theory bern\_tr\_soundpsd, 4 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms), 249-theory ancestry — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob c01e0b0c...). The shared semantic spine (sound\_psdCert0f) is displayed in the /Tri section per the family-preamble convention; this corollary discharges the quadratic-form fact via the ported kernel-extension bridge (psdCheckK\_quadForm, theory bern\_psd, where the witness check itself — psdCheckK, over the extension kernelExt — is ported with kernel-EVAL receipts). The payload record and its wellformedness arm ride the ported proof-object stratum (theory bern\_tr\_catdata2, 272 theorems, 0 offences, same run).

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — reduction to verified machinery. The certificate-family preamble is stated in the PSDCERTTRI section and applies verbatim. This is the *semidefinite* member: it certifies  $A \succcurlyeq 0$  together with an exactly-checked kernel, for the boundary cases the two strict tiers cannot touch.

**Recipe R30.1** (the PSDCERTKERNEL instance). Data:  $A \in \mathbb{Q}^{(m+k) \times (m+k)}$  with the  $k$  kernel coordinates *last* (the producer applies its permutation before emission — conjugation by a permutation is index renaming on matrix and term vector alike, so the identity-permutation form loses no generality);  $N \in \mathbb{Q}^{m \times k}$ ; the kernel-completion row matrix  $L := (I_m \mid N)$ ; a leading block  $B \in \mathbb{Q}^{m \times m}$  with a /T-or-/DD inner witness  $Z$ . Checked premises: the factorization  $A = L^T B L$  holds *entrywise exactly* (recomputed), and  $B$  passes the selected strict inner check. Then

$$\frac{\text{checkKernel}(A, N, B, Z) = \text{true} \quad c \equiv -\sum_{i,j} A_{ij} p_i p_j}{\Phi \vdash \Psi} \quad \text{PSDCERTKERNEL} \quad \rightsquigarrow \quad \text{discharged by sound\_psdCertKernel.}$$

The factorization check is — given the symmetry of  $B$ , which the selected inner check itself verifies — equivalent to the three separate conditions ( $A$  symmetric;  $A \cdot K = 0$  exactly for  $K = \begin{pmatrix} -N \\ I_k \end{pmatrix}$ ; leading block of  $A$  equal to  $B$ ): forward,  $B$ 's symmetry makes  $A = L^T B L$  symmetric and  $LK = 0$  kills  $AK$ ; backward,  $AK = 0$  forces the cross block  $BN$  and then the trailing block  $N^T B N$ . Either way the off-leading blocks are forced, so a forged  $N$  or  $B$  refuses.

**Proposition R30.2** (matrix warrant). If  $B \succ 0$  (hence  $\succcurlyeq 0$ ) and  $A = L^T B L$  for the rectangular  $L = (I \mid N)$ , then  $A \succcurlyeq 0$ . Moreover  $x^T A x = 0$  exactly on  $\text{span } K$ , so the certificate fixes  $\ker A = \text{span } K$  and  $\text{rank } A = m$ .

*Proof.* Positive semidefiniteness is rectangular congruence, with no invertibility anywhere:  $x^T A x = (Lx)^T B (Lx) \geq 0$  (the kernel proof is this one line, posSemidef\_of\_factor). For the kernel content, write  $x = (x_1, x_2)$  against the block split and note  $Lx = x_1 + Nx_2$ , while  $LK = -N + \bar{N} = 0$ ; with  $B \succ 0$ ,  $x^T A x = 0$  iff  $Lx = 0$  iff  $x = Kx_2$  — i.e. the zero set of the form is exactly  $\text{span } K$ . That zero set is the kernel: if  $Lx = 0$  then  $Ax = L^T B (Lx) = 0$ , and conversely  $Ax = 0$  forces  $x^T A x = 0$ , hence  $Lx = 0$ . The columns of  $K$  are independent (its lower block is  $I_k$ ), so  $\ker A = \text{span } K$  has dimension  $k$  and  $\text{rank } A = m$  by rank-nullity.  $\square$

**Remark R30.3** (exact kernels are forced, not a method limitation). A rounding-tolerant witness for kernel *directions* cannot exist for a residual-style check — one attesting the kernel claim only up to an  $\varepsilon$ -residual, with no exact kernel equation:  $\{A \succcurlyeq 0, \det A = 0\}$  is a boundary set with empty interior, so a check whose acceptance region contains an entrywise- $\varepsilon$  neighbourhood of a true instance also accepts some  $A - \varepsilon' I \not\succcurlyeq 0$ . The split is therefore forced, not preferred: exact kernel data plus a rounding-tolerant bulk (the /DD inner witness), with tolerance able to live in the bulk and nowhere else — and it matches the intended producers, whose kernels are *semantic* (a conservation law, a syzygy relation, the all-ones vector of a graph Laplacian):  $N$  comes from theorems, and floats only ever touch  $Z$ .

**Remark R30.4** (each hypothesis, and the corners). *Entrywise-exact factorization*: the whole kernel claim rides on  $AK = 0$  holding exactly; an  $\varepsilon$ -approximate factorization certifies nothing (previous remark). *The strict inner witness*:  $B \succ 0$  is what confines the zero set to  $\text{span } K$  — a merely semidefinite  $B$  would leave undeclared kernel directions inside the leading block. *Kernel coordinates last*: bookkeeping only, by the permutation argument in the recipe. Corners are structural, not special-cased:  $k = 0$  gives  $L = I$  and the rule degenerates to its inner strict tier ( $A = B$ );  $m = 0$  forces  $A = 0$  (empty  $B$ , no inner witness content — the factorization premise is the entire content).

*Remark* R30.5 (what reaches the formula layer). The checker verifies the kernel and rank content (a forged  $N$  refuses), but the formula language states only the quadratic-form conclusion  $\sum_{i,j} A_{ij} p_i p_j \geq 0$  — which for this rule is honest ( $\succ$ -shaped, per the preamble’s strictness discipline) and loses only the kernel/rank *statement*, not its checking. Making rank and kernel formula-expressible needs closed-tensor terms and a tensor-valued closed evaluator — the same named extension the PCONSTEXACT section records for exact matrix identities; short of that extension, the kernel content is certificate-level assurance surfaced here, deliberately not hand-faked into the formula grammar.

| proof step                           | formal anchor                                                                                                             |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| Recipe R30.1 warrant                 | sound_psdCertKernel (Soundness); shared transport sound_psdCertOf (see PSDCERTTRI)                                        |
| rectangular congruence (Prop. R30.2) | PsdCert.posSemidef_of_factor                                                                                              |
| check-to-theorem bridge              | PsdCert.checkKernel_posSemidef (inner tier via checkTri_posDef / checkDiag_posDef); corollary checkKernel_quadForm_nonneg |
| the executable check                 | PsdCert.checkKernel with the completion rows kernelExt                                                                    |
| conclusion identity + readings       | PsdCert.conclCheck; the SYZGVY section’s conclusion-form selector                                                         |



## Rule 31. ReLU-SPLIT (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *ReLU-Split*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \wedge (h \geq 0) \vdash \Psi[b := h] \quad \Phi \wedge (h \leq 0) \vdash \Psi[b := 0] \quad \text{relu}_{\{h\} \rightarrow \{y\}} \in H \quad (y \text{ typed at rank } 0)}{\Phi \vdash \Psi} \text{ReLU-SPLIT}$$

ReLU-SPLIT is the case-split rule of the ReLU family: from two branch proofs — the goal with the output position replaced by  $h$ , under  $\Phi$  strengthened by  $h \geq 0$ ; and the goal with the output position replaced by  $0$ , under  $\Phi$  strengthened by  $h \leq 0$  — conclude the goal outright, per rank-0 node: the tensor-shaped reading of the rule is *false*, not merely unrealized (the proof section's scope remark carries the mixed-sign countermodel), so a producer splits scalar by scalar. It is the *complete*-verification move of the neural-network literature (the Reluplex and branch-and-bound lineage): where the affine envelope of LiRPA loses too much, a producer splits on the neuron's sign and pays with proof size — each split doubles the branch count, which is exactly the branch-and-bound tradeoff.

Unlike every other rule in this book so far, ReLU-SPLIT has *no certificate node*: no payload constructor, no wire tag — a certificate stream never contains one, so there is nothing for this rule's checker column to recompute. A producer elaborates it away. The elaboration's target is the *derivation* the rule abbreviates (case analysis over the sign-split fact, equality transport in each branch, and structural plumbing), whose nodes the checker checks by their own rules — a certificate fragment that becomes emittable exactly when the context-carrying case split it opens with lands as a core constructor (today that form is a satellite soundness theorem only; the proof section states the dependency). What stands unconditionally is the rule-level semantic theorem of the proof section's mechanization remark.

Each ingredient of the two premises is load-bearing for the elaboration. The branch strengthenings ( $h \geq 0$ ,  $h \leq 0$ ) are what the case fact discharges into each branch context — remove one and that branch proof has no purchase on the sign information. The substitution alignment ( $\Psi$  with the output position filled by  $h$ , respectively  $0$ ) is what the equality transport rewrites through: inside the active branch the node's fact  $y = h$  carries  $\Psi[b := h]$  to  $\Psi$ , and inside the inactive branch  $y = 0$  carries  $\Psi[b := 0]$  to  $\Psi$ .

**notes** The checker column remains empty for the structural reason: no certificate stream ever contains a ReLU-SPLIT node, so there is nothing for the checker to recompute. The rule-level statement exists: `sound_reluSplit` takes the two branch validities to the conclusion's validity, at the rank-0 scope the recipe's own per-scalar route always meant — and the companion theorem `reluSplit_rank0_necessary` mechanizes the mixed-sign countermodel showing the tensor-shaped reading is false, upgrading the family's shape-scope remark from derivation-obstruction to semantic necessity for this rule. A verified elaboration function producing the derivation does not yet exist. The derivation requires the context-carrying form of disjunction elimination — the sign-split disjunction arrives as a sub-conjunct of the premise, which the plain sequent form of  $\vee E$  cannot reach; see the  $\vee E$  section's note.

### Lean statement

The rule-level semantic lemma, at the rank-0 scope: the template  $\Psi$  is an arbitrary formula with an arbitrary marker wire for  $b$ ; the three substitution instances are hypotheses via `substituteInFormula` (EqELIM's substitution predicate); the branch strengthenings are ReLU-Pos/ReLU-Neg's premise atoms verbatim; the linkage and its extraction are the family's (LiRPA section). The second theorem is the scope's necessity witness: a shape-[2] fixture with mixed-sign input on which both branch premises are valid and the conclusion fails.

BernsteinLogic.ReluSplit . **sound\_reluSplit**

statement bound

```
/-- ReLU-Split (rule-level semantic lemma; rank-0 scope, which
`reluSplit_rank0_necessary` shows is mandatory): from the validity of
the two branch sequents — the conclusion template at the preactivation
under ` $\Phi \wedge (h \geq 0)$ `, and at the constant 0 under ` $\Phi \wedge (h \leq 0)$ ` — the
conclusion sequent ` $\Phi \vdash \Psi$ ` is valid, where ` $\Psi$ ` is the template at the
cited ReLU application's output wire. -/
theorem sound_reluSplit {H : Hypernet}
  {seq activePremise inactivePremise : Sequent}
  {preact post marker : Wire} {bodyForm : Formula}
  (ihActive : activePremise.semanticallyValid H)
  (ihInactive : inactivePremise.semanticallyValid H)
  (hlink : reluLinked H preact post)
  (hrank0 :  $\exists \tau, H.wireType \text{ post} = \text{some } \tau \wedge \tau.shape = []$ )
```

```

(hActPre : activePremise.pre
  = .and seq.pre (.atom .ge (.var preact) (.const 0)))
(hInactPre : inactivePremise.pre
  = .and seq.pre (.atom .le (.var preact) (.const 0)))
(hActPost : substituteInFormula bodyForm marker (.var preact)
  activePremise.post)
(hInactPost : substituteInFormula bodyForm marker (.const 0)
  inactivePremise.post)
(hconcl : substituteInFormula bodyForm marker (.var post) seq.post) :
seq.semanticallyValid H

```

BernsteinLogic.ReluSplit . **reluSplit\_rank0\_necessary**

statement bound

```

/-- The rank-0 hypothesis of `sound_reluSplit` is NECESSARY: the same
rule schema with the shape scope deleted is refuted – at a shape-[2]
ReLU node with mixed-sign input, both branch premises are valid while
the conclusion fails. (The header records the shape-[1] countermodel,
via broadcast-arity mixing, in prose; this is the mechanized mixed-sign
witness.) -/
theorem reluSplit_rank0_necessary :
  - (∀ (H : Hypernet) (seq activePremise inactivePremise : Sequent)
    (preact post marker : Wire) (bodyForm : Formula),
    activePremise.semanticallyValid H →
    inactivePremise.semanticallyValid H →
    reluLinked H preact post →
    activePremise.pre
      = .and seq.pre (.atom .ge (.var preact) (.const 0)) →
    inactivePremise.pre
      = .and seq.pre (.atom .le (.var preact) (.const 0)) →
    substituteInFormula bodyForm marker (.var preact)
      activePremise.post →
    substituteInFormula bodyForm marker (.const 0)
      inactivePremise.post →
    substituteInFormula bodyForm marker (.var post) seq.post →
    seq.semanticallyValid H)

```

## Correspondence (Steele display ↔ Lean)

### NO CORRESPONDENCE DICTIONARY YET

no reviewer-certified dictionary maps the displayed rule onto the mechanized statements above; until one is written and certified, that mapping rests on this section's editorial care alone.

## Lean verifier receipts

module BernsteinLogic.ReluSplit; bound build 01611bc8a2c3... (bound from receipt 6dd1e30ab7bf0a19, 2026-07-20T05:56:22Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 6dd1e30ab7bf0a19 | 2026-07-20T05:56:22Z | build+decorrelated                 | current |
| f5e25a81aaa84445 | 2026-07-20T20:44:58Z | build+decorrelated                 | current |
| ca5158835179141d | 2026-07-20T21:01:26Z | build+replay+hardened+decorrelated | current |
| 1a900df2daf383f4 | 2026-07-21T05:28:56Z | build+replay+hardened+decorrelated | current |

gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator:** none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

ReLU-SPLIT has no certificate representation on either side of the seam: no payload constructor, no wire tag, no check function — a producer elaborates the rule into its derivation, so a certificate stream never contains a ReLU-SPLIT node and there is nothing for a HOL port to mirror. Any future HOL treatment would be a formalization of the rule-level statement or the derivation itself (the Lean side of the former exists — `sound_reluSplit`), not a port of checker code.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: admissible rule (tier 2) — but of the family’s third kind. PCONSTEXACT is admissible by reduction to verified machinery; its siblings here are admissible with shipped direct lemmas; ReLU-SPLIT is admissible in the purest sense: it has *no proof-term constructor and no wire tag* — a certificate stream never contains a ReLU-SPLIT node. The derivation below is the rule’s intended elaboration, and it becomes a shippable certificate fragment exactly when the context-carrying case split it opens with becomes a core constructor (Remark R31.2); the warrant that stands on its own is the rule-level theorem: `sound_reluSplit` states the rule itself, at the rank-0 scope Remark R31.4 proves is the rule’s true extent.

**Recipe R31.1** (the ReLU-SPLIT elaboration). The rule, as the source catalogue states it:

$$\frac{\mathcal{T}_+ \triangleright \Phi \wedge (h \geq 0) \vdash \Psi[b := h] \quad \mathcal{T}_- \triangleright \Phi \wedge (h \leq 0) \vdash \Psi[b := 0]}{\Phi \vdash \Psi} \text{ReLU-SPLIT}$$

where  $b$  marks the occurrences of the output wire  $y$  of a ReLU application  $\text{relu}_{\{h\} \rightarrow \{y\}} \in H$ , so that  $\Psi = \Psi[b := y]$ . Two side conditions ride under that “where”, made explicit by the realized statement exactly as the siblings’ displays make theirs: the linkage itself (the warrant consumes the family’s port lookup), and — semantic necessity, not caution — the output wire typed at rank 0 (Remark R31.4). The intended elaboration of one such node is the derivation:

1. the node’s sign-split fact —  $(h \leq 0 \wedge y = 0) \vee (0 \leq h \wedge y = h)$ , the packaged semantic contract stated in the LIRPA section’s family template — enters as a sub-conjunct of the premise (the sp-axiom convention: for an application in  $H$ , its packaged contract is available premise-side as an axiom, true at every admissible valuation — the same convention under which the cell rules’ one-hot disjunction enters);
2.  $\vee E$  in its context-carrying form splits on it (see Remark R31.2);
3. *active branch* (context gains  $0 \leq h$  and  $y = h$ ): the branch context is aligned with  $\mathcal{T}_+$ ’s premise  $\Phi \wedge (h \geq 0)$  and consulted, in five named moves —  $\wedge E$  extracts  $\Phi$  and the case conjunct  $0 \leq h$ ; the bound node’s polarity-flip reading re-orientes the latter to  $h \geq 0$  (PBOUND, reading (G), the one-node move displayed at the ReLU-NN section’s active branch; the ReLU-ABOVE section walks the toolkit’s longer chains);  $\wedge I$  rebuilds  $\Phi \wedge (h \geq 0)$ ; CUT against  $\mathcal{T}_+$  yields  $\Psi[b := h]$  in the branch context; and EQSYM on the context’s  $y = h$  followed by EQELIM along  $h = y$  (wire-to-wire, the node’s one shared shape) transports it to  $\Psi[b := y] = \Psi$ ;
4. *inactive branch* (context gains  $h \leq 0$  and  $y = 0$ ): the same route with the mirrored atoms, four grouped moves ( $h \leq 0$  needs no reorientation, so the PBOUND step drops out);  $\mathcal{T}_-$  yields  $\Psi[b := 0]$ , and EQSYM + EQELIM along  $0 = y$  transport it to  $\Psi$ ;
5. CUT and WEAKEN stitch the case analysis back under the original  $\Phi$ .

Step 4’s transport has a rank-0 constant side, so at tensor-shaped instances it is stopped by EQELIM’s broadcast-shape gate, exactly as the family’s shape-scope remark records (Remark R22.2, ReLU-NN section) — at such instances a producer splits per scalar node (ReLU-NEG’s direct node supplies the shape-general fact  $y = 0$ , but transporting it into an arbitrary  $\Psi$  rides the same gate, so the per-scalar split is the general route — and necessarily so: Remark R31.4 proves the tensor-shaped reading admits a countermodel).

**Remark R31.2** (why the context-carrying  $\vee E$ ). The disjunction being eliminated does not stand alone as the premise: it arrives *inside* the premise conjunction, next to everything else  $\Phi$  carries. The plain sequent form of  $\vee E$  ( $\Phi_1 \vdash \Psi$  and  $\Phi_2 \vdash \Psi$  give  $\Phi_1 \vee \Phi_2 \vdash \Psi$ ) cannot reach it — and this is not a stylistic gap: the  $\vee E$  section records the countermodel showing the context-carrying form is strictly stronger than the printed one (no elaboration of the former into the latter exists). This derivation is the load-bearing consumer of that form. The form is proven sound as a satellite lemma (`sound_orEctx`), and that is currently *all* it is: it has no core constructor and no wire tag, so the derivation above is checkable as a certificate fragment only once the constructor lands — until then the rule’s standing warrant is Remark R31.3’s theorem, and the elaboration is the design the constructor exists to serve.

**Remark R31.3** (mechanization status — the rule-level statement). The checker column remains empty for the structural reason it always was: no certificate ever contains the rule, so there is no payload to validate and no soundness case in the checker’s recursion. The rule-level statement exists: `sound_reluSplit` is the semantic lemma — from the validity of the two branch sequents and the

linkage, the conclusion sequent is valid — built from exactly the vocabulary the rest of the catalogue already uses (the linkage extraction `reluLinked_facts`, the substitution predicate `substituteInFormula` that EQELIM’s payload already uses, and the semantic transport `Formula.holds_subst_congr` that EQELIM’s soundness already rides). The template  $\Psi$  with its marked occurrences  $b$  is an arbitrary formula over an arbitrary marker wire, with the three substitution instances ( $\Psi[b := h]$ ,  $\Psi[b := 0]$ ,  $\Psi = \Psi[b := y]$ ) as hypotheses — no freshness condition, since the proof transports along value equality, never syntax. “Derivable, by inspection of this recipe” is thereby backed by a theorem at every instance the recipe itself reaches (rank-0 nodes — [Remark R31.4](#)); a verified elaboration function producing Recipe [R31.1](#)’s derivation does not yet exist.

*Remark R31.4* (the rank-0 scope is semantic necessity). `sound_reluSplit` hypothesizes the output wire typed at rank 0, and this is the rule’s true extent rather than proof convenience. The family’s shape-scope remark ([Remark R22.2](#)) records that the *derivation*’s constant-sided transport is stopped by EQELIM’s broadcast-shape gate at tensor shapes; for the siblings the direct lemma is nonetheless shape-general, so the gate reads there as a technicality of the derivation route. Not here: `reluSplit_rank0_necessary` (same module) mechanizes a countermodel showing the shape-general reading of `ReLU-SPLIT` is *false*. Template  $\Psi(b) = (b = 0) \vee (b = h)$  over a shape-[2] node,  $\Phi = \top$ : both premises are valid ( $\mathcal{T}_+$  through  $h = h$ ,  $\mathcal{T}_-$  through  $0 = 0$ ), yet at the mixed-sign valuation  $h = [1, -1]$ ,  $y = [1, 0]$  neither disjunct of the conclusion holds. Mixed signs are exactly what the rule’s case analysis cannot see from sign-definite branch hypotheses — the semantic content of “each split doubles the branch count”: a tensor node is  $2^n$  scalar cases, not 2. Even shape [1] fails, by a different leak (broadcast-arity mixing): take  $\Psi(b) = (b = h) \vee (b + w \geq 0)$  with  $w$  typed at [2] and premise  $\Phi = (w \geq 0)$ . Both branch sequents are valid —  $\mathcal{T}_+$  through  $h = h$ ,  $\mathcal{T}_-$  through  $0 + w \geq 0$ , the rank-0 constant broadcasting to  $w$ ’s shape — yet at  $h = [-1]$ ,  $y = [0]$ ,  $w = [1, 1]$  the conclusion fails:  $y = h$  fails entrywise, and the [1]-shaped  $y + w$  atom is unsatisfiable outright by the strict-shape arithmetic, which broadcasts rank-0 sides only (this leak is stated as prose beside the mechanized [2] witness, not itself mechanized). The recipe’s step-4 note (“the per-scalar split is the general route”) is thus not a workaround for a gate but the only sound reading; producers split per rank-0 node, and `sound_reluSplit` is the rule those splits instantiate.

*Remark R31.5* (when a producer splits). Splitting is the *complete*-verification move of the neural-network literature — Reluplex’s case analysis [134], systematized as branch and bound [33]: exact where LiRPA’s envelope is lossy, at the price of one branch doubling per split neuron, so proof size is exponential in the split count. The practical frontier — which neurons to split, which to relax — is the branch-and-bound literature’s whole subject; the calculus prices the choice honestly (a split is two subtrees, an envelope is one node) and takes no position on it.

| proof step                                | formal anchor                                                                                                       |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| the sign-split fact (step 1)              | <code>PrimApp.SemOk</code> — the packaged contract; conservativity via <code>relu_projection</code> (LiRPA section) |
| context-carrying case split (step 2)      | <code>sound_orCtx</code> (satellite lemma, <code>Admissible.lean</code> )                                           |
| equality transport (steps 3–4)            | <code>sound_eqElim</code> ; <code>substituteInFormula</code>                                                        |
| rule-level statement (rank-0 scope)       | <code>sound_reluSplit</code> ( <code>ReluSplit.lean</code> ) — <a href="#">Remark R31.3</a>                         |
| scope necessity (mixed-sign countermodel) | <code>reluSplit_rank0_necessary</code> ; fixture <code>ReluSplitScope.HSplit</code> — <a href="#">Remark R31.4</a>  |
| verified elaboration function             | <i>none exists</i> — the remaining gap, with the core constructor for the context-carrying case split               |

## 15.4 Assume–Guarantee Composition

### Rule 32. PROMISE (admissible rule)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *Promise*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi_1 \vdash Q \quad \Phi_2 \wedge Q \vdash \Psi \quad \text{wires}(Q) \subseteq Y \quad \text{wires}(\Phi_1) \cap (\text{wires}(\Phi_2) \cup \text{wires}(\Psi)) \subseteq Y}{\Phi_1 \wedge \Phi_2 \vdash \Psi} \text{PROMISE}$$

$Y$  = the declared interface (payload wire list; bundle entries addressed as  $\text{TIdx}(w, i)$ ). The two confinement checks are the locality warrant, not soundness needs: the contract  $Q$  is the *datasheet* — the public artefact whose vocabulary is the separator algebra — and the cost identities (LP size, ZK disclosure set) quantify over certificates carrying these checks.

a Bernstein Logic rule outright: the baked base theory lists it in its core roster (grouped there under an assume–guarantee comment), the certified checker decodes its wire node, and its certificate relations ride the pin table — there is no intermediate composition logic between this rule and the base calculus. The tier word above classifies proof theory (the unpack is **WEAKEN** +  $\wedge$ -restructuring + **CUT**); core-roster membership is wire placement.

**PROMISE** is the calculus's acyclic composition rule: it discharges an assumption one component makes about another. The supplier proves the contract  $Q$  from its own premises ( $\Phi_1 \vdash Q$ ); the consumer proves the goal *using*  $Q$  ( $\Phi_2 \wedge Q \vdash \Psi$ ); the composite concludes  $\Phi_1 \wedge \Phi_2 \vdash \Psi$  with the contract discharged internally — cut, wearing interface bookkeeping. A producer reaches for it wherever a system decomposes along a directed interface: certify the downstream component against a *datasheet* (the contract, written over the declared interface wires only), certify the upstream component as meeting the datasheet, and compose without ever re-deriving either side inside the other.

What the checker recomputes: the splice plumbing and the locality warrant. The plumbing — the contract sits at a payload-declared conjunct path in the consumer's premise; the composite premise is the consumer's with the contract replaced by the supplier's premise at that path — is verified by exact read/replace on the conjunction tree. The locality warrant is two wire-confinement checks: the contract's free wires lie inside the declared interface, and the supplier's wires meet the consumer's and the goal's only inside it.

The hypotheses split by role, and the split is the rule's point. Soundness consumes only the plumbing: drop the path checks and the “composite” premise could be unrelated to the two sub-proofs. The confinement checks are *warrant, not soundness* — cut is sound regardless — but dropping them destroys the datasheet reading: a contract mentioning supplier-private wires is no longer a public interface artefact, and every quantitative consequence hung on the interface (certificate size scaling with the separator rather than the components, bounded disclosure of component internals) quantifies over certificates that carry these checks. The proof section separates the two roles explicitly.

**notes** Tier reading: proof-theoretically the rule is admissible — one node unpacks into **CUT** against premise-transported copies of the cited sub-derivations, a global rewrite of those sub-proofs rather than a local chain (the proof section's recipe walks why: **WEAKEN** and **CUT** are head-positional) — while in the realized calculus it is a native constructor, kept for cost and for the locality-warrant payload. The tier column records the proof-theoretic status; constructor membership is wire placement.

#### Lean statement

BernsteinLogic.Soundness . **PromiseStep**

statement bound

```
/-- Applicability data of the Promise rule, ctx-style – field-for-field
the reference checker's obligations, and field-for-field `PPromiseData`
minus the two sub-proofs (which enter `sound_promise` as validity
hypotheses). The two confinement fields are checked side-conditions that
soundness does NOT consume (AG draft §1.2: cut is sound regardless) –
they are the locality warrant. -/
structure PromiseStep where
  seq : Sequent
  supplier : Sequent
  consumer : Sequent
  contract : Formula
```

```

path : ConjPath
iface : List Wire
ctx : Formula
supplierPost : supplier.post = contract
pathReads : consumer.pre.conjGet? path = some contract
prePlumb : consumer.pre.conjReplace? path supplier.pre = some seq.pre
postEq : seq.post = consumer.post
ctxReads : consumer.pre.conjReplace? path .tt = some ctx
contractConfined : formulaWiresWithin contract iface
separation : wiresSeparated supplier.pre ctx seq.post iface

```

BernsteinLogic.Soundness . **sound\_promise**

statement bound

```

/-- Soundness of Promise: valid supplier + valid consumer give a valid
composite. Read the supplier's premise at the path, obtain the contract
from supplier validity, replace it back to recover the consumer's
premise, conclude by consumer validity. Pure predicate logic over the
semantics; the confinement fields are untouched. -/
theorem sound_promise {H : Hypernet} (st : PromiseStep)
  (ihs : st.supplier.semanticallyValid H)
  (ihc : st.consumer.semanticallyValid H) :
  st.seq.semanticallyValid H

```

BernsteinLogic.Soundness . **Formula.holds\_of\_conjGet**

statement bound

```

/-- The conjunct addressed by a path is entailed: if  $\Gamma$  holds,  $\Gamma|\pi$  holds.
(Promoted from Admissible.lean – the vE-ctx kit; the Promise contract
path reuses it.) -/
theorem Formula.holds_of_conjGet {H : Hypernet} { $\sigma$  : H.Valuation} :
   $\forall \{\pi : \text{ConjPath}\} \{\Gamma \ d : \text{Formula}\},$ 
   $\Gamma.\text{conjGet? } \pi = \text{some } d \rightarrow \Gamma.\text{holds } \sigma \rightarrow d.\text{holds } \sigma$ 

```

BernsteinLogic.Soundness . **Formula.holds\_conjReplace**

statement bound

```

/-- Replacing the addressed conjunct by a formula that holds preserves
holding: if  $\Gamma$  holds and  $g$  holds,  $\Gamma[\pi := g]$  holds. (Promoted from
Admissible.lean.) -/
theorem Formula.holds_conjReplace {H : Hypernet} { $\sigma$  : H.Valuation} :
   $\forall \{\pi : \text{ConjPath}\} \{\Gamma \ \Gamma' \ g : \text{Formula}\},$ 
   $\Gamma.\text{conjReplace? } \pi \ g = \text{some } \Gamma' \rightarrow \Gamma.\text{holds } \sigma \rightarrow g.\text{holds } \sigma \rightarrow \Gamma'.\text{holds } \sigma$ 

```

BernsteinLogic.Soundness . **Formula.conjGet\_of\_conjReplace**

statement bound

```

/-- Reading at the path of a successful replacement returns the
replacement. (Promoted from Composition.lean §G2.) -/
theorem Formula.conjGet_of_conjReplace :
   $\forall \{\pi : \text{ConjPath}\} \{\Gamma \ \Gamma' \ g : \text{Formula}\},$ 
   $\Gamma.\text{conjReplace? } \pi \ g = \text{some } \Gamma' \rightarrow \Gamma'.\text{conjGet? } \pi = \text{some } g$ 

```

BernsteinLogic.Soundness . **Formula.conjReplace\_back**

statement bound

```

/-- Replacing back the conjunct a read found recovers the original: the
round trip  $\Gamma \rightarrow [\pi := g] \Gamma' \rightarrow [\pi := \Gamma|\pi] \Gamma$ . (Promoted from
Composition.lean §G2.) -/
theorem Formula.conjReplace_back :
   $\forall \{\pi : \text{ConjPath}\} \{\Gamma \ \Gamma' \ q \ g : \text{Formula}\},$ 
   $\Gamma.\text{conjGet? } \pi = \text{some } q \rightarrow \Gamma.\text{conjReplace? } \pi \ g = \text{some } \Gamma' \rightarrow$ 
   $\Gamma'.\text{conjReplace? } \pi \ q = \text{some } \Gamma$ 

```

**The confinement side conditions, as checked.** The proof's premise-transport induction leans on the wire-discipline predicates the checker recomputes; their definitions are displayed so the side conditions' meaning is on the page.

BernsteinLogic.Core.Catalogue . **premiseContains**

definition bound

```

/-- Conjunct-wise premise containment (geolog `premiseContains`): every
conjunct of `sub` is a conjunct of `pre`. Decouples component premises

```



```

from the composite premise – the core has no  $\wedge$ -restructuring rules, so
without this a component proved over its own box could not enter a
composite whose premise merely CONTAINS that box. -/
def premiseContains (pre sub : Formula) : Prop :=
   $\forall a \in \text{sub.conjunctList}, a \in \text{pre.conjunctList}$ 

```

BernsteinLogic.Core.Catalogue . formulaWiresWithin

definition bound

```

/-- Wire confinement (geolog `formulaWiresWithin`): every free wire of
the formula lies in the declared interface. Prover-side warrant that
demanding this costs no completeness: Lemma A, `farkas_interpolant_vars`
(contracts are mined confined). -/
def formulaWiresWithin (f : Formula) (iface : List Wire) : Prop :=
   $\forall w \in f.\text{freeWires}, w \in \text{iface}$ 

```

BernsteinLogic.Core.Catalogue . wiresSeparated

definition bound

```

/-- Separation (geolog `wiresSeparated`, ctx-form): every wire the
supplier's premise shares with the consumer's context or the conclusion
routes through the interface. The context formula is the consumer's
premise with the contract slot blanked (`conjReplace?` at `.tt`). -/
def wiresSeparated (left ctx post : Formula) (iface : List Wire) : Prop :=
   $\forall w \in \text{left.freeWires},$ 
   $(w \in \text{ctx.freeWires} \vee w \in \text{post.freeWires}) \rightarrow w \in \text{iface}$ 

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax; the statement bundles its data as PromiseStep (field-for-field the payload minus the two sub-proofs, which enter as ihs/ihc).

- $\Phi_1 \vdash Q \mapsto \text{ihs}$  with  $\text{supplierPost} : \text{supplier.post} = \text{contract}; Q \mapsto \text{contract} : \text{Formula}; \Phi_1 \mapsto \text{supplier.pre}$ .
- $\Phi_2 \wedge Q \vdash \Psi \mapsto \text{ihc}$ , with the certified statement taking the path-general form:  $Q$  sits at a *payload-declared conjunct path* ( $\text{pathReads} : \text{consumer.pre.conjGet? path} = \text{some contract}$ ), not at a fixed position — the  $\text{ctx-}\forall E$  style.  $\Phi_2 \mapsto \text{ctx}$ , the consumer premise with the contract deleted at the path ( $\text{ctxReads}$ ).
- conclusion  $\Phi_1 \wedge \Phi_2 \vdash \Psi \mapsto$  the goal for  $\text{st.seq}$ , whose premise is pinned by  $\text{prePlumb}$ : the consumer's premise with the supplier's premise spliced in *at the same path* where  $Q$  sat. Order honesty: at the displayed root-right instance ( $Q$  the right conjunct of  $\Phi_2 \wedge Q$ ), the splice therefore pins  $\text{seq.pre}$  to  $\Phi_2 \wedge \Phi_1$  — the displayed  $\Phi_1 \wedge \Phi_2$  is obtained by semantic conjunction commutativity, not by direct instantiation. Path generality remains strictly wider than the display, in the safe direction; the conjunct *order* is the one representational difference, named here.  $\text{postEq}$  carries  $\Psi$  across unchanged.
- the two confinement premises ( $\text{wires}(Q) \subseteq Y$ ; the interface-separation check)  $\mapsto$  the Step fields  $\text{contractConfined}$  and  $\text{separation over iface} \mapsto Y$  — carried, checked, and consumed by *no* step of  $\text{sound\_promise}$ : the displayed caption's claim that they are the locality warrant rather than soundness needs is visible in the statement itself: the proof never touches them, as on the QTSL pages.

Width note: beyond the deliberate path-generality and the conclusion-order transposition named above (the splice yields  $\Phi_2 \wedge \Phi_1$ ; the display's order needs conjunction commutativity, semantically free), neither side is wider; the soundness content is pure predicate-logic cut over the semantics, exactly as the witness column says.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt | minted | layers | status |
|---------|--------|--------|--------|
|---------|--------|--------|--------|



a9c0e34d3e3c9c8c 2026-08-03T21:30:36Z build+decorrelated current

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status               |
|------------------|----------------------|--------------------|----------------------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | <span>current</span> |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | <span>current</span> |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_fam\_resid, HOL4 development script bern\_fam\_residScript.sml script bound

#### Definition checkFam\_promise\_def

```

Definition checkFam_promise_def:
  checkFam_promise (cols:columns) (ct:ctx) (i:num) =>
    case frSeq cols ct i of
    | NONE => F
    | SOME sq =>
      case frChildSeq cols ct i 0 of
      | NONE => F
      | SOME ss =>
        case frChildSeq cols ct i 1 of
        | NONE => F
        | SOME cs =>
          case frFormula cols ct i 0 of
          | NONE => F
          | SOME ctr =>
            case path0f cols ct i of
            | NONE => F
            | SOME (stps, ifc) =>
              let pth = stepsToPath stps in
              ss.post = ctr ∧
              conjGet cs.pre pth = SOME ctr ∧
              conjReplace cs.pre pth ss.pre = SOME sq.pre ∧
              sq.post = cs.post ∧
              (case conjReplace cs.pre pth FTt of
              | SOME cx0 =>
                (EVERY (λw. MEM w ifc) (formulaFreeWires ctr) ∧
                 EVERY (λw. (MEM w (formulaFreeWires cx0) v
                           MEM w (formulaFreeWires sq.post)) =>
                           MEM w ifc)
                 (formulaFreeWires ss.pre))
              | NONE => F)
          | NONE => F)

```

End

#### Theorem famEq\_promise

```
Theorem famEq_promise:
  wfColumns cols ct ∧ mkCtx cols = SOME ct ∧
  parseCert (cols.hdrImage ++ rest) = SOME (ws, pf) ∧
  nodeCorr cols ct i (CertPf sq nd) ∧
  FST (THE (tagOf cols i)) = 25 →
  (checkFam_promise cols ct i ⇔ armPromise ws sq nd)
```

## HOL verifier receipts

**documentary** Documentary citations. Family validation: `bern_fam_resid` built GREEN with all ten `famEq` arms validated, `hol-strict` clean; spine census: `bern_joins` GREEN under `hol-strict` (55 theorems, 0 offences, 208-theory ancestry including this theory), `checkJoins_eq_checkPf` verified verbatim in the built signature. The cited `famEq` lemma is the kernel-checked per-node equivalence: under well-formed columns (`wfColumns/mkCtx/nodeCorr`) at this rule's wire tag, the columnar family check coincides with the flat reference walk's arm for this rule. There is no signed HOL receipt store yet — these citations are documentary. Semantic stratum, also ported: the soundness twin `sound_promise` (theory `bern_tr_b14: hol-strict` clean, 2 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources), consumed by the certified-fragment keystone (`checkPf_sound_frag`, theory `bern_tr_splice`, 20 theorems, 0 offences, same run): a reference-checker accept inside the proven wire fragment yields sequent validity.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. Proof-theoretically `PROMISE` unpacks into core rules (conjunct plumbing + `CUT`); in the realized calculus it is a native constructor, kept for cost and for the locality-warrant payload. The recipe shows the unpack; the semantic warrant is the kernel lemma `sound_promise`, which is the unpack's semantic shadow run in one pass.

**Setting.** The payload (the `PromiseStep` structure quoted above) carries a supplier sequent  $\Phi_1 \vdash Q$ , a consumer sequent whose premise contains the contract  $Q$  at a declared conjunct path  $\pi$  (write it  $\Phi_2 \wedge_\pi Q \vdash \Psi$  — the  $\pi$ -notation meaning  $Q$  sits at path  $\pi$  in the premise's conjunction tree,  $\Phi_2$  being the rest), a declared interface wire list  $Y$ , and the composite sequent, whose premise is the consumer's with  $Q$  replaced by  $\Phi_1$  at  $\pi$  and whose conclusion is  $\Psi$ . Four checked hypotheses: (P1) the path reads  $Q$  in the consumer's premise; (P2) replacing  $Q$  by  $\Phi_1$  at  $\pi$  yields exactly the composite premise; (P3)  $\text{wires}(Q) \subseteq Y$ ; (P4) the supplier's wires meet the consumer's and the goal's only inside  $Y$ . (P1)–(P2) are the splice plumbing; (P3)–(P4) are the locality warrant.

**Recipe R32.1** (the `PROMISE` instance — cut through a conjunct path). The general admissible template (one rule node  $\rightsquigarrow$  a derivation fragment over core rules with the same conclusion, applied leaf-first until no such node remains) instantiates as:

$$\frac{\mathcal{T}_s \triangleright \Phi_1 \vdash Q \quad \mathcal{T}_c \triangleright \Phi_2 \wedge_\pi Q \vdash \Psi}{\Phi_2 \wedge_\pi \Phi_1 \vdash \Psi} \text{PROMISE} \rightsquigarrow \frac{\frac{\mathcal{T}_s}{\Phi_2 \wedge_\pi \Phi_1 \vdash Q} \quad \frac{\mathcal{T}_c}{\Phi_2 \wedge_\pi Q \vdash \Psi}}{\Phi_2 \wedge_\pi \Phi_1 \vdash \Psi} \text{CUT}$$

where, writing  $C$  for the composite premise  $\Phi_2 \wedge_\pi \Phi_1$ : the left branch derives  $C \vdash Q$  by `CUT` from  $C \vdash \Phi_1$  (an `Ax` leaf  $C \vdash C$ , then a conclusion-side  $\wedge E$  chain along  $\pi$ ) against a *premise-transported* copy of  $\mathcal{T}_s$ , and `CUT`'s right premise proper is  $C \wedge Q \vdash \Psi$ , a *premise-transported* copy of  $\mathcal{T}_c$  — the display elides both transports. Premise transport (re-rooting a derivation  $X \vdash \Psi$  at a premise  $Y$  whose addressed conjuncts entail  $X$ ) is *not* a local chain in this calculus: `WEAKEN` and `CUT` are deliberately head-positional — each grafts one conjunct at the premise root's right — so the premises reachable from an opaque sub-proof  $X \vdash \Psi$  by insertion alone are exactly  $X$  with right-appended conjuncts (`CUT` can also *consume* an already-present root-right conjunct when a proof of it is supplied — removal, never interior insertion or permutation), and even  $Y \wedge X \vdash \Psi$  is locally out of reach. The transport is instead an induction over the sub-derivation itself; its principal cases: rules that copy their premise tree verbatim from their sub-proofs re-instantiate on the transported children; premise-free leaves conclude at an arbitrary premise by the `NORM` convention (an `Ax` leaf  $\Phi \vdash \Phi$  re-derives at  $Y$  as  $Y \vdash Y$  plus a conclusion-side  $\wedge E/\wedge I$  rebuild); and the two head-positional rules are their own inductive cases, because re-rooting commutes with the head graft: a transported `WEAKEN` grafts its conjunct onto the new root, and a transported `CUT`'s right premise re-forms as the new root with the cut formula appended. This list is not exhaustive over the catalogue: the *premise-forming* rules ( $\vee E$ 's conclusion premise is the disjunction of its branch premises; `SUBDIVIDE`'s branch premises conjoin the region data) and the context-carrying rules (`PROMISE`, `SZYGY`, whose path plumbing must compose with the splice — the  $\vee E$  section's machinery) are neither verbatim copies nor leaves; each needs its own commutation case, and those cases belong to the same transport metatheorem flagged as unmechanized below. Rewriting the premise field node by node under that case analysis yields the transported copies of  $\mathcal{T}_s$  and  $\mathcal{T}_c$ . Elimination costs the sizes of  $\mathcal{T}_s$  and  $\mathcal{T}_c$  plus the  $\wedge E$  chain along  $\pi$  at the auxiliary derivation plus the per-leaf `Ax` rebuilds — and each transported leaf pays its own conclusion-side rebuild chain (the proof-object grammar has no sharing), so the total is bounded by the certificate sizes plus the number of transported leaves times the addressing depth, linear in the certificate data only when that depth is bounded — and consumes `Ax` leaves throughout; conservativity holds with no new kernel semantic content, but as a global rewrite warranted by the transport induction (stated here as prose — the transport metatheorem is not yet mechanized), not as insertion of a constant-size fragment.

**Proposition R32.2** (semantic warrant). *If the supplier and consumer sequents are semantically valid and (P1)–(P2) hold, the composite sequent is semantically valid.*

*Proof.* Fix an admissible valuation  $\sigma$  satisfying the composite premise. Reading the composite premise at  $\pi$  returns  $\Phi_1$ : (P1) says  $\pi$  addresses  $Q$  in the consumer’s premise, so the replacement (P2) performs succeeded at  $\pi$ , and the read of a successful replacement returns the replacement (the round-trip lemma `Formula.conjGet_of_conjReplace`); an addressed conjunct of a holding conjunction holds (`Formula.holds_of_conjGet`); so  $\sigma \models \Phi_1$ , and supplier validity gives  $\sigma \models Q$ . Replacing  $\Phi_1$  back by  $Q$  at  $\pi$  recovers the consumer’s premise exactly (`Formula.conjReplace_back`, using (P1)–(P2)), and replacing the addressed conjunct of a holding conjunction by a formula that itself holds at  $\sigma$  preserves holding (`Formula.holds_conjReplace`, applied at  $\pi$  with  $Q$  for  $\Phi_1$ ); so  $\sigma \models \Phi_2 \wedge_\pi Q$ , and consumer validity gives  $\sigma \models \Psi$ . This is `sound_promise` verbatim: pure predicate logic over the semantics — admissibility of  $\sigma$  is passed through untouched, and neither confinement hypothesis is consulted.  $\square$

*Remark R32.3* (the confinement premises are warrant, not soundness). (P3)–(P4) appear in no step of Proposition R32.2, and dropping them alone — (P1)–(P2) retained — admits no false sequent: cut is sound regardless of where the contract’s wires live. What their removal *does* admit is a degenerate reading of the rule — a “contract” mentioning supplier–private wires, i.e. a composition whose interface is as wide as the components themselves. Everything the rule is *for* then fails: the contract stops being a datasheet (a public artefact whose vocabulary is the declared separator), certificate size stops scaling with the interface, and any bounded-disclosure reading of “what the consumer learns about the supplier” becomes vacuous. Those consequences are theorems proved elsewhere, *about certificates that carry the checks* — they quantify over exactly the certificates these hypotheses admit — which is why the checks are checked — a hypothesis whose necessity is economic and epistemic rather than semantic is exactly the kind a recipe must label, or the next reader deletes it as redundant.

*Remark R32.4* (relation to the literature). Assume–guarantee composition descends from composing-specifications calculi [2]; the subtlety those calculi negotiate is *circularity* — each component assuming what the other guarantees — which needs an induction over time or a fixpoint argument [177]. PROMISE deliberately takes the acyclic case, where plain cut suffices and the whole rule is bookkeeping around it; the circular case enters this calculus through the feedback rules — `SMALLGAIN` and `SMALLGAINN` at the sequent layer and the feedback cells of the satisfaction-layer composition rule, where the guarded-fixpoint structure is explicit, and `QWIRECOMPOSEMARKS` for liveness marks, where cyclic wiring needs no well-foundedness device at all and the wiring condition is a feasibility walk (schedules and pinned values must close around cycles) rather than a causality walk. Separating the cases — cut where cut suffices, guarded fixpoints where feedback threatens a safety face, a schedule walk where a cycle only has to deliver a mark — is the design’s answer to the classical observation that *unguarded* circular assume–guarantee reasoning about safety is unsound — there, some well-foundedness device is required.

| proof step                                                                                                                                                                               | formal anchor                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Recipe R32.1 payload & plumbing<br>path read/replace round trips<br>addressed conjunct entailed; replacement preserves holding<br>Proposition R32.2<br>confinement checks (warrant only) | <code>PromiseStep</code> (fields <code>path</code> , <code>pathReads</code> , <code>prePlumb</code> , <code>postEq</code> )<br><code>Formula.conjGet_of_conjReplace</code> , <code>Formula.conjReplace_back</code><br><code>Formula.holds_of_conjGet</code> , <code>Formula.holds_conjReplace</code><br><code>sound_promise</code> ; wired as core <code>RuleData.promise</code> into <code>ProofObject.sound</code><br><code>formulaWiresWithin</code> , <code>wiresSeparated</code> (payload fields <code>contractConfined</code> , <code>separation</code> ) |

## Rule 33. SYZYGY (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — `Syzygy`. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{c} \{\Phi_i \vdash e_i \leq 0\}_{i=1}^n \quad \{g_k = 0\}_k \text{ equality sub-conjuncts of } \Phi \text{ in wire variables} \quad \lambda_i \in \mathbb{Q}_{\geq 0} \quad \nu_k \in \mathbb{Q} \\ c \equiv \sum_i \lambda_i e_i + \sum_k \nu_k g_k \text{ (exact canonical polynomial)} \quad \Psi \in \{c \leq 0, \ e \geq 0 \text{ where } c = -e, \ l \leq r \text{ where } c = l - r\} \end{array}}{\Phi \vdash \Psi} \text{ SYZYGY}$$

$\Phi \supseteq$  each  $\Phi_i$  conjunct-wise (checked); the  $\nu_k$  are free-sign exactly as FARKAS frees  $\lambda_j$  on = atoms.

free-sign  $\nu_k$  is Lemma A's CommRing sign-blindness, one theorem for both dialects. Lifts FARKAS in exactly two ways: atoms from sub-*proofs* (not sub-conjuncts), atoms of arbitrary polynomial degree. The conclusion is exposed in any of the three `syzygyConclusionForm` readings — one exact  $\mathbb{Q}$  identity under a sign flip; the transport is the ge-bridge ring lemma, applied once inside `sound_syzygy`. A Bernstein Logic rule outright: the baked base theory lists it core (its wire node decoded by the certified checker, its combination relation pinned); “assume-guarantee” names its purpose, not a separate layer.

SYZYGY is the calculus's composition rule: it combines already-proven component bounds into a bound on a weighted sum. Each component sequent  $\Phi_i \vdash e_i \leq 0$  is a sub-proof — typically “this subsystem dissipates its storage function” — and the composite premise  $\Phi$  carries, among its conjuncts, *equality atoms*  $g_k = 0$  describing how the subsystems are wired together (power-preserving interconnection: what one port supplies, another consumes). A producer reaches for the rule wherever a system-level argument says “add the component certificates with weights  $\lambda_i$  and let the internal supplies cancel”: the weights are found by whatever method the producer likes (an LP, typically), and only the resulting exact identity ships in the certificate.

What the checker recomputes, given the payload (combination term, component list, conclusion term): that every entry weight is  $\geq 0$  — premise *equality atoms* expose both polarities to the matcher, so a free-sign multiplier on an equality is realized as a nonnegative weight on the appropriate polarity, and that freedom is what makes them syzygies; that every entry of the combination really is a component conclusion or a premise equality sub-conjunct, matched by exact canonical polynomial; and that the weighted sum of those polynomials is *identically* the conclusion polynomial as canonical forms over  $\mathbb{Q}$  — coefficient-for-coefficient equality, not evaluation at test points. It further checks the bookkeeping hypotheses: each component's premise is contained conjunct-wise in  $\Phi$  (`preContains`), the conclusion term has a determinate finite shape (`conclShape`, `conclFin`), and every wire in sight has positive size (`wiresPos`, `compPostWiresPos`).

Each hypothesis is load-bearing. Drop the sign condition on the inequality weights and a producer could multiply a bound by  $-1$  and conclude its negation. Drop the entry-sourcing check and the “combination” could smuggle in atoms nobody proved. Drop the exact-identity check and the conclusion would be unrelated to the premises. Drop premise containment and a component proved over its own operating box could be applied outside it. Drop the positive-size conditions and a component conclusion over a zero-size wire — vacuously true, since it quantifies over no tensor entries — could enter the combination with full algebraic weight and push the composite to a falsehood. The proof section walks these in order.

**notes** Port-Hamiltonian reading: with  $e_i = \mathcal{L}_i V_i - s_i^{\text{ext}} - s_i^{\text{int}}$ , the equality atoms  $g_k = 0$  cancel the internal supplies and the conclusion is a dissipation inequality for  $\sum_i \lambda_i V_i$  — Tellegen's theorem in certificate form. The free-body diagram is the junction tree; internal forces at the cuts are the separator algebra.

### Lean statement

BernsteinLogic.Soundness . SyzygyStep

statement bound

```
-- Applicability data of the Syzygy rule over a hypernet — field-for-field
the reference checker's obligations, and field-for-field `PSyzygyData`
minus the component-proof list (validity hypotheses of `sound_syzygy`)
and `denBoundLog2` (the Q7 family; zero proof-theoretic weight in v1).
B13-shaped conclusion fields and B14 (`wiresPos`) are inherited verbatim
from `PFarkasData`; `compPostWiresPos` is the honest EXTENSION of B14 to
component-conclusion wires (Composition.lean header: without it a
vacuously-held component conclusion over a zero-size wire poisons the
combination — counterexample kernel-checkable). -/
structure SyzygyStep (H : Hypernet) where
  seq : Sequent
  components : List Sequent
  combination : Term
```

```

conclusion : Term
preContains : ∀ c ∈ components, premiseContains seq.pre c.pre
conclShape : ∃ s, conclusion.bshape H = some s
conclFin : conclusion.finWires H
wiresPos : ∀ w ∈ seq.pre.freeWires ++ conclusion.freeWires,
  ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape
compPostWiresPos : ∀ c ∈ components, ∀ w ∈ c.post.freeWires,
  ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape
valid : syzygyCombinationValid combination components seq.pre conclusion
postForm : syzygyConclusionForm conclusion seq.post

```

BernsteinLogic.Soundness . **holds\_of\_syzygyConclusionForm**

statement bound

```

/-- The ge-bridge ring lemma (the both-polarity core change): every
reading admitted by
`szyzygyConclusionForm conclusion` holds whenever the zero-normal atom
`conclusion ≤ 0` holds – the readings are the same exact  $\mathbb{Q}$  identity
re-presented under a sign flip. The zero-form is the identity transport,
the ge-polarity flip is ` $-e ≤ 0 \rightarrow e ≥ 0$ `, and the literal atom is
`l - r ≤ 0 → l ≤ r` (ArithAtom's tensor-general sub-zero bridge).
Applied once inside `sound_syzygy`, so PSyzygy (tier-1), PBound's
compact spelling (tier-3), and any future  $≤ 0$ -shaped fact rule inherit
all readings with no separate polarity justification. -/
theorem holds_of_syzygyConclusionForm {H : Hypernet} {σ : H.Valuation}
  {conclusion : Term} {F : Formula}
  (hle : Formula.holds σ (.atom .le conclusion (.const 0)))
  (hform : syzygyConclusionForm conclusion F) :
  Formula.holds σ F

```

BernsteinLogic.Soundness . **sound\_syzygy**

statement bound

```

/-- Soundness of Syzygy (= Lemma B, dialect D): valid components give a
valid composite. The argument is `sound_farkas`'s, with the per-entry
source discharge generalized: a component-sourced entry's atom holds by
component validity (through `premiseContains`), a wiring-sourced entry's
atom holds as a premise sub-conjunct; either way
`atomLeZeroPolys_sound_entryC` delivers per-entry nonpositivity at every
clamped per-entry valuation, and the nonnegative fold closes. The
zero-normal bound then reaches `seq.post` through whichever
`szyzygyConclusionForm` reading the payload declared (both-polarity core
change, 2026-07-18). -/
theorem sound_syzygy {H : Hypernet} (st : SyzygyStep H)
  (hcomps : ∀ c ∈ st.components, c.semanticallyValid H) :
  st.seq.semanticallyValid H

```

BernsteinLogic.Soundness . **atomLeZeroPolys\_sound\_entryC**

statement bound

```

/-- Every ` $(e ≤ 0)$ `-normal-form polynomial a held atom entails evaluates
nonpositively at EVERY clamped per-entry valuation, given positive-size
wire types (general shapes – heterogeneous atoms read through the
clamp; the positivity is PROPOSED-B14's honest content: zero-size atoms
hold vacuously and entail nothing). -/
theorem atomLeZeroPolys_sound_entryC {H : Hypernet} (σ : H.Valuation)
  {F : Formula} (hF : Formula.holds σ F)
  (hw : ∀ w ∈ F.freeWires, ∃ τ, H.wireType w = some τ ∧
    0 < Shape.totalSize τ.shape)
  (hix : F.atomSidesIdxFree = true) (i : ℕ) :
  ∀ pa ∈ atomLeZeroPolys F, QArith.Poly.eval (σ.entryValC i) pa ≤ 0

```

BernsteinLogic.Soundness . **wire\_discipline\_of\_bshape\_fin**

statement bound

```

/-- The PROPOSED-B13 field pair (`bshape` + `finWires`) yields the wire
discipline that per-entry evaluation consumes: every free wire reads a
finite tensor of shape `[]` or `s`. -/
theorem wire_discipline_of_bshape_fin {H : Hypernet} (σ : H.Valuation)
  {t : Term} {s : Shape} (hb : t.bshape H = some s)

```

```
(hf : t.finWires H) (hix : t.idxFree = true) :
  ∀ w ∈ t.freeWires, ∃ qw, σ.wireQ w = some qw ∧
    (qw.shape = [] ∨ qw.shape = s)
```

BernsteinLogic.Soundness . Term.evalQ\_total\_singleton

statement bound

```
/-- Total singleton evaluation: if every free wire reads as a finite
singleton tensor, the term evaluates to a singleton tensor whose scalar
is the polynomial evaluation at the valuation read-out. (Definedness half
of the PROPOSED-B13 Farkas fields, and the workhorse behind every scalar
extraction below.) -/
theorem Term.evalQ_total_singleton {H : Hypernet} (σ : H.Valuation) :
  ∀ (t : Term),
    (∀ w ∈ t.freeWires, ∃ qw, σ.wireQ w = some qw ∧ SingletonShape qw.shape) →
    t.idxFree = true →
    ∃ q, Term.evalQ σ t = some q ∧ SingletonShape q.shape ∧
      q.scalar = QArith.Poly.eval σ.rho t.toPoly
```

**What `syzygyCombinationValid` means — and what it does not.** The rule's side condition appears in the payload structure above as a single opaque name, so this book binds its meaning by displaying the definition chain in full, from the checker sources. Reading the chain: `syzygyCombinationValid` is precisely the proposition “`syzygyCheck` returns true”, and `syzygyCheck` is the disjunction of three branches of one algorithm (a branch for index-free terms, a wire-coordinate branch for indexed terms, and a mixed tagged-union branch for heterogeneous combinations of both kinds). All three branches do the same three things: decode the combination term into weighted entries (`farkasEntries` — a fixed canonical encoding, anything else is rejected); require each weight nonnegative and each entry's polynomial to occur among the  $(\cdot \leq 0)$ -normal polynomials of a *component conclusion* or of an *equality sub-conjunct of the premise* (`atomLeZeroPolys` — note that equality atoms contribute both polarities, which is exactly the free-sign multiplier discipline); and require the weighted fold of those polynomials to equal the conclusion term's canonical polynomial.

What a skeptical reader should confirm by eye in the definitions below: the conclusion term enters the check *only* as the target of the exact-identity comparison (and through a syntactic index-freedom gate); the step's posted formula `seq.post` does not appear in the check at all; and nothing in the chain evaluates, assumes, or otherwise consults the *truth* of the conclusion — the check manipulates polynomials, never valuations, so it could not covertly “assert the conclusion is true”. `seq.post` connects to the conclusion term only through the separate, purely syntactic field `postForm` — the three readings of  $c \leq 0$  enumerated by `syzygyConclusionForm` below — whose semantic transport is the separately proven ring lemma `holds_of_syzygyConclusionForm` quoted above. Semantic truth is established only in `sound_syzygy`, from the component-validity hypotheses.

BernsteinLogic.Core.Catalogue . syzygyConclusionForm

definition bound

```
def syzygyConclusionForm (conclusion : Term) (F : Formula) : Prop :=
  F = .atom .le conclusion (.const 0)
  ∨ (∃ e, conclusion = .neg e ∧ F = .atom .ge e (.const 0))
  ∨ (∃ l r, conclusion = .sub l r ∧ F = .atom .le l r)
```

BernsteinLogic.Core.Catalogue . atomLeZeroPolys

definition bound

```
/-- The `(e ≤ 0)`-normal-form polynomials entailed by one premise atom
(exact polynomial forms — never rescaled; F7 discipline). -/
def atomLeZeroPolys : Formula → List QArith.Poly
| .atom .le l r => [QArith.Poly.sub l.toPoly r.toPoly]
| .atom .lt l r => [QArith.Poly.sub l.toPoly r.toPoly]
| .atom .ge l r => [QArith.Poly.sub r.toPoly l.toPoly]
| .atom .gt l r => [QArith.Poly.sub r.toPoly l.toPoly]
| .atom .eq l r => [QArith.Poly.sub l.toPoly r.toPoly,
  QArith.Poly.sub r.toPoly l.toPoly]
| _ => []
```

BernsteinLogic.Core.Catalogue . farkasEntries

definition bound

```
/-- Decode the combination-coefficient Term (pinned canonical encoding:
right-nested `.add` chain of `.mul (.const λj) ej`, `.const 0` end). -/
def farkasEntries : Term → Option (List (Rat × Term))
| .const c => if c = 0 then some [] else none
| .add (.mul (.const l) e) rest =>
  (farkasEntries rest).map ((l, e) :: ·)
| _ => none
```

BernsteinLogic.Core.Catalogue . **syzygyCheckCore**

definition bound

```

/-- The Syzygy combination check – `farkasCheck`'s shape, lifted the
AG draft's two ways: entry atoms may come from COMPONENT CONCLUSIONS
(the sub-proofs) as well as from equality sub-conjuncts of the composite
premise; atom polynomials are unrestricted in degree. Entry encoding,
fold, exactness discipline, and the G1 idx-freedom gates are inherited
verbatim from `farkasCheck`. -/
def syzygyCheckCore (cc : Term) (components : List Sequent) (premise : Formula)
  (conclusion : Term) : Bool :=
  match farkasEntries cc with
  | none => false
  | some entries =>
    conclusion.idxFree &&
    entries.all (fun (l, e) =>
      decide (0 ≤ l) &&
      ((components.any fun c =>
        c.post.atomSidesIdxFree &&
        ((atomLeZeroPolys c.post).any fun pa => decide (pa = e.toPoly))) ||
        (premise.conjunctList.any fun a =>
          a.isEqAtom && a.atomSidesIdxFree &&
          ((atomLeZeroPolys a).any fun pa => decide (pa = e.toPoly))))) &&
      decide ((entries.foldr
        (fun (l, e) acc => QArith.Poly.add (QArith.Poly.scale l e.toPoly) acc)
        []) = conclusion.toPoly)

```

BernsteinLogic.Core.Catalogue . **syzygyCheckW**

definition bound

```

/-- The idx-form branch of the Syzygy check: entries
match component conclusions or premise equality sub-conjuncts by their
wire-aware coordinate polynomials – the multi-wire composites (DP
Syzygy/SmallGain, distinct region wires per component) are exactly why
`wireCoordKey` keys variables by (wire, coordinate). -/
def syzygyCheckW (cc : Term) (components : List Sequent) (premise : Formula)
  (conclusion : Term) : Bool :=
  match farkasEntries cc with
  | none => false
  | some entries =>
    match farkasEntriesW entries with
    | none => false
    | some pentries =>
      match conclusion.asCoordPolyIdxW? with
      | none => false
      | some pc =>
        pentries.all (fun le =>
          decide (0 ≤ le.1) &&
          ((components.any fun c =>
            (atomLeZeroPolysW c.post).any fun pa => decide (pa = le.2)) ||
            (premise.conjunctList.any fun a =>
              a.isEqAtom &&
              ((atomLeZeroPolysW a).any fun pa => decide (pa = le.2))))) &&
          decide ((pentries.foldr
            (fun le acc => QArith.Poly.add (QArith.Poly.scale le.1 le.2) acc)
            []) = pc)

```

BernsteinLogic.Core.Catalogue . **syzygyCheckM**

definition bound

```

/-- The mixed-variable branch of the Syzygy check: entries match
component conclusions or premise equality sub-conjuncts by their MIXED
tagged polynomials – heterogeneous composites (a chart wire's TIdx
coordinates and bare scalar interface wires in one combination) are
exactly why the tagged union space exists. -/
def syzygyCheckM (cc : Term) (components : List Sequent) (premise : Formula)
  (conclusion : Term) : Bool :=
  match farkasEntries cc with
  | none => false

```



```

| some entries =>
  match farkasEntriesM entries with
  | none => false
  | some pentries =>
    match conclusion.asCoordPolyMixed? with
    | none => false
    | some pc =>
      pentries.all (fun le =>
        decide (0 ≤ le.1) &&
        ((components.any fun c =>
          (atomLeZeroPolysM c.post).any fun pa => decide (pa = le.2)) ||
          (premise.conjunctList.any fun a =>
            a.isEqAtom &&
            ((atomLeZeroPolysM a).any fun pa => decide (pa = le.2)))))) &&
        decide ((pentries.foldr
          (fun le acc => QArith.Poly.add (QArith.Poly.scale le.1 le.2) acc)
          []) = pc)

```

BernsteinLogic.Core.Catalogue . **syzygyCheck**

definition bound

```

/-- The Syzygy combination check: legacy idx-free branch OR the
wire-aware idx coordinate branch OR the mixed tagged branch (mirrors
`farkasCheck`'s three-way split). -/
def syzygyCheck (cc : Term) (components : List Sequent) (premise : Formula)
  (conclusion : Term) : Bool :=
  syzygyCheckCore cc components premise conclusion ||
  syzygyCheckW cc components premise conclusion ||
  syzygyCheckM cc components premise conclusion

```

BernsteinLogic.Core.Catalogue . **syzygyCombinationValid**

definition bound

```

def syzygyCombinationValid (cc : Term) (components : List Sequent)
  (premise : Formula) (conclusion : Term) : Prop :=
  syzygyCheck cc components premise conclusion = true

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Judgment frame as in Ax. The statement bundles its data as **SyzygyStep** — field-for-field the reference checker's obligations, and field-for-field the payload minus the component-proof list (which enters `sound_syzygy` as the validity hypothesis `hcomps`) and minus the denominator-bound declaration (resource accounting; zero proof-theoretic weight, so rightly absent from the soundness statement).

- the component judgments  $\{\Phi_i \vdash e_i \leq 0\}_{i=1}^n \mapsto \text{st.components} : \text{List Sequent}$  with `hcomps`; the atoms  $e_i \leq 0$  have no binders of their own — each combination entry tagged `FROM-COMPONENT` is checked against a component's conclusion inside the combination check, which is where the  $e_i$  live.
- $\Phi \supseteq \Phi_i$  conjunct-wise  $\mapsto \text{st.preContains} : \text{premiseContains seq.pre c.pre}$ , per component.
- the equality sub-conjuncts  $\{g_k = 0\}$  of  $\Phi \mapsto$  the `FROM-WIRING` entry tag: such an entry's atom is checked to be a sub-conjunct of `seq.pre` directly.
- $\lambda_i \in \mathbb{Q}_{\geq 0}, \nu_k \in \mathbb{Q}$ , and  $c \equiv \sum_i \lambda_i e_i + \sum_k \nu_k g_k \mapsto \text{st.combination}$  and `st.conclusion` under `st.valid : syzygyCombinationValid` — the canonical-polynomial identity is recomputed by one of *three* certified routes (the idx-free `syzygyCheckCore` replay, the wire-coordinate `syzygyCheckW`, and the mixed `syzygyCheckM`), each discharged separately in `sound_syzygy`.
- the evaluability and positive-size packs  $\mapsto \text{conclShape}, \text{conclFin}, \text{wiresPos}, \text{compPostWiresPos}$  — displayed inline at `PBOUND`'s cell (this page's cell leaves them to the caption); `compPostWiresPos` is the honest extension of the positive-size discipline to component conclusions, whose absence has a kernel-checkable vacuous-conclusion counterexample (the source docstring).
- $\Psi$ 's three readings  $\mapsto \text{st.postForm} : \text{syzygyConclusionForm}$ , literally the displayed three-membered disjunction; the transport of the zero-normal bound to every reading is the certified-alongside `ge-bridge lemma holds_of_syzygyConclusionForm`, applied once here, inherited by `PBOUND` and the `PSDCERT` trio.

Width note: nothing in the statement consumes the port-Hamiltonian reading — supply rates, storage functions, and the wiring identity are producer-side interpretations of which the certified content is exactly the polynomial identity and its component sourcing; and the statement is proven at every hypernet and component list, with every hypothesis consumed.

## Lean verifier receipts

module BernsteinLogic.Soundness; bound build a20d90e66c13... (bound from receipt a9c0e34d3e3c9c8c, 2026-08-03T21:30:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a9c0e34d3e3c9c8c | 2026-08-03T21:30:36Z | build+decorrelated | current |

21 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.Core.Catalogue; bound build cbc92e976a1d... (bound from receipt 5abdb3589707b5df, 2026-08-03T21:30:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 24804e66edba0bc9 | 2026-08-03T20:25:09Z | build+decorrelated | current |
| 5abdb3589707b5df | 2026-08-03T21:30:33Z | build+decorrelated | current |

18 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

|                                                           |            |
|-----------------------------------------------------------|------------|
| build (lake build (elaboration))                          | current    |
| replay (leanchecker (official kernel replay))             | stale only |
| hardened (safe_verify (hardened screen))                  | stale only |
| decorrelated (lean4export + nanoda (decorrelated kernel)) | current    |

**comparator: none for this rule** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises a SYZGY node yet; current fixtures cover BERNONE and NORM shapes. The end-to-end SYZGY certificate DpCompositeSyzygy.lean is a kernel-checked instance but rides the handshake harness, not the comparator.

## HOL4 statement

theory bern\_catalogue, HOL4 development script bern\_catalogueScript.sml **script bound**

### Definition syzygyCheckCore\_def

```

Definition syzygyCheckCore_def:
  syzygyCheckCore cc components premise conclusion =
    case farkasEntries cc of
    | NONE => F
    | SOME entries =>
      termIdxFree conclusion ∧
      EVERY (λ(l,e).
        0 ≤ l ∧
        (EXISTS (λc. atomSidesIdxFree c.post ∧
          EXISTS (λpa. pa = termToPoly e) (atomLeZeroPolys c.post))
          components v
        EXISTS (λa. isEqAtom a ∧ atomSidesIdxFree a ∧
          EXISTS (λpa. pa = termToPoly e) (atomLeZeroPolys a))
          (conjunctList premise))) entries ∧
      (FOLDER (λ(l,e) acc. polyAdd (polyScale l (termToPoly e)) acc)
        [] entries =
        termToPoly conclusion)
End

```

### Definition syzygyCheck\_def

```

Definition syzygyCheck_def:
  syzygyCheck cc components premise conclusion =
    syzygyCheckCore cc components premise conclusion v
    syzygyCheckW cc components premise conclusion
End

```

## HOL verifier receipts

**documentary** hol-strict CLEAN over the ported checker chain: theory bern\_catalogue, 472 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, scripts sha-matched verbatim to the sources cited here (git blob e41b9ed0...). Downstream, the same chain carries the checker to a CakeML binary with the machine\_code\_sound tag ([ "DISK\_THM", [] ] (theory bernCertProofARM8)). Family-level bridge lemmas for the SYZYGY node (famEq\_syz, bern\_fam\_syz) ride an in-flight assembly and are not yet in the census table. As on the FARKAS and PBOUND pages: the Lean check has since grown a third (mixed tagged-union) branch, added 2026-07-19 after this port run — the divergence is refusal-direction only, and its port is named residue. Semantic stratum, also ported: the soundness twins sound\_syzygy (the proof-object form) and sound\_pBound (theory bern\_tr\_soundsyz: hol-strict clean, 7 theorems, 0 offences, verification run of 2026-07-22 on an isolated build host, content-addressed at the same sources); the certified-fragment keystone (checkPf\_sound\_frag, theory bern\_tr\_splice, 20 theorems, 0 offences, same run) discharges this rule's wire arm through the latter.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail. Every hypothesis of the rule is used below and remarked on; the formal-anchor table maps each step to the Lean declaration that carries it.

**Setting and claim.** Fix a hypernet  $H$ . A *valuation*  $\sigma$  assigns each wire a rational tensor of its declared shape; a sequent  $\Phi \vdash \Psi$  is *semantically valid* when every admissible  $\sigma$  satisfying  $\Phi$  satisfies  $\Psi$ . Atoms  $e \leq 0$  over tensor-shaped terms hold per-entry: every entry of the evaluated tensor is  $\leq 0$ . The rule's data (the SyzygyStep payload quoted above) supplies component sequents  $\Phi_i \vdash e_i \leq 0$  for  $i = 1, \dots, n$ , a combination certificate with weights  $\lambda_i \in \mathbb{Q}_{\geq 0}$  on the components and free weights  $\nu_k \in \mathbb{Q}$  on equality sub-conjuncts  $g_k = 0$  of  $\Phi$ , a conclusion term  $c$ , and a checked *syntactic* identity

$$c \equiv \sum_i \lambda_i e_i + \sum_k \nu_k g_k \quad (\text{equality of exact canonical polynomials over } \mathbb{Q}). \quad (*)$$

The claim (sound\_syzygy): if every component sequent is semantically valid, the composite sequent  $\Phi \vdash \Psi$  is semantically valid, where  $\Psi$  is any reading of  $c \leq 0$  admitted by syzygyConclusionForm (Lemma R33.3).

**Lemma R33.1** (nonnegative combination). *In an ordered field, if  $a_i \leq 0$  and  $\lambda_i \geq 0$  for  $i = 1, \dots, n$ , and  $b_k = 0$  for  $k = 1, \dots, m$  with  $\nu_k$  arbitrary, then  $\sum_i \lambda_i a_i + \sum_k \nu_k b_k \leq 0$ .*

*Proof.* Each  $\lambda_i a_i \leq 0$ : multiplication by a nonnegative element preserves  $\leq$  (applied to  $a_i \leq 0$ ), so  $\lambda_i a_i \leq \lambda_i \cdot 0 = 0$ . Each  $\nu_k b_k = \nu_k \cdot 0 = 0$  regardless of the sign of  $\nu_k$  — this sign-blindness on equality atoms is precisely the freedom classical Farkas variants grant multipliers of equality constraints [216, §7.3]. A finite sum of nonpositive elements is nonpositive, by induction on  $n + m$ .  $\square$

**Lemma R33.2** (the certified identity evaluates). *Let  $(*)$  hold as an identity of canonical polynomials, and let  $\sigma$  be a valuation at which  $c$  and every  $e_i, g_k$  evaluate (per-entry, through the clamped per-entry valuation for tensor shapes). Then the rational identity  $c(\sigma) = \sum_i \lambda_i e_i(\sigma) + \sum_k \nu_k g_k(\sigma)$  holds at every entry.*

*Proof.* First fix what “per-entry” means. For an entry index  $j$  of the conclusion's shape, the *clamped per-entry valuation* reads each wire at position  $j$  when  $j$  is in range for that wire's shape and at its entry 0 otherwise — total on every wire in sight, because the B14 positivity pack guarantees each carries at least one entry, and agreeing with the plain positional read wherever  $j$  is in range. No shape-agreement hypothesis is needed beyond that: a sourced polynomial's wires share its own atom's shape (rank-0 broadcast aside), so they clamp in step, and its clamped value at  $j$  is an honest entry of its own evaluated tensor — which is where Step 4's per-entry case split picks it up. Evaluation at this scalar assignment is a ring homomorphism  $\mathbb{Q}[\text{wires}] \rightarrow \mathbb{Q}$ ; a syntactic identity of canonical forms is an identity of polynomials, and ring homomorphisms preserve polynomial identities. Two remarks make this step deserve its own lemma. First,  $(*)$  is checked by *exact canonical-form equality* over  $\mathbb{Q}$  (syzygyCombinationValid, replayed through toPoly), not by evaluation at test points — so no completeness, degree bound, or interpolation argument is needed anywhere in this proof; soundness is pure arithmetic. Second, the checker admits three certified routes to the same fact (the idx-free branch, the wire-coordinate branch, and the mixed tagged-union branch: syzygyCheck = Core  $\vee$  W  $\vee$  M); whichever branch fires, the certified fact consumed downstream is the same — the identity  $(*)$ , under the same per-entry reading. The branches differ in how terms are decoded, never in what is certified.  $\square$

**Lemma R33.3** (reading transport). *Suppose  $\sigma$  satisfies the zero-normal atom  $c \leq 0$ . Then  $\sigma$  satisfies every formula  $F$  admitted by syzygyConclusionForm( $c, F$ ), namely: (Z $\leq$ )  $F = (c \leq 0)$  itself; (G)  $F = (e \geq 0)$  where  $c$  is literally the negation term  $-e$ ; (L $\leq$ )  $F = (l \leq r)$  where  $c$  is literally the difference term  $l - r$  — structural identities of terms, not merely canonical-polynomial ones.*

*Proof.* ( $Z \leq$ ) is the identity transport. (G): per entry,  $-e(\sigma) \leq 0 \iff e(\sigma) \geq 0$  (negate and flip, an ordered-field axiom). ( $L \leq$ ): per entry,  $l(\sigma) - r(\sigma) \leq 0 \iff l(\sigma) \leq r(\sigma)$  (add  $r(\sigma)$  to both sides). Structurality is load-bearing: because  $e, l, r$  are subterms of  $c$ , the evaluability that the zero-normal atom's holding certifies for  $c$  passes to them — a merely canonical identity would not carry evaluation (a cancelled ill-typed subterm vanishes from the polynomial yet still poisons the term), which is why the admitted readings are stated on term structure. This is the ge-bridge ring lemma `holds_of_syzygyConclusionForm` (the both-polarity conclusion-form design): the readings are one exact  $\mathbb{Q}$  identity re-presented under a sign flip, so the polarity bridge lives *once*, here, and every  $\leq 0$ -shaped rule (SYZYGY itself, PBOUND's compact spelling, the `PsdCert` family) inherits all readings with no separate justification.  $\square$

**Theorem R33.4** (soundness of SYZYGY = Lemma B, dialect D). *Under the payload's checked hypotheses, if every component sequent  $\Phi_i \vdash e_i \leq 0$  is semantically valid, then so is  $\Phi \vdash \Psi$ .*

*Proof.* Fix an admissible valuation  $\sigma$  with  $\sigma \models \Phi$ ; we must show  $\sigma \models \Psi$ .

*Step 1 (components hold at  $\sigma$ ).* For each  $i$ , the payload checks  $\Phi \supseteq \Phi_i$  conjunct-wise (premiseContains — an ordered-tree containment certificate, and satisfaction descends along it by one  $\wedge$ -projection per step); hence  $\sigma \models \Phi_i$ , and by the assumed semantic validity of the  $i$ -th component,  $\sigma \models e_i \leq 0$ : every entry of  $e_i(\sigma)$  is  $\leq 0$ .

*Step 2 (wiring identities hold at  $\sigma$ ).* Each  $g_k = 0$  is an equality sub-conjunct of  $\Phi$ , so  $\sigma \models g_k = 0$  directly:  $g_k(\sigma) = 0$  at every entry.

*Step 3 (the conclusion evaluates).* The B13 shape fields (`conclShape`, `conclFin`) guarantee  $c$  has a determinate finite Bernstein shape over  $H$ , and the B14 positivity fields (Remark R33.5) guarantee every wire in sight has positive total size; together these give a total per-entry evaluation of  $c$  at  $\sigma$  — the two-case shape analysis (singleton vs. indexed) in `sound_syzygy`'s body.

*Step 4 (combine).* At each entry, Lemma R33.2 turns the certified identity  $(*)$  into the rational identity  $c(\sigma) = \sum_i \lambda_i e_i(\sigma) + \sum_k \nu_k g_k(\sigma)$ ; the per-entry source discharge is the case split of Steps 1–2 (a component-sourced entry's atom holds by component validity, a wiring-sourced entry's atom holds as a premise sub-conjunct — `atomLeZeroPolys_sound_entryC` delivers per-entry nonpositivity in both cases), and Lemma R33.1 closes the fold:  $c(\sigma) \leq 0$  at every entry.

*Step 5 (read off).* Lemma R33.3 transports the zero-normal bound  $c \leq 0$  to whichever reading `syzygyConclusionForm`( $c, \Psi$ ) the payload declared (`postForm`). Hence  $\sigma \models \Psi$ .  $\square$

*Remark R33.5* (why the B14 hypotheses are not decoration). `wiresPos` and its honest extension `compPostWiresPos` require every wire appearing in  $\Phi$ 's free wires, the conclusion, and each component conclusion to have positive total size. Drop `compPostWiresPos` and the rule is *unsound*: a component conclusion over a zero-size wire holds *vacuously* (its evaluated tensor has no entries, so “every entry  $\leq 0$ ” is empty), yet its polynomial enters the combination  $(*)$  with full algebraic weight — a vacuously-held component can then push the composite conclusion to something false. Concretely: over a hypernet with one zero-size wire  $z$ , take the single component  $\top \vdash z - z + 1 \leq 0$  — vacuously valid, its evaluated tensor having no entries — with weight 1 and empty  $\nu$ -row; the combination's canonical polynomial is the constant 1, so the checked identity admits the conclusion  $1 \leq 0$ , a rank-0 falsehood concluded from a vacuous truth. (The counterexample is also kernel-checkable, recorded at the field's introduction.) The B13 shape fields have the matching witness one step earlier: with  $c$  the term  $w - w$  for an untyped wire  $w$ , the canonical polynomial is 0 — identical to the empty combination's — yet the conclusion atom never evaluates and so never holds; `conclShape` and `conclFin` exclude exactly this. This is the classic mismatch between empty-product semantics and nonempty algebra, caught by making the degenerate case unrepresentable rather than discouraged.

*Remark R33.6* (relation to the literature). The rule is the *sound* (easy) direction of an affine Farkas-type combination, lifted twice: atoms come from sub-*proofs* rather than sub-conjuncts, and atoms may have arbitrary polynomial degree. The linear ancestor is Farkas' lemma [73]; the textbook treatment of multiplier sign conventions ( $\lambda \geq 0$  on inequalities, free on equalities) is [216, §7]. With polynomial atoms the combination sits in the Krivine–Handelman family of positivity certificates [110, 144] — of which, again, only the trivial direction is claimed or needed: no Positivstellensatz completeness enters the kernel. The wiring reading — components as open subsystems,  $g_k = 0$  as power-preserving interconnection, the composite as dissipation of  $\sum_i \lambda_i V_i$  — is Tellegen's theorem [231] in certificate form, i.e. compositional dissipativity in the sense of Willems [239] and Moylan–Hill [184]; the port-Hamiltonian formulation is surveyed in [215]. Within the accompanying development the rule is known as Lemma B (dialect D); a quantitative effectivity result for tree-structured interconnections — sparse Handelman certificates with running-intersection support — is developed in a companion note.

| proof step                                             | formal anchor                                                                                                                                                                                            |
|--------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Step 1: conjunct-wise premise containment              | <code>premiseContains</code> (payload field <code>preContains</code> )                                                                                                                                   |
| Steps 1–2, per-entry atom discharge                    | <code>atomLeZeroPolys_sound_entryC</code> ; <code>source tags FROM-COMPONENT/FROM-WIRING</code>                                                                                                          |
| Step 3: conclusion evaluates                           | <code>conclShape</code> , <code>conclFin</code> (B13); <code>wiresPos</code> , <code>compPostWiresPos</code> (B14); <code>wire_discipline_of_bshape_fin</code> , <code>Term.evalQ_total_singleton</code> |
| Step 4: certified identity, all three checker branches | <code>syzygyCombinationValid</code> , <code>syzygyCheck</code> = <code>syzygyCheckCore</code> $\vee$ <code>syzygyCheckW</code> $\vee$ <code>syzygyCheckM</code>                                          |
| Lemma R33.1 fold                                       | the nonnegative fold in <code>sound_syzygy</code> 's body                                                                                                                                                |
| Lemma R33.3                                            | <code>holds_of_syzygyConclusionForm</code>                                                                                                                                                               |
| Theorem R33.4                                          | <code>sound_syzygy</code> ; wired as core <code>RuleData.syzygy</code> into <code>ProofObject.sound</code>                                                                                               |

## Rule 34. SMALLGAIN (alias for SYZGYG)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *SmallGain*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi_1 \vdash e_1 \leq 0 \quad \Phi_2 \vdash e_2 \leq 0 \quad \Phi_w \vdash \lambda \sigma_2 - (1-\theta) \alpha_1 \leq 0 \quad \Phi_w \vdash \sigma_1 - \lambda(1-\theta) \alpha_2 \leq 0 \quad \Phi \supseteq \Phi_1, \Phi_2, \Phi_w \text{ conjunct-wise (checked)} \quad \lambda \in \mathbb{Q}_{>0} \quad \theta \in \mathbb{Q} \cap (0, 1)}{\Phi \vdash \mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2) \leq 0} \text{ SMALLGAIN}$$

$e_i \equiv \mathcal{L}_i V_i + \alpha_i - \sigma_i$ ;  $m_1 \equiv \lambda \sigma_2 - (1-\theta) \alpha_1$ ,  $m_2 \equiv \sigma_1 - \lambda(1-\theta) \alpha_2$ ; conclusion identity, checked as exact canonical polynomial:  $e_1 + \lambda e_2 + m_1 + m_2 = \mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2)$ .

the two membership sequents are ordinary BL sequents over the declared operating box  $\Phi_w$ , their BERNCONe multipliers *are* the LP witness; in the conclusion identity the cross-supplies cancel by construction — the checker never sees a fixpoint. In the baked base theory this rule sits in the admissible roster — no proof-term sort, no wire tag: it elaborates to one SYZGYG node at weights  $(1, \lambda, 1, 1)$ , which this catalogue renders as the alias tier.

SMALLGAIN is the calculus's feedback-composition rule in its classical costume: two subsystems, each dissipating its own storage function except for a supply fed by the other, compose into a single dissipation certificate for  $V_1 + \lambda V_2$  when the loop gain is strictly less than one ( $\theta$  is the margin). It is an *alias*: the wire format never carries a SMALLGAIN node. A producer that finds  $(\lambda, \theta)$  — an LP feasibility problem over the membership certificates — elaborates the step to the single SYZGYG node displayed in the recipe below before emission, so what the checker recomputes is exactly SYZGYG's obligation set: the exact conclusion identity at weights  $(1, \lambda, 1, 1)$ , in which the cross-supplies cancel by construction. Nothing fixpoint-shaped survives to checking time; the classical contraction argument lives entirely on the producer's side, in the choice of  $(\lambda, \theta)$ .

What would break without each hypothesis is inherited from SYZGYG through the unpack — that section's commentary applies verbatim to the unpacked node. The two conditions native to this rule:  $\lambda > 0$ , of which soundness uses only  $\lambda \geq 0$  (the strictness backs the rule's published meaning), and  $\theta \in (0, 1)$ , which is well-formedness for  $\theta$ 's published meaning as a robustness margin rather than a soundness need — the membership sequents self-police, and an infeasible pair simply yields unprovable memberships.

**notes** The four component sequents are ordinary sequents over the declared operating box; their cone-membership multipliers *are* the LP witness. Signed polynomial contracts (rather than norm gains) are constitutive, not stylistic: swing-up analyses in the Åström–Furuta family exploit the *sign* of  $\dot{\vartheta} \cos \vartheta$ , which no norm-gain contract can express.  $(\lambda, \theta)$ -existence stays on the producer side, deliberately outside the checker; the  $n$ -component generalization is the SMALLGAINN section — the same alias discipline over an arbitrary interconnection graph, elaborating to one SYZGYG node, of which this rule is the  $n = 2$  instance.

### Lean statement

BernsteinLogic.Composition . SmallGainStep

statement bound

```
-- Applicability data of the SmallGain rule – field-for-field the
-- draft's `smallGainShapes` obligations (§3.3):

* (a) the dissipation conclusions are `(e_i ≤ 0)` atoms – the residual
    reading  $e_i = \mathcal{L}_i V_i + \alpha_i - \sigma_i$  is definitional at the checker level
    ( $\mathcal{L}_i V_i := e_i - \alpha_i + \sigma_i$ ), so no separate field pins it;
* (b) the membership conclusions are `(m_i ≤ 0)` atoms whose polynomials
    equal their schema instances at  $(\lambda, \theta)$  – exact canonical polynomial;
* (c)  $\lambda > 0$  and  $0 < \theta < 1$  – three rational comparisons. The  $\theta$  range is
    NOT consumed by soundness (the memberships self-police; the range backs
    the published guard-modulus semantics – draft §3.2's honesty note);
* (d) the conclusion identity: the  $(1, \lambda, 1, 1)$  fold of the four posted
    polynomials equals the conclusion schema  $\mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2)$  – the
    checker replays the cancellation (`smallGain_is_syzygy` proves the
    schema coherent: given (b) and the residual reading, (d) is a theorem,
    not a search).

B13/B14 fields as in `SyzygyStep`, component-post coverage included. -/
structure SmallGainStep (H : Hypernet) where
  seq : Sequent
```

```

diss1 : Sequent
diss2 : Sequent
mem1 : Sequent
mem2 : Sequent
lambda : ℚ
theta : ℚ
alpha1 : Term
alpha2 : Term
sigma1 : Term
sigma2 : Term
lyapTerm : Term
e1 : Term
e2 : Term
m1 : Term
m2 : Term
concl : Term
-- (a) dissipation shapes
diss1Post : diss1.post = .atom .le e1 (.const 0)
diss2Post : diss2.post = .atom .le e2 (.const 0)
-- (b) membership shapes and schemas
mem1Post : mem1.post = .atom .le m1 (.const 0)
mem2Post : mem2.post = .atom .le m2 (.const 0)
mem1Poly : m1.toPoly = Poly.sub (Poly.scale lambda sigma2.toPoly)
(Poly.scale (1 - theta) alpha1.toPoly)
mem2Poly : m2.toPoly = Poly.sub sigma1.toPoly
(Poly.scale (lambda * (1 - theta)) alpha2.toPoly)
-- G1 rider: the posted residuals and the conclusion are idx-free --
-- their canonical polynomials carry the rule's meaning (`Term.toPoly`'s
-- G1 invariant; mirrors `farkasCheck`'s gates)
e1IdxFree : e1.idxFree = true
e2IdxFree : e2.idxFree = true
m1IdxFree : m1.idxFree = true
m2IdxFree : m2.idxFree = true
conclIdxFree : concl.idxFree = true
-- (c) the three rational comparisons
lambdaPos : 0 < lambda
thetaLo : 0 < theta
thetaHi : theta < 1
-- (d) the conclusion identity (checker-replayed cancellation), and the
-- conclusion term's tie to the schema
foldEq : Poly.add e1.toPoly (Poly.add (Poly.scale lambda e2.toPoly)
(Poly.add m1.toPoly m2.toPoly))
= Poly.add lyapTerm.toPoly
(Poly.scale theta
(Poly.add alpha1.toPoly (Poly.scale lambda alpha2.toPoly)))
conclPoly : concl.toPoly = Poly.add lyapTerm.toPoly
(Poly.scale theta
(Poly.add alpha1.toPoly (Poly.scale lambda alpha2.toPoly)))
postEq : seq.post = .atom .le concl (.const 0)
-- premise containment for all four components
pre1 : premiseContains seq.pre diss1.pre
pre2 : premiseContains seq.pre diss2.pre
preM1 : premiseContains seq.pre mem1.pre
preM2 : premiseContains seq.pre mem2.pre
-- B13/B14 (component-post coverage per the §R2 finding)
conclShape : ∃ s, concl.bshape H = some s
conclFin : concl.finWires H
wiresPos : ∀ w ∈ seq.pre.freeWires ++ concl.freeWires,
  ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape
compPostWiresPos : ∀ c ∈ [diss1, diss2, mem1, mem2],
  ∀ w ∈ c.post.freeWires,
  ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape

```

BernsteinLogic.Composition . **toSyzygy**

statement bound

| /-- The SmallGain node, AS the Syzygy instance it is (weights



```
(1, λ, 1, 1), components in combination order). -/
def toSyzygy (st : SmallGainStep H) : SyzygyStep H
```

BernsteinLogic.Composition . **sound\_smallGain**

statement bound

```
/-- Soundness of SmallGain (= Lemma B, dialect G), as the Syzygy
corollary: the four component validities feed `sound_syzygy` on the
collapsed instance. The three rational comparisons ( $\lambda > 0$ ,  $0 < \theta < 1$ )
enter only through  $\lambda \geq 0$  inside the combination check;  $\theta$ 's range is
well-formedness, exactly as the draft's honesty note states. -/
theorem sound_smallGain {H : Hypernet} (st : SmallGainStep H)
  (h1 : st.diss1.semanticallyValid H)
  (h2 : st.diss2.semanticallyValid H)
  (hm1 : st.mem1.semanticallyValid H)
  (hm2 : st.mem2.semanticallyValid H) :
  st.seq.semanticallyValid H
```

BernsteinLogic.Composition . **toSyzygy\_components**

statement bound

```
theorem toSyzygy_components (st : SmallGainStep H) :
  st.toSyzygy.components = st.components
```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The alias's Lean vehicle is the payload structure `SmallGainStep H` (quoted in full above — its fields carry the display's metavariables), the verified unpack `toSyzygy`, and the soundness corollary `sound_smallGain`.

- $\lambda \mapsto \text{lambda} : \mathbb{Q}$  with  $\text{lambdaPos} : 0 < \lambda$ ;  $\theta \mapsto \text{theta}$  with  $\text{thetaLo}, \text{thetaHi} : 0 < \theta < 1$ ;  $\alpha_i, \sigma_i \mapsto \text{alpha1/alpha2}, \text{sigma1/sigma2}$ ; the composite storage  $V_1 + \lambda V_2 \mapsto \text{lyapTerm}$ .
- the dissipation premises  $\Phi_i \vdash e_i \leq 0 \mapsto \text{diss1/diss2}$  with the shape ties  $\text{diss1Post/diss2Post}$  (each  $e_i$  is a field; its published spelling  $\mathcal{L}_i V_i + \alpha_i - \sigma_i$  lives on the producer's side). The operating-box membership premises  $\mapsto \text{mem1/mem2}$  with  $\text{mem1Poly/mem2Poly}$  tying their residuals to the displayed schemas  $\lambda \sigma_2 - (1-\theta)\alpha_1$  and  $\sigma_1 - \lambda(1-\theta)\alpha_2$  as canonical polynomials.
- the display's side text — the exact conclusion identity (the cross-supply-cancellation gloss is the flavor text's) —  $\mapsto \text{foldEq}$  and  $\text{conclPoly}$  (the checker-replayed cancellation, exact over canonical polynomials) plus  $\text{postEq}$  (the conclusion tie); the displayed containment premise ( $\Phi \supseteq \Phi_1, \Phi_2, \Phi_w$  conjunct-wise)  $\mapsto \text{pre1/pre2/preM1/preM2}$ .
- conclusion  $\Phi \vdash \mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2) \leq 0 \mapsto$  the goal `st.seq.semanticallyValid H` of `sound_smallGain`, whose four hypotheses are exactly the component validities.
- the alias artefacts: the unpack function is `toSyzygy` (weights (1, λ, 1, 1), components in combination order); the soundness corollary routes through the target rule's lemma on the collapsed instance — the displayed rule never reaches the wire.

Width notes, both directions: soundness uses only  $\lambda \geq 0$  of `lambdaPos` (the strictness backs the rule's published meaning), and  $\theta$ 's range is well-formedness rather than a soundness need — both as the commentary states. Conversely the structure carries `idxFree` and wire-shape fields (`e1IdxFree` through `compPostWiresPos`) that the display leaves to the ambient calculus's conventions: the mechanization is narrower there, naming every side condition the checker replays.

## Lean verifier receipts

module BernsteinLogic.Composition; bound build 9116f454beb0... (bound from receipt 548ea3dda8ed73a2, 2026-08-03T21:30:29Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 548ea3dda8ed73a2 | 2026-08-03T21:30:29Z | build+decorrelated | current |

10 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.



### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: structurally n/a the alias unpacks *before* emission (no rule tag; producers elaborate a SMALLGAIN step to its SYZYGY node), so no wire certificate — and hence no comparator fixture — can ever contain a SMALLGAIN node. Comparator coverage for this rule is SYZYGY's. The end-to-end instance `DpCompositeSmallGain.lean` is kernel-checked upstream of the unpack.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*, not by backlog: the wire format never carries a SMALLGAIN node (no tag — producers elaborate to the  $(1, \lambda, 1, 1)$  SYZYGY node before emission, warrant `SmallGainStep.toSyzygy`), and the HOL4 development's object is the wire checker. The HOL statement that verifies a composed small-gain certificate is therefore SYZYGY's (`syzygyCheck`, previous section); an independent HOL port of the alias's unpack function would be new assurance surface only if HOL-side producers ever emit SMALLGAIN steps directly.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template. An alias owes the book five artefacts: the verified  $1 \rightarrow 1$  unpack function, the filled-in application of the target rule, the payload-transport table, the exact algebraic identity that makes the respelling correct, and the soundness corollary through the target rule's lemma. All five follow. (The tier criterion counts derivation nodes, not instantiation work: the unpack *computes* the weighting, but its image is a single SYZYGY node, so the rule is an alias.)

**Recipe R34.1** (the alias template). An alias rule  $A$  is warranted by a verified function  $\text{unpack} : \text{Step}_A \rightarrow \text{Step}_T$  into the payload of a *single* target node  $T$  (core or admissible), such that the  $A$ -node and its image conclude the same sequent from the same premises: one node  $\rightarrow$  one node, payload transported, nothing derived. Soundness is then a corollary,  $\text{sound}_A = \text{sound}_T \circ \text{unpack}$ , and the wire format need never carry  $A$  at all — producers elaborate before emission, which is why SMALLGAIN has no `RuleTag`.

**Recipe R34.2** (the filled-in application — SMALLGAIN's premises rewritten *into* the SYZYGY instance). A SMALLGAIN step carries four component proofs: the two dissipation sequents (over their own boxes  $\Phi_1, \Phi_2$ ) and the two gain-membership sequents (over the declared operating box  $\Phi_w$ ). Define the four atom terms and the substitution into SYZYGY's metavariables:

$$e_1 \equiv \mathcal{L}_1 V_1 + \alpha_1 - \sigma_1, \quad e_2 \equiv \mathcal{L}_2 V_2 + \alpha_2 - \sigma_2, \quad m_1 \equiv \lambda \sigma_2 - (1-\theta) \alpha_1, \quad m_2 \equiv \sigma_1 - \lambda(1-\theta) \alpha_2,$$

$$n := 4, \quad (e_1, e_2, e_3, e_4) := (e_1, e_2, m_1, m_2), \quad (\Phi_1, \Phi_2, \Phi_3, \Phi_4) := (\Phi_1, \Phi_2, \Phi_w, \Phi_w),$$

$$(\lambda_1, \lambda_2, \lambda_3, \lambda_4) := (1, \lambda, 1, 1), \quad \{g_k\}_k := \emptyset \text{ (no equality entries)}, \quad c := \mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2),$$

with  $\lambda \in \mathbb{Q}_{>0}$  and  $\theta \in \mathbb{Q} \cap (0, 1)$  from the payload. Under this substitution the SMALLGAIN node is the following SYZYGY application (compare the rule display in the SYZYGY section, field for field):

$$\frac{\begin{array}{c} \Phi_1 \vdash e_1 \leq 0 \quad \Phi_2 \vdash e_2 \leq 0 \quad \Phi_w \vdash m_1 \leq 0 \quad \Phi_w \vdash m_2 \leq 0 \\ (1, \lambda, 1, 1) \in \mathbb{Q}_{\geq 0}^4 \quad c \equiv 1 \cdot e_1 + \lambda \cdot e_2 + 1 \cdot m_1 + 1 \cdot m_2 \text{ (exact canonical polynomial)} \end{array}}{\Phi \vdash c \leq 0} \text{SYZYGY}$$

with  $\Phi \supseteq \Phi_1, \Phi_2, \Phi_w$  conjunct-wise (the four-way `preContains` split). Discharging SYZYGY's obligations at this instance: the nonnegativity checks are  $1 \geq 0$  (three times) and  $\lambda \geq 0$  (from  $\lambda > 0$  — the *only* use soundness makes of  $\lambda$ 's sign); the sourcing check is immediate (each of the four entries is the corresponding component's posted atom, verbatim); and the exact-identity check is Proposition R34.3 below; the five index-freeness gates (the four posted residuals and the composite conclusion carry no free indices — recomputed syntactic checks, the G1 rider that lets canonical polynomials carry the rule's meaning) discharge by recomputation on the five terms; the remaining obligations — the B13 shape pack, the B14 positivity pack, and the zero-normal `postForm` declaration — transport verbatim, per Recipe R34.4. The dead premise slots of SYZYGY are honestly dead: no equality sub-conjuncts of  $\Phi$  enter (the  $\nu$ -row is empty), because the cross-supply cancellation is performed by the membership atoms themselves. The verified function computing this rewriting is `SmallGainStep.toSyzygy`, quoted above.

**Proposition R34.3** (the conclusion identity). *As an identity of exact canonical polynomials,*

$$e_1 + \lambda e_2 + m_1 + m_2 = \mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2).$$

*Proof.* Expand and collect — two lines, which is the point:

$$\begin{aligned} e_1 + \lambda e_2 + m_1 + m_2 &= (\mathcal{L}_1 V_1 + \lambda \mathcal{L}_2 V_2) + \alpha_1 + \lambda \alpha_2 \underbrace{- \sigma_1 - \lambda \sigma_2 + \lambda \sigma_2 + \sigma_1}_{= 0 \text{ (cross-supplies cancel)}} - (1-\theta)\alpha_1 - \lambda(1-\theta)\alpha_2 \\ &= \mathcal{L}(V_1 + \lambda V_2) + (1 - (1-\theta))\alpha_1 + \lambda(1 - (1-\theta))\alpha_2 = \mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2). \end{aligned}$$

The final rewrite reads  $\mathcal{L}(V_1 + \lambda V_2)$  for  $\mathcal{L}_1 V_1 + \lambda \mathcal{L}_2 V_2$ , and that is a definition, not a step: over disjoint variable sets the composite generator acts coordinate-wise, so the composite storage-rate term — the payload’s `lyapTerm`, pinned to exactly this sum by the exact canonical ties (`foldEq · conclPoly`) — has  $\mathcal{L}(V_1 + \lambda V_2)$  as its published spelling. The cross-supply cancellation is *by construction of the multipliers*: the membership sequents were chosen so that their polynomials supply exactly  $+\lambda\sigma_2$  and  $+\sigma_1$ . The checker replays this as `Syzygy`’s exact canonical-polynomial identity (\*) and **never sees a fixpoint** — the classical small-gain contraction argument is dissolved into two lines of ring arithmetic at certificate-checking time.  $\square$

**Recipe R34.4** (payload transport — `toSyzygy`, field for field).

| SmallGainStep                                | SyzygyStep                                                                 |
|----------------------------------------------|----------------------------------------------------------------------------|
| <code>seq</code>                             | <code>seq</code>                                                           |
| <code>diss1, diss2, mem1, mem2</code>        | <code>components</code> (that order = combination order)                   |
| <code>combTerm (weights (1, λ, 1, 1))</code> | <code>combination</code>                                                   |
| <code>concl</code>                           | <code>conclusion</code>                                                    |
| <code>pre1, pre2, preM1, preM2</code>        | <code>preContains</code> (four-way case split)                             |
| <code>conclShape, conclFin</code>            | <code>conclShape, conclFin</code> (verbatim)                               |
| <code>wiresPos, compPostWiresPos</code>      | <code>wiresPos, compPostWiresPos</code> (verbatim)                         |
| <code>e1/e2/m1/m2/conclIdxFree</code>        | consumed by <code>combValid</code> $\mapsto$ <code>valid</code> (G1 gates) |
| <code>combValid</code>                       | <code>valid</code>                                                         |
| <code>postEq</code>                          | <code>postForm = (Z≤)</code> , the zero-normal reading                     |

The alias declares the *zero-normal* reading of the both-polarity conclusion form: `SMALLGAIN`’s own contract stays  $\leq$ -only, and any polarity flexibility is inherited from `Syzygy`’s side of the fence (the reading-transport lemma `holds_of_syzygyConclusionForm`, proved in the `Syzygy` section), not re-justified here.

**Proposition R34.5** (soundness corollary). *If the four component sequents are semantically valid, the composite sequent  $\Phi \vdash \mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2) \leq 0$  is semantically valid.*

*Proof.* `sound_smallGain` = `sound_syzygy` applied to `st.toSyzygy`, feeding the four validities through the four-way membership case split. Beyond that routing there is no further argument — the alias genre working as intended.  $\square$

**Remark R34.6** ( $\theta$ ’s range is well-formedness, not soundness). The identity of Proposition R34.3 holds at *any*  $\theta$ ; the memberships self-police: an infeasible  $(\lambda, \theta)$  makes the required membership inequalities false over the operating box, so no sound certificate can discharge them. The  $(0, 1)$  range check backs the *published meaning* of  $\theta$  as a guard modulus — a first-class robustness radius travelling in the proof term, per-node in the payload, with the certificate header publishing the composite modulus as the min over nodes — and  $\lambda > 0$  enters soundness only as  $\lambda \geq 0$ .

**Remark R34.7** (relation to the literature). The rule certificate-izes the Lyapunov small-gain construction: two ISS-style dissipation inequalities with gain interconnection conditions yield a composite Lyapunov function  $V_1 + \lambda V_2$  — the nonlinear small-gain theorem of Jiang–Teel–Prly [126] in its Lyapunov formulation [125], sitting inside the dissipativity-composition tradition [184, 239]. Where the classical theorems close a fixpoint/contraction argument, the certificate form closes a ring identity: the analytic content is moved into the *choice* of  $(\lambda, \theta)$  and the membership certificates, and the kernel checks only arithmetic. The contracts are *signed polynomial* inequalities rather than norm gains for cause: Åström–Furuta’s swing-up analysis [9] exploits the sign of  $\dot{\vartheta} \cos \vartheta$ , information a pure norm-gain contract does not record.  $(\lambda, \theta)$ -existence results are deliberately outside the checker; the  $n$ -component generalization over arbitrary interconnection graphs [59] is catalogued as `SMALLGAINN` — the same alias discipline, one `Syzygy` node, with this rule as its  $n = 2$  instance.

| proof step             | formal anchor                                                                                                       |
|------------------------|---------------------------------------------------------------------------------------------------------------------|
| Recipe R34.4 unpack    | <code>SmallGainStep.toSyzygy</code> ( <code>Composition.lean</code> ); <code>toSyzygy_components</code>             |
| Proposition R34.3      | <code>combValid</code> ’s fold identity ( <code>foldEq · conclPoly</code> ), replayed as <code>Syzygy</code> ’s (*) |
| Proposition R34.5      | <code>sound_smallGain</code> (a <code>sound_syzygy</code> corollary)                                                |
| no wire tag, by design | <code>RuleTag</code> comment block ( <code>Core/Catalogue.lean</code> ): producers elaborate before emission        |
| end-to-end instance    | <code>Handshake/DpCompositeSmallGain.lean</code> (kernel-checked upstream of the unpack)                            |

## Rule 35. SMALLGAINN (alias for SYZGY)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *SmallGainN*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\Phi \supseteq \Phi_1, \dots, \Phi_n, \Phi_w \text{ conjunct-wise (checked)} \quad \frac{\{\Phi_i \vdash e_i \leq 0\}_{i=1}^n \quad \lambda \in \mathbb{Q}_{>0}^n \quad \frac{\{\Phi_w \vdash m_i \leq 0\}_{i=1}^n \quad B \in \mathbb{Q}_{\geq 0}^{n \times n} \quad \sum_i B_{ij} = \lambda_j \ (\forall j) \quad \theta \in \mathbb{Q} \cap (0, 1)}{\Phi \vdash \mathcal{L}(\sum_i \lambda_i V_i) + \theta \sum_j \lambda_j \alpha_j \leq 0}}{\text{SMALLGAINN}}$$

$e_i \equiv \mathcal{L}_i V_i + \alpha_i - \sigma_i$ ;  $m_i \equiv \lambda_i \sigma_i - (1 - \theta) \sum_j B_{ij} \alpha_j$  (the repayment matrix's membership rows); conclusion identity, checked as exact canonical polynomial:  $\sum_i \lambda_i e_i + \sum_i m_i = \mathcal{L}(\sum_i \lambda_i V_i) + \theta \sum_j \lambda_j \alpha_j$  under the column condition.

the  $n$  membership sequents are ordinary BL sequents over the declared operating box  $\Phi_w$ , their BERNCONe multipliers *are* the LP witness; the repayment matrix  $B$  routes each component's supply through its neighbours, and the column condition  $\sum_i B_{ij} = \lambda_j$  makes every cross-supply cancel in the conclusion identity — the checker never sees a fixpoint, a spectral radius, or the interconnection graph. Like the binary rule it is an alias: no proof-term sort, no wire tag; it elaborates to one SYZGY node at weights  $(\lambda_1, \dots, \lambda_n, 1, \dots, 1)$ , and  $B$  itself never reaches the wire (the membership rows carry it; the exact identity is the whole obligation). The binary rule is the  $n = 2$  instance on the nose:  $\lambda = (1, \lambda)$ ,  $B_{21} = 1$ ,  $B_{12} = \lambda$  reproduce SMALLGAIN's membership rows and its  $(1, \lambda, 1, 1)$  weights.

SMALLGAINN is the  $n$ -component quantitative small-gain rule:  $n$  subsystems on an arbitrary directed interconnection graph, each dissipating its own storage function except for supplies fed by its neighbours, compose into a single dissipation certificate for the weighted sum  $\sum_i \lambda_i V_i$  when a weighted loop-gain condition holds with margin  $\theta$ . The graph structure enters through the *repayment matrix*  $B$ : the membership row  $m_i \equiv \lambda_i \sigma_i - (1 - \theta) \sum_j B_{ij} \alpha_j$  charges component  $i$ 's incoming supply against a  $(1 - \theta)$ -discounted draw on its neighbours' decay budgets, and the column condition  $\sum_i B_{ij} = \lambda_j$  says every component's budget is drawn on at exactly its weight in the composite — which is what makes the cross-supplies cancel identically in the conclusion.

Like the binary rule, it is an *alias*: the wire format never carries a SMALLGAINN node, and the checker gains no new obligation kind. A producer that finds  $(\lambda, B, \theta)$  — an LP feasibility problem over the membership certificates; when the gain matrix has spectral radius below  $1 - \theta$ , a Neumann-series weight vector witnesses feasibility, so the classical spectral small-gain condition is exactly an existence theory for this rule's side data — elaborates the step to a single SYZGY node at weights  $(\lambda_1, \dots, \lambda_n, 1, \dots, 1)$  before emission. What the checker recomputes is exactly SYZGY's obligation set: the exact conclusion identity over canonical polynomials. Neither the graph, nor  $B$ , nor anything fixpoint- or spectrum-shaped survives to checking time.

What would break without each hypothesis is inherited from SYZGY through the unpack — that section's commentary applies verbatim to the unpacked node. The conditions native to this rule:  $\lambda_i > 0$ , of which soundness uses only  $\lambda_i \geq 0$  (strictness backs the published meaning: every component genuinely participates);  $\theta \in (0, 1)$ , well-formedness for  $\theta$ 's published meaning as a robustness margin (the identity holds at any  $\theta$ ; the memberships self-police); and the column condition, which is well-formedness at the *schema* level — it is what makes the membership rows deliver the stated conclusion for every decay budget  $\alpha$ , while at any concrete instance the exact identity subsumes it: an emitted node whose identity checks is sound regardless of how the producer bookkept  $B$ .

The binary rule is the  $n = 2$  instance on the nose:  $\lambda = (1, \lambda)$ ,  $B_{21} = 1$ ,  $B_{12} = \lambda$  reproduce SMALLGAIN's membership rows (as a set — the two rows arrive in the opposite order, a bookkeeping difference in combination order) and its  $(1, \lambda, 1, 1)$  weights.

**notes** The  $n$ -component small-gain theorem for general interconnection topologies is Dashkovskiy–Rüffer–Wirth [59]; this rule is its certificate form, with the cycle/spectral conditions of that tradition dissolved into the producer's choice of  $(\lambda, B, \theta)$  and the checker replaying only ring arithmetic. Signed polynomial contracts rather than norm gains, for the same cause as the binary section. The dedicated  $n$ -ary payload vehicle is mechanized as the binary rule's pattern at arbitrary  $n$ : the exact identity (`smallGainN_is_syzgy` — the cross-supplies cancel under the column condition, by canonical-polynomial extensionality), the payload structure with its verified collapse (`SmallGainNStep.toSyzgy`, weights  $(\lambda_1, \dots, \lambda_n, 1, \dots, 1)$ ), the soundness corollary (`sound_smallGainN`), and the column condition's necessity as a machine-checked mutant (`smallGainN_colSum_mutant_refuted`: delete it and a one-component instance separates fold from conclusion, by decide). The semantic face lives beside the binary one's in the discrete-kernel module: `smallGainN_drift` with its assembled avoid bound, and the binary recovery on the nose (`smallGain_drift_of_N`:  $\lambda = (1, \lambda)$ ,  $B_{21} = 1$ ,  $B_{12} = \lambda$  reproduce the binary rows — order-swapped, set-equal, as the commentary states). Every *emitted* instance remains checked and warranted as SYZGY's node: the alias unpacks before emission.

### Lean statement

BernsteinLogic.Composition . SmallGainNStep

statement bound

```

/-- The n-ary payload vehicle: `SmallGainStep`'s pattern at arbitrary `kk`.
Every field is the checker's obligation for the rule; the collapse target is
one `Syzygy` node (`toSyzygy`). `lambdaPos`, `thetaLo/Hi`, and `colSum` are
rule-level well-formedness exactly as in the binary structure (soundness
uses only ` $\theta \leq \lambda_i$ ` inside the combination check; an emitted node whose
`foldEq` checks is sound however the producer bookkept `B` – the identity
`smallGainN_is_syzygy` is what guarantees the row shapes always discharge
`foldEq` under the column condition). -/
structure SmallGainNStep (H : Hypernet) (kk : ℕ) where
  seq : Sequent
  diss : Fin kk → Sequent
  mem : Fin kk → Sequent
  lambda : Fin kk → ℚ
  B : Fin kk → Fin kk → ℚ
  theta : ℚ
  e : Fin kk → Term
  m : Fin kk → Term
  concl : Term

  dissPost : ∀ i, (diss i).post = .atom .le (e i) (.const 0)
  memPost : ∀ i, (mem i).post = .atom .le (m i) (.const 0)

  eIdxFree : ∀ i, (e i).idxFree = true
  mIdxFree : ∀ i, (m i).idxFree = true
  conclIdxFree : concl.idxFree = true

  lambdaPos : ∀ i, 0 < lambda i
  thetaLo : 0 < theta
  thetaHi : theta < 1
  colSum : ∀ j, ((List.finRange kk).map (fun i => B i j)).sum = lambda j

  foldEq : (((List.finRange kk).map (fun i => ((lambda i : ℚ), e i))
    ++ (List.finRange kk).map (fun i => ((1 : ℚ), m i))).foldr
    (fun pr acc => Poly.add (Poly.scale pr.1 pr.2.toPoly) acc) [])
    = concl.toPoly
  postEq : seq.post = .atom .le concl (.const 0)

  preDiss : ∀ i, premiseContains seq.pre (diss i).pre
  preMem : ∀ i, premiseContains seq.pre (mem i).pre

  conclShape : ∃ s, concl.bshape H = some s
  conclFin : concl.finWires H
  wiresPos : ∀ w ∈ seq.pre.freeWires ++ concl.freeWires,
    ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape
  compPostWiresPos : ∀ c ∈ (List.finRange kk).map diss
    ++ (List.finRange kk).map mem,
    ∀ w ∈ c.post.freeWires,
    ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape

```

BernsteinLogic.Composition . **smallGainN\_is\_syzygy**

statement bound

```

/-- **The n-ary exact identity** (the producer warrant that `foldEq` is
always dischargeable from the row shapes): under the column condition, the
 $(\lambda_1, \dots, \lambda_{kk}, 1, \dots, 1)$ -weighted entry fold over the dissipation and
membership polynomials equals the conclusion polynomial
 $\sum_i \lambda_i \cdot LV_i + \theta \cdot \sum_j \lambda_j \cdot A_j$  – the cross-supplies cancel identically. Reduces
by canonical-polynomial extensionality to per-key rational arithmetic:
distribute, exchange the double sum, apply the column condition. -/
theorem smallGainN_is_syzygy (kk : ℕ) (LV A Sg : Fin kk → Poly)
  (lam : Fin kk → ℚ) (B : Fin kk → Fin kk → ℚ) (th : ℚ)
  (hcol : ∀ j, ((List.finRange kk).map (fun i => B i j)).sum = lam j) :
  (((List.finRange kk).map
    (fun i => (lam i, Poly.sub (Poly.add (LV i) (A i)) (Sg i)))
    ++ (List.finRange kk).map
    (fun i => ((1 : ℚ), smallGainNMemPoly kk A Sg lam B th i)))).foldr
  (fun pr acc => Poly.add (Poly.scale pr.1 pr.2) acc) [])

```

```

= Poly.add
  ((List.finRange kk).foldr
   (fun i acc => Poly.add (Poly.scale (lam i) (LV i)) acc) [])
  (Poly.scale th
   ((List.finRange kk).foldr
    (fun i acc => Poly.add (Poly.scale (lam i) (A i)) acc) [])))

```

BernsteinLogic.Composition . **sound\_smallGainN**

statement bound

```

/-- **Soundness of SmallGainN**, as the Syzygy corollary: the `2.kk`
component validities feed `sound_syzygy` on the collapsed instance. The
rational comparisons (`λi > 0`, `0 < θ < 1`) enter only through `λi ≥ 0`
inside the combination check; `θ`'s range and the column condition are
well-formedness, exactly as the catalogue commentary states. -/
theorem sound_smallGainN {H : Hypernet} {kk : ℕ} (st : SmallGainNStep H kk)
  (hdiss : ∀ i, (st.diss i).semanticallyValid H)
  (hmem : ∀ i, (st.mem i).semanticallyValid H) :
  st.seq.semanticallyValid H

```

BernsteinLogic.Composition . **smallGainN\_colSum\_mutant\_refuted**

statement bound

```

/-- **M-N2 (the column condition deleted): the identity is REFUTED** – by
the B-swap, computation-free: the mutant at `B = 0` and the PROVED identity
at `B = 1` (column sums check: `1 = λ`) share every other datum
(`kk = 1`, `LV = Sg = 0`, `A = const 1`, `λ = 1`, `θ = 0`), so their entry
folds would be equal; reading coefficients through the fold lemmas, the two
membership polynomials differ by exactly the decay budget's coefficient –
forcing `coeffSum (const 1) = 0` everywhere, against canonicity. The column
condition is what makes the membership rows deliver the stated conclusion
for every decay budget. -/
theorem smallGainN_colSum_mutant_refuted :
  ¬ (∀ (kk : ℕ) (LV A Sg : Fin kk → Poly) (lam : Fin kk → ℚ)
    (B : Fin kk → Fin kk → ℚ) (th : ℚ),
    (((List.finRange kk).map
     (fun i => (lam i, Poly.sub (Poly.add (LV i) (A i)) (Sg i)))
    ++ (List.finRange kk).map
     (fun i => ((1 : ℚ), smallGainNMemPoly kk A Sg lam B th i))).foldr
     (fun pr acc => Poly.add (Poly.scale pr.1 pr.2) acc) [])
    = Poly.add
     ((List.finRange kk).foldr
      (fun i acc => Poly.add (Poly.scale (lam i) (LV i)) acc) [])
     (Poly.scale th
      ((List.finRange kk).foldr
       (fun i acc => Poly.add (Poly.scale (lam i) (A i)) acc) [])))

```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **smallGainN\_drift**

statement bound

```

/-- **B3', the cancellation row at any n** (the semantic face of the
checker-level `(λ1, ..., λn, 1, ..., 1)` syzygy fold): per-component dissipation
rows with cross-supplies on the joint closed-loop kernel, under the n-ary
membership rows and the exact column condition, yield the `θ`-margin drift
row for `W = ∑ i, lam i * V i`. The cross-supplies cancel by construction:
`∑ i, lam i * sigma i ≤ (1-θ) ∑ i ∑ j, B i j * alpha j = (1-θ) ∑ j, lam j * alpha j`
(swap the double sum, then the column condition). -/
theorem smallGainN_drift (hlam : ∀ i, 0 ≤ lam i)
  (hcol : ∀ j, ∑ i, B i j = lam j)
  (hint : ∀ i, ∀ y, y ∉ A → Integrable (V i) (K y))
  (hd : ∀ i, ∀ y, y ∉ A →
    ∫ z, V i z ∂K y ≤ V i y - alpha i y + sigma i y)
  (hm : ∀ i, ∀ y, y ∉ A →
    lam i * sigma i y ≤ (1 - theta) * ∑ j, B i j * alpha j y) :
  ∀ y, y ∉ A → ∫ z, (∑ i, lam i * V i z) ∂K y
    ≤ (∑ i, lam i * V i y) - theta * ∑ j, lam j * alpha j y

```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **smallGain\_drift\_of\_N**

statement bound

```

/-- **Binary recovery, on the nose**: `smallGain_drift`'s exact statement
re-derived from `smallGainN_drift` at `m = 2`, `lam = ![1, λ]`,
`B = ![![0, λ], ![1, 0]]` (column sums `0 + 1 = 1` and `λ + 0 = λ`). One
bookkeeping disanalogy, inherited from the catalogue commentary: the n-ary
membership row at `i = 0` reads `1 * σ₁ ≤ (1-θ)·λ·α₂` — the binary `hm₂`
with its factors reassociated — and at `i = 1` reads `λ * σ₂ ≤ (1-θ)·α₁` =
the binary `hm₁`; the two rows arrive order-swapped vs the binary labels,
set-equal. -/
theorem smallGain_drift_of_N {V₁ V₂ alpha₁ alpha₂ sigma₁ sigma₂ : S → ℝ}
  {lam theta : ℝ} (hlam : 0 ≤ lam)
  (hint₁ : ∀ y, y ∉ A → Integrable V₁ (K y))
  (hint₂ : ∀ y, y ∉ A → Integrable V₂ (K y))
  (hd₁ : ∀ y, y ∉ A → ∫ z, V₁ z ∂K y ≤ V₁ y - alpha₁ y + sigma₁ y)
  (hd₂ : ∀ y, y ∉ A → ∫ z, V₂ z ∂K y ≤ V₂ y - alpha₂ y + sigma₂ y)
  (hm₁ : ∀ y, y ∉ A → lam * sigma₂ y ≤ (1 - theta) * alpha₁ y)
  (hm₂ : ∀ y, y ∉ A → sigma₁ y ≤ lam * ((1 - theta) * alpha₂ y)) :
  ∀ y, y ∉ A → ∫ z, (V₁ z + lam * V₂ z) ∂K y
    ≤ (V₁ y + lam * V₂ y) - theta * (alpha₁ y + lam * alpha₂ y)

```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete. **avoid\_smallGainN**

statement bound

```

/-- **B3', assembled**: on the closed loop, the n-ary small-gain data yields
the composite avoid bound for the weighted-sum certificate (the discrete face
of the catalogue rule end-to-end; Foster–Lyapunov metatheory through
`Battery.measure_reach_le`, exactly as the binary `avoid_smallGain`). -/
theorem avoid_smallGainN (hlam : ∀ i, 0 ≤ lam i) (htheta : 0 ≤ theta)
  {C : Fin m → ℝ} (hA : MeasurableSet A)
  (hcol : ∀ j, ∑ i, B i j = lam j)
  (hV : ∀ i, Measurable (V i))
  (hV0 : ∀ i y, 0 ≤ V i y) (hVC : ∀ i y, V i y ≤ C i)
  (hbar : ∀ y ∈ A, 1 ≤ ∑ i, lam i * V i y)
  (halpha : ∀ i, ∀ y, y ∉ A → 0 ≤ alpha i y)
  (hd : ∀ i, ∀ y, y ∉ A →
    ∫ z, V i z ∂K y ≤ V i y - alpha i y + sigma i y)
  (hm : ∀ i, ∀ y, y ∉ A →
    lam i * sigma i y ≤ (1 - theta) * ∑ j, B i j * alpha j y) (x : S) :
  pathMeasure K x {ω | ∃ n, ω n ∈ A}
    ≤ ENNReal.ofReal (∑ i, lam i * V i x)

```

## Correspondence (Steele display ↔ Lean)

### NO CORRESPONDENCE DICTIONARY YET

no reviewer-certified dictionary maps the displayed rule onto the mechanized statements above; until one is written and certified, that mapping rests on this section's editorial care alone.

## Lean verifier receipts

module BernsteinLogic.Composition; bound build 9116f454beb0... (bound from receipt 548ea3dda8ed73a2, 2026-08-03T21:30:29Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 548ea3dda8ed73a2 | 2026-08-03T21:30:29Z | build+decorrelated | current |

10 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.QtslCompose.AGComposeDiscrete; bound build d5233632f95a... (bound from receipt 88ff74180adf5b95, 2026-08-03T20:13:53Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).



| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 88ff74180adf5b95 | 2026-08-03T20:13:53Z | build+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

|                                                           |            |
|-----------------------------------------------------------|------------|
| build (lake build (elaboration))                          | current    |
| replay (leanchecker (official kernel replay))             | stale only |
| hardened (safe_verify (hardened screen))                  | stale only |
| decorrelated (lean4export + nanoda (decorrelated kernel)) | current    |

**comparator:** **structurally n/a** the alias unpacks *before* emission (no rule tag; producers elaborate a SMALLGAINN step to its SYZYGY node), so no wire certificate — and hence no comparator fixture — can ever contain a SMALLGAINN node. Comparator coverage for this rule is SYZYGY's.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*, not by backlog, and for exactly the binary rule's reason: the wire format never carries a SMALLGAINN node — producers elaborate to the  $(\lambda_1, \dots, \lambda_n, 1, \dots, 1)$  SYZYGY node before emission — and the HOL4 development's object is the wire checker. The HOL statement that verifies a composed  $n$ -component small-gain certificate is therefore SYZYGY's (syzygyCheck).

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template. The five artefacts an alias owes the book: the  $1 \rightarrow 1$  unpack, the filled-in application of the target rule, the payload-transport table, the exact algebraic identity that makes the respelling correct, and the soundness corollary through the target rule's lemma. All five follow; the unpack and transport table are stated here as the producer's contract, their dedicated mechanized vehicle being the one artefact not yet in the Lean tree (stated loudly below and in the section's Lean cell) — soundness of every emitted node meanwhile rests on the target rule's already-mechanized lemma, because the alias unpacks before emission.

**Recipe R35.1** (the unpack — one node, any  $n$ ). A SMALLGAINN step carries  $2n$  component proofs: the  $n$  dissipation sequents (each over its own box  $\Phi_i$ ) and the  $n$  membership sequents (over the declared operating box  $\Phi_w$ ), with side data  $\lambda \in \mathbb{Q}_{>0}^n$ ,  $B \in \mathbb{Q}_{\geq 0}^{n \times n}$  with  $\sum_i B_{ij} = \lambda_j$  for every  $j$ , and  $\theta \in \mathbb{Q} \cap (0, 1)$ . Define the atom terms

$$e_i \equiv \mathcal{L}_i V_i + \alpha_i - \sigma_i, \quad m_i \equiv \lambda_i \sigma_i - (1-\theta) \sum_j B_{ij} \alpha_j,$$

and the substitution into SYZYGY's metavariables:

$$N := 2n, \quad (e_1, \dots, e_N) := (e_1, \dots, e_n, m_1, \dots, m_n), \quad (\Phi_1, \dots, \Phi_N) := (\Phi_1, \dots, \Phi_n, \underbrace{\Phi_w, \dots, \Phi_w}_n),$$

$$(\lambda_1, \dots, \lambda_N) := (\lambda_1, \dots, \lambda_n, 1, \dots, 1), \quad \{g_k\}_k := \emptyset, \quad c := \mathcal{L}(\sum_i \lambda_i V_i) + \theta \sum_j \lambda_j \alpha_j.$$

The repayment matrix  $B$  is consumed here and does not appear in the image: it is the producer's bookkeeping for *finding* membership rows that make the identity of Proposition R35.3 true, and the identity is the only trace of it the checker ever replays.

**Recipe R35.2** (the filled-in application). Under the substitution of Recipe R35.1 the SMALLGAINN node is the following SYZYGY application:

$$\frac{(\lambda_1, \dots, \lambda_n, 1, \dots, 1) \in \mathbb{Q}_{\geq 0}^{2n} \quad \{ \Phi_i \vdash e_i \leq 0 \}_{i=1}^n \quad \{ \Phi_w \vdash m_i \leq 0 \}_{i=1}^n}{c \equiv \sum_i \lambda_i e_i + \sum_i 1 \cdot m_i \text{ (exact canonical polynomial)}}_{\Phi \vdash c \leq 0} \text{SYZYGY}$$

with  $\Phi \supseteq \Phi_1, \dots, \Phi_n, \Phi_w$  conjunct-wise (the  $2n$ -way containment split). Discharging SYZYGY's obligations at this instance: the nonnegativity checks are  $\lambda_i \geq 0$  (from  $\lambda_i > 0$  — the only use soundness makes of the signs) and  $1 \geq 0$ ,  $n$  times each; the sourcing check is immediate (each of the  $2n$  entries is the corresponding component's posted atom, verbatim); the exact-identity check is Proposition R35.3; the index-freeness gates discharge by recomputation on the  $2n + 1$  terms; and the shape, positivity, and zero-normal postForm obligations transport exactly as in the binary section. The  $\nu$ -row is empty for the same reason as there: the cross-supply cancellation is performed by the membership atoms themselves, so no equality sub-conjuncts of  $\Phi$  enter.



**Proposition R35.3** (the conclusion identity). *Under the column condition  $\sum_i B_{ij} = \lambda_j$  (all  $j$ ), as an identity of exact canonical polynomials,*

$$\sum_i \lambda_i e_i + \sum_i m_i = \mathcal{L}(\sum_i \lambda_i V_i) + \theta \sum_j \lambda_j \alpha_j.$$

*Proof.* Expand and collect — still two lines at any  $n$ , which is the point:

$$\begin{aligned} \sum_i \lambda_i e_i + \sum_i m_i &= \sum_i \lambda_i \mathcal{L}_i V_i + \sum_i \lambda_i \alpha_i - \underbrace{\sum_i \lambda_i \sigma_i + \sum_i \lambda_i \sigma_i}_{= 0 \text{ (cross-supplies cancel)}} - (1-\theta) \sum_j \left( \sum_i B_{ij} \right) \alpha_j \\ &= \mathcal{L}(\sum_i \lambda_i V_i) + \sum_j (1 - (1-\theta)) \lambda_j \alpha_j = \mathcal{L}(\sum_i \lambda_i V_i) + \theta \sum_j \lambda_j \alpha_j, \end{aligned}$$

where the second line swaps the summation order in the  $B$ -term and applies the column condition  $\sum_i B_{ij} = \lambda_j$  — the one place it is used, and exactly the discount that makes the neighbour draws rejoin the decay budgets at weight  $\lambda_j$ . As in the binary section, reading  $\mathcal{L}(\sum_i \lambda_i V_i)$  for  $\sum_i \lambda_i \mathcal{L}_i V_i$  is a definition, not a step: over disjoint variable sets the composite generator acts coordinate-wise. The checker replays this as SYZYGY's exact canonical-polynomial identity and never sees a fixpoint, a spectral radius, or the interconnection graph — the  $n$ -component small-gain argument is dissolved into ring arithmetic at certificate-checking time.  $\square$

**Proposition R35.4** (binary recovery, on the nose). *At  $n = 2$  with  $\lambda = (1, \lambda)$ ,  $B_{21} = 1$ ,  $B_{12} = \lambda$  (and  $B_{11} = B_{22} = 0$ ), the column condition holds, the membership rows are exactly the binary section's  $\{\lambda \sigma_2 - (1-\theta)\alpha_1, \sigma_1 - \lambda(1-\theta)\alpha_2\}$ , the conclusion is exactly  $\mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2) \leq 0$ , and the SYZYGY weights are  $(1, \lambda, 1, 1)$  — i.e. SMALLGAIN is the  $n = 2$  instance of this rule.*

*Proof.* Column sums:  $B_{11} + B_{21} = 1 = \lambda_1$  and  $B_{12} + B_{22} = \lambda = \lambda_2$ . Membership rows:  $m_1 = \sigma_1 - \lambda(1-\theta)\alpha_2$  and  $m_2 = \lambda \sigma_2 - (1-\theta)\alpha_1$  — the binary section's rows as a *set*, arriving in the opposite order (its  $m_1$  is this rule's  $m_2$ ): a bookkeeping difference in combination order, visible in the node bytes and irrelevant to validity, since SYZYGY's identity is a sum. Conclusion and weights read off directly.  $\square$

**Recipe R35.5** (payload transport — the producer's contract). The  $n$ -ary payload transports into SyzygyStep exactly as the binary payload does, with the four fixed slots generalized to  $n$ -vectors:

| SMALLGAINN step (contract)                                                            | SyzygyStep                                        |
|---------------------------------------------------------------------------------------|---------------------------------------------------|
| seq                                                                                   | seq                                               |
| diss <sub>1</sub> , ..., diss <sub>n</sub> , mem <sub>1</sub> , ..., mem <sub>n</sub> | components (that order = combination order)       |
| weights $(\lambda_1, \dots, \lambda_n, 1, \dots, 1)$                                  | combination                                       |
| concl                                                                                 | conclusion                                        |
| 2n-way containment split                                                              | preContains                                       |
| shape/positivity/idx-freeness packs $(\lambda, B, \theta)$                            | verbatim, as in the binary table                  |
| postEq                                                                                | consumed by the unpack; $B$ leaves no image field |
|                                                                                       | postForm = the zero-normal reading                |

This table is the contract of the dedicated mechanized vehicle — the  $n$ -ary payload structure with its verified unpack and soundness corollary, the binary section's SmallGainStep apparatus at arbitrary  $n$  — which is not yet in the Lean tree (the section's Lean cell states the absence). Nothing checked on the wire waits on it: the unpack happens producer-side, and the emitted node is SYZYGY's.

**Proposition R35.6** (soundness corollary). *If the  $2n$  component sequents are semantically valid, the composite sequent  $\Phi \vdash \mathcal{L}(\sum_i \lambda_i V_i) + \theta \sum_j \lambda_j \alpha_j \leq 0$  is semantically valid.*

*Proof.* The target rule's kernel lemma applied to the unpacked instance of Recipe R35.2, feeding the  $2n$  validities through the containment split; the exact-identity obligation is Proposition R35.3. Beyond that routing there is no further argument — the alias genre working as intended, at every  $n$  at once.  $\square$

**Remark R35.7** (what stays outside the checker).  $(\lambda, B, \theta)$ -existence is the producer's problem, and the classical spectral conditions are its solution theory: when the gain matrix  $\Gamma$  has spectral radius  $\rho(\Gamma) < 1 - \theta$ , the Neumann-series vector  $\lambda := \sum_k (1 - \theta)^{-k} (\Gamma^\top)^k \mathbb{1}$  converges and witnesses  $\Gamma^\top \lambda \leq (1 - \theta)\lambda$ , from which a feasible  $B$  is assembled column by column (under-allocation is repaired by diagonal padding, which only weakens the membership row it feeds). The  $n$ -component ISS small-gain theorem for general networks is Dashkovskiy–Rüffer–Wirth [59]; where that tradition closes a monotone-operator fixpoint argument over the interconnection graph, the certificate form closes the ring identity of Proposition R35.3, and the graph combinatorics survive only as the producer's search space. The signed polynomial contract discipline is the binary section's, unchanged.

| proof step             | formal anchor                                                                                                      |
|------------------------|--------------------------------------------------------------------------------------------------------------------|
| Recipe R35.1 unpack    | producer-side elaboration; dedicated mechanized vehicle = named successor (binary vehicle: SmallGainStep.toSyzygy) |
| Proposition R35.3      | combValid's fold identity, replayed as SYZYGY's (*) on the emitted node                                            |
| Proposition R35.4      | the binary section's SmallGainStep apparatus at $n = 2$ (weights $(1, \lambda, 1, 1)$ )                            |
| Proposition R35.6      | sound_syzygy on the unpacked instance ( $n$ -ary corollary = named successor)                                      |
| no wire tag, by design | RuleTag comment block (Core/Catalogue.lean): producers elaborate before emission                                   |

## Rule 36. PRODUCTLATTICE (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `rules.toml` — *ProductLattice*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{B_1 \text{ a certified attracting-block lattice for } \varphi_1 \text{ over } H_1 \quad B_2 \text{ likewise for } \varphi_2 \text{ over } H_2 \quad \text{wires}(H_1) \cap \text{wires}(H_2) = \emptyset}{\{b_1 \times b_2 \mid b_1 \in B_1, b_2 \in B_2\} \text{ (the meet-generators) certified for } \varphi_1 \times \varphi_2 \text{ over } H_1 \sqcup H_2} \text{PRODUCTLATTICE}$$

the displayed family is the list of pairwise-product meet-generators.

the full lattice  $\langle B_1 \times B_2 \rangle_\vee$  (join-closure of the pairwise products) is the paper-strength reading — the bound theorem quantifies over the meet-generators, and the finite-union/boundary-decomposition step remains paper-strength, exactly as the correspondence concedes. Every composite facet sequent is discharged BY CITATION: a composite facet is normal to one factor's axis, its inwardness polynomial IS that factor's facet polynomial with the other factor's coordinates riding along as free box variables (the rider, conjoined onto the premise). Bernstein-wise the composite facet grid is the factor grid tensored with all-ones — a coefficient identity; nothing for a checker to recompute beyond the ordinary BERNCONC recompute pattern. Declared in the baked base theory itself (its first certificate-family rule), not in any separate composition layer.

PRODUCTLATTICE composes at the certificate-*family* level, one judgment up from the sequent rules: given certified attracting-block lattices for two systems over wire-disjoint hypernets, it certifies the join-closure of pairwise products as an attracting-block lattice for the product system over the disjoint union. The geometric fact that powers it: every internal facet of an axis-aligned product block is normal to one factor's axis, so its inwardness polynomial *is* that factor's facet polynomial, with the other factor's coordinates riding along as free box variables. Each composite facet sequent is therefore discharged *by citation*: conclusion kept verbatim, the other factor's box conjoined onto the premise as a rider.

What the checker recomputes: no new Bernstein arithmetic — Bernstein-wise the composite facet grid is the factor grid tensored with all-ones, a coefficient identity, so the per-facet discharge is syntactic identity of the cited conclusion. What remains checked is the citation plumbing itself: wire-disjointness of the two hypernets, factor scopedness, and the rider conjunction on each cited premise.

Wire-disjointness is load-bearing twice. Structurally, the disjoint union of the two hypernets is only well-formed without shared wires (linearity and acyclicity of the union are proven *from* disjointness). Semantically, no single transported facet sequent's validity fails under a shared wire — the weaken lemma conjoins any rider — but the assembled object stops being a product of the factors: the identification of every composite facet with a cited factor facet, the other factor entering only as a rider over fresh coordinates, is proven from disjointness, and with a shared wire the recomputed readout misdescribes the composite. A single shared wire is precisely an interface, and composition along an interface is PROMISE/small-gain territory, not this rule's.

**notes** The soundness statement is dynamics-free — sound for flows and maps alike, because it certifies a *lower bound* family (“these blocks are certified”), never “these are all the attractors”. The completeness reading — that the generated lattice is the whole attractor lattice of the product — is a flows-only theorem of the source literature and lives paper-side; the proof section exhibits the map-dynamics counterexample that forces the fence. No wire tag is allocated yet: soundness lands first, executable checking follows the family-object wire-format work, and the emit path refuses the rule until then.

### Lean statement

**What “discharge by citation” means, stated in the flavor text and bound by the declarations below.** `citeFacet` keeps the cited factor facet's conclusion verbatim and conjoins the rider onto the premise; `prodBlock` assembles a product block's facet list as exactly the two citation families; `prodBlocks` enumerates only the meet-generators of the join-closure — the exponentially many join blocks are never materialized; their sub-facets discharge through the paper-strength boundary decomposition (which selects a factor facet and rider), whose final transport-and-weaken step is the same citation lemmas with smaller riders. A skeptical reader should confirm by eye that no definition below evaluates dynamics anywhere: the rule's semantic content is entirely the transport-and-weaken pair quoted above.

BernsteinLogic.ProductLattice . BlockCert

statement bound

```
-- One certified attracting block, certificate view (the recon's Route-2
shape): its region — the compact axis-aligned box(-union), as the formula
the producer wrote it with — and its internal-facet inwardness sequents
(strict inwardness via non-strict predicates with explicit positive
rational margins; block-hood is elementary for strictly inward box unions,
no combinatorial index data claimed — note §1.1). -/
```

```

structure BlockCert where
  region : Formula
  facets : List Sequent

```

#### BernsteinLogic.ProductLattice . CertifiedBlockLattice

statement bound

```

/-- A certified attracting-block lattice over a hypernet – the catalogue's
first certificate-FAMILY object (premise object of ProductLattice).
The family carries its lattice structure explicitly at the index level:
`join`/`meet` with the six standard lattice laws (idempotence is
derivable). These are the certificate-level shadow of the note §1.1
closure ("closed under u and regularized n"); that a join index denotes
the set-union of the denoted blocks is producer-side geometry in v1 –
kernel teeth for the denotation arrive with the family-object wire format
(C-4/format cycle). Wire confinement makes the family transportable into
any extension (§3) – it is what `sound_productLattice` consumes. -/
structure CertifiedBlockLattice (H : Hypernet) where
  blocks : List BlockCert
  join : Fin blocks.length → Fin blocks.length → Fin blocks.length
  meet : Fin blocks.length → Fin blocks.length → Fin blocks.length
  join_comm : ∀ i j, join i j = join j i
  join_assoc : ∀ i j k, join (join i j) k = join i (join j k)
  meet_comm : ∀ i j, meet i j = meet j i
  meet_assoc : ∀ i j k, meet (meet i j) k = meet i (meet j k)
  absorb_join : ∀ i j, join i (meet i j) = i
  absorb_meet : ∀ i j, meet i (join i j) = i
  regionConfined : ∀ b ∈ blocks, ∀ w ∈ b.region.freeWires, (H.wireType w).isSome
  facetsScoped : ∀ b ∈ blocks, ∀ s ∈ b.facets, s.wellScoped H

```

#### BernsteinLogic.ProductLattice . PProductLatticeData

statement bound

```

/-- Applicability data of the ProductLattice rule – the two factor
families and the side condition. Field-for-field what a checker would
verify: the factor certificates (whose facet-sequent validity enters
`sound_productLattice` as hypotheses, Step-style) and wire-disjointness.
No first-class `RuleTag`/wire constructor in v1 (note §1.4: emission-layer
posture; BINARY.md §6.2 reserves tag 27 for the family-object format
cycle); correspondingly NOT in `v0frag` – like the ReLU family
pre-enactment, soundness lands first and executable checking follows the
format work. -/
structure PProductLatticeData (H1 H2 : Hypernet) where
  disjoint : H1.WireDisjoint H2
  left : CertifiedBlockLattice H1
  right : CertifiedBlockLattice H2

```

#### BernsteinLogic.ProductLattice . Sequent.semanticallyValid\_extend

statement bound

```

/-- TRANSPORT: semantic validity is monotone under hypernet extension –
a sequent valid over H, well-scoped in H, is valid over any H' that
extends H's wire typing and contains H's applications. (For ProductLattice
this is "the factor's certified sign is the composite's inwardness": the
facet polynomial reads only the factor's wires, so the other factor's
coordinates ride along invisibly.) -/
theorem Sequent.semanticallyValid_extend {H H' : Hypernet} {seq : Sequent}
  (hext : H.WireExtends H')
  (happs : ∀ app ∈ H.applications, app ∈ H'.applications)
  (hws : seq.wellScoped H) (hval : seq.semanticallyValid H) :
  seq.semanticallyValid H'

```

#### BernsteinLogic.ProductLattice . Sequent.semanticallyValid\_weaken

statement bound

```

/-- WEAKEN, semantic and standalone: any conjunct may join a valid
sequent's premise (PWeaken's semantic content – the rider needs no
side condition). -/
theorem Sequent.semanticallyValid_weaken {H : Hypernet}
  {pre extra post : Formula}

```

```
(h : Sequent.semanticallyValid H (pre, post)) :
  Sequent.semanticallyValid H (.and pre extra, post)
```

BernsteinLogic.ProductLattice . **sound\_productLattice\_citeLeft**

statement bound

```
/-- The standing citation discharge, left factor: ANY sequent citing a
valid, well-scoped left-factor facet – conclusion verbatim, premise
= factor premise  $\wedge$  an arbitrary rider – is valid over the union.
Soundness is transport (§3) then weaken: the note §1.2's one-line
argument, in two lines. A join block's sub-facet, once the paper-strength
boundary decomposition has selected its factor facet and rider, is
discharged by this lemma (rider = the sub-facet's free-coordinate cell);
the decomposition itself is not mechanized here. -/
theorem sound_productLattice_citeLeft {H1 H2 : Hypernet}
  (hd : H1.WireDisjoint H2) {s0 : Sequent} (rider : Formula)
  (hws : s0.wellScoped H1) (hval : s0.semanticallyValid H1) :
  (citeFacet s0 rider).semanticallyValid (H1.disjUnion H2 hd)
```

BernsteinLogic.ProductLattice . **sound\_productLattice\_citeRight**

statement bound

```
/-- The standing citation discharge, right factor. -/
theorem sound_productLattice_citeRight {H1 H2 : Hypernet}
  (hd : H1.WireDisjoint H2) {s0 : Sequent} (rider : Formula)
  (hws : s0.wellScoped H2) (hval : s0.semanticallyValid H2) :
  (citeFacet s0 rider).semanticallyValid (H1.disjUnion H2 hd)
```

BernsteinLogic.ProductLattice . **sound\_productLattice**

statement bound

```
/-- SOUNDNESS of ProductLattice (note §1.2, written properly): given two
certified block lattices over wire-disjoint hypernets, every facet
sequent of every product block in the conclusion family is semantically
valid over the disjoint union. Each facet is a factor facet discharged by
citation; validity = transport of the factor's validity along the union
injection (§3), then weakening in the rider. Dynamics-free – sound for
flows AND maps; the flows-only fence (header) concerns completeness,
which this statement deliberately does not express: the conclusion is a
certified LOWER BOUND family, and "the whole attractor lattice" is a
paper-side, flows-only reading (KW `cor_flows_att_iso`). -/
theorem sound_productLattice {H1 H2 : Hypernet}
  (d : PProductLatticeData H1 H2)
  (hL : d.left.Valid) (hR : d.right.Valid) :
   $\forall b \in d.prodBlocks, \forall s \in b.facets,$ 
  s.semanticallyValid (H1.disjUnion H2 d.disjoint)
```

BernsteinLogic.ProductLattice . **Hypernet.disjUnion**

statement bound

```
/-- The disjoint union of two wire-disjoint hypernets: union typing,
concatenated applications; the three well-formedness predicates are
proven from the factors' plus disjointness. This is the composite system
 $\phi_1 \times \phi_2$ 's hypernet,  $H_1 \sqcup H_2$  (note §1.1). -/
def Hypernet.disjUnion (H1 H2 : Hypernet) (hd : H1.WireDisjoint H2) : Hypernet
```

BernsteinLogic.ProductLattice . **Hypernet.Valuation.restrict**

statement bound

```
/-- Restrict a valuation of an extending hypernet to the extended one. -/
def Hypernet.Valuation.restrict {H H' : Hypernet} ( $\sigma$  : H'.Valuation)
  (hext : H.WireExtends H') : H.Valuation
```

BernsteinLogic.ProductLattice . **Formula.holds\_restrict**

statement bound

```
/-- Formulas scoped to the extended hypernet's wires hold identically
under restriction. -/
theorem Formula.holds_restrict {H H' : Hypernet} ( $\sigma$  : H'.Valuation)
  (hext : H.WireExtends H') :
   $\forall (F : Formula), (\forall w \in F.freeWires, (H.wireType w).isSome) \rightarrow$ 
  (Formula.holds ( $\sigma.restrict$  hext) F  $\leftrightarrow$  Formula.holds  $\sigma$  F)
```

BernsteinLogic.ProductLattice . **PrimApp.semOk\_restrict**

statement bound

```

/-- Per-application semantics survive restriction: every entry SemOk reads
is at a port wire, typed in the extended hypernet by type consistency. -/
theorem PrimApp.semOk_restrict {H H' : Hypernet} (σ : H'.Valuation)
  (hext : H.WireExtends H') {app : PrimApp}
  (hTC : app.TypeConsistent H.wireType) (h : app.SemOk σ) :
  app.SemOk (σ.restrict hext)

```

BernsteinLogic.ProductLattice . **prodBlocks\_confined**

statement bound

```

/-- The conclusion family is itself wire-confined over the union – the
confinement half of what a NESTED application (product of products,
AG-inside/product-outside, note §2) consumes from its premise object. -/
theorem prodBlocks_confined {H1 H2 : Hypernet}
  (d : PProductLatticeData H1 H2) :
  ∀ b ∈ d.prodBlocks,
    (∀ w ∈ b.region.freeWires,
      ((H1.disjUnion H2 d.disjoint).wireType w).isSome) ∧
    ∀ s ∈ b.facets, s.wellScoped (H1.disjUnion H2 d.disjoint)

```

BernsteinLogic.ProductLattice . **Hypernet.WireDisjoint**

definition bound

```

/-- No wire is typed in both hypernets – ProductLattice's side condition
(note §1.1: wires(H1) ∩ wires(H2) = ∅). A single shared wire demotes the
composition to the interface axis (Promise/SmallGain – note §2). -/
def Hypernet.WireDisjoint (H1 H2 : Hypernet) : Prop :=
  ∀ w : Wire, (H1.wireType w).isSome → (H2.wireType w).isSome → False

```

BernsteinLogic.ProductLattice . **CertifiedBlockLattice.Valid**

definition bound

```

/-- The family is certified when every block's every facet sequent is
semantically valid. Validity enters `sound_productLattice` as a
hypothesis – the Step pattern: sub-proof warrants are supplied at the
use site (by `sound_v0` on the factor certificates, or manually). -/
def CertifiedBlockLattice.Valid {H : Hypernet}
  (L : CertifiedBlockLattice H) : Prop :=
  ∀ b ∈ L.blocks, ∀ s ∈ b.facets, s.semanticallyValid H

```

BernsteinLogic.ProductLattice . **citeFacet**

definition bound

```

/-- Discharge BY CITATION (note §1.1): the composite facet sequent keeps
the cited factor facet's conclusion VERBATIM – polynomial and predicate
unchanged; the Bernstein grid of the composite facet is the factor's grid
replicated along the free coordinates, a coefficient identity, so
syntactic identity of the conclusion is the whole discharge – and
conjoins the rider (the other factor's box, riding along as free box
variables) onto the premise. -/
def citeFacet (s : Sequent) (rider : Formula) : Sequent :=
  (.and s.pre rider, s.post)

```

BernsteinLogic.ProductLattice . **prodBlock**

definition bound

```

/-- The product of two blocks: region = the conjunction of boxes; facets =
each factor's facets, cited with the other factor's region as rider.
(Every internal facet of an axis-aligned product is normal to one
factor's axis – note §1.2.) -/
def prodBlock (b1 b2 : BlockCert) : BlockCert :=
  (.and b1.region b2.region,
    b1.facets.map (fun s => citeFacet s b2.region)
    ++ b2.facets.map (fun s => citeFacet s b1.region))

```

BernsteinLogic.ProductLattice . **PProductLatticeData.prodBlocks**

definition bound

```

/-- The conclusion family's meet-generators: all pairwise products.
(B1 × B2)v = the join-closure of THIS list (KW Prop)

```

```
`prop:Lwedgevee-generated`); the joins — exponentially many — are never
enumerated: their sub-facets discharge via the paper-strength boundary
decomposition (unmechanized — it selects the factor facet and rider),
whose final step is `sound_productLattice_citeLeft/Right` below with
smaller riders. -/
def PProductLatticeData.prodBlocks {H1 H2 : Hypernet}
  (d : PProductLatticeData H1 H2) : List BlockCert :=
  d.left.blocks.flatMap fun b1 => d.right.blocks.map fun b2 => prodBlock b1 b2
```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The one rule of this book whose judgment sits a layer up: the conclusion is not a sequent but a certified *family*, so the  $Ax$  frame applies one level in — per facet sequent.

- $B_1, B_2 \mapsto d.\text{left}, d.\text{right} : \text{CertifiedBlockLattice over } H_1, H_2$ ; “certified for  $\varphi_i$ ”  $\mapsto hL, hR : \text{Valid}$  — every block's every facet sequent semantically valid. Note what does *not* appear: any semiflow. The certified statement is dynamics-free (the note above) — the display's  $\varphi_1 \times \varphi_2$  is the intended reading of the facet sequents, not a consumed hypothesis.
- $\text{wires}(H_1) \cap \text{wires}(H_2) = \emptyset \mapsto d.\text{disjoint} : H_1.\text{WireDisjoint } H_2$ , and the composite carrier  $H_1 \sqcup H_2 \mapsto \text{Hypernet}.\text{disjUnion}$ .
- the displayed conclusion (the meet-generator family certified)  $\mapsto \text{sound\_productLattice}$ , which discharges every facet of every *pairwise product* block ( $\text{prodBlocks}$  — the join-closure's meet-generators) — the displayed family and the bound quantification now coincide. The full join-closure  $\langle B_1 \times B_2 \rangle_\vee$  is the flavor text's paper-strength reading: the joins, never enumerated, discharge facet-by-facet through the standing citation lemmas  $\text{sound\_productLattice\_citeLeft/Right}$  (certified alongside, at *arbitrary* rider), with the boundary decomposition itself paper-strength — no separate join statement is bound.
- with no displayed row: well-scopedness — the factor lattices' facets and regions are confined to their own hypernets ( $\text{facetsScoped}/\text{regionConfined}$  fields of  $\text{CertifiedBlockLattice}$ ), which is what makes transport along the union injection sound; the undisplayed-pack class of the  $\text{PsdCERT}$  evaluability rows.

Width note: the citation lemmas accept *any* rider formula where the rule's conclusion only ever conjoins the other factor's region — wider machinery in the safe direction, and exactly the generality the join-block sub-facets consume.

## Lean verifier receipts

module BernsteinLogic.ProductLattice; bound build 84b7018f79da... (bound from receipt 9ebc3d9af324d1f7, 2026-07-21T04:48:38Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 224af3a8595b550f | 2026-07-21T01:49:09Z | build+decorrelated                 | current |
| 7187ff3aa085cd58 | 2026-07-21T01:58:41Z | build+decorrelated                 | current |
| 9ebc3d9af324d1f7 | 2026-07-21T04:48:38Z | build+decorrelated                 | current |
| fd5ec354b89369c0 | 2026-07-21T05:28:29Z | build+replay+hardened+decorrelated | current |

7 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement



**NO HOL STATEMENT YET**

The certificate-object stratum of this rule now has HOL twins, with kernel-evaluated transcription receipts, across five ported theories: the certified-block record and the citation constructor, the product-block assembly, the certified block lattice with its wellformedness and validity predicates, the applicability payload record with its wellformedness predicate, the meet-generator enumeration, and the valuation-restriction lemmas the transport argument reads (verified against the scripts this writing; hol-strict clean at the same 2026-07-22 run as the ported entries). What has no HOL twin yet is exactly the rule's semantic content — the transport-and-weaken citation lemmas and the soundness theorem they assemble into (the commentary's own point: nothing in the certificate stratum evaluates dynamics anywhere, so the semantic layer IS the rule). The rule also has no arm in the ported proof-object family and no wire tag on either side. Absent until the citation/soundness stratum ports; the certificate-object groundwork above is in place for that seating.

**HOL verifier receipts****NONE**

no HOL verification receipt exists for this rule.

**Proof**

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe, at the certificate-family layer: one lattice node unpacks into a family of cited facet sequents. Like PCONSTEXACT, part of the discharge is by reduction to verified machinery rather than to catalogue nodes — the transport step is a semantic monotonicity lemma with no catalogue counterpart, because hypernet extension is invisible to the sequent calculus — and the recipe says so.

**Setting.** A *certified attracting-block lattice* over a hypernet  $H$  (the `CertifiedBlockLattice` structure quoted above) is a finite family of blocks — each a region box formula plus its internal-facet inwardness sequents — carrying explicit index-level lattice structure and wire confinement; the family is *valid* when every facet sequent is semantically valid over  $H$ . The rule's premises: valid families  $B_1$  over  $H_1$  and  $B_2$  over  $H_2$ , and wire-disjointness  $\text{wires}(H_1) \cap \text{wires}(H_2) = \emptyset$ . Conclusion: the join-closure  $\langle B_1 \times B_2 \rangle_\vee$  is certified over the disjoint union  $H_1 \sqcup H_2$ .

**Recipe R36.1** (the `PRODUCTLATTICE` instance — one node, a citation family). The rewrite replaces the lattice node by one derivation fragment *per facet of per product block*, each of the same two-move shape:

$$\frac{s_0 \triangleright \text{a factor facet sequent (cited sub-proof)}}{\langle \text{pre}(s_0) \wedge \text{rider}, \text{post}(s_0) \rangle \text{ over } H_1 \sqcup H_2} \text{CITE}$$

where the rider is the *other* factor's box. The conclusion is kept verbatim — the polynomial and predicate of the cited facet, unchanged — because a composite facet of an axis-aligned product is normal to one factor's axis: its inwardness polynomial *is* that factor's, the other factor's coordinates riding along as free box variables. Only the meet-generators  $\{b_1 \times b_2\}$  are enumerated ( $O(|B_1| \cdot |B_2|)$  blocks,  $O(|B_1| + |B_2|)$  distinct cited factor sub-proofs, each reused across its products with varying riders); the join-closure's remaining elements — exponentially many — are never materialized. They discharge by the same mechanism: a boundary point of a finite union of axis-aligned blocks lies on the boundary of some member containing it (member interiors are interior to the union), so a union block's boundary decomposes into sub-facets, each a subset of some generator's facet and hence again normal to one factor's axis; inwardness for the member implies inwardness for the union (monotonicity under  $\sqsubseteq$ ), so the sub-facet's sequent is the same factor citation with its premise further constrained to the exposed portion — and the citation lemmas accept these further-constrained premises because they accept an *arbitrary* rider.

**Lemma R36.2** (transport: validity is monotone under hypernet extension). *Let  $H'$  extend  $H$ 's wire typing and contain  $H$ 's applications, and let  $s$  be semantically valid over  $H$  and well-scoped in  $H$ . Then  $s$  is semantically valid over  $H'$ .*

*Proof.* An admissible valuation  $\sigma$  of  $H'$  restricts to a valuation of  $H$  (read each  $H$ -typed wire through the extension), and the restriction is admissible: every entry a per-application semantic predicate reads is at a port wire, typed in  $H$  by type consistency, where the restriction agrees with  $\sigma$  by construction. Formulas whose free wires are  $H$ -typed therefore evaluate identically under  $\sigma$  and its restriction — by induction on terms and formulas, the only base case being a wire read, which is where well-scopedness is consumed. So from  $\sigma \models \text{pre}(s)$ : the restriction satisfies  $\text{pre}(s)$ , validity over  $H$  gives the restriction satisfies  $\text{post}(s)$ , and evaluation-agreement carries it back to  $\sigma$ . This is `Sequent.semanticallyValid_extend`; the well-scopedness hypothesis is not decoration — a sequent reading a wire *untyped* in  $H$  evaluates that read differently in an extension that types it, and the induction's base case fails exactly there.  $\square$

**Lemma R36.3** (weaken, standalone). *If  $\langle \text{pre}, \text{post} \rangle$  is semantically valid over  $H$ , so is  $\langle \text{pre} \wedge \text{extra}, \text{post} \rangle$ , for any formula  $\text{extra}$ .*

*Proof.* Project the premise (`Sequent.semanticallyValid_weaken`). The rider needs no side condition — in particular it may mention wires the cited factor has never heard of, which is the whole point.  $\square$

**Proposition R36.4** (soundness of `PRODUCTLATTICE`). *Given valid families over wire-disjoint hypernets, every facet sequent of every product block is semantically valid over  $H_1 \sqcup H_2$ .*



*Proof.* A product block’s facet list is, by construction (`prodBlock`), the left factor’s facets cited with the right factor’s region as rider, concatenated with the symmetric family. For a left-cited facet: the disjoint union extends  $H_1$ ’s wire typing and contains its applications (disjointness makes the union’s linearity, acyclicity, and type consistency inherit from the factors — this is where the side condition does structural work), so Lemma R36.2 transports the factor’s validity to the union, and Lemma R36.3 conjoins the rider. Symmetrically on the right. That is the entire proof (`sound_productLattice`, through the two standing citation lemmas `sound_productLattice_citeLeft/citeRight`). For the un-enumerated join blocks’ sub-facets the same two lemmas supply the per-generator transport and arbitrary-rider steps, with the exposure constraints as riders — but the join-closure extension itself (the boundary decomposition of Recipe R36.1’s argument) is paper-strength: no bound declaration represents finite unions, their boundary decomposition, or the union-inwardness step, and the mechanized theorem quantifies over the generator blocks only.  $\square$

*Remark R36.5* (why wire-disjointness is not decoration). With a shared wire the recipe fails twice. Structurally, the disjoint union is no longer well-formed as built (linearity of the concatenated application lists is proven *from* disjointness). Semantically, no single transported sequent’s validity would fail — Lemma R36.3 conjoins *any* rider, shared wires included — but the assembled *readout* would misdescribe the composite: the identification of every product-block facet with a cited factor facet (the other factor entering only as a rider) is proven from disjointness, and with a shared wire the built object is simply not a product of the factors. A single shared wire is precisely an *interface*, and composition along an interface is PROMISE/small-gain territory — the rule’s applicability condition is the boundary between the two composition regimes, not a technicality.

*Remark R36.6* (the completeness fence: flows only, and the map counterexample). Proposition R36.4 is dynamics-free — no premise mentions a flow or a map, so the rule is sound for both. What is *flows-only* is the completeness reading from the source literature [133]: for flows, the attractor lattice of a product is the coproduct of the factors’ lattices, so the generated family is everything. For maps it is not: with  $f(x) = -\tanh(2x)$  on  $[-p, p]$  (where  $\tanh(2p) = p$ ), the factor attractor lattice is  $\{\emptyset, P, X\}$  with  $P = \{p, -p\}$ , yet  $f \times f$  has the phase-locked period-two attractors  $\{(p, p), (-p, -p)\}$  and  $\{(p, -p), (-p, p)\}$  — neither generated by factor attractors; the coproduct is a proper sublattice. (The arithmetic:  $\tanh(2x)$  leaves the diagonal with slope 2, so a fixed point  $p \approx 0.96$  of  $\tanh(2p) = p$  exists; 0 repels,  $|f'(0)| = 2$ ; each product two-cycle attracts,  $(f^2)'(\pm p) = 4(1 - p^2)^2 < 1$ ; and that the factor lattice has nothing beyond  $\emptyset, P, X$  is the source example’s own analysis [133].) The Lean statement is shaped so this cannot overreach: it certifies a lower bound family, and any decomposition-level readout from a map instance must be stated as “at least this structure,” never “exactly this.”

| proof step                              | formal anchor                                                                                                                                                                                                           |
|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Recipe R36.1: citation & product blocks | <code>citeFacet, prodBlock, PProductLatticeData.prodBlocks</code>                                                                                                                                                       |
| Lemma R36.2                             | <code>Sequent.semanticallyValid_extend;</code> <code>restriction</code> <code>machinery</code><br><code>Hypernet.Valuation.restrict,</code> <code>Formula.holds_restrict,</code><br><code>PrimApp.semOk_restrict</code> |
| Lemma R36.3                             | <code>Sequent.semanticallyValid_weaken</code>                                                                                                                                                                           |
| union well-formedness from disjointness | <code>Hypernet.disjUnion (fields linearity, acyclic, typeConsistent);</code><br><code>Hypernet.WireDisjoint</code>                                                                                                      |
| Proposition R36.4                       | <code>sound_productLattice; sound_productLattice_citeLeft/citeRight</code>                                                                                                                                              |
| nested-application confinement          | <code>prodBlocks_confined</code>                                                                                                                                                                                        |
| family validity (premise form)          | <code>CertifiedBlockLattice.Valid</code>                                                                                                                                                                                |

## 15.5 Judgment Structure and Batteries

### Rule 37. QSATFROMREGION (alias for PCONSTEXACT)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QSatFromRegion*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{child} : \text{QRegionSat}(\varphi, R) \quad \text{eval} \vdash x_0 \in R \text{ (BL leaf; sublevel} \rightarrow \text{PCONSTEXACT, box} \rightarrow \text{registry interval check)} \quad \text{Sinit}(\varphi) \in s.\forall}{\mathcal{M} \models s}$$

QSATFROMREGION is the extension theory's closing move: it turns a region-level certificate into the root satisfaction verdict. The workhorse judgment of the extension page concludes region-level facts — every point of a region  $R$  satisfies the formula  $\varphi$ , with  $R$  under-approximating the formula's denotation — and this rule is the single bridge from that shape to the top-level claim  $\mathcal{M} \models s$ : cite the child, check that the specification's initial point  $x_0$  lies in  $R$  by exact rational arithmetic, and require that the warranted safety component of the specification is the initial-state injection  $\text{Sinit}(\varphi)$  of the child's formula. A producer uses it exactly once per initialized certificate, as the last row.

What the checker recomputes: the membership leaf — for a sublevel atom  $\{x : V(x) \leq y\}$ , exact substitution of  $x_0$  into  $V$  and one closed comparison, which is precisely the ground-fact rule PCONSTEXACT's own check; for a box atom, per-coordinate containment recomputed by the checker's side-condition registry — and the component tie: the warrant row names which safety component it warrants, anything other than the  $\text{Sinit}$  image of the child's formula refuses at load (a menu of one), and that the named component is among the specification's consumed components is admission conformance.

What would break without each hypothesis: drop the child and  $R$  is an arbitrary set nobody certified — the conclusion is unmoored from any proof; drop the eval leaf and  $x_0$  may lie outside  $R$ , so the child's universal never instantiates (a certificate could certify a region the system never starts in); drop the component tie and a certificate about some subformula — or an unrelated formula — could claim a component it never certified.

**notes** The soundness content is one line — instantiation of the child's universal at  $x_0$  — which is why the rule prices as an alias even though it has two premise fields: the child is a judgment *citation* (an edge under the tower discipline, the PROOFWELLFORMEDNESS section), and the sole derivation node the respelling produces is the PCONSTEXACT leaf. Mechanization state, plainly: the bound statements below are that one line and its assembly. `sound_QSatFromRegion` (module `BernsteinLogicAux.QtslAliases`) is the instantiation itself, generic in the satisfaction predicate and profile-free — axiom-free, even: the kernel certifies it depends on no axioms at all — and `sound_QSatFromRegion_sublevel` composes it over the exact-Q membership leaf (the `ROOTEval` identities, this page), which is the sole derivation node named above. The payload-level elaborator is bound beside them (module `BernsteinLogic.QtslElaborators`, the batch template's unpack made concrete): `qsatFromRegionElab` takes the rule payload to the one PCONSTEXACT payload — an inhabitant of the target's own payload type, comparator confinement discharged at elaboration — with `qsatFromRegionElab_sound` the factoring through `sound_pConstExact` and `qsatFromRegionElab_sat` the judgment-level composition over the elaborated leaf; the semantic content carries no profile residue, so the entry carries no conditional flag. Inherited flag, travelling with the alias: the target excludes the  $\neq$  comparator from its admissible set (a second-soundness-invariant exclusion, unpacked in the target's section); the membership leaf uses  $\leq$ , and any port must preserve the exclusion.

#### Lean statement

`BernsteinLogicAux.QtslAliases . sound_QSatFromRegion`

statement bound

```
/-- **`sound_QSatFromRegion`** (QTSL-rules.toml row `QSatFromRegion`; the
catalogue's "Soundness: one line – instantiation of the universal"): the
child certificate concludes region-level satisfaction (`R` under-approximates
the truth set: every point of `R` satisfies), the eval leaf places the
specification's initial point in `R`, and the root claim follows by
instantiation. Generic in the satisfaction predicate: for the probabilistic
judgments the extension page concludes, `sat x` reads as the concluded
bound at start `x` (e.g. `sound_QSafetyBattery`'s conclusion), and the rule
is profile-free. The root tie `s.root = φ` is a syntactic side condition
recomputed by the checker; it selects *which* predicate this lemma is
instantiated at, and carries no semantic content of its own. -/
theorem sound_QSatFromRegion {X : Type*} {R : Set X} {sat : X → Prop} {x₀ : X}
  (hchild : ∀ x ∈ R, sat x) (heval : x₀ ∈ R) : sat x₀
```

BernsteinLogicAux.QtslAliases . sound\_QSatFromRegion\_sublevel

statement bound

```

/-- **`QSatFromRegion` assembled over its `RootEval` leaf, sublevel shape**:
the child's region-level universal on the cast sublevel atom, instantiated
at the cast initial point through the exact-Q evaluation – the sole
derivation node the respelling produces, composed. -/
theorem sound_QSatFromRegion_sublevel {σ : Type*} {sat : (σ → ℝ) → Prop}
  {P : MvPolynomial σ ℚ} {thr : ℚ} {xq : σ → ℚ}
  (hchild : ∀ x ∈ {x : σ → ℝ |
    MvPolynomial.eval x (MvPolynomial.map (Rat.castHom ℝ) P) ≤ (thr : ℝ)}, sat x)
  (heval : MvPolynomial.eval xq P ≤ thr) :
  sat (fun i => (xq i : ℝ))

```

**The payload-level elaborator, displayed whole.** The unpack builds the target's payload from the rule's wire data — the eval field *is* the `RootEval` leaf (that section binds the closed-term builder and the wire-level exactness identity it rides).

BernsteinLogic.QtslElaborators . qsatFromRegionElab\_sound

statement bound

```

/-- The factoring, BL face – through `sound_pConstExact`, as the amber
box demanded. -/
theorem qsatFromRegionElab_sound {H : Hypernet} {seq : Sequent}
  {x0 : Nat → Rat} {V : Poly} {thr : Rat}
  (hEval : (rootEvalFormula x0 V thr).evalClosed = some true)
  (hpost : seq.post = rootEvalFormula x0 V thr) :
  seq.semanticallyValid H

```

BernsteinLogic.QtslElaborators . qsatFromRegionElab\_sat

statement bound

```

/-- The judgement-level composition over the elaborated leaf
(`prop:qsr-sound` at the wire): instantiation of the child's universal
at  $x_0$ , the membership hypothesis supplied by the checked leaf through
the  $\mathbb{Q}$ -face exactness identity. `sound_QSatFromRegion` is generic in the
satisfaction predicate and the carrier, so the composition consumes the
bound lemma verbatim – sound_A = (instantiation) ◦ (elaborated leaf). -/
theorem qsatFromRegionElab_sat {x0 : Nat → Rat} {V : Poly} {thr : Rat}
  {sat : (Nat → Rat) → Prop}
  (hchild : ∀ x ∈ {x : Nat → Rat | Poly.eval x V ≤ thr}, sat x)
  (hEval : (rootEvalFormula x0 V thr).evalClosed = some true) :
  sat x0

```

BernsteinLogic.QtslElaborators . qsatFromRegionElab

definition bound

```

/-- **The QSatFromRegion payload elaborator**, sublevel case: the
rule's wire payload elaborates to exactly the RootEval leaf's payload –
the leaf-level respelling has its own row (`RootEval`), and this rule's
eval field IS that leaf (the consumption-site anchor of the RootEval
section). -/
def qsatFromRegionElab (H : Hypernet) (seq : Sequent) (x0 : Nat → Rat)
  (V : Poly) (thr : Rat)
  (hEval : (rootEvalFormula x0 V thr).evalClosed = some true)
  (hpost : seq.post = rootEvalFormula x0 V thr) :
  PConstExactData H seq :=
  rootEvalElab H seq x0 V thr hEval hpost

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed premises map onto `sound_QSatFromRegion` verbatim: the child  $\mathbb{Q}$ RegionSat( $\varphi, R$ )  $\mapsto$  `hchild : ∀ x ∈ R, sat x` (region-level satisfaction, compiled); the eval leaf  $\vdash x_0 \in R \mapsto$  `heval : x_0 ∈ R`; the conclusion  $\mathcal{M} \models s \mapsto$  `sat x_0` — universal instantiation, one line, axiom-free. The component tie ( $\text{SInit}(\varphi) \in s.\forall$ , the refuse-mode warrant naming) has deliberately no Lean

counterpart: it is the syntactic side condition that selects *which* predicate `sat` the lemma is instantiated at, recomputed by the checker, carrying no semantic content — the statement's docstring says exactly this. `sound_QSatFromRegion_sublevel` is the assembled form over the exact- $\mathbb{Q}$  `RootEval` leaf: child universal on the cast sublevel atom + exact- $\mathbb{Q}$  evaluation `heval : MvPolynomial.eval xq P ≤ thr ⇒ satisfaction at the cast initial point` — the sole derivation node the respelling produces, composed. Width note: `sat : X → Prop` is generic — probabilistic conclusions read `sat x` as the concluded bound at start `x`, so the alias is profile-free by binder shape; wider than any single displayed instantiation, safe direction.

## Lean verifier receipts

module `BernsteinLogicAux.QtslAliases`; bound build 570599779c29... (bound from receipt 5a60f4d01fd81ac5, 2026-07-21T05:39:02Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0a3819a978e31127 | 2026-07-21T04:51:36Z | build+decorrelated                 | current |
| 0bb1074df705d863 | 2026-07-21T05:21:58Z | build+decorrelated                 | current |
| 5a60f4d01fd81ac5 | 2026-07-21T05:39:02Z | build+replay+hardened+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.QtslElaborators`; bound build 0205ec213b91... (hashed from the built olean at bind time). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a237f4cff27cb15a | 2026-07-22T10:06:47Z | build+decorrelated | current |

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current  
 replay (leanchecker (official kernel replay)) current  
 hardened (safe\_verify (hardened screen)) current  
 decorrelated (lean4export + nanoda (decorrelated kernel)) current

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development's object is the base calculus's wire-format checker (theory `bern_catalogue` ports the base catalogue's check-function cone); no HOL development of the extension theory exists, so there is no HOL statement for any extension rule to bind. Absent rather than out-of-scope: the bridge's baked realization is the extension theory's warrant seam (`qsat_warrant`), so a future HOL port of an extension-theory checker would state it. The one leaf the alias adds checks through the base ground-fact rule `PConstExact`, whose own HOL statement is not yet writable either (it postdates the HOL4 port and its wire tag is unallocated — see that rule's section).

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template: the 1→1 unpack, the filled-in target application, the payload-transport table, the exactness identity, and the soundness corollary through the target's lemma. Two riders. First, the tier criterion counts derivation *nodes*, not premise fields: this rule has two premise fields, but the child is a judgment *citation* — an edge under the tower discipline (the `PROOFWELLFORMEDNESS` section), not a node any unpack must produce — so the rule's proof content per instance is at most one node: in the sublevel branch, the single `PConstExact`

application discharging its membership leaf; in the box branch, no node at all (the leaf is registry recomputation). The child instantiation and the component tie are definitional transports along edges, not nodes any unpack produces — that, and nothing stronger, is the sense in which the rule is an alias. Second, the semantic composition is mechanized and bound in the entry, and so — now that the extension certificate schema exists — is the payload-level elaborator (the batch template's unpack, at the wire level; the remark below names the bound functions).

**Recipe R37.1** (the alias template, one extends-edge up). An alias at the satisfaction level is warranted by a  $1 \rightarrow 1$  respelling into a single application of a target rule, with judgment citations transported as *edges* and the concluded judgment recovered definitionally from the cited child. Here the target is the base calculus's ground-fact rule  $\text{PCONSTEXACT}$ ; the one new node is the membership leaf; and the  $\models$ -level conclusion is an instantiation of the child's universal (Proposition R37.4), not a derived fact needing catalogue structure.

**Recipe R37.2** (the filled-in application). The rule's fields: a child judgment  $\text{child} : \text{QRegionSat}(\varphi, R)$  — every point of  $R$  satisfies  $\varphi$ , with  $R$  under-approximating the formula's denotation — a membership leaf eval, and the component tie  $\text{SInit}(\varphi) \in s.\forall$  (the warranted safety component is the initial-state injection of the child's formula — the record carries component lists, not a single root, and the warrant row names its component). For a sublevel region atom  $R = \{x : V(x) \leq y\}$ , with  $V$  a polynomial over  $\mathbb{Q}$  and  $x_0$  the specification's rational initial point, the eval leaf is the  $\text{PCONSTEXACT}$  node

$$\frac{V(x_0) \leq y \text{ holds under exact evaluation in } \mathbb{Q}}{\Phi \vdash V(x_0) \leq y} \text{PCONSTEXACT}$$

with arbitrary premise  $\Phi$ , as the target licenses.  $V(x_0)$  is the closed term obtained by exact substitution of  $x_0$ 's coordinates — wire-free, so the target's verified evaluator decides it, with comparator  $\leq$  inside the target's admissible set. The leaf-level respelling has its own row on this page (ROOTEVAL), whose section carries the transport and exactness detail; this section carries the judgment-level composition. For a *box* atom the leaf is not a  $\text{PCONSTEXACT}$  node: per-coordinate interval containment is recomputed directly by the checker's side-condition registry — the alias reading covers the sublevel branch, and the box branch is stated here so the respelling never overclaims.

**Recipe R37.3** (payload transport — field for field).

| $\text{QSATFROMREGION}$ field                       | destination                                                         |
|-----------------------------------------------------|---------------------------------------------------------------------|
| child                                               | not transported — a judgment citation (edge; the tower discipline)  |
| eval                                                | the $\text{PCONSTEXACT}$ payload: closed comparison $V(x_0) \leq y$ |
| component tie $\text{SInit}(\varphi) \in s.\forall$ | the concluded judgment's formula                                    |
| concluded $\mathcal{M} \models s$                   | instantiation of the child's universal at $x_0$                     |

**Proposition R37.4** (the instantiation identity). *If  $R \subseteq \{x : \mathcal{M}, x \models \varphi\}$  and  $x_0 \in R$ , then  $\mathcal{M}, x_0 \models \varphi$ ; with the component tie, this is  $\mathcal{M} \models s$ . Moreover, for the sublevel atom the membership hypothesis is exactly the checked leaf, as an identity of exact rationals:*

$$x_0 \in R \iff V(x_0) \leq y,$$

*because substituting a rational point into a rational polynomial evaluates with no rounding — the checker's true verdict is membership, not an approximation to it (the ROOTEVAL section proves this as its exactness identity).*

*Proof.* Instantiate the universal at  $x_0$ ; with the component tie the instance is  $\mathcal{M} \models s$ . For the moreover clause, two equivalences compose:  $x_0 \in R \iff V(x_0) \leq y$  is the sublevel region's membership predicate read at  $x_0$  — a definition, not a computation — and the identification of the checker's true verdict with that exact inequality is the ROOTEVAL exactness identity, cited rather than re-proved. The instantiation is the rule's entire own content — one node's worth, which is why it prices as an alias.  $\square$

**Proposition R37.5** (soundness corollary). *In the sublevel branch,  $\text{sound}_{\text{QSATFROMREGION}} = \text{sound\_pConstExact}$  (applied to the eval leaf) ; (instantiation at  $x_0$ ); in the box branch the recomputed containment supplies the membership fact and the same instantiation finishes. In both branches the child's validity is supplied by the induction hypothesis of the tower discipline's induction (PROOFWELLFORMEDNESS section).*

*Proof.* Sublevel branch:  $\text{sound\_pConstExact}$  (the target's kernel-verified lemma, quoted in the  $\text{PCONSTEXACT}$  section) turns the true evaluation into the validity of  $V(x_0) \leq y$ ; Proposition R37.4 finishes. Box branch: the recomputed containment plays the leaf's role — per-coordinate comparison of exact rationals, the box face of the same no-rounding exactness the ROOTEVAL section proves for sublevel leaves, so the recompute's verdict is membership itself; Proposition R37.4's instantiation, which never asked how membership was obtained, finishes identically.  $\square$

**Remark R37.6** (the payload-level elaborator, mechanized). The *semantic* composition above is bound in the entry:  $\text{sound\_QSATFROMREGION}$  is the instantiation (axiom-free), and  $\text{sound\_QSATFROMREGION\_sublevel}$  is Proposition R37.5's statement over the exact- $\mathbb{Q}$  leaf (the ROOTEVAL identities). The payload-level elaborator — previously this page's one absent artefact, held back until the extension certificate schema existed — is bound with it:  $\text{qsatFromRegionElab}$  (module  $\text{BernsteinLogic.QtsLElaborators}$ ) takes the rule's wire payload to (in the sublevel case) exactly the one  $\text{PCONSTEXACT}$  payload of Recipe R37.3, as an inhabitant of the target's own payload type — the comparator confinement discharged at elaboration, once — with  $\text{qsatFromRegionElab\_sound}$  the factoring through  $\text{sound\_pConstExact}$  and  $\text{qsatFromRegionElab\_sat}$  the judgment-level composition over the elaborated leaf, the checker's verdict entering as membership through the wire-level exactness identity  $\text{rootEvalFormula\_true\_iff}$  (the  $\mathbb{Q}$  face of the ROOTEVAL identities). The box branch remains checker-side registry recompute, by design — nothing to elaborate.

**Remark R37.7** (the inherited comparator confinement).  $\text{PCONSTEXACT}$  excludes one comparator ( $\neq$ ) from its admissible set, for reasons of a second soundness invariant unpacked in that rule's section. The membership leaf uses  $\leq$  only — comfortably inside — but any port or mechanization of this rule must *preserve* the exclusion rather than re-derive the comparator set from the semantic lemma alone; the flag travels with the alias.

| proof step                              | formal anchor                                                                                                                      |
|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| target lemma (Prop. R37.5)              | sound_pConstExact; Formula.holds_of_evalClosed_true (quoted in the PCONSTEXACT section)                                            |
| leaf-level transport + exactness        | the ROOTEval section (this page)                                                                                                   |
| child validity                          | the tower discipline's induction (PROOFWELLFORMEDNESS section)                                                                     |
| alias-level corollary                   | sound_QSatFromRegion + sound_QSatFromRegion_sublevel (bound in the entry)                                                          |
| payload-level elaborator (Remark R37.6) | qsatFromRegionElab + qsatFromRegionElab_sound + qsatFromRegionElab_sat (module BernsteinLogic.QtslElaborators, bound in the entry) |
| box-atom branch                         | checker-side registry recompute, by design — no rule anchor                                                                        |

## Rule 38. QGUARDINTRO (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QGuardIntro*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{child} : \text{QRegionSat}(\varphi, R) \quad \text{SInit}(\varphi) \in s. \forall}{\mathcal{M} \models \text{Guard}(R, s)}$$

QGUARDINTRO is implication introduction at the judgment stratum: from a region-level certificate alone — every state of  $R$  satisfies  $\varphi$  — it concludes the *guarded* claim  $\mathcal{M} \models \text{Guard}(R, s)$ , with no initial-membership premise anywhere. The hypothesis that the unguarded route discharges at the ground seam ( $X_0 \subseteq R$ , checked by exact rational arithmetic) is instead discharged *into the specification*: a guarded specification demands its body's guarantee only from initial states inside  $R$  and promises nothing off-region. The guard region imports the certificate-side region grammar (sublevel, box, affine) into the specification record at exactly one slot; the record still binds no model and fixes no initial condition — the guard is a predicate *on* the subject's initial data, not initial data.

What the checker recomputes: nothing beyond the child's own obligations — that is the point. The two ties are load-time conformance, not computation: the guarded warrant row names its safety component, and anything other than the initial-state injection of the warrant's formula refuses at load (a menu of one); the guard region stored on the record is derived from the warrant row itself (store-and-check: a disagreeing producer row is a second region on a one-region slot, refused at conformance).

What would break without each hypothesis: drop the child and the guarded claim is unmoored from any proof; drop the component tie and a certificate about an unrelated formula could claim the guarded specification; drop the disjointness from the unguarded warrant relation and the initial-membership demand would fire on a guarded introduction — exactly the hypothesis the former exists to internalize.

**notes** The semantics is *pointwise* over the credal stack: a behavior object satisfies  $\text{Guard}(R, s)$  iff every admitted law whose initial marginal is supported in  $\llbracket R \rrbracket$  meets  $s$ 's demand — at demonic-box initial data this is exactly conditioning the belief evaluation on initial membership. Pointwise is *strictly stronger* than the blanket conditional (“if all of  $X_0$  lies in  $R$  then...”), and the strictness is a theorem, not a taste: the blanket reading would vacuously discharge any guard the initial set merely straddles. The class algebra stays a complement-free distributive lattice — the guard's antecedent is a ground condition on system data, not a behavior class, which is why the conditional lives at the record stratum and no formula-level implication former exists or is introduced.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The baked realization: the guard slot on the specification record, the guarded-route warrant relation (disjoint-by-relation from the unguarded one, so the initial-membership axiom structurally never fires on a guarded introduction), and the ties.

theory Qtsl . spec\_guard

quoted from the theory source at render

| spec\_guard : {spec: Spec, region: QRegionAtom} -> Prop;

theory Qtsl . qsat\_guard\_warrant

quoted from the theory source at render

| qsat\_guard\_warrant : {sat: QSat, warrant: QForallSat} -> Prop;

theory Qtsl . ax/qguard/region\_tie

quoted from the theory source at render

| chase ax/qguard/region\_tie : forall r : QSat, c : QForallSat,  
| sp : Spec.  
| {sat: r, warrant: c} qsat\_guard\_warrant,  
| r QSat/spec = sp  
|- {spec: sp, region: c QForallSat/region} spec\_guard;

theory Qtsl . ax/wf/qguard\_warrant\_member

quoted from the theory source at render



```

report ax/wf/qguard_warrant_member : forall r : QSat, sp : Spec,
  c : QForallSat.
{sat: r, warrant: c} qsat_guard_warrant,
r QSat/spec = sp
|- {spec: sp, f: c QForallSat/component} qtsl_warrant_member;

```

## Lean statement

The guarded denotation and the introduction's soundness, displayed whole. GuardSat is the pointwise reading; the strictness witness against the blanket conditional and the wrong-region countermodel bind beside the soundness statements.

QtslAux.QGuard.sound\_QGuardIntro

statement bound

```

/-- Soundness of the introduction rule: a region warrant – the body's
demand holds from every state of `R` – yields the guarded claim with no
initial-membership premise, for any initial set. The membership hypothesis
that region-warranted satisfaction discharges at the ground seam is here
discharged into the former (implication introduction). -/
theorem sound_QGuardIntro {init R : Set X} {sat : X → Prop}
  (hchild : ∀ x ∈ R, sat x) : GuardSat init R sat

```

QtslAux.QGuard.sound\_QGuardIntro\_sublevel

statement bound

```

/-- The rational-sublevel guard (a certificate family's sublevel set at an
exact-rational threshold, single-piece form): a warrant on the real
sublevel set of the coefficient-cast polynomial yields the guarded claim
with that sublevel set in the guard slot, for any initial set. -/
theorem sound_QGuardIntro_sublevel {σ : Type*} {init : Set (σ → ℝ)}
  {sat : (σ → ℝ) → Prop} {P : MvPolynomial σ ℚ} {thr : ℚ}
  (hchild : ∀ x ∈ {x : σ → ℝ |
    MvPolynomial.eval x (MvPolynomial.map (Rat.castHom ℝ) P) ≤ (thr : ℝ)},
    sat x) :
  GuardSat init
  {x : σ → ℝ |
    MvPolynomial.eval x (MvPolynomial.map (Rat.castHom ℝ) P) ≤ (thr : ℝ)}
  sat

```

QtslAux.QGuard.sound\_QGuardIntro\_prob

statement bound

```

/-- Kernel-form corollary in the battery idiom: a per-initial-state
probability floor warranted on the region yields the guarded floor. -/
theorem sound_QGuardIntro_prob [MeasurableSpace X] {init R : Set X}
  {μ : X → Measure (ℕ → X)} {p : ENNReal} {A : Set (ℕ → X)}
  (hchild : ∀ x ∈ R, p ≤ μ x A) :
  GuardSat init R fun x => p ≤ μ x A

```

QtslAux.QGuard.exists\_blanket\_not\_guardSat

statement bound

```

/-- Strictness witness for the semantics adjudication, and the permanent
twin of the dropped-membership mutant: the blanket reading does not imply
the pointwise one. On `Bool` with the full initial set, guard region
`{true}`, and an everywhere-false demand, the blanket conditional holds
vacuously (the initial set is not contained in the region) while the
pointwise guard is refuted at `true`. A blanket semantics would license
claims the pointwise semantics rejects; the former adopts the pointwise
reading. -/
theorem exists_blanket_not_guardSat :
  ∃ (init R : Set Bool) (sat : Bool → Prop),
    GuardSatBlanket init R sat ∧ ¬ GuardSat init R sat

```

QtslAux.QGuard.exists\_not\_guardSat\_wrong\_region

statement bound

```

/-- Wrong-region countermodel (the permanent twin of the region-substitution
mutant): a warrant on one region does not license a guard on an unrelated
one – an everywhere-false demand is vacuously warranted on the empty

```

```
region, and the guard on the full region is refuted. -/
theorem exists_not_guardSat_wrong_region :
  ∃ (init R R' : Set Bool) (sat : Bool → Prop),
    (∀ x ∈ R, sat x) ∧ ¬ GuardSat init R' sat
```

QtslAux.QGuard.GuardSat

definition bound

```
-- The guarded-satisfaction denotation, pointwise (demonic-initial-set)
form: every admitted initial point inside the guard region meets the body's
demand; initial points outside the region are unconstrained. -/
def GuardSat (init R : Set X) (sat : X → Prop) : Prop :=
  ∀ x₀ ∈ init, x₀ ∈ R → sat x₀
```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The display maps onto `sound_QGuardIntro` verbatim: the child  $\text{QRegionSat}(\varphi, R) \mapsto h : \forall x \in R, \text{sat } x$ ; the conclusion  $\mathcal{M} \models \text{Guard}(R, s) \mapsto \text{GuardSat init } R \text{ sat}$  (pointwise: every admitted initial point inside  $R$  meets the body's demand), generic in the initial set `init` — the rule is sound for *any* initial data, which is what “no membership premise” means formally. The component tie  $(\text{SInit}(\varphi) \in s.\forall)$  has deliberately no Lean counterpart: it is the load-time refusal selecting *which* predicate `sat` the lemma is instantiated at. Width note: `sat : X → Prop` is generic, so probabilistic bodies read `sat x` as the concluded bound at start  $x$  — wider than any single displayed instantiation, safe direction. The specializations bind beside it: `sublevel-atom` guards over exact- $\mathbb{Q}$  polynomial evaluation (`sound_QGuardIntro_sublevel`, `guard_eval_sublevel`) and the probability-threshold body form (`sound_QGuardIntro_prob`).

## Lean verifier receipts

module QtslAux.QGuard; bound build 3eb84fb9b559... (bound from receipt bd99b3a9109c4daf, 2026-07-28T18:45:47Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| bd99b3a9109c4daf | 2026-07-28T18:45:47Z | build+decorrelated | current |

gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **none**

hardened (safe\_verify (hardened screen)) **none**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator:** **none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

No HOL development of the extension theory exists (see the region-to-satisfaction bridge's section); the guarded former's realization is extension-theory record structure, so a future HOL port of an extension-theory checker would state it.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), structural group. The proof is deliberately one step — the rule exists to *relocate* a hypothesis, not to add content — and the substantive work is the semantics adjudication in the second remark, certified by the strictness witness and the wrong-region countermodel bound at this section.

**Claim (sound\_QGuardIntro).** Let  $R$  be the warrant’s region,  $s$  the specification whose warranted safety component is  $\text{SInit}(\varphi)$ , and suppose the region warrant: every state of  $R$  satisfies  $\varphi$ . Then for *any* initial set  $X_0$ , the guarded claim holds: every admitted initial point inside  $R$  meets the body’s demand.

*Proof.* Let  $x \in X_0 \cap R$ . The warrant instantiates at  $x$ :  $x \in R$ , so  $x \models \varphi$ , which is the body’s demand at  $x$ . Since  $x$  was an arbitrary admitted initial point inside the guard region, the pointwise guarded denotation holds. No hypothesis on  $X_0$  was used — the rule is sound for arbitrary initial data, which is exactly what licenses dropping the membership premise.  $\square$

*Remark R38.1* (why pointwise, not blanket). The blanket conditional — *if all of  $X_0$  lies in  $R$ , then the body holds from all of  $X_0$*  — is strictly weaker: on an initial set that merely straddles  $R$ , the blanket antecedent is false and the blanket conditional vacuously true, while the pointwise reading still demands the body from the inside points. The strictness is mechanized (`exists_blanket_not_guardSat`); the pointwise reading is the adopted semantics, and it is the one the introduction rule is sound for.

*Remark R38.2* (wrong region refuted). Substituting any region other than the warrant’s own breaks soundness, by countermodel (`exists_not_guardSat_wrong_region`); the load-time tie (the stored guard region is derived from the warrant row, store-and-check) is what makes the substitution unrepresentable rather than detected.

## Rule 39. QGUARDELIM (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QGuardElim*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\mathcal{M} \models \text{Guard}(R, s) \quad \text{eval} : \vdash X_0 \subseteq R}{\mathcal{M} \models s}$$

QGUARDELIM is the elimination: a guarded claim plus initial containment yields the body's claim. The membership premise is the same ground computation the unguarded route uses — per region-constructor pair, a degenerate box against a sublevel atom is the exact evaluation leaf, a fat box against a sublevel atom is an enclosure family, and box against box is interval containment — re-entering at exactly one place. Introduction followed by elimination recovers the dissolved region-warranted route whole, so the guarded former adds expressive structure without adding any new soundness surface: everything the pair can conclude, the old route could conclude given the same membership evidence.

What would break without the containment premise is the consumption fence's whole reason: a guarded claim consumed as if unguarded is an unsound *strengthening* — the guarded claim promises nothing off-region, so treating it as the body's claim smuggles in the very hypothesis the former internalized.

**notes** Realized *at consumption*, deliberately — there is no standalone elimination node. Guard-presence triggers the membership demand on any consumer of the claim: the composition rule walks its premise list and demands initial membership for every guarded premise (the quoted consumer-fence axiom below), and the verdict-compile and equivalence evaluators are guard-aware. A consumer that cannot discharge the membership defers loudly rather than assuming it.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: *chase* = the engine derives; *report* = a violation is a loud judgment; *refuse* = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The baked realization of elimination-at-consumption: list membership over the composition rule's premises, the consumer-fence axiom demanding initial membership for guarded premises, and the ground membership relation it demands.

theory Qtsl . **qsat\_list\_member**

quoted from the theory source at render

```
| qsat_list_member : {list: QSatList, sat: QSat} -> Prop;
```

theory Qtsl . **ax/wf/qwirecompose\_guard\_elim**

quoted from the theory source at render

```
| report ax/wf/qwirecompose_guard_elim : forall n : QWireCompose,
  l : QSatList, q : QSat, y : System, sp : Spec,
  g : QRegionAtom.
  n QWireCompose/components = l,
  {list: l, sat: q} qsat_list_member,
  q QSat/sys = y,
  q QSat/spec = sp,
  {spec: sp, region: g} spec_guard
  |- {region: g, sys: y} qtsl_init_member;
```

theory Qtsl . **qtsl\_init\_member**

quoted from the theory source at render

```
| qtsl_init_member : {region: QRegionAtom, sys: System} -> Prop;
```

### Lean statement

QtslAux.QGuard . **sound\_QGuardElim**

statement bound

```

/-- Soundness of the elimination rule: the initial-membership evaluation
re-enters exactly here – a guarded claim together with containment of the
initial set in the guard region yields the unguarded claim. -/
theorem sound_QGuardElim {init R : Set X} {sat : X → Prop}
  (hg : GuardSat init R sat) (heval : init ⊆ R) : ∀ x₀ ∈ init, sat x₀

```

QtslAux.QGuard.**qsatFromRegion\_derived**

statement bound

```

/-- Coherence: introduction followed by elimination recovers the
region-warranted record route (warrant plus initial containment yields
satisfaction from every initial point). -/
theorem qsatFromRegion_derived {init R : Set X} {sat : X → Prop}
  (hchild : ∀ x ∈ R, sat x) (heval : init ⊆ R) : ∀ x₀ ∈ init, sat x₀

```

QtslAux.QGuard.**exists\_guardSat\_not\_unguarded**

statement bound

```

/-- The permanent twin of the dropped-membership mutant, stated directly:
elimination WITHOUT its containment premise is false – a guarded claim
does not yield the unguarded one. On `Bool` with the full initial set and
guard region `{true}`, the demand "the state is `true`" is guarded-satisfied
(vacuously off-region, directly on it) yet fails from the initial point
`false`. -/
theorem exists_guardSat_not_unguarded :
  ∃ (init R : Set Bool) (sat : Bool → Prop),
    GuardSat init R sat ∧ ¬ ∀ x₀ ∈ init, sat x₀

```

QtslAux.QGuard.**guardSat\_singleton\_iff**

statement bound

```

/-- At a singleton initial set the guarded denotation is the pointwise
conditional – the shape `Qtsl.Aliases.sound_QSatFromRegion` consumes. -/
theorem guardSat_singleton_iff {R : Set X} {sat : X → Prop} {x₀ : X} :
  GuardSat {x₀} R sat ↔ (x₀ ∈ R → sat x₀)

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable ↔ Lean binder, each display hypothesis ↔ the premise or side condition carrying it, the display conclusion ↔ the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The display maps onto `sound_QGuardElim` verbatim: the guarded premise ↔ `hg : GuardSat init R sat`; the containment premise  $\vdash X_0 \subseteq R$  ↔ `hsub : init ⊆ R`; the conclusion  $\mathcal{M} \models s \mapsto \forall x \in \text{init}, \text{sat } x$  (the unguarded pointwise demand). The dropped-membership variant is refuted by countermodel (`exists_guardSat_not_unguarded`); the coherence statement `qsatFromRegion_derived` is the factoring of the dissolved region-warranted route through the pair.

## Lean verifier receipts

module QtslAux.QGuard; bound build 3eb84fb9b559... (bound from receipt bd99b3a9109c4daf, 2026-07-28T18:45:47Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| bd99b3a9109c4daf | 2026-07-28T18:45:47Z | build+decorrelated | current |

gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **none**

hardened (safe\_verify (hardened screen)) **none**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator:** none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

As at the introduction rule: no HOL development of the extension theory exists; the consumption fence is extension-theory walk structure.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), structural group. One-step proof; the load-bearing content is the consumption fence (a guarded claim consumed as if unguarded is an unsound strengthening), certified by the dropped-membership countermodel.

**Claim (sound\_QGuardElim).** If the guarded claim holds (every admitted initial point inside  $R$  meets the body's demand) and  $X_0 \subseteq R$ , then the body's claim holds from every admitted initial point.

*Proof.* Let  $x \in X_0$ . By containment  $x \in R$ , so  $x \in X_0 \cap R$ , and the guarded claim yields the body's demand at  $x$ .  $\square$

*Remark R39.1* (the fence, and its countermodel). Dropping the containment premise is refuted by countermodel (exists\_guardSat\_not\_unguarded): a guarded claim promises nothing off-region, so an initial point outside  $R$  can violate the body while the guarded claim holds. This is why the elimination is realized *at consumption* — guard-presence triggers the membership demand on any consumer — rather than trusting producers to remember it.

*Remark R39.2* (coherence). Introduction followed by elimination recovers the dissolved region-warranted route whole (qsatFromRegion\_derived): warrant in, guarded claim out, membership in, body's claim out — the same conclusion the old single rule drew from the same two pieces of evidence. The former adds a place to *stand* between them, not a new way to conclude.

## Rule 40. QSafetyBattery (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QSafetyBattery*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\begin{array}{c}
 V : \text{QCertFam} \\
 \{D_m \vdash \mathcal{A}_m V_m \leq 0\}_m \text{ (interiors)} \quad \{\text{slab}_j \vdash g_j \circ V_{m'_j} \leq V_{m_j}\}_j \text{ (guards, per jump } j: \text{ source mode } m_j, \text{ target } m'_j, \text{ reset } g_j) \\
 \{U \cap \bar{D}_m \vdash V_m \geq 1\}_m \text{ (floors)} \quad \{\bar{D}_m \vdash V_m \geq 0\}_m \text{ (nonnegs)} \\
 \text{scope\_warrant} : \text{QInvScope (optional — the dissolved ScopeRule)} \quad p + y \leq 1 \text{ (threshold tie)} \\
 \hline
 \mathcal{M}, x_0 \in \{V \leq y\} \models \text{SProbGE}(p, \text{PAlways } \varphi)
 \end{array}$$

a region written left of the turnstile scopes the row:  $R \vdash P$  asserts  $P$  at every point of  $R$  (each such row is an inherited BL BERNcone-family sequent). The certificate family  $V$  enters as certificate data — a shared reference with no claim content of its own; the rows impose every claim — never as a derivation edge. The threshold tie  $p + y \leq 1$  is the soundness theorem's own hypothesis, displayed as a side condition: unlike the deadline and until nodes (whose baked sorts carry a tie proof field), the baked safety node carries no tie field yet — a named schema divergence, not a checked leaf. The conclusion is the theory row's own judgment, in constructor vocabulary: the region-conditioned probability formula ( $\text{SProbGE}(p, \text{PAlways } \varphi)$ ) on the region  $\text{qraSublevel}(V, y)$ . The record's licensed alias tie is  $\text{SProbGE}(p, \psi) = \text{SReturnGE}(p, \text{retPath } \psi)$  — a return respelling, which does not touch the state-region antecedent; the initial-set hypothesis  $x_0 \in \{V \leq y\}$  left of the judgment is the satisfaction binding's own guard (the theory demands the subject's initial data lie in the concluded region), rendered in the same hypothesis convention as every BERNcone-family sequent. The internalized variant LANDED at the grid re-sort revision: the guarded-spec former (*Guard*, a certificate-stratum record slot carrying the initial-region antecedent) with its introduction rule *QGUARDINTRO* (this catalogue, the structural group). The hypothesis form remains this catalogue's rendering convention (one convention catalogue-wide); the guarded former is its internalized spelling, one *QGUARDINTRO* application away, and no belief-gauge reading is displayed.

QSafetyBattery is the probabilistic safety battery: from a family of purely polynomial premise rows about a certificate  $V$ , it concludes that with probability at least  $p$  the system satisfies  $\text{PAlways } \varphi$  (surface syntax  $G \varphi$ ) — *forever* — from every start state in the sublevel region  $\{V \leq y\}$ . The rule concludes a satisfaction judgment ( $\models$ ), one theory-extension edge above the base calculus, and every premise row is an inherited base-calculus sequent: *interior* rows say the certificate does not increase along the in-mode dynamics, *guard* rows say jumps do not increase it, *floor* rows say  $V \geq 1$  on the avoid set (the compiled over-approximation of where  $\varphi$  fails), and *nonneg* rows say  $V \geq 0$  on the domain hull. The battery adds zero kernel arithmetic; its entire content is the supermartingale assembly one level up:  $V$  read along the process and stopped at the first failure is a nonnegative supermartingale, and Ville's inequality converts its initial value into a bound on the probability of ever failing.

A producer reaches for the rule wherever a quantitative safety guarantee over a hybrid stochastic system is wanted. The certificate is found by whatever synthesis method the producer likes (typically linear or sum-of-squares programming over the same polynomial cones the premise rows are checked in); only the rows ship in the certificate.

What the checker recomputes are the side conditions: the domain-hull condition (one step from any in-scope state stays inside the hull  $\bar{D}$ , which is exactly where the floor and nonneg rows were checked), the fail-closed containment (one-step mass stays in  $D \cup U$ ; leaving the modelled domain is counted as *failure*, never as silent success), and the floor-scope resolution (the floor rows' scopes resolve to an exact cover of the avoid set within the hull). The threshold tie  $p + y \leq 1$  is displayed as a side condition, not a checked leaf: it is the soundness theorem's own hypothesis, and the baked safety node carries no tie proof field yet (contrast the deadline and until nodes) — a named schema divergence, as the rule's flavor text states.

Each hypothesis is load-bearing. Drop the floors and failure is free of charge: a certificate identically 0 satisfies every other row and would certify almost-sure safety of a chain that fails at its first step. Drop the nonnegs and the maximal inequality itself fails — a signed process can pay for a barrier visit with mass borrowed from below. Drop the fail-closed containment and probability mass could exit the modelled domain unaccounted. Drop the threshold tie and the conclusion would assert success probability  $p$  alongside a failure budget  $y$  with  $p + y > 1$ . The proof section walks each of these, with counterexamples.

The dashboard row's mark states the status honestly: the hash-bound theorems warrant the rule at the *discrete-kernel profile* — system semantics a one-step Markov kernel on the flattened hybrid state space, interior rows read as one-step expected decrease (the Foster–Lyapunov reading) — with the PDMP sub-profile of the continuous-time semantics additionally machine-checked end-to-end. The general continuous-time profile (jump–diffusion) is *proven*, at the martingale-problem presentation: the system enters through its generator drift data,  $\mathcal{A}V$  as data with the martingale-problem hypothesis for the pair, and the battery's conclusion follows end-to-end (sound\_GSHS\_safety\_mp, sound\_GSHS\_safety\_battery\_mp, module QtslAux.LayerD.MPDDischarge of the semantic-universe strand's auxiliary package) — the same epistemic shape as the staged-row theorem's (sound\_QSafetyBattery\_staged) mixture bridge: an explicit presentation hypothesis, quantifying over exactly the systems whose presentation satisfies it. The martingale-problem theorems are hash-bound below like every other block — their package, the semantic-universe strand's auxiliary package in the public tree, is registered with this catalogue's binder — so the dashboard row carries no conditional mark: each named profile is proven, and its theorems are bound.



At the general stratum the battery's shape is also stated over the semantic universe itself, in the semantic-universe functor package (hash-bound below): trajectories live as closed record sets under the Fell topology, and the undisplayed regularity pack — the measurability the profile proofs *assume* of their region events — restates there as a theorem of the site's topology: the confinement event of a closed safe region is Borel over every trajectory window, with no compactness and no  $\sigma$ -compactness anywhere (measurableSet\_confinedEvent\_cylinder). On that carrier the safety conclusion is a *wall* — a  $\forall$ -face spec: per-window confinement classes are closed under record restriction, their persistence follows from window-calculus transitivity alone, and the judgment bridge is an equivalence — satisfaction *equals* almost-sure confinement at the window (satAtM\_safetySpecF\_iff), with the indicator calculus admitted exactly by the measurability above (pull\_indPenM). Two further site-level facts sharpen the story. Record confinement of a trajectory's closed-up graph to a closed wall is *pointwise* safety at every stage, unconditional in the trajectory (graphCl\_mem\_confinedEvent\_iff) — the segment-constancy side condition a skeleton-based route would carry is absent by construction. And composing two wall demands along a window compositor yields exactly the wall demand of the glued record class (mboxplusWF\_indPenM): wall specs are closed under composition, and persistence survives it. These statements bind the site-general stratum — the acceptance events and spec classes that every profile's conclusion instantiates; the displayed judgment's warrant remains with the profile theorems above.

**notes** Fail-closed as a design reading: the underlying theorem's concluded event is *stronger* than the safety formula's truth set — never fail *and* never leave the modelled domain — and the formula tie then weakens it. The node presents its premises per mode and per jump (mode-indexed certificate families); the bound theorem for that shape assembles the flattened generator row through a sub-convex mixture decomposition of the one-step kernel — the mode's flow step plus the enabled guard jumps — which is the presentation-to-semantics bridge, carried as an explicit hypothesis and quoted in the proof's final remark.

## Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

In the geolog theory the checker loads, every object this rule touches is a declared row, and the display's remaining ambient notation resolves as follows. The concluded formula  $\text{SProbGE}(p, \text{PALways } \varphi)$  (surface syntax  $P_{\geq p}[G\varphi]$ ) is the safety-face cap former  $\text{SProbGE}$  (concluding into the record's  $\forall$ -component sort) applied at the path former  $\text{PALways}$  — the probability-floor former whose threshold canonicity is the range axiom quoted below. The displayed  $\mathcal{M}$  is the paired subject: a `System` element, bound to the claim by the `QSat` pairing (the specification consumes this formula among its safety components; the warrant row names it). The generator reads  $\mathcal{A}_m V_m$  are not free-floating operator applications: `qtsl_gen_form` is the derived generator-form seam — given the subject system and a certificate component (a hypernet from the state space to the nonnegative extended reals), the row names the hypernet saying how the certificate can change with duration, recomputed from the subject's content rows and never trusted as stored data. The battery's E2 payload is the `QSafetyBattery` entity below, field for field the display's premise kit; the conclusion region `qraSublevel(V, y)` is the region-atom constructor of the same name.

theory Qtsl . **SProbGE**

quoted from the theory source at render

```
| SProbGE      : {p: Q, path: PathPred} -> ForallSpec;
```

theory Qtsl . **ax/wf/prob\_range**

quoted from the theory source at render

```
| report ax/wf/prob_range : forall p : Q, ps : PathPred, f : ForallSpec.
|   f = {p: p, path: ps} SProbGE
| - {l: 0, r: p} q_lt, {l: p, r: 1} q_le;
```

theory Qtsl . **PALways**

quoted from the theory source at render

```
| PALways : StatePred -> PathPred;                               // G phi
```

theory Qtsl . **QSafetyBattery**

quoted from the theory source at render

```
| QSafetyBattery : Set;
```

theory Qtsl . **QSafetyBattery/\***

quoted from the theory source at render

```
| QSafetyBattery/proof : QSafetyBattery -> QProof;
| QSafetyBattery/p : QSafetyBattery -> Q;
| QSafetyBattery/arg : QSafetyBattery -> StatePred;
| QSafetyBattery/cert : QSafetyBattery -> QCertFam;
| QSafetyBattery/interiors : QSafetyBattery -> QProofList;
| QSafetyBattery/guards : QSafetyBattery -> QProofList;
| QSafetyBattery/lowbounds : QSafetyBattery -> QProofList;
```

```

QSafetyBattery/nonnegs : QSafetyBattery -> QProofList;
QSafetyBattery/avoid : QSafetyBattery -> QRegionAtom;
QSafetyBattery/scope_warrant : QSafetyBattery -> QInvScope;

```

theory Qtsl.qraSublevel

quoted from the theory source at render

```

graSublevel : {cert: QCertFam, thr: Q} -> QRegionAtom;

```

theory Qtsl.qtsl\_gen\_form

quoted from the theory source at render

```

qtsl_gen_form : {sys: System, cert: HypernetRef, gen: HypernetRef}
-> Prop; // side-condition (oracle discharge / checker

```

## Lean statement

BernsteinLogicAux.Battery.sound\_QSafetyBattery

statement bound

```

/-- **`sound_QSafetyBattery`, discrete profile** (QTSL-rules.toml row `QSafetyBattery`
at the discrete-kernel profile; the concluded judgment reads as region
under-approximation – every start in the region satisfies the formula). Over the
fail-closed form, the formula tie (F) – the avoid region over-approximates the falsity
set of the `G`-argument `φ`, compiled as `U := cml(φ)` – weakens the concluded event to
the truth of `G φ` along the path:
`Pr_{x}[∀ t, X t ≡ φ] ≥ p` for every start `x` in the concluded region
`D n {V ≤ thr}`. No measurability is asked of the satisfaction predicate: the `G φ`
event is bounded below through `measure_mono` on the (measurable) fail-closed event. -/
theorem sound_QSafetyBattery {sat : S → Prop}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV : Measurable V)
  (hF : ∀ y, ¬ sat y → y ∈ U)
  (hθ : p + thr ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ U → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ U → K y (D ∪ U) = 1)
  (hB12int : ∀ y ∈ D, y ∉ U → Integrable V (K y))
  (hB12 : ∀ y ∈ D, y ∉ U → ∫ z, V z ∂K y ≤ V y)
  (hB3 : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V y)
  (hB4 : ∀ y ∈ Dbar, 0 ≤ V y)
  (hx : x ∈ D) (hthr : V x ≤ thr) :
  ENNReal.ofReal p ≤ pathMeasure K x {ω | ∀ n, sat (ω n)}

```

BernsteinLogicAux.Battery.sound\_QSafetyBattery\_failClosed

statement bound

```

/-- **`sound_QSafetyBattery`, discrete profile, fail-closed form** (the discrete-kernel
profile: the hybrid model flattened to one measurable state space with a single one-step
Markov kernel `K`). Hypotheses
are the checked rows (B1)^(B2) as the kernel drift row, (B3) floors, (B4) nonnegs, the
side conditions (R3) (hull: `D ⊆ D̄` and one-step mass in `D̄`) and (R4) (fail-closed:
one-step mass in `D ∪ U`), the threshold arithmetic (θ) `p + thr ≤ 1` (the staged
`q_add`/`q_le` vocabulary, in exact  $\mathbb{Q}$ ), and a start state in the concluded region
`D n {V ≤ thr}` (`qraSublevel`). Conclusion: with probability at least `p` the chain
remains in `D \ U` **forever** – never reaching the avoid region and never leaving the
modeled domain. This is the strong (fail-closed) reading of the concluded `QRegionSat`;
`sound_QSafetyBattery` weakens it through the formula tie (F).

```

The `Integrable V (K y)` hypothesis is the discrete reading of  
"domain membership of `V` in the extended generator's domain"; for the intended  
instances (`V` polynomial, one-step images in the compact  
`D̄` by (R3)) it is automatic. -/

```

theorem sound_QSafetyBattery_failClosed
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV : Measurable V)
  (hθ : p + thr ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ U → K y Dbar = 1)

```

```

(hR4 :  $\forall y \in D, y \notin U \rightarrow K y (D \cup U) = 1$ )
(hB12int :  $\forall y \in D, y \notin U \rightarrow \text{Integrable } V (K y)$ )
(hB12 :  $\forall y \in D, y \notin U \rightarrow \int z, V z \partial K y \leq V y$ )
(hB3 :  $\forall y \in U, y \in Dbar \rightarrow 1 \leq V y$ )
(hB4 :  $\forall y \in Dbar, 0 \leq V y$ )
(hx :  $x \in D$ ) (hthr :  $V x \leq thr$ ) :
ENNReal.ofReal p  $\leq \text{pathMeasure } K \times \{\omega \mid \forall n, \omega n \in D \setminus U\}$ 

```

BernsteinLogicAux.LayerD.Skeleton . sound\_GSHS\_safety\_battery

statement bound

```

/-- **sound_QSafetyBattery**, GSHS instance** (T5; lifetime-relative in the
value-presentation reading – and unconditional for W-Z-free models,
those a.s. free of the two stall phenomena, Zeno accumulation and
vanishing into a never-terminating segment): with the threshold arithmetic `p + thr ≤ 1` and a
start whose certificate is under the threshold, the chain remains outside
the avoid region – at every alive entry *and* every halt value – with
probability at least `p`. -/
theorem sound_GSHS_safety_battery [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {U : Set State} {p thr : ℝ}
  (hV : Measurable V) (hU : MeasurableSet U)
  (hSegD : SegmentDynkin M 0 V) (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) {C : ℝ} (hC0 : 0 ≤ C) (hVC :  $\forall y, V y \leq C$ )
  (hFloor :  $\forall y \in U, 1 \leq V y$ )
  (h0 : p + thr ≤ 1) (x : State) (hthr :  $V x \leq thr$ ) :
ENNReal.ofReal p  $\leq \text{Battery.pathMeasure (skel M) (SkelState.alive x)}$ 
  { $\omega \mid \forall n, \omega n \notin \text{skelBad } U$ }

```

**Discharging the segment field, corner-immune route.** The per-model segment-Dynkin field the statement above consumes is itself a theorem for piecewise-deterministic segments: the statement below derives it from upper-Dini rate rows alone — no differentiability of the certificate along the flow is assumed anywhere, so corner points, absolutely-continuous solutions, and Filippov-style trajectories are in scope exactly as the module's header fences state.

BernsteinLogicAux.LayerD.DiniDischarge . segmentDynkin\_of\_pdmp\_dini

statement bound

```

/-- **[SegD] discharged at Dini strength** for models whose entry states
carry terminating Dini flow data. Mixing with the landed smooth data is
free – the field is a conjunction over states, so discharge is pointwise;
this theorem states the all-Dini case, and mixed cases compose by
`rcases` at the call site exactly as `segmentDynkin_of_pdmp` does. -/
theorem segmentDynkin_of_pdmp_dini (M : SegmentModel State) (0 : ℝ)
  (V : State → ℝ) [IsSFiniteKernel M.jump] (hV : Measurable V)
  (h :  $\forall x : \text{State}, \text{Nonempty (PDMPFlowFinDini M 0 V x)}$ ) :
SegmentDynkin M 0 V

```

BernsteinLogicAux.LayerD.DiniDischarge . pdmp\_dynkin\_core\_dini

statement bound

```

/-- **The corner-immune [SegD] core** : the B1 row at upper-right-Dini
strength – stated in the eventual/liminf-slope form the mean-value fence
consumes, so it survives switch corners, Zeno accumulation of corners,
and (through the AC/Filippov reading) sliding – still bounds the
0-discounted, survival-weighted terminal read plus the accumulated jump
read by the entry value. No derivative of `v` is hypothesized anywhere. -/
theorem pdmp_dynkin_core_dini {v r j : ℝ → ℝ} {0 T : ℝ} (hT : 0 ≤ T)
  (hr : Continuous r) (hj : Continuous j)
  (hv_cont : ContinuousOn v (Icc 0 T))
  (hrow :  $\forall t \in \text{Ico } 0 T, \forall \rho : \mathbb{R}, (r t - 0) * v t - j t < \rho \rightarrow$ 
 $\exists^\dagger z \text{ in } \mathcal{M}[>] t, \text{slope } v t z < \rho$ ) :
jint r j 0 T + survWeight r 0 T * v T  $\leq v 0$ 

```

**Discharging the segment field, diffusion route.** For jump-diffusion segments the same field follows from a single quarantined per-model input — the sampled-supermartingale inequality — stated one stratum below as an interface structure; the theorem below converts any inhabitant into the segment-Dynkin field, with a deterministic positive control included in the module.

BernsteinLogicAux.LayerD.ItoInterface . segmentDynkin\_of\_ito

statement bound

```

/-- **[SegD] discharged for the diffusion profile against the stated
interface** (assault doc §3 row 1(a)): a model all of whose entry states

```

```

carry Itô-interface data satisfies the `SegmentDynkin` hypothesis field.
Mixing with PDMP/Dini data is free at call sites, exactly as for
`segmentDynkin_of_pdmp_dini` – the field is a conjunction over states,
discharged pointwise. -/
theorem segmentDynkin_of_ito (M : SegmentModel State) (θ : ℝ)
  (V : State → ℝ) [IsSFiniteKernel M.jump] (hV : Measurable V)
  (h : ∀ x : State, Nonempty (ItoInterface M θ V x)) :
  SegmentDynkin M θ V

```

**The interface's own discharge.** The quarantined inequality is in turn delivered by optional sampling: from a supermartingale property of the certificate process (plus the début-mesurability family), the assembly below inhabits the full interface — leaving, per diffusion entry state, exactly the supermartingale property and the début family. The blocks that follow discharge each in turn.

BernsteinLogicAux.LayerD.ItoSampling . **ItoInterface.ofSupermartingale**

statement bound

```

/-- **The interface, discharged down to the supermartingale property**:
a full `ItoInterface` instance from the non-analytic pathwise data plus
`Supermartingale` + the début family. Composing with
`segmentDynkin_of_ito`, the chain
`Supermartingale + début family = [SegD]` is Lean end-to-end; what a
concrete diffusion model still owes is exactly the supermartingale
property (theorem 6's shape / upstream Itô). -/
noncomputable def ItoInterface.ofSupermartingale
  (M : SegmentModel State) [IsProbabilityMeasure P]
  (hT : ∀ ω, 0 ≤ T ω)
  (hk : ∀ ω, k ω = ExitKind.guard v k ω = ExitKind.halt
    v k ω = ExitKind.live)
  (hV : ∀ ω, Measurable (γ ω))
  (hvc : ∀ ω, Continuous fun u => V (γ ω u))
  (hVγ : ∀ ω u, 0 ≤ V (γ ω u))
  (hr : ∀ ω, Continuous (r ω)) (hr0 : ∀ ω u, 0 ≤ r ω u)
  (hjr : ∀ ω, Continuous (jr ω)) (hjr0 : ∀ ω u, 0 ≤ jr ω u)
  (hjr_couple : ∀ ω, ∀ u ∈ segDom (T ω) (k ω),
    ENNReal.ofReal (jr ω u)
      = ENNReal.ofReal (r ω u)
      * ∫- z, ENNReal.ofReal (V z) ∂(M.jump (γ ω u, ExitKind.alarm)))
  (hread : ∫- o, outcomeRead M θ V o ∂(M.seg x)
    ≤ ∫- ω, ∫- o, outcomeRead M θ V o
      ∂(segMeasureIto (γ ω) (r ω) (T ω) (k ω)) ∂P)
  (hy0 : ∀ ω, γ ω 0 = x)
  (hZ : Supermartingale (certProcess θ V γ r jr) ℱ P)
  (hτ : ∀ s : ℝ≥0, MeasurableSet[ℱ s]
    {ω | exitDebut T k ω ≤ (s : ℝ≥0∞)}) :
  ItoInterface M θ V x

```

BernsteinLogicAux.LayerD.ItoSampling . **sampled\_le\_of\_supermartingale**

statement bound

```

/-- **Partial discharge of (A.1)** from the banked optional-sampling
suite: the supermartingale property of the certificate process (the
[WR-2]/[WR-3] residue) plus the exit début's measurability family imply
the `hsampled` field, with the start pinned at `x` and `P` a probability
measure supplying `Y₀ = V x`. -/
theorem sampled_le_of_supermartingale [IsProbabilityMeasure P]
  (hT : ∀ ω, 0 ≤ T ω)
  (hvc : ∀ ω, Continuous fun u => V (γ ω u))
  (hVγ : ∀ ω u, 0 ≤ V (γ ω u))
  (hr : ∀ ω, Continuous (r ω)) (hjr : ∀ ω, Continuous (jr ω))
  (hjr0 : ∀ ω u, 0 ≤ jr ω u)
  (hy0 : ∀ ω, γ ω 0 = x)
  (hZ : Supermartingale (certProcess θ V γ r jr) ℱ P)
  (hτ : ∀ s : ℝ≥0, MeasurableSet[ℱ s]
    {ω | exitDebut T k ω ≤ (s : ℝ≥0∞)}) :
  ∀ t : ℝ, 0 ≤ t →
  ∫- ω, sampledRead θ V (γ ω) (r ω) (jr ω) (T ω) (k ω) t ∂P
    ≤ ENNReal.ofReal (V x)

```

**The residue, bundled.** The per-model inputs the two routes leave open are packaged as a structure whose only analytic fields are the certificate-process supermartingale property and the exit début's measurability family; the theorems below convert per-entry-state families of such bundles (all-diffusion, or a per-state diffusion/Dini mixture) into the segment field, and the class constructor builds the battery's Ville class with that field proven rather than assumed.

BernsteinLogicAux.GshsItoDischarge . **ItoSupermartingaleData**

statement bound

```

/-- **The supermartingale-floor bundle**: exactly
`ItoInterface.ofSupermartingale`'s inputs as a structure, so that
per-entry-state families can be quantified. The two analytic fields are
`hZ` and `ht`; everything else is the pathwise bookkeeping the interface
already carried. -/
structure ItoSupermartingaleData (M : SegmentModel State) (θ : ℝ)
  (V : State → ℝ) (x : State) where
  /-- the driving path-data space -/
  Ω : Type*
  /-- its σ-algebra -/
  [mΩ : MeasurableSpace Ω]
  /-- the law of the driving data issued from `x` -/
  P : Measure Ω
  /-- `P` is a probability measure (`Y₀ = V x` is the model's business –
  the optional-sampling route consumes this) -/
  hprob : IsProbabilityMeasure P
  /-- the filtration the certificate process rides -/
  ℱ : Filtration ℝ≥0 mΩ
  /-- exit time of each path (unread on live paths) -/
  T : Ω → ℝ
  hT : ∀ ω, 0 ≤ T ω
  /-- terminator species of each path -/
  k : Ω → ExitKind
  hk : ∀ ω, k ω = ExitKind.guard v k ω = ExitKind.halt
    v k ω = ExitKind.live
  /-- the trajectory of each path (clamped extension off the segment) -/
  γ : Ω → ℝ → State
  hγ : ∀ ω, Measurable (γ ω)
  hvc : ∀ ω, Continuous fun u => V (γ ω u)
  hVγ : ∀ ω u, 0 ≤ V (γ ω u)
  /-- paths start at the entry state -/
  hγ0 : ∀ ω, γ ω 0 = x
  /-- alarm intensity along each path -/
  r : Ω → ℝ → ℝ
  hr : ∀ ω, Continuous (r ω)
  hr0 : ∀ ω u, 0 ≤ r ω u
  /-- λ-weighted post-jump read along each path -/
  jr : Ω → ℝ → ℝ
  hjr : ∀ ω, Continuous (jr ω)
  hjr0 : ∀ ω u, 0 ≤ jr ω u
  hjr_couple : ∀ ω, ∀ u ∈ segDom (T ω) (k ω),
    ENNReal.ofReal (jr ω u)
    = ENNReal.ofReal (r ω u)
    * ∫- z, ENNReal.ofReal (V z) ∂(M.jump (γ ω u, ExitKind.alarm))
  /-- the read-level disintegration of `M.seg x` along paths
  (definitional for `Rep`-constructed models) -/
  hread : ∫- o, outcomeRead M θ V o ∂(M.seg x)
    ≤ ∫- ω, ∫- o, outcomeRead M θ V o
      ∂(segMeasureIto (γ ω) (r ω) (T ω) (k ω)) ∂P
  /-- **analytic field 1**: the certificate process is a supermartingale
  – the exact [WR-2]/[WR-3] residue -/
  hZ : Supermartingale (certProcess θ V γ r jr) ℱ P
  /-- **analytic field 2**: the exit début's measurability family -/
  ht : ∀ s : ℝ≥0, MeasurableSet[ℱ s]
    {ω | exitDebut T k ω ≤ (s : ℝ≥0∞)}

```

BernsteinLogicAux.GshsItoDischarge . **segmentDynkin\_of\_supermartingale**

statement bound

```

/-- **[SegD] from the supermartingale floor, all-diffusion form**: a
model all of whose entry states carry supermartingale-floor bundles

```

```
satisfies the `SegmentDynkin` hypothesis field. -/
theorem segmentDynkin_of_supermartingale (M : SegmentModel State) (θ : ℝ)
  (V : State → ℝ) [IsSFiniteKernel M.jump] (hV : Measurable V)
  (h : ∀ x : State, Nonempty (ItoSupermartingaleData M θ V x)) :
  SegmentDynkin M θ V
```

BernsteinLogicAux.GshsItoDischarge . **segmentDynkin\_of\_mixed**

statement bound

```
/-- **The pointwise diffusion/Dini mixture, enacted** (both discharge
docstrings promised this "free at call sites"; here it is as one
theorem): each entry state carries either a supermartingale-floor bundle
(diffusion segments) or terminating Dini flow data (deterministic/PDMP
segments, switch corners included) – mixed freely, [SegD] follows. Two
disjuncts only: Dini is the deterministic floor (smooth data restates at
it, or lifts by `dini_row_of_deriv_row`). -/
theorem segmentDynkin_of_mixed (M : SegmentModel State) (θ : ℝ)
  (V : State → ℝ) [IsSFiniteKernel M.jump] (hV : Measurable V)
  (h : ∀ x : State, Nonempty (ItoSupermartingaleData M θ V x)
    ∨ Nonempty (PDMPFlowFinDini M θ V x)) :
  SegmentDynkin M θ V
```

BernsteinLogicAux.GshsItoDischarge . **GshsSegmentVille.ofDischarge**

statement bound

```
/-- **The discharge constructor – `gshsGen:segment-dynkin`'s diffusion
sub-case retired to the floor**: the rung's class with its quarantined
field PROVEN from row-conditional per-state discharge data, never
assumed. `hdata`'s conditional shape mirrors the field's own: the
certified θ-discounted generator rows are where a concrete model's
analytic side takes its input (Itô's formula consumes `λV ≤ 0` on the
cells to prove `hZ`; the Dini row data is the compiled rows themselves).
What a model owes per diffusion entry state is exactly `hZ` + `ht` – the
`sdeGen:dynkin-supermartingale` route-(i) floor. -/
noncomputable def GshsSegmentVille.ofDischarge
  [MeasurableSpace Mode] [DecidableEq Mode]
  {P : GshsProfile n m T Mode} {V : Mode → State n → ℝ}
  {gradV : Mode → State n → State n}
  {hessV : Mode → State n → Fin n → Fin n → ℝ} {θ : ℝ}
  (M : LayerD.SegmentModel (Flat n Mode)) [IsSFiniteKernel M.jump]
  (rd : Flat n Mode → Flat n Mode)
  (jump_guard_det : ∀ p, M.jump (p, LayerD.ExitKind.guard)
    = Measure.dirac (rd p))
  (dispatch_cover : ∀ p : Flat n Mode,
    (∃ e : Fin T, (P.trans e).forced = true ∧ (P.trans e).src = p.1
      ∧ p.2 ∈ (P.trans e).guard
      ∧ rd p = ((P.trans e).dst, (P.trans e).reset p.2))
    ∨ rd p = p)
  (hV : Measurable (flatCert V))
  (hdata : (∀ mo : Mode, ∀ z ∈ P.cell mo,
    θ * V mo z + generator P V gradV hessV mo z ≤ 0) →
    ∀ p : Flat n Mode,
    Nonempty (LayerD.ItoSupermartingaleData M θ (flatCert V) p)
    ∨ Nonempty (LayerD.PDMPFlowFinDini M θ (flatCert V) p)) :
  GshsSegmentVille P V gradV hessV θ
```

**Discharging the début family.** The second analytic field is a library application: exit from a closed certified cell is the début of the open complement, and the début of an open set under right-continuous paths is a stopping time of the right-continuous filtration — no completion, no metric, no per-model measure theory.

BernsteinLogicAux.Debut . **isStoppingTime\_debut**

statement bound

```
/-- The début of an open set by a right-continuous adapted process is a
bona-fide mathlib **stopping time** for any right-continuous filtration
(`ℝ≥0∞ := WithTop ℝ≥0` definitionally, so no translation layer): the pin's
`isStoppingTime_of_measurableSet_lt_of_isRightContinuous` upgrades the
`<-family. This is the wide-sense-to-strict step the census's G3 entry
priced; it costs an instance, never a per-model proof – models on a raw
```

```

filtration `F` use `F.rightCont` (`F`), whose `IsRightContinuous`
instance mathlib provides. -/
theorem isStoppingTime_debut [F.IsRightContinuous]
  (hX :  $\forall t : \mathbb{R}_{\geq 0}, \text{Measurable}[F\ t] (X\ t)$ )
  (hrc :  $\forall \omega, \forall s : \mathbb{R}_{\geq 0}, \text{ContinuousWithinAt} (\text{fun } u \Rightarrow X\ u\ \omega) (\text{Set.Ici } s)\ s$ )
  (hU : IsOpen U) :
  IsStoppingTime F (debut X U)

```

BernsteinLogicAux.Debut.exitDebut\_le\_family\_of\_isClosed\_cell

statement bound

```

/-- The certified-cell instantiation: exit from a CLOSED cell (the
box-union geometry – finite unions of closed boxes are closed by
`isClosed_Icc`/`isClosed_set_pi` + `Set.Finite.isClosed_biUnion`, one line
per model) is the début of the open complement, so the `ht` field is a
library application. -/
theorem exitDebut_le_family_of_isClosed_cell [F.IsRightContinuous]
  {T :  $\Omega \rightarrow \mathbb{R}$ } {k :  $\Omega \rightarrow \text{ExitKind}$ } {X :  $\mathbb{R}_{\geq 0} \rightarrow \Omega \rightarrow E$ } {cell : Set E}
  (hcell : IsClosed cell)
  (hEq :  $\forall \omega, \text{exitDebut } T\ k\ \omega = \text{debut } X\ \text{cell}^c\ \omega$ )
  (hX :  $\forall t : \mathbb{R}_{\geq 0}, \text{Measurable}[F\ t] (X\ t)$ )
  (hrc :  $\forall \omega, \forall s : \mathbb{R}_{\geq 0}, \text{ContinuousWithinAt} (\text{fun } u \Rightarrow X\ u\ \omega) (\text{Set.Ici } s)\ s$ ) :
   $\forall s : \mathbb{R}_{\geq 0}, \text{MeasurableSet}[F\ s] \{ \omega \mid \text{exitDebut } T\ k\ \omega \leq (s : \mathbb{R}_{\geq 0\infty}) \}$ 

```

**Discharging the supermartingale property, generic half.** The first analytic field's martingale theory is also a library application: the exponential-discounting theorem produces the required supermartingale from a martingale-problem hypothesis (the compensated certificate process is a martingale — what Itô's formula and the interior rows establish per model, with no stochastic integration in the certified statement), the bounded-horizon and horizon-free optional-sampling inequalities are the theorems the interface discharge above consumes, and the continuous-time Doob/Ville maximal inequality converts supermartingale level crossings into probability bounds. What remains per model is exactly the martingale-problem hypothesis and the uniform bounds it consumes.

BernsteinLogicAux.OptionalStoppingRC.supermartingale\_exp\_discount

statement bound

```

/-- **Exponential discounting of a drift-dominated process.** Suppose
`W :  $\mathbb{R}_{\geq 0} \rightarrow \Omega \rightarrow \mathbb{R}$ ` is adapted with continuous paths, `g :  $\mathbb{R}_{\geq 0} \rightarrow \Omega \rightarrow \mathbb{R}$ ` is
jointly measurable, both are uniformly bounded, the compensated process
`fun t  $\omega \Rightarrow W\ t\ \omega - \int s \text{ in } (0 : \mathbb{R})..(t : \mathbb{R}), g\ s.\text{toNNReal } \omega` is a
martingale (the martingale-problem formulation of `g` being the generator
drift of `W`), and `g t  $\omega \leq c * W\ t\ \omega` pointwise. Then

`fun t  $\omega \Rightarrow \text{Real.exp } (-c * t) * W\ t\ \omega` is a supermartingale.

This is the discounting step of Dynkin-formula arguments, carried out with
pathwise Riemann–Stieltjes integration by parts against the `C` function
`s  $\mapsto \text{exp } (-c * s)$ ` and a conditional Fubini argument – no stochastic
integral is involved. -/
theorem supermartingale_exp_discount
  [IsProbabilityMeasure  $\mu$ ]
  (W :  $\mathbb{R}_{\geq 0} \rightarrow \Omega \rightarrow \mathbb{R}$ ) (g :  $\mathbb{R}_{\geq 0} \rightarrow \Omega \rightarrow \mathbb{R}$ ) (c :  $\mathbb{R}$ ) (C :  $\mathbb{R}$ )
  (hWadp : Adapted F W)
  (hWcont :  $\forall \omega, \text{Continuous } \text{fun } t \Rightarrow W\ t\ \omega$ )
  (hWb :  $\forall t\ \omega, |W\ t\ \omega| \leq C$ )
  (hgmeas : Measurable (Function.uncurry g))
  (hgb :  $\forall t\ \omega, |g\ t\ \omega| \leq C$ )
  (hmart : Martingale
    (fun (t :  $\mathbb{R}_{\geq 0}$ ) ( $\omega : \Omega$ ) =>
       $W\ t\ \omega - \int s \text{ in } (0 : \mathbb{R})..(t : \mathbb{R}), g\ s.\text{toNNReal } \omega$ ) F  $\mu$ )
  (hdom :  $\forall t\ \omega, g\ t\ \omega \leq c * W\ t\ \omega$ ) :
  Supermartingale
    (fun (t :  $\mathbb{R}_{\geq 0}$ ) ( $\omega : \Omega$ ) =>  $\text{Real.exp } (-c * (t : \mathbb{R})) * W\ t\ \omega$ ) F  $\mu$$$$ 
```

BernsteinLogicAux.OptionalStoppingRC.lintegral\_stoppedValue\_min\_le

statement bound

```

/-- The upper dyadic approximation to a nonnegative real. -/
noncomputable def dyadicCeil (n :  $\mathbb{N}$ ) (x :  $\mathbb{R}_{\geq 0}$ ) :  $\mathbb{N} :=
  \lceil x * (2 ^ n : \mathbb{R}_{\geq 0}) \rceil +$ 
```



```

lemma dyadicCeil_time_tendsto (x : ℝ≥0) :
  Filter.Tendsto (fun n => dyadicTime n (dyadicCeil n x)) Filter.atTop (nhds x) := by
  -- Use the fact that  $x \lfloor \log_2 x \rfloor \leq x \leq x \lceil \log_2 x \rceil$ 
  -- Use the fact that  $x \lfloor \log_2 x \rfloor \leq x \leq x \lceil \log_2 x \rceil$ 
  have h_bounds : ∀ n : ℕ, x ≤ (dyadicCeil n x : ℝ≥0) / 2^n ∧ (dyadicCeil n x : ℝ≥0) / 2^n ≤ x + 1 / 2^n := by
    intro n
    simp [dyadicCeil, dyadicTime];
    rw [le_div_iff, div_le_iff] <|> norm_num;
    exact (Nat.le_ceil _, by rw [add_mul, inv_mul_cancel (by positivity)] ; exact_mod_cast Nat.ceil_lt_add_one (by
    refine' tendsto_of_tendsto_of_tendsto_of_le_of_le' tendsto_const_nhds _ (Filter.Eventually.of_forall fun n => h_bounds
    convert tendsto_const_nhds.add (tendsto_inv_atTop_zero.comp (tendsto_pow_atTop_atTop_of_one_lt (show (1 : ℝ≥0) < 2
    exacts [rfl, by rw [add_zero]] ] ]

lemma dyadicCeil_time_mem_Ici (n : ℕ) (x : ℝ≥0) :
  dyadicTime n (dyadicCeil n x) ∈ Set.Ici x := by
  unfold dyadicTime dyadicCeil;
  exact Set.mem_Ici.mpr (by rw [le_div_iff (by positivity)] ; exact Nat.le_ceil _)

lemma dyadicCeil_le_iff (n k : ℕ) (x : ℝ≥0) :
  dyadicCeil n x ≤ k ↔ x ≤ dyadicTime n k := by
  rw [dyadicCeil, Nat.ceil_le, dyadicTime, le_div_iff (by positivity)]

lemma dyadicCeil_stoppingTime
  (τ : Ω → ℝ≥0) (hτ : ∀ t : ℝ≥0, MeasurableSet[ℱ t] {ω | τ ω ≤ t})
  (T : ℝ≥0) (n : ℕ) :
  IsStoppingTime (dyadicFiltration (ℱ := ℱ) n)
  (fun ω => (dyadicCeil n (min (τ ω) T) : ℕ)) := by
  intro k
  by_cases hT : T ≤ dyadicTime n k
  · refine MeasurableSet.univ.congr ?_
    refine (Set.eq_univ_of_forall fun ω => ?_).symm
    show (dyadicCeil n (min (τ ω) T) : ℕ) ≤ (k : ℕ)
    exact_mod_cast (dyadicCeil_le_iff n k (min (τ ω) T)).mpr ((min_le_right _ _).trans hT)
  · refine (hτ (dyadicTime n k)).congr ?_
    ext ω
    simp only [Set.mem_setOf_eq]
    constructor
    · intro h
      show (dyadicCeil n (min (τ ω) T) : ℕ) ≤ (k : ℕ)
      exact_mod_cast (dyadicCeil_le_iff n k (min (τ ω) T)).mpr ((min_le_left _ _).trans h)
    · intro h
      have h' : (dyadicCeil n (min (τ ω) T) : ℕ) ≤ (k : ℕ) := h
      have hnk : dyadicCeil n (min (τ ω) T) ≤ k := by exact_mod_cast h'
      have := (dyadicCeil_le_iff n k (min (τ ω) T)).mp hnk
      exact ((min_le_iff.mpr this).resolve_right hT)

lemma dyadicCeil_bounded (τ : Ω → ℝ≥0) (T : ℝ≥0) (n : ℕ) (ω : Ω) :
  (dyadicCeil n (min (τ ω) T) : ℕ) ≤ dyadicCeil n T := by
  exact_mod_cast Nat.ceil_mono (mul_le_mul_of_nonneg_right (min_le_right _ _)) (by positivity)

lemma integral_dyadic_stopped_le
  [IsProbabilityMeasure μ]
  (Z : ℝ≥0 → Ω → ℝ) (hZ : Supermartingale Z ℱ μ)
  (τ : Ω → ℝ≥0)
  (hτ : ∀ t : ℝ≥0, MeasurableSet[ℱ t] {ω | τ ω ≤ t})
  (T : ℝ≥0) (n : ℕ) :
  ∫ ω, Z (dyadicTime n (dyadicCeil n (min (τ ω) T))) ω dμ
  ≤ ∫ ω, Z 0 ω dμ := by
  have h_submartingale :
    Submartingale (fun k => -Z (dyadicTime n k)) (dyadicFiltration (ℱ := ℱ) n) μ :=
    (supermartingale_dyadic Z hZ n).neg
  have h_stopping_time : IsStoppingTime (dyadicFiltration (ℱ := ℱ) n)
    (fun ω => (dyadicCeil n (min (τ ω) T) : ℕ)) := dyadicCeil_stoppingTime τ hτ T n
  have h0 : IsStoppingTime (dyadicFiltration (ℱ := ℱ) n) (fun _ : Ω => ((0 : ℕ) : ℕ)) := by
    intro k
    refine MeasurableSet.univ.congr ?_

```

```

    refine (Set.eq_univ_of_forall fun ω => ?_).symm
    show ((0 : ℕ) : ℕ∞) ≤ (k : ℕ∞)
    exact_mod_cast Nat.zero_le k
  have key := h_submartingale.expected_stoppedValue_mono h0 h_stopping_time
  (fun ω => by
    show ((0 : ℕ) : ℕ∞) ≤ (dyadicCeil n (min (τ ω) T) : ℕ∞)
    exact_mod_cast Nat.zero_le _)
  (N := dyadicCeil n T) (fun ω => by exact_mod_cast dyadicCeil_bounded τ T n ω)
  have hL : stoppedValue (fun k => -Z (dyadicTime n k)) (fun _ : Ω => ((0 : ℕ) : ℕ∞))
    = fun ω => -Z (dyadicTime n 0) ω := rfl
  have hd0 : dyadicTime n 0 = 0 := by simp [dyadicTime]
  have hR : stoppedValue (fun k => -Z (dyadicTime n k))
    (fun ω => (dyadicCeil n (min (τ ω) T) : ℕ∞))
    = fun ω => -Z (dyadicTime n (dyadicCeil n (min (τ ω) T))) ω := rfl
  rw [hL, hR, hd0] at key
  simp only [integral_neg] at key
  exact neg_le_neg_iff.mp key

lemma integrable_dyadic_stopped
  [IsProbabilityMeasure μ]
  (Z : ℝ≥0 → Ω → ℝ) (hZ : Supermartingale Z ℱ μ)
  (τ : Ω → ℝ≥0)
  (ht : ∀ t : ℝ≥0, MeasurableSet[ℱ t] {ω | τ ω ≤ t})
  (T : ℝ≥0) (n : ℕ) :
  Integrable (fun ω => Z (dyadicTime n (dyadicCeil n (min (τ ω) T))) ω) μ := by
  have h_sub : Submartingale (fun k => -Z (dyadicTime n k)) (dyadicFiltration (ℱ := ℱ) n) μ :=
    (supermartingale_dyadic Z hZ n).neg
  have h_int := h_sub.integrable_stoppedValue (dyadicCeil_stoppingTime τ ht T n)
  (N := dyadicCeil n T) (fun ω => by exact_mod_cast dyadicCeil_bounded τ T n ω)
  have hR : stoppedValue (fun k => -Z (dyadicTime n k))
    (fun ω => (dyadicCeil n (min (τ ω) T) : ℕ∞))
    = fun ω => -Z (dyadicTime n (dyadicCeil n (min (τ ω) T))) ω := rfl
  rw [hR] at h_int
  simp using h_int.neg

lemma dyadic_stopped_tendsto
  (Z : ℝ≥0 → Ω → ℝ)
  (hZrc : ∀ ω, ∀ t : ℝ≥0,
    ContinuousWithinAt (fun s => Z s ω) (Set.Ici t) t)
  (τ : Ω → ℝ≥0) (T : ℝ≥0) (ω : Ω) :
  Filter.Tendsto
    (fun n => Z (dyadicTime n (dyadicCeil n (min (τ ω) T))) ω)
    Filter.atTop (nhds (Z (min (τ ω) T) ω)) := by
  refine' Filter.Tendsto.comp (hZrc ω (min (τ ω) T)) _;
  rw [tendsto_nhdsWithin_iff];
  exact (dyadicCeil_time_tendsto _, Filter.Eventually.of_forall fun n => dyadicCeil_time_mem_Ici _ _)

/-
**Optional sampling inequality at a bounded horizon.** For a nonnegative
supermartingale `Z` over `ℝ≥0` with right-continuous paths, a stopping time
`τ` (given by its measurability family) and a deterministic horizon `T`,


$$\int^- \omega, \text{ENNReal.ofReal } (Z (\min (\tau \omega) T) \omega) \partial \mu \leq \int^- \omega, \text{ENNReal.ofReal } (Z 0 \omega) \partial \mu.$$


Stated with the lower Lebesgue integral so that no integrability of the
stopped value needs to be assumed.
-/
theorem lintegral_stoppedValue_min_le
  [IsProbabilityMeasure μ]
  (Z : ℝ≥0 → Ω → ℝ) (hZ : Supermartingale Z ℱ μ)
  (hZnn : ∀ t ω, 0 ≤ Z t ω)
  (hZrc : ∀ ω, ∀ t : ℝ≥0,
    ContinuousWithinAt (fun s => Z s ω) (Set.Ici t) t)
  (τ : Ω → ℝ≥0)
  (ht : ∀ t : ℝ≥0, MeasurableSet[ℱ t] {ω | τ ω ≤ t})
  (T : ℝ≥0) :

```

$$\int^- \omega, \text{ENNReal.ofReal } (Z (\min (\tau \ \omega) \ T) \ \omega) \ \partial \mu \\ \leq \int^- \omega, \text{ENNReal.ofReal } (Z \ 0 \ \omega) \ \partial \mu$$

BernsteinLogicAux.OptionalStoppingRC.lintegral\_stoppedValue\_le

statement bound

```

/-- Truncate a wide-sense nonnegative stopping time at a finite horizon. -/
noncomputable def ennrealTrunc (τ : Ω → ℝ≥0∞) (T : ℝ≥0) (ω : Ω) : ℝ≥0 :=
  if τ ω ≤ (T : ℝ≥0∞) then (τ ω).toNNReal else T

lemma ennrealTrunc_le_iff (τ : Ω → ℝ≥0∞) (T t : ℝ≥0) (ht : t < T) :
  {ω | ennrealTrunc τ T ω ≤ t} = {ω | τ ω ≤ (t : ℝ≥0∞)} := by
  ext ω;
  by_cases h : τ ω ≤ T <|> simp +decide [ h, ennrealTrunc ];
  · cases h' : τ ω <|> aesop;
  · exact iff_of_false ( not_le_of_gt ht ) ( fun h' => h <| le_trans h' <| mod_cast ht.le )

lemma ennrealTrunc_eq_of_le (τ : Ω → ℝ≥0∞) (T : ℝ≥0) (ω : Ω)
  (h : τ ω ≤ (T : ℝ≥0∞)) :
  ennrealTrunc τ T ω = (τ ω).toNNReal := by
  exact if_pos h

/-
**Optional sampling inequality, horizon-free form.** For a nonnegative
right-continuous supermartingale `Z` and a wide-sense stopping time `τ` with
values in `ℝ≥0∞` (given by its measurability family), the contribution of
the event `{τ < ∞}` satisfies

`∫^- ω, (if τ ω = ∞ then 0 else ENNReal.ofReal (Z (τ ω).toNNReal ω)) ∂μ
≤ ∫^- ω, ENNReal.ofReal (Z 0 ω) ∂μ`.

Follows from `lintegral_stoppedValue_min_le` by monotone convergence along
an increasing sequence of horizons, using nonnegativity to discard the
`{τ > T}` remainder.
-/
theorem lintegral_stoppedValue_le
  [IsProbabilityMeasure μ]
  (Z : ℝ≥0 → Ω → ℝ) (hZ : Supermartingale Z ℱ μ)
  (hZnn : ∀ t ω, 0 ≤ Z t ω)
  (hZrc : ∀ ω, ∀ t : ℝ≥0,
    ContinuousWithinAt (fun s => Z s ω) (Set.Ici t) t)
  (τ : Ω → ℝ≥0∞)
  (ht : ∀ t : ℝ≥0, MeasurableSet[ℱ t] {ω | τ ω ≤ (t : ℝ≥0∞)}) :
  ∫^- ω, (if τ ω = ∞ then 0
    else ENNReal.ofReal (Z (τ ω).toNNReal ω)) ∂μ
  ≤ ∫^- ω, ENNReal.ofReal (Z 0 ω) ∂μ

```

BernsteinLogicAux.OptionalStoppingRC.measure\_sup\_ge\_le\_of\_supermartingale

statement bound

```

/-- Restrict a supermartingale along a monotone map of time indices. -/
private lemma Supermartingale.comp_monotone
  {ι κ : Type*} [Preorder ι] [Preorder κ]
  {ℱι : Filtration ι m0} {Z : ι → Ω → ℝ}
  (hZ : Supermartingale Z ℱι μ) (f : κ → ι) (hf : Monotone f) :
  Supermartingale (fun k ↦ Z (f k))
  ({ seq := fun k ↦ ℱι (f k)
    mono' := fun _ _ h ↦ ℱι.mono (hf h)
    le' := fun k ↦ ℱι.le (f k) } : Filtration κ m0) μ := by
  exact (fun k ↦ hZ.stronglyAdapted (f k),
    fun i j hij ↦ hZ.condExp_ae_le (hf hij), fun k ↦ hZ.integrable (f k))

/-
A continuous path hitting `c` on `[0,T]` is approximated on uniform grids
at every strictly smaller level.
-/
private lemma exists_grid_hit_of_continuous
  (z : ℝ≥0 → ℝ) (hz : Continuous z) (T : ℝ≥0) (c d : ℝ) (hdc : d < c)

```

```

(h :  $\exists s : \mathbb{R}_{\geq 0}, s \leq T \wedge c \leq z s$ ) :
 $\exists m : \mathbb{N}, 0 < m \wedge \exists k \leq m,$ 
 $d \leq z ((k : \mathbb{R}) * T / m, \text{by positivity}) : \mathbb{R}_{\geq 0}$ ) := by
by_contra h_contra;
obtain ( s, hs1, hs2 ) := h;
-- By the continuity of  $z$ , for any  $\epsilon > 0$ , there exists a  $\delta > 0$  such that  $|z(t) - z(s)| < \epsilon$  w
obtain (δ, hδ_pos, hδ) :  $\exists \delta > 0, \forall t : \mathbb{R}_{\geq 0}, |t - s| < \delta \rightarrow |z t - z s| < (c - d) / 2$  := by
obtain (δ, hδ, hδ') :=
Metric.continuous_iff.mp hz s ((c - d) / 2) (half_pos (sub_pos.mpr hdc))
refine (δ, hδ, fun t ht  $\mapsto$  ?_)
simp only [Real.dist_eq] using hδ' t (by simp only [NNReal.dist_eq] using ht);
-- Choose  $m$  large enough such that  $T/m < \delta$ .
obtain (m, hm_pos, hm) :  $\exists m : \mathbb{N}, 0 < m \wedge (T : \mathbb{R}) / m < \delta$  := by
exact (  $\lfloor T / \delta \rfloor + 1, \text{Nat.succ_pos } \_, \text{by rw [div_lt_iff]} <=> \text{push\_cast } <=> \text{nlinarith [Nat.lt\_floor\_add\_one (T /$ 
-- Choose  $k$  such that  $k * T / m$  is within  $\delta$  of  $s$ .
obtain (k, hk) :  $\exists k : \mathbb{N}, k \leq m \wedge |(k * T / m : \mathbb{R}) - s| < \delta$  := by
refine' (  $\lfloor (s : \mathbb{R}) * m / T \rfloor + \_, \_$  );
· exact Nat.floor_le_of_le (div_le_of_le_mul (by positivity) (by positivity) (by nlinarith [show (s :  $\mathbb{R}$ )  $\leq$ 
· by_cases hT :  $T = 0$  <=> simp_all +decide [div_lt_iff];
rw [abs_lt] ; constructor <=> nlinarith [Nat.floor_le (show  $0 \leq (s : \mathbb{R}) * m / T$  by positivity), Nat.lt\_floor,
exact h_contra ( m, hm_pos, k, hk.1, by linarith [abs_lt.mp (hδ (k * T / m, by positivity) hk.2) ] ) )
/-
**Doob-type maximal inequality, continuous time.** For a nonnegative
supermartingale  $Z$  over the index set  $\mathbb{R}_{\geq 0}$  with continuous paths, the
probability that the running supremum over  $[0, T]$  reaches a level  $c > 0$ 
is at most  $(\int \omega, Z_0 \omega \, d\mu) / c$ .
-/
theorem measure_sup_ge_le_of_supermartingale
[IsProbabilityMeasure  $\mu$ ]
( $Z : \mathbb{R}_{\geq 0} \rightarrow \Omega \rightarrow \mathbb{R}$ ) (hZ : Supermartingale  $Z \mathcal{F} \mu$ )
(hZnn :  $\forall t \omega, 0 \leq Z t \omega$ )
(hZcont :  $\forall \omega, \text{Continuous fun } t \Rightarrow Z t \omega$ )
( $T : \mathbb{R}_{\geq 0}$ ) ( $c : \mathbb{R}$ ) (hc :  $0 < c$ ) :
 $\mu \{ \omega \mid \exists s : \mathbb{R}_{\geq 0}, s \leq T \wedge c \leq Z s \omega \}$ 
 $\leq \text{ENNReal.ofReal } ((\int \omega, Z_0 \omega \, d\mu) / c)$ 

```

**The martingale-problem profile, end to end.** The general continuous-time theorems: the inter-jump motion enters through its generator —  $\mathcal{A}V$  as data, tied to the process only by the martingale hypothesis for the discounted compensated certificate process — the segment field is derived rather than assumed, and the battery conclusion follows for every system whose presentation satisfies the martingale-problem hypothesis.

QtslAux.LayerD.MPDischarge . **sound\_GSHS\_safety\_mp**

statement bound

```

/-- Safety (avoidance) at the martingale-problem profile: the probability
that the segment skeleton ever enters the bad set is at most  $\int V x$ . -/
theorem sound_GSHS_safety_mp [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
{V : State  $\rightarrow$   $\mathbb{R}$ } {U : Set State} (hV : Measurable V) (hU : MeasurableSet U)
(hmp :  $\forall x : \text{State}, \text{Nonempty (MPFlow M } \emptyset V x)$ )
(hForce : ForcedNonexpansion M V)
(hNonneg : NonnegV V) {C :  $\mathbb{R}$ } (hC0 :  $0 \leq C$ ) (hVC :  $\forall y, V y \leq C$ )
(hFloor :  $\forall y \in U, 1 \leq V y$ ) (x : State) :
Battery.pathMeasure (skel M) (SkelState.alive x)
 $\{ \omega \mid \exists n, \omega n \in \text{skelBad } U \}$ 
 $\leq \text{ENNReal.ofReal } (V x)$ 

```

QtslAux.LayerD.MPDischarge . **sound\_GSHS\_safety\_battery\_mp**

statement bound

```

/-- The safety battery at the martingale-problem profile. -/
theorem sound_GSHS_safety_battery_mp
[IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
{V : State  $\rightarrow$   $\mathbb{R}$ } {U : Set State} {p thr :  $\mathbb{R}$ }
(hV : Measurable V) (hU : MeasurableSet U)
(hmp :  $\forall x : \text{State}, \text{Nonempty (MPFlow M } \emptyset V x)$ )
(hForce : ForcedNonexpansion M V)
(hNonneg : NonnegV V) {C :  $\mathbb{R}$ } (hC0 :  $0 \leq C$ ) (hVC :  $\forall y, V y \leq C$ )

```

```
(hFloor :  $\forall y \in U, 1 \leq V y$ )
(h0 :  $p + thr \leq 1$ ) (x : State) (hthr :  $\forall x \leq thr$ ) :
ENNReal.ofReal p  $\leq$  Battery.pathMeasure (skel M) (SkelState.alive x)
{ $\omega \mid \forall n, \omega n \notin \text{skelBad } U$ }
```

The staged-row form binds the rule's premise shapes as the schema realizes them: per-mode interior rows against the mode's flow mechanism, per-jump guard rows, per-mode floors and nonnegs, and the exact- $\mathbb{Q}$  threshold row cast once — assembled onto the flattened battery through the mixture bridge discussed in the proof section's final remark.

BernsteinLogicAux.QtslBattery . sound\_QSafetyBattery\_staged

statement bound

```
/-- **sound_QSafetyBattery_staged** – the `QSafetyBattery` node's staged
right column (QTSL-rules.toml row `QSafetyBattery`,
amended side) as a bound theorem, discrete-kernel profile. Hypothesis
families mirror the staged rosters:

* `hB1` **interiors** – per mode `m`, the flow chart's one-step drift
   $\int V_m d(\text{flow}_m z) \leq V_m z$  on  $D_m \setminus U_m$  (the refined admissible scope;
  the B1 row), with `hint` its domain-membership integrability;
* `hB2` **guards** – per jump `j`, the boundary row
   $V_{\{dst\ j\}}(g_j z) \leq V_{\{src\ j\}} z$  on the guard slab (the B2 row);
* `hB3` **floors** and `hB4` **nonnegs** – per mode on  $U \cap D_m / D_m$ 
  (the B3/B4 rows);
* `h0` – the threshold row  $p + thr \leq 1$  in **exact  $\mathbb{Q}$ **, the staged
  `q_add`/`q_le` vocabulary, cast once here;
* `hR3`/`hR4` – the registry side conditions at the compiled kernel;
* `hmix` – **the System bridge**: the presentation's
  charts compile, at this profile, to the step kernel `K` – pointwise, the
  one-step measure from any in-scope state is a sub-convex mixture of the
  mode's flow step and the jumps enabled there (weights supported on
   $\{src\ j = m \wedge z \in G\ j\}$ ). A System whose compilation satisfies `hmix` is
  exactly what stands left of the concluded  $\models$ .
```

Conclusion: region-level, as the concluded `QRegionSat` with the  
`qraSublevel(V, thr)` atom reads – from every start in  
 $D_m \cap \{V_m \leq thr\}$ , the chain satisfies  $G \emptyset$  with probability at least  
`p`. Assembled onto `sound\_QSafetyBattery` through `mixture\_drift` /  
`mixture\_integrable`; the GSHS-profile transport is the standing  
residue. -/

```
theorem sound_QSafetyBattery_staged
  {K : Kernel (Md × X) (Md × X)} [IsMarkovKernel K]
  {Dm Dbarm Um : Md → Set X} {p thr :  $\mathbb{Q}$ } {sat : Md × X → Prop}
  (hDm :  $\forall m$ , MeasurableSet (Dm m)) (hDbarm :  $\forall m$ , MeasurableSet (Dbarm m))
  (hUm :  $\forall m$ , MeasurableSet (Um m)) (hVmeas :  $\forall m$ , Measurable (V m))
  (hF :  $\forall m z, \neg \text{sat } (m, z) \rightarrow z \in Um\ m$ )
  (h0 :  $p + thr \leq 1$ )
  (hDsub :  $\forall m, Dm\ m \subseteq Dbarm\ m$ )
  (hR3 :  $\forall m z, z \in Dm\ m \rightarrow z \notin Um\ m \rightarrow K\ (m, z)\ \{y \mid y.2 \in Dbarm\ y.1\} = 1$ )
  (hR4 :  $\forall m z, z \in Dm\ m \rightarrow z \notin Um\ m \rightarrow$ 
     $K\ (m, z)\ (\{y \mid y.2 \in Dm\ y.1\} \cup \{y \mid y.2 \in Um\ y.1\}) = 1$ )
  (hint :  $\forall m z, z \in Dm\ m \rightarrow z \notin Um\ m \rightarrow \text{Integrable } (V\ m)\ ((\text{flow}_m\ m)\ z)$ )
  (hB1 :  $\forall m z, z \in Dm\ m \rightarrow z \notin Um\ m \rightarrow \int \zeta, V\ m\ \zeta\ \partial(\text{flow}_m\ m)\ z \leq V\ m\ z$ )
  (hB2 :  $\forall j, \forall z \in G\ j, V\ (dst\ j)\ (g\ j\ z) \leq V\ (src\ j)\ z$ )
  (hB3 :  $\forall m z, z \in Um\ m \rightarrow z \in Dbarm\ m \rightarrow 1 \leq V\ m\ z$ )
  (hB4 :  $\forall m z, z \in Dbarm\ m \rightarrow 0 \leq V\ m\ z$ )
  (hmix :  $\forall m z, z \in Dm\ m \rightarrow z \notin Um\ m \rightarrow$ 
     $\exists (w_0 : \mathbb{R}_{\geq 0}) (w : J \rightarrow \mathbb{R}_{\geq 0}), w_0 + \sum j, w\ j = 1 \wedge$ 
     $(\forall j, w\ j \neq 0 \rightarrow \text{src}\ j = m \wedge z \in G\ j) \wedge$ 
     $K\ (m, z) = (w_0 : \mathbb{R}_{\geq 0\infty}) \cdot ((\text{flow}_m\ m)\ z).map\ (\text{Prod.mk}\ m)$ 
     $+ \sum j, (w\ j : \mathbb{R}_{\geq 0\infty}) \cdot \text{Measure.dirac}\ (dst\ j, g\ j\ z)) :$ 
     $\forall m z, z \in Dm\ m \rightarrow V\ m\ z \leq (thr : \mathbb{R}) \rightarrow$ 
    ENNReal.ofReal (p :  $\mathbb{R}$ )
     $\leq \text{pathMeasure}\ K\ (m, z)\ \{\omega \mid \forall n, \text{sat}\ (\omega\ n)\}$ 
```

**The general stratum: acceptance events on the record carrier.** Over the semantic universe's Fell carrier the safety acceptance event is measurable by the topology of the site rather than by assumption: confinement events of closed walls are Fell-closed, hence

Borel under the Effros structure — over every trajectory window, with no compactness hypothesis anywhere.

SuFunctor.Site.FellAcceptance . **measurableSet\_confinedEvent**

statement bound

```
/-- The safety acceptance event of a closed wall is Borel – no
compactness, no  $\sigma$ -compactness, no Hausdorffness. -/
theorem measurableSet_confinedEvent {A : Set  $\alpha$ } (hA : IsClosed A) :
  MeasurableSet (confinedEvent A)
```

SuFunctor.Site.FellAcceptance . **measurableSet\_confinedEvent\_cylinder**

statement bound

```
/-- **The safety acceptance event is Borel over every window**: the
confinement event of a closed safe region's cylinder, with no
compactness and no  $\sigma$ -compactness – the A8 discharge of the safety
battery's measurability pack. -/
theorem measurableSet_confinedEvent_cylinder {D : Set X}
  (hD : IsClosed D) :
  MeasurableSet
    (confinedEvent ((Set.univ : Set  $\iota$ W)  $\times^s$  D) : Set (FTraj W X))
```

**Walls over the record carrier.** The battery's conclusion shape at the general stratum: per-window confinement classes with the full judgment bridge (satisfaction equals almost-sure confinement), the indicator transport law that admits them, the closed-graph strengthening (record confinement is pointwise safety, unconditional in the trajectory), and the composition seam (the window compositor of two wall demands is the wall demand of the glued class — persistence survives composition).

SuFunctor.Site.FellBehaviorSpec . **indPenM**

statement bound

```
/-- The indicator penalty of a class at the general stratum: permitted
outright iff the joint's mass avoids the class's complement, forbidden
outright otherwise. Total in the class (mathlib's outer-measure
evaluation); its transport law below is where measurability binds. -/
def indPenM (S : Set K) : MeasPen K
```

SuFunctor.Site.FellBehaviorSpec . **pull\_indPenM**

statement bound

```
/-- **Transport of the indicator along a measurable map is the
indicator of the preimage** – the general-stratum form of
`pull_indPen`, with the two measurability hypotheses that stratum
charges: the map's, and the class's. The latter is the A8 admission
ticket: only Borel acceptance events enter the indicator calculus. -/
theorem pull_indPenM (h : K  $\rightarrow$  L) (hm : Measurable h) {S : Set L}
  (hS : MeasurableSet S) :
  MeasPen.pull h (indPenM S) = indPenM (h  $^{-1}$ ' S)
```

SuFunctor.Site.FellBehaviorSpec . **isPersistentMF\_clipDemandsF**

statement bound

```
/-- **Persistence of clip demands**: window-calculus transitivity
alone (`penExtMF_penExtMF`), with openness of the site's windows as
the only hypothesis. No condition on the bases whatsoever. -/
theorem isPersistentMF_clipDemandsF (U :  $\iota \rightarrow$  Set T)
  (hU :  $\forall i$ , IsOpen (U i))
  (base :  $\forall i$ , MeasPen (FTraj (U i) X)) :
  IsPersistentMF U (clipDemandsF U base)
```

SuFunctor.Site.FellBehaviorSpec . **satAtM\_specOfBehaviorF\_iff**

statement bound

```
/-- **The judgment bridge at the general stratum**: a joint satisfies
the spec of a Fell behavior subtype at a window iff its mass lies in
the class there. Forward consumes the reflexive demand; backward
consumes the subpresheaf condition, clip measurability (openness of
the site's windows), and class measurability (the A8 ticket) – each
exactly once. -/
theorem satAtM_specOfBehaviorF_iff (U :  $\iota \rightarrow$  Set T)
  (hU :  $\forall i$ , IsOpen (U i)) (B : WBehaviorF U X)
  (hB :  $\forall i$ , MeasurableSet (B.cls i)) (j :  $\iota$ )
  ( $\mu$  : MeasJoint (FTraj (U j) X)) :
  SatAtM  $\mu$  (specOfBehaviorF U B j)  $\leftrightarrow$ 
  ( $\mu.1$  : Measure (FTraj (U j) X)) (B.cls j) $^c$  =  $\emptyset$ 
```

SuFunctor.Site.FellBehaviorSpec . satAtM\_safetySpecF\_iff

statement bound

```

/-- The safety wall's judgment bridge, with its measurability paid by
A8: for a closed safe region, satisfaction of the safety spec at a
window is almost-sure confinement there – the SU-general form of the
safety battery's acceptance reading, with the measurability pack
discharged by the topology of the site. -/
theorem satAtM_safetySpecF_iff (U :  $\mathbb{1}$  → Set T)
  (hU :  $\forall i$ , IsOpen (U i)) {D : Set X} (hD : IsClosed D) (j :  $\mathbb{1}$ )
  ( $\mu$  : MeasJoint (FTraj (U j) X)) :
  SatAtM  $\mu$  (specOfBehaviorF U (safetyBehaviorF U D) j)  $\leftrightarrow$ 
  ( $\mu.1$  : Measure (FTraj (U j) X))
  (confinedEvent ((Set.univ : Set  $\mathbb{1}$ (U j))  $\times^s$  D))c = 0

```

SuFunctor.Site.FellBehaviorSpec . graphCl\_mem\_confinedEvent\_iff

statement bound

```

/-- **The A5 strengthening**: record confinement of a trajectory's
closed-up graph to a closed wall is pointwise confinement at every
stage – unconditional in the trajectory. The skeleton route's
segment-constancy side condition is absent by construction: the record
constrains every `(t, x)` of the closed graph, mid-segment stages
included, and the closure completion costs nothing against a closed
wall. -/
theorem graphCl_mem_confinedEvent_iff {D : Set X} (hD : IsClosed D)
  (f : Traj W X) :
  graphCl f  $\in$  confinedEvent ((Set.univ : Set  $\mathbb{1}$ W)  $\times^s$  D)  $\leftrightarrow$ 
   $\forall w, f w \in D$ 

```

SuFunctor.Site.FellBehaviorSpec . mboxplusWF\_indPenM

statement bound

```

/-- **The record-carrier composition seam on indicators** (the
general-stratum sibling of `boxplusW_indPen`): the window compositor
of two wall demands is the wall demand of the glued record class –
the pullback intersection along the two state transports. Properness
feeds the transports' measurability, and the classes' measurability
is the A8 ticket again. This is the A4 transport core: composing
wall specs along the compositor never leaves the wall fragment, so
the composed conclusion is again a per-window class demand, and
persistence survives by `IsPersistentMF.image2_mboxplusWF`. The
per-profile grid cells of the wire-composition rule (ODE, sampled
data, gauge-form feedback) are drift-assembly content at the
presentation profiles, deliberately not restated here. -/
theorem mboxplusWF_indPenM {W : Set T} { $\pi_1$  : Z → U} { $\pi_2$  : Z → V}
  (hp1 : IsProperMap  $\pi_1$ ) (hp2 : IsProperMap  $\pi_2$ )
  {S1 : Set (FTraj W U)} {S2 : Set (FTraj W V)}
  (hS1 : MeasurableSet S1) (hS2 : MeasurableSet S2) :
  mboxplusWF  $\pi_1$   $\pi_2$  (indPenM S1) (indPenM S2)
  = indPenM (fellMap (W

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Model frame as at QBASIN, probabilistically read: the displayed  $\mathcal{M}$ ,  $x_0 \in \{V \leq y\} \models \text{SProbGE}(p, \text{PAlways } \varphi) \mapsto$  the goal ofReal  $p \leq \text{pathMeasure } K \times \{\omega \mid \forall n, \text{sat } (\omega n)\}$  for every start in the concluded region — sat the  $\varphi$ -satisfaction predicate, of which *no measurability is asked* (the fail-closed event is the measurable one; the tie weakens by `measure_mono`). The staged-row theorem `sound_QSafetyBattery_staged` is the amended display's binding; the flattened `sound_QSafetyBattery` is its engine.

- profile, stated plainly: the map splits in two. The bullets below marked *discrete map* bind the rule at the *discrete-kernel profile* —  $\mathcal{M}$ 's step semantics is a Markov kernel chain, its trajectory measure rendered as `pathMeasure` in the bound statements



— via `sound_QSafetyBattery_staged`, with the PDMP sub-profile carried end-to-end by `sound_GSHS_safety_battery`. The martingale-problem theorems (`sound_GSHS_safety_mp`, `sound_GSHS_safety_battery_mp` — hash-bound below) are the *continuous-time avoidance engine*, mapped separately in the *MP map* bullet; they do not themselves bind the full displayed judgment.

- *MP map*, its own binders and its own goal: the generator presentation and the interior rows  $\mapsto \text{hmp} : \forall x, \text{Nonempty } (\text{MPFlow } M \ 0 \ V \ x)$  (the martingale-problem presentation hypothesis, carried as data); guards  $\mapsto \text{hForce} : \text{ForcedNonexpansion } M \ V$ ; nonnegs  $\mapsto \text{hNonneg} : \text{NonnegV } V$ ; floors  $\mapsto \text{hFloor} : \forall y \in U, 1 \leq V \ y$ ; the threshold row  $\mapsto \text{h}\theta$ , and the start  $\mapsto x/\text{hthr}$ . Undisplayed MP regularity, named outright:  $\text{hV}/\text{hU}$  (measurability),  $\text{hC}\theta/\text{hVC}$  (a global certificate bound  $C'$ ), and the `IsMarkovKernel` instances on `M.seg/M.jump`. Honesty about the goal: the MP battery's literal conclusion is skeleton avoidance — of  $\text{Real } p \leq \text{Battery.pathMeasure } (\text{skel } M) (\text{SkelState.alive } x) \{ \omega \mid \forall n, \omega \ n \notin \text{skelBad } U \}$  — avoidance of the  $U$ -bad set along the embedded skeleton, not the displayed  $\text{SProbGE}(p, \text{PAways } \varphi)$  judgment verbatim; the formula tie from  $\varphi$  to  $U$  is the discrete map's  $\text{hF}$  business, and only the discrete staged theorem binds the displayed judgment itself.
- *discrete map* — what stands left of the  $\models \mapsto \text{hmix}$ , the System bridge: the presentation's charts compile to the kernel as a sub-convex mixture of the mode's flow step and the enabled jumps — an explicit hypothesis, so the certified statement quantifies over exactly the Systems whose compilation satisfies it.
- *discrete map* — the four premise families map one-to-one: interiors  $\mapsto \text{hB1}$  (per-mode one-step flow drift on  $D_m \setminus U_m$ ) with the undisplayed integrability rider `hint`; guards  $\mapsto \text{hB2}$  (per-jump boundary rows on the guard slab); floors  $\mapsto \text{hB3}$  (on  $U \cap \bar{D}_m$ , the ruled fail-closed scope); nonnegs  $\mapsto \text{hB4}$ .
- *discrete map* — the threshold tie  $p + y \leq 1$  (a side condition in the display)  $\mapsto \text{h}\theta$ , in exact  $\mathbb{Q}$ , cast once inside the statement.
- *discrete map* — side conditions  $\text{R3}/\text{R4} \mapsto \text{hR3}/\text{hR4}$  at the compiled kernel (hull containment; fail-closed reset containment); the formula tie  $U := \text{cmpl}(\varphi) \mapsto \text{hF} : \neg \text{sat } y \rightarrow y \in U$  — note the certified hypothesis asks only that  $U$  *over-approximate* the falsity set: strictly wider than the displayed equation, in the safe direction.
- *discrete map* — conclusion region  $\text{qraSublevel}(V, y) \mapsto$  the start conditions  $z \in D_m \ m$  and  $V \ m \ z \leq \text{thr}$  (the Lean source's own identifier for the display's  $y$ ), region-level as the judgment reads.

Width note: measurability of the region/certificate data ( $\text{hDm}/\text{hDbarm}/\text{hUm}/\text{hVmeas}$ ) and the interior integrability rider are profile-level regularity with no displayed rows — the probabilistic sibling of the `PsdCERT` evaluability pack; every mandatory displayed row is consumed — the display's *optional* `scope_warrant` slot (the dissolved `ScopeRule`'s interface) is taken by neither certified statement and has no entry in this map — and the two widenings (over-approximating  $U$ , sub-convex  $\text{hmix}$ ) both enlarge the certified reach beyond the display's letter.

## Lean verifier receipts

module `BernsteinLogicAux.Battery`; bound build `2fd1c031c80b...` (bound from receipt `8d9ccb68993ea0ec`, 2026-08-04T01:56:55Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 8d9ccb68993ea0ec | 2026-08-04T01:56:55Z | build+decorrelated | current |
| 5075586e4813a7c5 | 2026-08-04T03:11:30Z | build+decorrelated | current |

13 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogicAux.LayerD.Skeleton`; bound build `9a4f5ca8bc26...` (bound from receipt `f3d40eda96c4513b`, 2026-08-04T01:57:01Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| f3d40eda96c4513b | 2026-08-04T01:57:01Z | build+decorrelated | current |
| 5966aa764026dde9 | 2026-08-04T03:11:31Z | build+decorrelated | current |

13 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogicAux.LayerD.DiniDischarge`; bound build `db41370b5e77...` (bound from receipt `2a294d6f4fc9a7b`, 2026-07-21T04:51:51Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 073080d556f0cf08 | 2026-07-20T22:02:26Z | build+decorrelated                 | current |
| 6f4929afcf219d67 | 2026-07-20T22:52:48Z | build+decorrelated                 | current |
| f258f966d21ec502 | 2026-07-21T00:03:10Z | build+decorrelated                 | current |
| 2a294d6f4ffc9a7b | 2026-07-21T04:51:51Z | build+decorrelated                 | current |
| 277ec2c0fa8227a9 | 2026-07-21T05:38:41Z | build+replay+hardened+decorrelated | current |
| a142e9bdad29ed20 | 2026-07-22T18:27:03Z | build+decorrelated                 | current |
| 6aeb017a3939a647 | 2026-07-31T21:59:58Z | build+decorrelated                 | current |
| 85b98a8efabbfc19 | 2026-07-31T22:28:52Z | build+decorrelated                 | current |
| fc8973bd3d8aa0b  | 2026-08-02T23:31:02Z | build+decorrelated                 | current |
| 45fcb5d9746e88b  | 2026-08-03T16:27:06Z | build+decorrelated                 | current |
| 5a035508867866dd | 2026-08-03T22:45:49Z | build+decorrelated                 | current |

module BernsteinLogicAux.LayerD.ItoInterface; bound build 9b18e4e710f1... (bound from receipt 8b9184a6df93a83e, 2026-07-21T04:51:52Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 4e3a32599ecb1605 | 2026-07-21T00:03:12Z | build+decorrelated                 | current |
| 8b9184a6df93a83e | 2026-07-21T04:51:52Z | build+decorrelated                 | current |
| 04aa3a28f81ea9f9 | 2026-07-21T05:38:43Z | build+replay+hardened+decorrelated | current |
| d95924f2dd7f318f | 2026-07-22T18:27:04Z | build+decorrelated                 | current |
| c78dba97e2eee2ba | 2026-07-31T21:59:59Z | build+decorrelated                 | current |
| ca54011b021b532e | 2026-07-31T22:28:52Z | build+decorrelated                 | current |
| bdf792a36abf7da8 | 2026-08-02T23:31:02Z | build+decorrelated                 | current |
| e8478cde1b56df65 | 2026-08-03T16:27:06Z | build+decorrelated                 | current |
| e987ba886a91fc22 | 2026-08-03T22:45:50Z | build+decorrelated                 | current |

module BernsteinLogicAux.LayerD.ItoSampling; bound build fb8de8b804d7... (bound from receipt 467820f314b0750c, 2026-07-21T08:16:24Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 467820f314b0750c | 2026-07-21T08:16:24Z | build+decorrelated | current |

9 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.GshsItoDischarge; bound build 858d3234fc4f... (bound from receipt d6088699caa41da2, 2026-08-03T20:13:46Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 1d9003db41305809 | 2026-08-02T12:53:42Z | build+decorrelated | current |
| 615e37d28395ee85 | 2026-08-02T23:30:51Z | build+decorrelated | current |
| 306da0b697e1ba52 | 2026-08-03T16:26:54Z | build+decorrelated | current |
| d6088699caa41da2 | 2026-08-03T20:13:46Z | build+decorrelated | current |
| b13efcc7c507c534 | 2026-08-03T22:45:38Z | build+decorrelated | current |

7 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.Debut; bound build cc670db616db... (bound from receipt 38571a839f9da48c, 2026-07-24T09:28:37Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a7c2ce15b3ea5ce7 | 2026-07-22T16:23:09Z | build+decorrelated | current |
| e8af72cc3335497f | 2026-07-22T18:26:50Z | build+decorrelated | current |
| 38571a839f9da48c | 2026-07-24T09:28:37Z | build+decorrelated | current |
| d0e0aa26ac8f8d34 | 2026-07-31T21:59:41Z | build+decorrelated | current |
| b5cc2fcbabda9723 | 2026-07-31T22:28:35Z | build+decorrelated | current |
| f494a0acd9a534b4 | 2026-08-02T23:30:43Z | build+decorrelated | current |
| 72ed3556543f0ada | 2026-08-03T16:26:47Z | build+decorrelated | current |
| aea75952b3723556 | 2026-08-03T22:45:30Z | build+decorrelated | current |

module BernsteinLogicAux.OptionalStoppingRC; bound build 3f566c19afe8... (bound from receipt 704016ef1d4bd23d, 2026-07-24T09:28:43Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 704016ef1d4bd23d | 2026-07-24T09:28:43Z | build+decorrelated | current |

6 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module QtslAux.LayerD.MPDischarge; bound build bade74dd943... (bound from receipt 7228834dea10e5f9, 2026-07-24T11:04:04Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 7228834dea10e5f9 | 2026-07-24T11:04:04Z | build+decorrelated | current |

module BernsteinLogicAux.QtslBattery; bound build 5c75f9923f27... (bound from receipt 3fd830b4914bdeda, 2026-08-04T01:56:58Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 3fd830b4914bdeda | 2026-08-04T01:56:58Z | build+decorrelated | current |
| 96b18d8572451f65 | 2026-08-04T03:11:31Z | build+decorrelated | current |

13 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module SuFunctor.Site.FellAcceptance; bound build 651fb557bbfe... (bound from receipt aa2169560f8f571a, 2026-08-03T20:19:04Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 7ad85720de6e419e | 2026-08-03T10:13:50Z | build+decorrelated | current |
| 94a408109996d1bf | 2026-08-03T13:41:21Z | build+decorrelated | current |
| 580a7e551fab03b6 | 2026-08-03T15:39:44Z | build+decorrelated | current |
| aa2169560f8f571a | 2026-08-03T20:19:04Z | build+decorrelated | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module SuFunctor.Site.FellBehaviorSpec; bound build 345f9caf9453... (bound from receipt 72f2c3b62b6771fc, 2026-08-03T20:19:08Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| d7b846e785c1d7fb | 2026-08-03T10:13:53Z | build+decorrelated | current |
| c0e3dc15419765aa | 2026-08-03T13:41:24Z | build+decorrelated | current |
| 9a62b29c08b63b4f | 2026-08-03T15:39:47Z | build+decorrelated | current |
| 72f2c3b62b6771fc | 2026-08-03T20:19:08Z | build+decorrelated | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development ports the base calculus's checker chain (theory `bern_catalogue`, developed side by side with the base catalogue sources, stopping below the proof-object layer); it contains no probability or measure theory. `QSAFETYBATTERY` lives one theory-extension edge above the base calculus — it concludes a satisfaction judgment, not a sequent — and its soundness content is path-space measure theory (trajectory kernels, stopped supermartingales, Ville's inequality), for which no counterpart exists anywhere in the ported development. A HOL statement of this rule would be a new measure-theoretic campaign, not an extension of the ported checker chain; until one exists, this box records the current state of the edifice.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail. This rule concludes a *satisfaction judgment* ( $\models$ ), one theory-extension edge above the base calculus: every premise row is an inherited base-calculus sequent, and soundness is proved *given* base soundness (the extension-edge convention). The theorems quoted above warrant the rule at the **discrete-kernel profile**; the profile remark at the end of this section states exactly what remains a hypothesis beyond it. Lemmas [R40.1–R40.3](#) and Proposition [R40.4](#) are the battery family's shared probabilistic preamble.

**Setting and claim.** Fix a measurable state space  $S$  — the flattened hybrid state space, i.e. the disjoint sum of the per-mode domains — and a Markov one-step kernel  $K$  on  $S$ , carrying both discretized flow steps and guarded jumps. For  $x \in S$  write  $\Pr_x$  for the path measure on  $S^{\mathbb{N}}$ : the trajectory law of the time-homogeneous chain driven by  $K$ , started deterministically at  $x$  (the Ionescu–Tulcea construction, with the constant kernel family).  $(\mathcal{F}_n)_n$  is the canonical filtration,  $\mathcal{F}_n$  generated by the coordinates  $\leq n$ .

The rule's data: a modelled domain  $D$ , a hull  $\bar{D} \supseteq D$  (the region on which the polynomial rows were actually checked), an avoid set  $U$  compiled to over-approximate the falsity set of the G-argument —  $D, \bar{D}, U$  all measurable subsets of  $S$ , as is every region a row speaks on, here and in the sibling rules riding this substrate — a measurable certificate  $V : S \rightarrow \mathbb{R}$ , and rationals  $p, y$ . The checked rows and side conditions, read semantically at this profile:

(B1 $\wedge$ B2) *generator*: for every  $z \in D \setminus U$ ,  $V$  is integrable against  $K z$  and  $\int V dK z \leq V(z)$ . (At this profile the interior and guard rows land *together*: the one-step kernel carries flow and enabled jumps in a single step, so the two row families compile to one kernel-level generator row.)

(B3) *floors*:  $V \geq 1$  on  $U \cap \bar{D}$ .

(B4) *nonnegs*:  $V \geq 0$  on  $\bar{D}$ .

(R3) *hull*: for every  $z \in D \setminus U$ ,  $K z(\bar{D}) = 1$  — one-step mass stays where the rows were checked.

(R4) *fail-closed containment*: for every  $z \in D \setminus U$ ,  $K z(D \cup U) = 1$ .

( $\Theta$ ) *threshold*:  $p + y \leq 1$ , checked in exact rational arithmetic.

(F) *formula tie*:  $\neg \text{sat}(z) \Rightarrow z \in U$ , where  $\text{sat}$  is the state-satisfaction predicate of the G-argument  $\varphi$ .

The claim (`sound_QSafetyBattery`): for every  $x \in D$  with  $V(x) \leq y$ ,

$$\Pr_x^*[\forall n, \text{sat}(\omega_n)] \geq p$$

— outer measure, because no measurability is ever asked of the satisfaction predicate (Lemma R40.3 states the convention); it coincides with the ordinary probability whenever the satisfaction event is measurable.

**Lemma R40.1** (the stopped certificate). *Let  $f : S \rightarrow \mathbb{R}$  and let  $A \subseteq S$  be a stop set. Define the stopped certificate process  $X_n(\omega) := f(\omega_{n \wedge \tau_A})$ , where  $\tau_A$  is the first visit to  $A$  — equivalently, by the primitive recursion that freezes the value once  $A$  has been visited. Then: (i) if  $A$  is measurable and  $f$  is measurable,  $X$  is adapted to  $(\mathcal{F}_n)$ ; (ii) any pointwise property of  $f$ 's values holds of  $X$ 's values; (iii) barrier transport: if  $f \geq 1$  on  $A$ , then  $X_n \geq 1$  on the event  $\{\tau_A \leq n\}$ ; (iv)  $X_n$  depends only on the coordinates  $\leq n$ .*

*Proof.* All four are elementary inductions on  $n$  over the two branches of the recursion (surviving:  $X_{n+1} = f(\omega_{n+1})$ ; frozen:  $X_{n+1} = X_n$ ). No stopping-time API is needed anywhere — the recursion is the stopped process, and (iii) is proved by chasing the first index at which  $A$  was visited.  $\square$

**Lemma R40.2** (supermartingale bridge). *Let  $f$  be measurable with  $0 \leq f \leq C$ , let  $A$  be measurable, and suppose the one-step generator row holds off  $A$ : for all  $z \notin A$ ,  $\int f dK z \leq f(z)$ . Then for every start  $x$ , the stopped certificate  $X$  of Lemma R40.1 is a supermartingale with respect to  $(\mathcal{F}_n, \Pr_x)$ .*

*Proof.* Integrability is the bound  $0 \leq X_n \leq C$  (Lemma R40.1(ii)). For the supermartingale inequality, condition on  $\mathcal{F}_n$ : by the Markov property of the trajectory construction, the conditional expectation of a bounded  $\mathcal{F}_{n+1}$ -measurable function given  $\mathcal{F}_n$  is its integral against the trajectory kernel continued from the  $n$ -prefix. Split on the survival event  $S_n := \{\omega : \omega_k \notin A \text{ for all } k \leq n\}$  —  $\mathcal{F}_n$ -measurable,  $A$  being measurable, and determined by the coordinates  $\leq n$ , so paths patched to agree with  $\omega$  up to  $n$  have the same survival status. On the surviving branch the integrand is  $f$ (next coordinate), whose one-step marginal is  $K(\omega_n)$ ; since survival means  $\omega_n \notin A$ , the generator row applies and the conditional expectation is  $\leq f(\omega_n) = X_n(\omega)$ . On the frozen branch the integrand is the constant  $X_n(\omega)$ . Note that every hypothesis here is pointwise — there is no almost-everywhere invariance argument anywhere in the bridge; this is what the fail-closed design buys, and the reachability sibling (`QDeadlineBattery`) will have to pay precisely this cost when it can no longer enlarge its stop set.  $\square$

**Lemma R40.3** (Ville's inequality, barrier form). *Let  $X$  be a nonnegative supermartingale with  $X_n \geq 1$  on the event  $\{\text{some coordinate } \leq n \text{ lies in } A\}$ . Then  $\Pr_x^*[\exists n, \omega_n \in A] \leq \mathbb{E} X_0$ , where  $\Pr_x^*$  is the outer measure — equal to the ordinary probability whenever the visit event is measurable, as it is at every consumption in this family ( $A$  is measurable there).*

*Proof.* This is the maximal inequality for nonnegative supermartingales at barrier 1 [234] (textbook treatment: [64]): the probability that a nonnegative supermartingale ever reaches level  $a$  is at most  $\mathbb{E} X_0/a$ . The mechanized form deliberately asks no measurability of the visit event itself — the bad event is bounded by inclusion in the measurable event  $\{\exists n, X_n \geq 1\}$ , whose probability the maximal inequality bounds, and only the process needs to be adapted; the outer-measure phrasing of the conclusion is exactly this inclusion, spelled.  $\square$

**Proposition R40.4** (the battery pattern). *Let  $A$  be measurable and let  $f$  be measurable with  $0 \leq f \leq C$  globally,  $f \geq 1$  on  $A$ , and one-step drift off  $A$ . Then for every  $x$ ,*

$$\Pr_x[\exists n, \omega_n \in A] \leq f(x).$$

*Proof.* Lemmas R40.1–R40.3 assembled; the initial value is  $\mathbb{E} X_0 = f(x)$  because coordinate 0 is deterministically  $x$ . This proposition — nonnegative supermartingale + barrier + initial value  $\Rightarrow$  probability bound — is the shape every rule in the battery family consumes; the siblings cite it as “the battery pattern (`QSafetyBattery`, Proposition R40.4)”.  $\square$

**Lemma R40.5** (the clamp: from checked rows to global hypotheses). *The rows (B3), (B4) speak only on  $\bar{D}$ , while Proposition R40.4 wants its bounds globally. Set  $A := U \cup D^c$  (the fail-closed stop set),  $C := \max(y, 1)$ , and let  $g$  be the clamped certificate: on  $D$ , clamp  $V$  into  $[0, C]$ ; off  $D$ , clamp  $\max(V, 1)$  into  $[0, C]$  (so the floor is forced off the modelled domain). Then: (i)  $0 \leq g \leq C$  globally; (ii)  $g \geq 1$  on  $A$ ; (iii) on  $\bar{D} \cap (D \cup U)$  the clamp is invisible:  $g = \min(V, C)$  there; (iv) under (R3) + (R4),  $g$  satisfies the one-step generator row off  $A$ ; (v)  $g(x) = V(x)$  at any  $x \in D$  with  $V(x) \leq y$ . Throughout,  $g$  is measurable: a case split between measurable pieces ( $V$ ,  $D$  measurable) composed with clamps — so  $(g, A)$  is an admissible input to Proposition R40.4.*

*Proof.* (i), (ii), (v) are case checks from (B3), (B4) and the definition — (ii) uses (B3) on  $U \cap \bar{D}$  and the forced floor off  $D$ . (iii): on  $D$  the lower clamp is inert by (B4); on  $(\bar{D} \cap U) \setminus D$  the forced floor is inert by (B3). For (iv), fix  $z \in D \setminus U$  (i.e.  $z \notin A$ ). By (R3) and (R4) the one-step measure  $K z$  is carried by  $\bar{D} \cap (D \cup U)$  — the complement has measure  $\leq K z(\bar{D}^c) + K z((D \cup U)^c) = 0$  — so by (iii) the integrand may be replaced by  $\min(V, C)$ . Then  $\int \min(V, C) dK z \leq \min(\int V dK z, C) \leq \min(V(z), C) = g(z)$ , using monotonicity of the integral, the generator row (B1 $\wedge$ B2), and (iii) at  $z$  itself.  $\square$

**Theorem R40.6** (fail-closed form). *Under (B1)–(B4), (R3), (R4), ( $\Theta$ ): for every  $x \in D$  with  $V(x) \leq y$ ,*

$$\Pr_x[\forall n, \omega_n \in D \setminus U] \geq p.$$

*Proof.* Apply Proposition R40.4 to  $(g, A)$  of Lemma R40.5:  $\Pr_x[\text{reach } A] \leq g(x) = V(x) \leq y$ . The complement of ever visiting  $A = U \cup D^c$  is exactly  $\{\forall n, \omega_n \in D \setminus U\}$ , so its probability is at least  $1 - y \geq p$  by ( $\Theta$ ). Note the concluded event is stronger than the safety formula's truth set: the chain never fails and never leaves the modelled domain. Leaving counts as failure — that is the fail-closed reading, and it is what (R4) is recomputed for.  $\square$



**Proposition R40.7** (soundness of `QSafetyBattery`). *Under the hypotheses above including the formula tie (F): for every  $x \in D$  with  $V(x) \leq y$ ,  $\Pr_x^*[\forall n, \text{sat}(\omega_n)] \geq p \rightarrow$  the concluded satisfaction judgment  $\mathcal{M}, x_0 \in \{V \leq y\} \models \text{SProbGE}(p, \text{PAlways } \varphi)$ , read on the sublevel region  $\{V \leq y\}$ .*

*Proof.* By the contrapositive of (F),  $\omega_n \in D \setminus U$  implies  $\text{sat}(\omega_n)$ ; so the event of Theorem R40.6 is contained in  $\{\forall n, \text{sat}(\omega_n)\}$ , and monotonicity of the (outer) measure finishes. No measurability is ever asked of the satisfaction predicate — the smaller, fail-closed event is the measurable one.  $\square$

**Remark R40.8** (why the floors are not decoration). Drop (B3) and take  $V \equiv 0$ : drift, nonneg, and threshold rows all hold, with  $V(x) = 0 \leq y$  for any  $y \geq 0$ , so with  $y = 0$ , (Θ) permits  $p = 1$ : the rule would certify *almost-sure* safety — of any chain at all, including the two-state chain  $D = \{d\}$ ,  $U = \{u\}$ ,  $K d = \delta_u$ , which fails at its first step. With the floor present, this certificate dies immediately: (B3) forces  $V(u) \geq 1$ , and then the generator row at  $d$  demands  $\int V dK d = V(u) \leq V(d) = 0$  — contradiction. Failure must cost at least one unit of certificate value; the floor is the price list.

**Remark R40.9** (why the nonnegs are not decoration). The maximal inequality of Lemma R40.3 is *false* for signed supermartingales, and a two-branch chain isolates the sign hypothesis exactly — every other row intact, boundedness included. Take  $D = \{s, b\}$ ,  $U = \{a\}$ ,  $\bar{D} = \{s, a, b\}$ , with  $K s = \frac{1}{2}\delta_a + \frac{1}{2}\delta_b$ ,  $K b = \delta_b$ , and the signed certificate  $V(s) = 0$ ,  $V(a) = 1$ ,  $V(b) = -1$ . The generator row holds on  $D \setminus U$  ( $\int V dK s = 0 \leq 0$ ;  $\int V dK b = -1 \leq -1$ ), the floor holds on  $U \cap \bar{D}$ , the hull and fail-closed rows hold, and  $|V| \leq 1$ . Were Proposition R40.4 available without nonnegativity, the certificate would conclude  $\Pr_s[\text{reach } a] \leq V(s) = 0$  — but the chain reaches  $a$  with probability  $\frac{1}{2}$ : the  $a$ -branch's barrier visit is subsidized by the mass parked at  $V = -1$  on the other branch, and (B4) is what forbids the loan. The classical unbounded picture is the simple symmetric random walk on  $\mathbb{Z}$  with  $V(z) = z$  and  $A = \{1\}$  — same phenomenon, though as a drop-one witness it is impure: it violates the boundedness hypothesis along with the sign, which is why the two-branch chain is the recorded counterexample.

**Remark R40.10** (the hull and fail-closed conditions (R3), (R4)). Both are recomputed by the checker, never taken from the producer, and each blocks a distinct hole. Without (R3), the one-step integral in the generator row could consult states outside  $\bar{D}$  — states where the floor and nonneg rows were *never checked* — and Lemma R40.5(iv)'s clamp-invisibility computation collapses. Without (R4), one-step mass could leave  $D \cup U$  entirely: the concluded event counts leaving the modelled domain as failure, and (R4) is exactly what makes the clamped certificate's drift hold at the *enlarged* stop set  $U \cup D^c$  — the design that let every hypothesis in Lemma R40.2 stay pointwise. A companion condition (floor-scope resolution) is discharged at row-compilation time: it is what makes the checked floor rows imply (B3) as stated here, and it has no semantic residue at this layer.

**Remark R40.11** (threshold and integrability). (Θ) is bookkeeping, and it is sharp: the certificate buys  $\Pr[\text{fail}] \leq y$ , so asserting success  $\geq p$  requires  $p \leq 1 - y$ , and  $y = 1 - p$  is the tightest admissible choice — producer weakening ( $y < 1 - p$ ) is sound for free. The integrability half of (B1 $\wedge$ B2) is the discrete-profile reading of “the certificate lies in the generator's domain”; for the intended instances — polynomial  $V$ , one-step images inside the compact hull by (R3) — it is automatic.

**Remark R40.12** (**profile** — what is proven, at which presentations). Proven, and quoted above this proof: the discrete-kernel profile — one-step Markov kernel semantics over the trajectory construction, the theorems `sound_QSafetyBattery` and `sound_QSafetyBattery_failClosed`, sorry-free. Also proven, end-to-end: the *PDMP sub-profile* of the continuous-time semantics — piecewise-deterministic flow with guarded jumps, no diffusion (`sound_GSHS_safety_battery`). And proven, at the profile that carries the general continuous-time semantics, jump-diffusions included: the *martingale-problem profile* — the system presented by its generator drift data,  $\mathcal{AV}$  carried as data beside the certificate with the martingale-problem hypothesis for the pair, càdlàg paths per the presentation — the theorems `sound_GSHS_safety_mp` and `sound_GSHS_safety_battery_mp`, machine-checked sorry-free in the semantic-universe strand's auxiliary package (module `QtsLAux.LayerD.MPDDischarge`). The three ingredients an earlier edition of this remark filed as missing are each discharged there: continuous-time generator domains ride the presentation — for the calculus's own denoted class, compositional symbolic piecewise-polynomial hypernets,  $\mathcal{AV}$  is again polynomial data and domain bookkeeping dissolves; fine path properties ride the presentation's càdlàg clause; and optional stopping for right-continuous nonnegative supermartingales is a library theorem, consumed at the exit début (`OptionalSamplingRC`, with `supermartingale_of_compensator_nonpos` and the exponential-discount construction). The epistemic shape is the discrete profile's own: exactly as the staged theorem quantifies over systems whose compilation satisfies the mixture bridge, the martingale-problem theorems quantify over systems whose presentation satisfies the martingale-problem hypothesis — an explicit bridge carried as data, never a silent assumption, so the argument this section walks is now the theorem's own at every profile. The martingale-problem theorems live in the semantic-universe strand's auxiliary package, which is registered with this catalogue's binder: their statements are extracted and hash-bound like every other block in this section, so the dashboard row carries no conditional mark — each named profile is proven, and its theorems are bound.

**Remark R40.13** (the staged premise shapes and the system bridge). The rule's node presents its premises per mode and per jump: mode-indexed certificate family ( $V_m$ ), interior rows against each mode's flow mechanism, guard rows per jump, floors and nonnegs per mode, and the threshold row in exact rational arithmetic, cast once. The bound theorem for that shape (`sound_QSafetyBattery_staged`) closes the gap to the flattened hypotheses used above through an explicit bridge hypothesis: at every in-scope state, the one-step kernel decomposes as a sub-convex mixture of the mode's flow step and the enabled guard-reset jumps (weights supported on enabled jumps). Under that decomposition the mode's interior row and the enabled guard rows are the flattened generator row (`mixture_drift`): the one-step integral splits over the mixture, each jump term is a guard row evaluation, and the weights sum to one. What stands left of the concluded  $\models$  is any system presentation whose compilation at this profile satisfies the bridge.

**Remark R40.14** (relation to the literature). Supermartingale safety certificates for stochastic systems go back to Kushner [147]; the modern certificate-form idiom — synthesize a barrier function, check polynomial rows, conclude a probability bound — is

Prajna–Jadbabaie–Pappas [202]. The probability engine is Ville's inequality [234], with the systematic supermartingale treatment in Doob [64]. The reading of the interior rows as one-step expected decrease is the Foster–Lyapunov drift idiom [181], whose geometric form powers the deadline sibling. What this rule adds to that literature is not mathematics but *trust structure*: every analytic ingredient lives in a verified lemma, every polynomial ingredient is an inherited base-calculus row replayed by the kernel, and the checker recomputes the side conditions rather than trusting the producer.

| proof step                                                       | formal anchor                                                                                                                                                |
|------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lemma R40.1: stopped certificate, adaptedness, barrier transport | stopped, survive, stronglyAdapted_stopped, one_le_stopped, stopped_congr                                                                                     |
| Lemma R40.2: supermartingale bridge                              | supermartingale_stopped (Markov property via Kernel.condExp_traj; prefix patching via Kernel.integral_traj; one-step marginal via Kernel.map_traj_succ_self) |
| Lemma R40.3: maximal inequality at barrier 1                     | BernsteinLogic.Ville.reach_avoid, consumed at $a = 1$                                                                                                        |
| Proposition R40.4: the battery pattern                           | measure_reach_le                                                                                                                                             |
| Lemma R40.5: the clamp                                           | the construction $g$ , hclamp, hgdrift inside                                                                                                                |
| Theorem R40.6                                                    | sound_QSafetyBattery_failClosed                                                                                                                              |
| Proposition R40.7: formula tie, monotonicity                     | sound_QSafetyBattery                                                                                                                                         |
| Remark R40.13: staged shapes, mixture bridge                     | mixture_integrable, mixture_drift, sound_QSafetyBattery_staged                                                                                               |
| Remark R40.12: PDMP sub-profile                                  | sound_GSHS_safety_battery                                                                                                                                    |
| Remark R40.12: martingale-problem profile                        | sound_GSHS_safety_mp, sound_GSHS_safety_battery_mp<br>(QtSLAux.LayerD.MPDDischarge; hash-bound via the qtsl-aux package registration)                        |



## Rule 41. QDEADLINEBATTERY (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — *QDeadlineBattery*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} V : \text{QCertFam} \quad R \subseteq \llbracket \varphi \rrbracket \text{ (target formula tie)} \quad \{D_m \setminus R \vdash \mathcal{A}_m V_m \leq -\theta V_m\}_m \text{ (decay)} \\ \{\text{slab}_j \vdash g_j \ ; \ V_{m'_j} \leq V_{m_j}\}_j \text{ (guards, per jump } j: \text{ source mode } m_j, \text{ target } m'_j, \text{ reset } g_j) \\ \{\bar{D}_m \setminus R \vdash V_m \geq 1\}_m \text{ (floors, off-target only)} \quad \{\bar{D}_m \vdash V_m \geq 0\}_m, \{\bar{D}_m \vdash V_m \leq M\}_m \text{ (nonnegs; bound)} \\ \vdash q \leq \sum_{k \leq n} (\theta T)^k / k! \text{ (one TAYLORBOUND leaf)} \quad 0 \leq \beta, \beta \leq e^{-\theta}, \theta \geq 0, 0 \leq q, p \leq 1 \text{ (drift tie; sign gates)} \end{array}}{\mathcal{M}, x_0 \in \{V \leq (1-p)q\} \models \text{SProbGE}(p, \text{PDeadline}(T, \varphi, b))}$$

rule sort baked in theory `qtsl (QDeadlineBattery; the tabular tier's QDeadlineTable beside it)` — the display is the realized form, as the discrete-profile theorems bind it. A region left of the turnstile scopes the row  $(R \vdash P: P \text{ at every point of } R)$ ; the certificate family  $V$  enters as certificate data — a shared reference with no claim content of its own; the rows impose every claim — never as a derivation edge. In the drift tie and sign gates, both transcendental sides ( $\beta \leq e^{-\theta}$  and the Taylor leaf's  $e^{\theta T}$  enclosure) live in the soundness lemma, never in the checker. The conclusion is the region-conditioned judgment in constructor vocabulary, as at `QSAFETYBATTERY`;  $b$  is `PDeadline`'s producer-chosen-horizon mark.

**QDEADLINEBATTERY** is the bounded-reachability battery: it concludes that with probability at least  $p$  the system reaches the target — satisfies `PDeadline( $T, \varphi, b$ )` (surface syntax  $F_{\leq T} \varphi$ ) — from every start in the sublevel region  $\{V \leq (1-p)q\}$ . Its engine is exponential decay: decay rows say the certificate shrinks by a factor  $\beta$  per step off the target, so the discounted process  $\beta^{-t}V$  is a supermartingale off the target, while the off-target floor  $V \geq 1$  makes *missing* the deadline cost at least one unit of discounted certificate value; Markov's inequality at the horizon gives  $\Pr[\text{miss}] \leq \beta^T V(x)$ .

The rule's division of labour between checker and lemma is its most instructive feature. The checker replays only exact rational comparisons: the one arithmetic leaf is the Taylor comparison  $q \leq \sum_{k \leq n} (\theta T)^k / k!$  — a closed, wire-free inequality discharged by the ground-fact rule — while both transcendental facts ( $\sum_{k \leq n} (\theta T)^k / k! \leq e^{\theta T}$  and  $e^{-\theta} e^{\theta} = 1$ ) live inside the soundness lemma. The checker never needs to know  $e$  exists.

What the checker recomputes beyond that leaf: the domain-hull and fail-closed side conditions (as in the safety battery), and the threshold arithmetic in the exact rational vocabulary ( $\text{thr}_F = \text{thr} \cdot q \text{ atop } p + \text{thr} \leq 1$ ). Soundness is independent of the producer knobs  $\theta$  and  $q$ : any discrete rate  $0 \leq \beta \leq e^{-\theta}$  and any  $q$  below the Taylor sum are admissible.

Each hypothesis is load-bearing. Drop the off-target floor and a certificate identically 0 would certify certain reachability of an unreachable target. Drop the boundedness row and the decay row no longer transfers to the clamped certificate — the discount multiplies the certificate by  $\beta^{-t}$ , so the ceiling must be invisible wherever one-step mass reaches, which is exactly what  $V \leq M$  buys (and, in the mechanized engine, the integrability of the discounted certificate rides the same bound). Drop the fail-closed containment and escape from the modelled domain would count as success: a reach guarantee must show the escape event *null*, where the safety battery could simply enlarge its stop set (leaving counts as failing is conservative for safety, but would be unsound generosity for reachability). Drop the threshold arithmetic and the concluded region over-budgets. The proof section walks each.

The dashboard mark states the status: the hash-bound theorems warrant the rule at the discrete-kernel profile (geometric drift,  $\beta = e^{-\theta}$  discretized); the continuous-time profile is proven at the martingale-problem presentation — the discounted supermartingale built from the undiscounted martingale-problem hypothesis by the exponential-discount construction, stopped at the début by the right-continuous optional-sampling library theorem (`sound_GSHS_deadline_mp`, `sound_GSHS_deadline_battery_mp`, `QtslAux.LayerD.MPD`ischarge) — hash-bound below via the `qtsl-aux` package registration, so the dashboard row carries no conditional mark: each named profile is proven, and its theorems are bound.

At the general stratum, over the semantic universe's record carrier (the semantic-universe functor package, hash-bound below), the deadline conclusion is a *mark* — an  $\exists$ -face spec: the per-window family of acceptance-event demands is persistent, and satisfaction forces almost-sure acceptance at the window (`satAtM_acceptanceSpecF_ae`). The backward direction is deliberately absent, and the absence is the semantics: a record that reaches its target only in a superwindow is invisible to the subwindow, so a subwindow's reach demand is a genuinely stronger demand — a deadline — and extension is exactly how a met deadline persists to every superwindow. The measurability the mark face charges is issued by the site's topology rather than assumed: hit-events commute with countable unions in the set argument (`hitEvent_iUnion`), turning grid decompositions into event decompositions, and a compact target cell's visit event over a  $\sigma$ -compact stage window is Borel (`measurableSet_hitEvent_univ_prod_of_isCompact`). One typing note rides the transposition: the site-general grids are indexed by bare naturals; at clocked continuum strata the legs re-type to rational-duration stage cuts — the tick-indexed reading of a windowed horizon is a forcing of the finitary record site, not of the grammar.

**notes** The checked floor is *off-target only*: on the target the certificate may take any value, since success needs no budget. This is deliberately weaker than a global floor, and the proven statement binds the weaker form; a catalogued rendering that states a global floor is sound but relaxable

to match. The concluded threshold  $(1 - p)q$  is the compile's choice; the underlying theorem consumes only the single inequality  $\beta^T \cdot \text{thr} \leq 1 - p$ . At this profile, guard rows land inside the one-step decay row (the kernel carries flow and jumps together); the instantaneous-jump reading, where the discount factor does not tick during a jump, is carried at the continuous-time profile by the martingale-problem discharge's no-alarm segment assembly.

## Lean statement

BernsteinLogicAux.Battery . **sound\_QDeadline**

statement bound

```
/-- **`sound_QDeadline`, discrete profile** (T6): over the target-hit form, the deadline
formula tie – the target region *under*-approximates the truth set of the
`F_{≤T}`-argument – weakens the concluded event to the truth of the reach formula:
`Pr_x[∃ t ≤ T, X t = φ] ≥ p` on the concluded region `D n {V ≤ thr}`. No measurability
is asked of the satisfaction predicate (`measure_mono` on the measurable hit event). -/
theorem sound_QDeadline {T : ℕ} {M : ℝ} {sat : S → Prop}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hB : MeasurableSet B)
  (hV : Measurable V)
  (hF : ∀ y ∈ B, sat y)
  (hβ : 0 ≤ β)
  (hθF : β ^ T * thr ≤ 1 - p)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ B → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ B → K y (D ∪ B) = 1)
  (hint : ∀ y ∈ D, y ∉ B → Integrable V (K y))
  (hdecay : ∀ y ∈ D, y ∉ B → ∫ z, V z ∂K y ≤ β * V y)
  (hfloor : ∀ y ∈ Dbar, y ∉ B → 1 ≤ V y)
  (hnonneg : ∀ y ∈ Dbar, 0 ≤ V y)
  (hM : ∀ y ∈ Dbar, V y ≤ M)
  (hx : x ∈ D) (hthr : V x ≤ thr) :
  ENNReal.ofReal p ≤ pathMeasure K x {ω | ∃ k ≤ T, sat (ω k)}
```

BernsteinLogicAux.Battery . **sound\_QDeadline\_hit**

statement bound

```
/-- **`sound_QDeadline`, discrete profile, target-hit form** (T6 – the safety battery's
deadline successor – at the discrete-kernel profile). Hypotheses are the
decay rows (`∫ V ∂K y ≤ β·V y` off the target `B`, the discrete `QV ≤ -θV` with
`β = e^{-θ}`), floors **off** the target, nonnegs, a certificate bound `M` on the cover,
side conditions (R3)/(R4), and the deadline threshold arithmetic
`(θ_F): β ^ T * thr ≤ 1 - p` – the inequality form of the compile's `thr = (1-p)·q` with
`q ≤ e^{θT}`. Conclusion: from any start in `D n {V ≤ thr}`, the chain visits `B` by time
`T` with probability at least `p`. Consumes the landed geometric-drift engine
`BernsteinLogic.Deadline.deadline_tail` at the enlarged target `B ∪ D^c`,
then removes the escape branch with `ae_no_escape`. -/
theorem sound_QDeadline_hit {T : ℕ} {M : ℝ}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hB : MeasurableSet B)
  (hV : Measurable V)
  (hβ : 0 ≤ β)
  (hθF : β ^ T * thr ≤ 1 - p)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ B → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ B → K y (D ∪ B) = 1)
  (hint : ∀ y ∈ D, y ∉ B → Integrable V (K y))
  (hdecay : ∀ y ∈ D, y ∉ B → ∫ z, V z ∂K y ≤ β * V y)
  (hfloor : ∀ y ∈ Dbar, y ∉ B → 1 ≤ V y)
  (hnonneg : ∀ y ∈ Dbar, 0 ≤ V y)
  (hM : ∀ y ∈ Dbar, V y ≤ M)
  (hx : x ∈ D) (hthr : V x ≤ thr) :
  ENNReal.ofReal p ≤ pathMeasure K x {ω | ∃ k ≤ T, ω k ∈ B}
```

The catalogue form: the  $(\theta, q)$  parameterization, concluding on  $\{V \leq (1 - p)q\}$  from the Taylor leaf and the drift tie  $\beta \leq e^{-\theta}$ , with both transcendental facts inside the lemma. It is a wrapper over `sound_QDeadline`, which carries the general inequality form  $\beta^T \cdot \text{thr} \leq 1 - p$ .

BernsteinLogicAux.QtslBattery . **sound\_QDeadlineBattery**

statement bound

```
/-- **`sound_QDeadlineBattery`** (QTSL-rules.toml row `QDeadlineBattery` –
the catalogue's  $(\theta, q)$  parameterization of T6, discrete-kernel profile).
```

The concluded region is  $\{V \leq (1 - p) \cdot q\}$ : the deadline threshold  
 $\text{thr}_F = \text{thr} \cdot q$  in the staged  $\text{q\_mul}$  vocabulary atop  $\text{thr} = 1 - p$ .  
 Its arithmetic premises are

- \* the **\*\*T7 Taylor leaf\*\***  $q \leq \sum_{k \leq n} (\theta T)^k / k!$  – the one closed wire-free comparison the checker replays in exact  $\mathbb{Q}$  (rule row  $\text{TaylorBound } (T7)$ , alias  $\rightarrow \text{PConstExact}$ ), and
- \* the **\*\*drift tie\*\***  $\beta \leq e^{-\theta}$  – the producer's discrete geometric rate matching the catalogue's  $\beta = e^{-\theta}$  (any smaller rate is sound).

The transcendental facts –  $\sum_{k \leq n} (\theta T)^k / k! \leq e^{\theta T}$  ( $\text{Real.sum\_le\_exp\_of\_nonneg}$ ) and  $e^{-\theta} \cdot e^{\theta} = 1$  – live in this lemma, never in the checker (the catalogue's "the checker never needs to know  $e$  exists"). Wrapper over  $\text{sound\_QDeadline}$ , which carries the general inequality form  $\beta \wedge T \cdot \text{thr} \leq 1 - p$ .

System bridge: for any System whose chart compilation at the discrete-kernel profile yields  $K$ , this is the concluded  $\text{QRegionSat}$ 's reading on that System's records. -/

```
theorem sound_QDeadlineBattery {n : ℕ} {sat : S → Prop}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hB : MeasurableSet B)
  (hV : Measurable V)
  (hF : ∀ y ∈ B, sat y)
  (hβ : 0 ≤ β) (hβθ : β ≤ Real.exp (-θ)) (hθ : 0 ≤ θ)
  (hq0 : 0 ≤ q)
  (hq : q ≤ ∑ k ∈ range (n + 1), (θ * T) ^ k / (Nat.factorial k : ℝ))
  (hp : p ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ B → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ B → K y (D ∪ B) = 1)
  (hint : ∀ y ∈ D, y ∉ B → Integrable V (K y))
  (hdecay : ∀ y ∈ D, y ∉ B → ∫ z, V z ∂K y ≤ β * V y)
  (hfloor : ∀ y ∈ Dbar, y ∉ B → 1 ≤ V y)
  (hnonneg : ∀ y ∈ Dbar, 0 ≤ V y)
  (hM : ∀ y ∈ Dbar, V y ≤ M)
  (hx : x ∈ D) (hthr : V x ≤ (1 - p) * q) :
  ENNReal.ofReal p ≤ pathMeasure K x {ω | ∃ k ≤ T, sat (ω k)}
```

**The general-stochastic instance, assembled.** The deadline bound on the timed skeleton of the segment model: the  $\theta$ -discounted generator rows enter per segment, the clock is a coordinate of the timed chain, and the conclusion is the displayed deadline-overrun bound on that chain's events — headline, off-target, and battery forms.

BernsteinLogicAux.LayerD.Deadline . **sound\_GSHS\_deadline**

statement bound

```
/-- **The GSHS deadline tail bound** (T6, headline form). Inherited
premises, stated (the leg's discipline – never silent):

* `hSegD : SegmentDynkin M θ V` at the discount `θ` – **the  $\theta > 0$  premise**:  

  the B1 row compiled at positive discount; the  

  statement is typed at  $\theta \leq \theta$  and degenerates to the trivial floor bound  

  at  $\theta = 0$ .

* `hFloor : ∀ y, ε ≤ V y` – **the F7 floor premise**:  

  the positive floor on the working region, conclusion scaled by  $1/\epsilon$ .  

  Without it the conclusion is unsound (the review's two-point witness:  $V \equiv 0$   

  off-target satisfies every remaining hypothesis while the deadline fails  

  surely).
```

Conclusion: the probability that the timed skeleton chain ever presents a value stamped past the deadline  $T$  is at most  $e^{-\theta T} V(x)/\epsilon$  – the T6 display  $\text{Pr}[\tau_B > T] \leq e^{-\theta T} V(x_0)/\epsilon$  in the value-presentation reading: instantiated at the stopped presentation  $\text{Rep}(P^{\{tB\}})$  with halts into the target ("the stop IS the target" – a design choice made here and visible), a path outside this event either reached the target by  $T$  or stalled (Zeno / vanished – the residues emptied a.s. for in-scope models by the prose W-Z-freedom theorem). Consumes  $\text{Battery.measure\_reach\_le}$  (=  $\text{Ville.reach\_avoid}$ ) over the

```

timed chain: the deadline battery rides literally the same engine as T5,
with the capped certificate  $\min(e^{\{0s\}V}, \epsilon e^{\{0T\}}/\epsilon e^{\{0T\}})$  – and no
global bound  $V \leq C$  is demanded (the cap manufactures its own). -/
theorem sound_GSHS_deadline [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {θ ε : ℝ} {T : ℝ≥0}
  (hV : Measurable V) (hθ : 0 ≤ θ)
  (hSegD : SegmentDynkin M θ V) (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) (hε : 0 < ε) (hFloor : ∀ y, ε ≤ V y) (x : State) :
  Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
    {ω | ∃ n, T < tclock (ω n)}
    ≤ ENNReal.ofReal (Real.exp (-(θ * T)) * V x / ε)

```

BernsteinLogicAux.LayerD.Deadline . **sound\_GSHS\_deadline\_offTarget**

statement bound

```

/-- **T6, off-target-floor variant** (the sharper premise, the narrower
event): the floor is demanded off the target only – the F7 floor premise
read narrowly – and the conclusion event asks for a past-deadline
presentation *outside* `B`. What this variant gives up is the crossing-jump
corner: a path whose first past-`T` presentation is the target hit itself
(the dispatch that crossed the deadline landed in `B`) misses the deadline
without entering this event – `exceed_cases` makes the dichotomy explicit.
The headline `sound_GSHS_deadline` closes the corner by flooring the whole
working region instead. -/
theorem sound_GSHS_deadline_offTarget [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {θ ε : ℝ} {T : ℝ≥0} {B : Set State}
  (hV : Measurable V) (hθ : 0 ≤ θ) (hB : MeasurableSet B)
  (hSegD : SegmentDynkin M θ V) (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) (hε : 0 < ε) (hFloorB : ∀ y ∉ B, ε ≤ V y)
  (x : State) :
  Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
    {ω | ∃ n, T < tclock (ω n) ∧ ω n ∉ tbadIn B}
    ≤ ENNReal.ofReal (Real.exp (-(θ * T)) * V x / ε)

```

BernsteinLogicAux.LayerD.Deadline . **sound\_GSHS\_deadline\_battery**

statement bound

```

/-- **The deadline battery form** (T6 as `sound_QDeadline`'s GSHS instance,
complement arithmetic): with the threshold accounting
 $p + e^{-\{0T\}} \cdot \text{thr}/\epsilon \leq 1$  and a start under the threshold, the chain's
presented clock stays  $\leq T$  at every index with probability at least  $p$ .
On a stopped-at-target instantiation with halts into the target, that event
reads *reached the target by the deadline* – up to the two stall residues
(forced-jump Zeno accumulation; vanishing into a never-terminating
segment), which are inside the event, named, and emptied a.s. for in-scope
models by the prose W-Z-freedom theorem: the honest
one-sided residue, visible rather than erased. -/
theorem sound_GSHS_deadline_battery [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {θ ε p thr : ℝ} {T : ℝ≥0}
  (hV : Measurable V) (hθ : 0 ≤ θ)
  (hSegD : SegmentDynkin M θ V) (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) (hε : 0 < ε) (hFloor : ∀ y, ε ≤ V y)
  (hθ : p + Real.exp (-(θ * T)) * thr / ε ≤ 1) (x : State) (hthr : V x ≤ thr) :
  ENNReal.ofReal p ≤ Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
    {ω | ∀ n, tclock (ω n) ≤ T}

```

**The martingale-problem profile, end to end.** The general continuous-time theorems: the discounted supermartingale is built from the undiscounted martingale-problem hypothesis by the exponential-discount construction, stopped at the début by the right-continuous optional-sampling library theorem, and the deadline conclusion follows for every system whose presentation satisfies the martingale-problem hypothesis.

QtslAux.LayerD.MPDDischarge . **sound\_GSHS\_deadline\_mp**

statement bound

```

/-- The deadline reach bound at the martingale-problem profile. -/
theorem sound_GSHS_deadline_mp [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {θ ε : ℝ} {T : ℝ≥0}
  (hV : Measurable V) (hθ : 0 ≤ θ)
  (hmp : ∀ x : State, Nonempty (MPFlow M θ V x))

```

```

(hForce : ForcedNonexpansion M V)
(hNonneg : NonnegV V) (hε : 0 < ε) (hFloor : ∀ y, ε ≤ V y) (x : State) :
Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
  {ω | ∃ n, T < tclock (ω n)}
  ≤ ENNReal.ofReal (Real.exp (-(θ * T)) * V x / ε)

```

QtslAux.LayerD.MPDischarge . **sound\_GSHS\_deadline\_battery\_mp**

statement bound

```

/-- The deadline battery at the martingale-problem profile. -/
theorem sound_GSHS_deadline_battery_mp
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {θ ∈ p thr : ℝ} {T : ℝ≥0}
  (hV : Measurable V) (hθ : 0 ≤ θ)
  (hmp : ∀ x : State, Nonempty (MPFlow M θ V x))
  (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) (hε : 0 < ε) (hFloor : ∀ y, ε ≤ V y)
  (hθ : p + Real.exp (-(θ * T)) * thr / ε ≤ 1) (x : State)
  (hthr : V x ≤ thr) :
ENNReal.ofReal p ≤ Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
  {ω | ∀ n, tclock (ω n) ≤ T}

```

**The general stratum: grid tickets by topology.** The countable-combination engine and the reach leg's measurability ticket over the record carrier: hitting commutes with unions in the set argument, and a compact target cell's visit event over a  $\sigma$ -compact stage window is Borel — issued by the site's topology, not assumed.

SuFunctor.Site.FellAcceptance . **hitEvent\_iUnion**

statement bound

```

/-- **Hitting commutes with unions in the set argument** — the engine
that turns countable set decompositions into countable event
decompositions. Arbitrary index, pure set algebra. -/
theorem hitEvent_iUnion {ι : Sort*} (A : ι → Set α) :
  hitEvent (⋃ i, A i) = ⋃ i, hitEvent (A i)

```

SuFunctor.Site.FellAcceptance . **measurableSet\_hitEvent\_iUnion**

statement bound

```

/-- Hit events pass countable unions of measurable pieces. -/
theorem measurableSet_hitEvent_iUnion {ι : Sort*} [Countable ι]
  {A : ι → Set α} (h : ∀ i, MeasurableSet (hitEvent (A i))) :
  MeasurableSet (hitEvent (⋃ i, A i))

```

SuFunctor.Site.FellAcceptance . **measurableSet\_hitEvent\_univ\_prod\_of\_isCompact**

statement bound

```

/-- **Hit-events of certified cells are Effros-measurable for free**
(the memo's headline, cached): a compact cell's visit event over a
σ-compact stage window is a countable union of Fell-closed hit
events. -/
theorem measurableSet_hitEvent_univ_prod_of_isCompact
  [SigmaCompactSpace ιW] {R : Set X} (hR : IsCompact R) :
  MeasurableSet
    (hitEvent ((Set.univ : Set ιW) ×s R) : Set (FTraj W X))

```

**Marks over the record carrier.** The deadline conclusion's general-stratum shape: acceptance-event spec families, persistent by window-calculus transitivity, with the forward judgment lemma; the backward direction is deliberately absent — deadline semantics, as the commentary states.

SuFunctor.Site.FellBehaviorSpec . **acceptanceSpecF**

statement bound

```

/-- The spec family of a per-window acceptance event (deadline, until,
Streett): clip demands of the events' indicators. -/
def acceptanceSpecF (U : ι → Set T)
  (E : ∀ i, Set (FTraj (U i) X)) (j : ι) :
  Set (MeasPen (FTraj (U j) X))

```

SuFunctor.Site.FellBehaviorSpec . **isPersistentMF\_acceptanceSpecF**

statement bound

```

theorem isPersistentMF_acceptanceSpecF (U : ι → Set T)
  (hU : ∀ i, IsOpen (U i)) (E : ∀ i, Set (FTraj (U i) X)) :
  IsPersistentMF U (acceptanceSpecF U E)

```

SuFunctor.Site.FellBehaviorSpec . satAtM\_acceptanceSpecF\_ae

statement bound

```

/-- The forward judgment: satisfying an acceptance spec at a window
forces almost-sure acceptance there. The backward direction is
deliberately absent – liveness events do not clip-restrict, so
subwindow demands are genuinely stronger (a reach demand within a
subwindow is a deadline); the module docstring carries the reading. -/
theorem satAtM_acceptanceSpecF_ae (U :  $\iota \rightarrow \text{Set } T$ )
  (E :  $\forall i, \text{Set } (\text{FTraj } (U \ i) \ X)$ ) (j :  $\iota$ )
  { $\mu$  : MeasJoint (FTraj (U j) X)}
  (h : SatAtM  $\mu$  (acceptanceSpecF U E j)) :
  ( $\mu.1$  : Measure (FTraj (U j) X)) (E j)c = 0

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Model frame and profile as at QSAFETYBATTERY: hash-bound at the discrete-kernel profile (the amended cell's own caveat), with the continuous-time profile proven at the martingale-problem presentation (hash-bound below, per the profile remark). The catalogue form binds sound\_QDeadlineBattery; sound\_QDeadline carries the general inequality it wraps.

- the displayed judgment  $\mathcal{M}, x_0 \in \{V \leq (1-p)q\} \models \text{SProbGE}(p, \text{PDeadline}(T, \varphi, b)) \mapsto$  the goal of  $\text{Real } p \leq \text{pathMeasure } K \times \{\omega \mid \exists k \leq T, \text{sat } (\omega \ k)\}$  from every start with  $x \in D$  and  $V \times \leq (1-p) \cdot q$ ; the target tie  $\mapsto \text{hF} : \forall y \in B, \text{sat } y \text{ — } B$  under-approximates the target truth set (the polarity opposite to QSAFETYBATTERY's avoid tie, both safe directions), and again no measurability is asked of sat.
- decay rows  $\mapsto \text{hdecay}$  (one-step  $\int V \leq \beta V$  on  $D \setminus B$ ; at this profile guard rows land inside it, per the note) with integrability rider hint; floors *off-target only*  $\mapsto \text{hfloor}$  (on  $\bar{D} \setminus B$  — the display's weaker, deliberately relaxable form is exactly what is bound); nonnegs  $\mapsto \text{hnonneg}$ ; the bound row  $V \leq M \mapsto \text{hM}$ .
- the TaylorBound leaf  $q \leq \sum_{k \leq n} (\theta T)^k / k! \mapsto \text{hq}$ , verbatim — the one closed comparison a checker replays in exact  $\mathbb{Q}$  — and the drift tie  $\beta \leq e^{-\theta} \mapsto \text{h}\beta\theta$ , with both transcendental facts ( $\text{Real.sum\_le\_exp\_of\_nonneg}$ ;  $e^{-\theta}e^\theta = 1$ ) discharged *inside* the lemma: the certified statement is the witness that the checker never needs to know  $e$  exists.
- sign gates  $0 \leq \beta, \theta \geq 0, 0 \leq q, p \leq 1 \mapsto \text{h}\beta/\text{h}\theta/\text{hq}/\text{hp}$ , verbatim — the base sign gate  $\text{h}\beta$  is consumed by the power-order step; side conditions and region measurability as at QSAFETYBATTERY ( $\text{hR3}/\text{hR4}$ ;  $\text{hD}/\text{hDbar}/\text{hB}/\text{hV}$ , the undisplayed regularity pack).

Width note: the wrapped sound\_QDeadline consumes only  $\beta^T \cdot \text{thr} \leq 1 - p$  — every  $(\theta, q)$  satisfying the displayed leaves yields it, so the catalogue parameterization is an instance of a strictly wider certified core; producer knobs, exactly as the roster note says.

## Lean verifier receipts

module BernsteinLogicAux.Battery; bound build 2fd1c031c80b... (bound from receipt 8d9ccb68993ea0ec, 2026-08-04T01:56:55Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 8d9ccb68993ea0ec | 2026-08-04T01:56:55Z | build+decorrelated | current |
| 5075586e4813a7c5 | 2026-08-04T03:11:30Z | build+decorrelated | current |

13 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.QtslBattery; bound build 5c75f9923f27... (bound from receipt 3fd830b4914bdeda, 2026-08-04T01:56:58Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).



| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 3fd830b4914bdeda | 2026-08-04T01:56:58Z | build+decorrelated | current |
| 96b18d8572451f65 | 2026-08-04T03:11:31Z | build+decorrelated | current |

13 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.LayerD.Deadline; bound build ae6767f0b59d... (bound from receipt 1c87bcb62534ed03, 2026-08-04T01:57:04Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 1c87bcb62534ed03 | 2026-08-04T01:57:04Z | build+decorrelated | current |
| ea6411e23e04ba3c | 2026-08-04T03:11:30Z | build+decorrelated | current |

14 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module QtslAux.LayerD.MPDIscharge; bound build bade74dd943... (bound from receipt 7228834dea10e5f9, 2026-07-24T11:04:04Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 7228834dea10e5f9 | 2026-07-24T11:04:04Z | build+decorrelated | current |

module SuFunctor.Site.FellAcceptance; bound build 651fb557bbfe... (bound from receipt aa2169560f8f571a, 2026-08-03T20:19:04Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 7ad85720de6e419e | 2026-08-03T10:13:50Z | build+decorrelated | current |
| 94a408109996d1bf | 2026-08-03T13:41:21Z | build+decorrelated | current |
| 580a7e551fab03b6 | 2026-08-03T15:39:44Z | build+decorrelated | current |
| aa2169560f8f571a | 2026-08-03T20:19:04Z | build+decorrelated | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module SuFunctor.Site.FellBehaviorSpec; bound build 345f9caf9453... (bound from receipt 72f2c3b62b6771fc, 2026-08-03T20:19:08Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| d7b846e785c1d7fb | 2026-08-03T10:13:53Z | build+decorrelated | current |
| c0e3dc15419765aa | 2026-08-03T13:41:24Z | build+decorrelated | current |
| 9a62b29c08b63b4f | 2026-08-03T15:39:47Z | build+decorrelated | current |
| 72f2c3b62b6771fc | 2026-08-03T20:19:08Z | build+decorrelated | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current



**comparator:** none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development ports the base calculus's checker chain (theory `bern_catalogue`, developed side by side with the base catalogue sources, stopping below the proof-object layer); it contains no probability or measure theory. QDEADLINEBATTERY lives one theory-extension edge above the base calculus and its soundness content is path-space measure theory (discounted supermartingales, optional stopping, an escape-null argument), for which no counterpart exists anywhere in the ported development. A HOL statement of this rule would be a new measure-theoretic campaign, not an extension of the ported checker chain; until one exists, this box is the honest state of the edifice.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail. Substrate, stopped-process machinery, and the battery pattern are stated with QSAFETYBATTERY (Lemmas R40.1–R40.3, Proposition R40.4) and cited here by reference. The theorems quoted above warrant the rule at the discrete-kernel profile; the profile remark at the end states what remains a hypothesis beyond it.

**Setting and claim.** The substrate of QSAFETYBATTERY: state space  $S$ , one-step Markov kernel  $K$ , path measure  $\Pr_x$ , canonical filtration. The rule's data: modelled domain  $D$ , hull  $\bar{D} \supseteq D$ , a target  $B$  compiled to *under*-approximate the truth set of the  $F_{\leq T}$ -argument — note the orientation flip against the safety battery's avoid set: the formula tie here is  $(F') \forall y \in B, \text{sat}(y) \rightarrow$  a measurable certificate  $V$ , a discrete decay factor  $\beta \geq 0$ , a horizon  $T \in \mathbb{N}$ , and  $p, \text{thr}$ . The checked rows, read semantically:

- (D1) *decay*: for  $y \in D \setminus B$ ,  $V$  integrable against  $K y$  and  $\int V dK y \leq \beta V(y)$  — the discrete reading of  $AV \leq -\theta V$  with  $\beta = e^{-\theta}$ . (At this profile guard rows land inside this one row; see Remark R41.8.)
- (D2) *floor, off-target*:  $V \geq 1$  on  $\bar{D} \setminus B$ .
- (D3) *nonneg*:  $V \geq 0$  on  $\bar{D}$ ; *bound*:  $V \leq M$  on  $\bar{D}$ .
- (R3)/(R4) *hull and fail-closed containment at target  $B$* : from  $y \in D \setminus B$ , one-step mass stays in  $\bar{D}$  and in  $D \cup B$ .
- ( $\Theta_F$ ) *deadline threshold*:  $\beta^T \cdot \text{thr} \leq 1 - p$ .

The claim (`sound_QDeadline`): for every  $x \in D$  with  $V(x) \leq \text{thr}$ ,

$$\Pr_x^*[\exists k \leq T, \text{sat}(\omega_k)] \geq p$$

— outer measure, as everywhere in the family: no measurability is asked of the satisfaction predicate, and the mark is inert whenever its truth set is measurable.

**Lemma R41.1** (discounted tail bound at the enlarged target). *Let  $B' := B \cup D^c$ . Under (D1)–(D3), (R3), (R4): for every  $x \in D \setminus B$ ,*

$$\Pr_x[\forall k \leq T, \omega_k \notin B'] \leq \beta^T V(x).$$

*Proof.* Clamp as in the safety battery, but with the floor forced on  $D \setminus B$  rather than off  $D$ : let  $C := \max(M, 1)$  and let  $g$  clamp  $\max(V, 1)$  into  $[0, C]$  on  $D \setminus B$  and  $\max(V, 0)$  into  $[0, C]$  elsewhere. As in Lemma R40.5:  $0 \leq g \leq C$ ;  $g \geq 1$  off  $B'$  (here the floor (D2) does the work on  $\bar{D} \setminus B$ , and the forced branch covers the rest); on  $\bar{D} \cap (D \cup B)$  — which carries all one-step mass from  $D \setminus B$ , by (R3)+(R4) — the clamp is invisible,  $g = \min(V, C)$ . And by the bound row (D3),  $V \leq M \leq C$  on the hull, so the ceiling never truncates there:  $g = V$  on the whole mass-carrying set and at  $y$  itself. The decay row therefore transfers,  $\int g dK y = \int V dK y \leq \beta V(y) = \beta g(y)$  for  $y \in D \setminus B$  — and this is where the bound row is load-bearing, not an integrability convenience: without it the transfer would need  $\min(\beta V(y), C) \leq \beta \min(V(y), C)$ , false for  $0 < \beta < 1$  wherever  $V > C$  (at  $\beta = 0$  both sides vanish; that branch is handled separately below). Likewise  $g(x) = V(x)$ . Assume first  $\beta > 0$ . Now the discounted process  $M_n := \beta^{-(n \wedge \tau_{B'})} g(\omega_{n \wedge \tau_{B'}})$  is a nonnegative supermartingale (the decay row supplies exactly the extra factor  $\beta$  the discount consumes at each surviving step), and on the miss event  $\{\forall k \leq T, \omega_k \notin B'\}$  we have  $\tau_{B'} > T$  and  $g(\omega_T) \geq 1$ , so  $M_T \geq \beta^{-T}$ . Markov's inequality at time  $T$  gives  $\Pr[\text{miss}] \leq \beta^T \mathbb{E} M_0 = \beta^T g(x) = \beta^T V(x)$ . For  $\beta = 0$  the inverse discount is unavailable and the bound holds directly: at  $T = 0$  the right side is  $V(x) = g(x) \geq 1 \geq \Pr[\text{miss}]$  (the floor off  $B'$ ), while for  $T \geq 1$  the decay row at  $\beta = 0$  forces  $\int g dK_y = 0$  for every  $y$  off  $B'$ , so —  $g \geq 1$  off  $B'$  — the kernel from any off- $B'$  point charges  $B'$  with full mass and the miss event is null, by the one-step marginal of the trajectory law — the same prefix-step disintegration Lemma R41.2 rides. The mechanized route packages this computation as the geometric-drift engine (the discounted supermartingale and its hitting-time moment bound), consumed at the enlarged target  $B'$ .  $\square$

**Lemma R41.2** (no escape). *Under (R4) at target  $B$ : for  $x \in D$ ,*

$$\Pr_x \left[ \exists n, (\forall k \leq n, \omega_k \notin B) \wedge \omega_n \notin D \right] = 0.$$

*Proof.* Write  $E_n$  for the event that some escape as displayed occurs by time  $n$ . Induction on  $n$  gives  $\Pr_x[E_n] = 0$  for every  $n$ . Base: coordinate 0 is deterministically  $x \in D$ . Step: an escape at time  $n + 1$  that has not yet visited  $B$  either already escaped by time  $n$  (null by the inductive hypothesis) or steps from a state in  $D \setminus B$  to a state outside  $D \cup B$  — and that step is null, because the joint law of the  $n$ -prefix and the next step disintegrates through the one-step kernel, at which (R4) says the section  $(D \cup B)^c$  has measure zero from every surviving prefix endpoint. The displayed event is  $\bigcup_n E_n$ , null as a countable union of null events. This lemma is the almost-everywhere invariance content of (R4) that a reach conclusion cannot avoid: the safety battery got fail-closure for free by enlarging its stop set — leaving counts as failing is conservative there — but a reach guarantee cannot count escape as success, so the escape event must be shown null outright.  $\square$

**Theorem R41.3** (target-hit form). *Under (D1)–(D3), (R3), (R4),  $(\Theta_F)$ : for every  $x \in D$  with  $V(x) \leq \text{thr}$ ,  $\Pr_x[\exists k \leq T, \omega_k \in B] \geq p$ .*

*Proof.* If  $x \in B$ , the deadline is met at time 0 almost surely (coordinate 0 is deterministic) and  $p \leq 1$  suffices — the degenerate corner, handled first so the clamp may assume  $x \notin B$ . Otherwise decompose the miss event:  $\{\text{miss } B \text{ through } T\} \subseteq \{\text{miss } B' \text{ through } T\} \cup \{\text{escape before visiting } B\}$ . The second is null (Lemma R41.2); the first has probability  $\leq \beta^T V(x) \leq \beta^T \text{thr} \leq 1 - p$  (Lemma R41.1 and  $(\Theta_F)$ ). Complement.  $\square$

**Proposition R41.4** (soundness of QDeadlineBattery). *With the formula tie  $(F')$ :  $\Pr_x^*[\exists k \leq T, \text{sat}(\omega_k)] \geq p$  for every  $x \in D$  with  $V(x) \leq \text{thr}$  — the concluded judgment read on the sublevel region.*

*Proof.*  $B$  under-approximates the truth set, so the hit event of Theorem R41.3 is contained in the reach-formula event; monotonicity of the (outer) measure. As always, no measurability is asked of the satisfaction predicate.  $\square$

**Proposition R41.5** (catalogue form: the  $(\theta, q)$  parameterization). *Suppose additionally  $\beta \leq e^{-\theta}$  with  $\theta \geq 0$  (the drift tie),  $0 \leq q \leq \sum_{k \leq n} (\theta T)^k / k!$  for some  $n$  (the Taylor leaf — the one closed, wire-free comparison the checker replays in exact rational arithmetic, discharged as a ground-fact-rule application), and  $p \leq 1$ . Then the conclusion of Proposition R41.4 holds on the region  $\{V \leq (1 - p)q\}$ .*

*Proof.*  $q \leq \sum_{k \leq n} (\theta T)^k / k! \leq e^{\theta T}$  — the first transcendental fact, living here and not in the checker, applied at the nonnegative exponent  $\theta T$  — and  $\beta^T \leq e^{-\theta T}$ , powers preserving order between nonnegative bases ( $0 \leq \beta \leq e^{-\theta}$ ,  $T \in \mathbb{N}$ ); so  $\beta^T q \leq e^{-\theta T} e^{\theta T} = 1$  — the second. Hence  $\beta^T \cdot (1 - p)q \leq 1 - p$ : hypothesis  $(\Theta_F)$  holds at  $\text{thr} = (1 - p)q$ , and Proposition R41.4 applies. The checker never needs to know  $e$  exists; soundness is indifferent to the producer knobs  $\theta$  and  $q$  — any smaller rate or budget is admissible.  $\square$

**Remark R41.6** (why the off-target floor is not decoration). Drop (D2) and take  $V \equiv 0$  with the identity kernel  $Ky = \delta_y$  at some  $x \in D \setminus B$ : decay holds ( $0 \leq \beta \cdot 0$ ), nonneg and bound hold, and  $(\Theta_F)$  permits any  $p \leq 1$  since  $\beta^T \cdot 0 = 0$ . The rule would then certify that a chain frozen forever outside the target reaches it with probability one. With the floor present the certificate dies:  $V \geq 1$  on  $\bar{D} \setminus B \ni x$  contradicts  $V(x) = 0$ . Missing the deadline must cost at least one unit of discounted certificate value — that is precisely the step  $M_T \geq \beta^{-T}$  in Lemma R41.1. And the floor is asked off the target only: on  $B$  the certificate may take any value, because success needs no budget. This checked form is deliberately weaker than a global floor, and the proven statement binds the weaker form.

**Remark R41.7** (the two intrinsic asymmetries against the safety battery). Both are consequences of concluding reach rather than avoid, and neither is an artefact of the mechanization. First, the boundedness row  $V \leq M$ : the safety battery's additive drift survives its ceiling for free —  $\int \min(V, C) dKy \leq \min(\int V dKy, C) \leq \min(V(y), C)$  — so a ceiling at  $\max(\text{thr}, 1)$  cost nothing; here the discount multiplies the certificate by  $\beta^{-n}$ , and the transfer of the decay row to the clamped certificate needs the ceiling to be invisible wherever one-step mass reaches — exactly (D3)'s bound, consumed inside Lemma R41.1 (and in the mechanized geometric-drift engine it additionally supplies integrability of the discounted certificate) — for polynomial certificates over a compact hull it is free. Second, escape: the safety battery counts leaving the modelled domain as failure by enlarging its stop set, which is conservative for an avoid conclusion but would be unsound generosity for a reach conclusion — hence Lemma R41.2, whose almost-everywhere argument is the one place this section's hypotheses stop being pointwise.

**Remark R41.8** (guards at this profile). The catalogued rule lists guard rows (jumps instantaneous). At the discrete-kernel profile the one-step kernel carries flow and enabled jumps together, so the guard rows land inside the single decay row (D1) — exactly as the safety battery's interior and guard rows land together. The instantaneous-jump reading — the discount factor does not tick during a jump, so the  $e^{\theta t}$  factor cancels across it — belongs to the continuous-time profile, where it is carried by the martingale-problem discharge of Remark R41.9.

**Remark R41.9** (profile — what is proven, at which presentations). Proven: the discrete-kernel profile, sorry-free (sound\_QDeadline\_hit, sound\_QDeadline, sound\_QDeadlineBattery). Also proven: the continuous-time profile at the martingale-problem presentation (sound\_GSHS\_deadline\_mp, sound\_GSHS\_deadline\_battery\_mp, module QtslAux.LayerD.MPDDischarge of the semantic-universe strand's auxiliary package) — and the three ingredients this remark once filed are exactly what that discharge supplies: the  $e^{\theta t}V$  supermartingale over continuous time is the exponential-discount construction from the undiscounted martingale-problem hypothesis; optional stopping for right-continuous nonnegative supermartingales is the library theorem, consumed at the début; and the instantaneous-jump reading rides the presentation's no-alarm segment assembly. The martingale-problem theorems are hash-bound by this catalogue's binder via the qtsl-aux package registration, as for the safety sibling, so the dashboard row carries no conditional mark — each named profile is proven, and its theorems are bound.

*Remark* R41.10 (relation to the literature). The engine is the geometric drift condition of the Foster–Lyapunov family [181]: expected decrease by a factor per step off the target, yielding exponential hitting-time tails. The discounted supermartingale and its optional-stopping consumption are classical [64]; the barrier mechanics are the same Ville-type argument [234] the whole family rides. The certificate-form packaging — exact rational rows, transcendental facts quarantined inside a verified lemma — is this calculus's contribution rather than the literature's.

| proof step                                                           | formal anchor                                                                                                                        |
|----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| Lemma R41.1: discounted engine, clamp for the deadline reading       | BernsteinLogic.Deadline.deadline_tail (via supermartingale_inv_pow_mul, expMoment_hitting); the clamp $g$ inside sound_QDeadline_hit |
| Lemma R41.2: escape-null induction                                   | ae_no_escape (file-local; joint prefix-step law via Kernel.partialTraj_compProd_eq_map_traj); base case ae_coord_zero_mem            |
| Theorem R41.3: assembly, degenerate corner                           | sound_QDeadline_hit                                                                                                                  |
| Proposition R41.4: formula tie                                       | sound_QDeadline                                                                                                                      |
| Proposition R41.5: Taylor leaf, drift tie, transcendental quarantine | sound_QDeadlineBattery (via Real.sum_le_exp_of_nonneg)                                                                               |

## Rule 42. QUnionBound (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — *QUnionBound*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\mu(E_1) \leq a \quad \mu(E_2) \leq b \quad \text{fail} \subseteq E_1 \cup E_2}{\mu(\text{fail}) \leq a + b}$$

rule sort baked in theory `Qtsl` — realized measure-generically (arbitrary measure; no profile residue). Packaged instance (the début decomposition, pure path-set algebra over sequence space): failure of  $(\varphi_s \cup_{\leq T} \varphi)$ , at stop set *Stop* and good-exit set *R*, is contained in  $\{\text{some first Stop-visit presents a bad state}\} \cup \{\text{no Stop-visit through } T\}$  — the containment the until battery cites as its  $\text{fail} \subseteq E_1 \cup E_2$  premise.

QUnionBound is the battery family's glue: from two probability bounds and an event decomposition it concludes that the sum of the budgets bounds the failure event —  $\Pr[E_1] \leq a$ ,  $\Pr[E_2] \leq b$ , and  $\text{fail} \subseteq E_1 \cup E_2$  give  $\Pr[\text{fail}] \leq a + b$ . Its semantic content is monotonicity plus subadditivity (Boole's inequality), stated over an *arbitrary* measure: the rule is profile-generic, and its instance at any system-semantics profile is the same lemma applied at that profile's path measure. Its dashboard row accordingly carries an unconditional check with no profile discharge even needed — there was never a profile residue to discharge (the battery siblings' rows earned theirs through the martingale-problem discharge; this one is measure-generic from birth).

The hypothesis  $\text{fail} \subseteq E_1 \cup E_2$  is where all the content concentrates. The rule is sound for *any* decomposition; the citing rule must justify the containment. The packaged instance — the one the bounded-until battery cites — is the début decomposition: failure of “stay safe until the target, by the deadline” is contained in  $\{\text{some first stop-set visit presents a bad state}\} \cup \{\text{no stop-set visit by the horizon}\}$ . That containment is pure path-set algebra over sequence space: no measure, no measurability, only the well-ordering of the time axis.

A producer rarely writes this node by hand; it appears where a composite temporal property splits its failure modes across two certificates, and what the checker recomputes at such a node is precisely the containment side of the decomposition (or it discharges it by citing the packaged path-set lemma). The same additivity doubles as the slack-composition device in assume-guarantee composition, where interface slack budgets add across a wiring — one lemma, two consumers.

Why core rather than admissible: there is nothing to unpack into. Subadditivity of measure is not derivable from the other battery rules; it is new — deliberately small — semantic content at the measure level, and keeping it a separate node keeps every consumer's soundness proof honest about where its additivity comes from.

notes Stated in two forms: an extended-nonneg form (the primitive), and a real-valued form in the shape battery bounds actually arrive (certificate initial values), whose nonnegativity hypotheses are free at every use site.

### Lean statement

BernsteinLogicAux.QtslBattery . sound\_QUnionBound

statement bound

```
-- **`sound_QUnionBound`** ( $\mathbb{R}_{\geq 0\infty}$  form). Subadditivity through an event
decomposition — the entire semantic content of the `QUnionBound` rule
(QTSL-rules.toml row `QUnionBound`). -/
theorem sound_QUnionBound {Ω : Type*} [MeasurableSpace Ω] (μ : Measure Ω)
  {E₁ E₂ F : Set Ω} {a b : ℝ≥0∞}
  (h₁ : μ E₁ ≤ a) (h₂ : μ E₂ ≤ b) (hsub : F ⊆ E₁ ∪ E₂) :
  μ F ≤ a + b
```

BernsteinLogicAux.QtslBattery . sound\_QUnionBound\_ofReal

statement bound

```
-- **`sound_QUnionBound`** (`ENNReal.ofReal` form), in the shape the battery
bounds arrive: real-valued certificate bounds `a`, `b` combine to `a + b`.
The nonnegativity hypotheses are free at every use site (certificate initial
values, nonneg rows). -/
theorem sound_QUnionBound_ofReal {Ω : Type*} [MeasurableSpace Ω] (μ : Measure Ω)
  {E₁ E₂ F : Set Ω} {a b : ℝ} (ha : 0 ≤ a) (hb : 0 ≤ b)
  (h₁ : μ E₁ ≤ ENNReal.ofReal a) (h₂ : μ E₂ ≤ ENNReal.ofReal b)
  (hsub : F ⊆ E₁ ∪ E₂) :
  μ F ≤ ENNReal.ofReal (a + b)
```

BernsteinLogicAux.QtslBattery . compl\_firstExit\_subset

statement bound

```

/-- **The first-exit event decomposition** (the `QUnionBound` premise's
début decomposition): over
sequence space, the complement of the first-exit-success event — some time
`n ≤ T` whose state is in the good exit set `Good` with no earlier visit to
the stop set `Stop` — is contained in the union of {some `Stop`-visit is a
*first* visit presenting a bad state (`Stop \ Good`)} and {no `Stop`-visit
through `T`}. Pure path-set algebra, no measure and no measurability; the
until battery instantiates `Stop := (A ∪ R) ∪ D^c`, `Good := R`. This is the
lemma the `QUnionBound` premise `fail ⊆ E1 ∪ E2` cites for T8. -/
theorem compl_firstExit_subset {S : Type*} (Stop Good : Set S) (T : ℕ) :
  {ω : ℕ → S | ∃ n ≤ T, ω n ∈ Good ∧ ∀ j < n, ω j ∉ Stop}^c ⊆
  {ω : ℕ → S | ∃ n, ω n ∈ Stop \ Good ∧ ∀ j < n, ω j ∉ Stop} ∪
  {ω : ℕ → S | ∀ k ≤ T, ω k ∉ Stop}

```

## Correspondence (Steele display ↔ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The cleanest dictionary in the Q-family — the certified statement is the displayed rule at full measure-generality, with no profile residue.

- $\mu \mapsto$  an arbitrary Measure  $\mu$  on an arbitrary measurable space — the amended display's own generality (*wider* than the original display's path-probability reading, in the safe direction): the rule's GSHS instance and the AG slack-composition use are the same lemma at other measures (the roster's “one item, two consumers”).
- the three premises  $\mu(E_1) \leq a$ ,  $\mu(E_2) \leq b$ ,  $\text{fail} \subseteq E_1 \cup E_2 \mapsto h_1, h_2, \text{hsub}$ , verbatim; conclusion  $\mu(\text{fail}) \leq a + b \mapsto$  the goal, verbatim. No hypothesis is unused and nothing undisplayed is demanded — not even measurability of the events (monotonicity and subadditivity hold for outer measure).
- the ofReal companion carries the same rule in the shape battery bounds arrive (real-valued  $a, b$ ), its two nonnegativity riders free at every use site, per the note.
- the packaged instance — the flavor text's début decomposition —  $\mapsto$  compl\_firstExit\_subset, certified alongside: pure path-set algebra over sequence space (no measure, no measurability), delivering exactly the  $\text{fail} \subseteq E_1 \cup E_2$  premise the until battery cites at  $\text{Stop} := (A \cup R) \cup D^c$ ,  $\text{Good} := R$ .

## Lean verifier receipts

module BernsteinLogicAux.QtslBattery; bound build 5c75f9923f27... (bound from receipt 3fd830b4914bdeda, 2026-08-04T01:56:58Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 3fd830b4914bdeda | 2026-08-04T01:56:58Z | build+decorrelated | current |
| 96b18d8572451f65 | 2026-08-04T03:11:31Z | build+decorrelated | current |

13 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development ports the base calculus's checker chain (theory `bern_catalogue`, developed side by side with the base catalogue sources, stopping below the proof-object layer); it contains no probability or measure theory. `QUNIONBOUND` lives one theory-extension edge above the base calculus, and even its small soundness content (measure subadditivity, a path-set decomposition over sequence space) has no counterpart anywhere in the ported development. A HOL statement of this rule would open a measure-theory development, not extend the ported checker chain; until one exists, this box is the honest state of the edifice.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1) — the battery family's smallest member, and its only *measure-generic* one: the theorems below are stated over an arbitrary measure, so the rule carries no profile conditionality at all — its instance at any system-semantics profile is the same lemma applied at that profile's path measure. A Putnam grader would accept the one-line proof and then ask where the content went; the honest answer — into the decomposition hypothesis — is [Remark R42.4](#).

**Setting and claim.** Let  $(\Omega, \mu)$  be any measurable space with a measure, read throughout as the induced outer measure on arbitrary subsets (coinciding with  $\mu$  on measurable sets) — no measurability is asked of any event below, and both monotonicity and finite subadditivity hold in that reading. The rule: from  $\mu(E_1) \leq a$ ,  $\mu(E_2) \leq b$ , and a decomposition  $\text{fail} \subseteq E_1 \cup E_2$ , conclude  $\mu(\text{fail}) \leq a + b$ . In its battery consumption,  $\mu$  is the path measure  $\text{Pr}_x$  of the profile at hand,  $\text{fail}$  is the failure event of a composite temporal property, and  $a, b$  are the initial values of two certificates.

**Theorem R42.1** (soundness of `QUNIONBOUND`).  $\mu(E_1) \leq a, \mu(E_2) \leq b$ , and  $\text{fail} \subseteq E_1 \cup E_2$  imply  $\mu(\text{fail}) \leq a + b$ .

*Proof.* Monotonicity, then subadditivity (Boole's inequality):  $\mu(\text{fail}) \leq \mu(E_1 \cup E_2) \leq \mu(E_1) + \mu(E_2) \leq a + b$ . Note what is *not* used: no measurability of *any* of the three events (the Setting's outer-measure reading supplies monotonicity and finite subadditivity on arbitrary subsets), no disjointness of  $E_1, E_2$ , no probability normalization — the statement holds for any measure, which is exactly what makes the rule profile-generic.  $\square$

**Proposition R42.2** (real-valued form). *With  $a, b \geq 0$  real and the bounds given as  $\mu(E_i) \leq \text{ofReal-embedded budgets}$ , the conclusion is  $\mu(\text{fail}) \leq a + b$  under the same embedding.*

*Proof.* The embedding of nonnegative reals into extended nonnegative reals is additive on nonnegatives; apply Theorem [R42.1](#). This is the shape battery bounds actually arrive in — certificate initial values — and its nonnegativity hypotheses are free at every use site (certificate nonneg rows).  $\square$

**Lemma R42.3** (the *début* decomposition — the packaged instance). *Over sequence space  $S^{\mathbb{N}}$ , fix a stop set  $\text{Stop} \subseteq S$ , a good-exit set  $\text{Good} \subseteq S$ , and a horizon  $T$ . Then the complement of the first-exit-success event — some  $n \leq T$  with  $\omega_n \in \text{Good}$  and no earlier visit to  $\text{Stop}$  — is contained in*

$$\{\exists n, \omega_n \in \text{Stop} \setminus \text{Good} \wedge \forall j < n, \omega_j \notin \text{Stop}\} \cup \{\forall k \leq T, \omega_k \notin \text{Stop}\},$$

*i.e. in  $\{\text{some first Stop-visit presents a bad state}\} \cup \{\text{no Stop-visit through } T\}$ .*

*Proof.* Suppose the path is not a first-exit success. If it never visits  $\text{Stop}$  through  $T$ , the second disjunct holds. Otherwise some  $k \leq T$  has  $\omega_k \in \text{Stop}$ ; let  $n$  be the *least* index with  $\omega_n \in \text{Stop}$  (the well-ordering of  $\mathbb{N}$  — this is the entire “*début*” content; at continuous time the first-entry construction is a genuine theorem, while over  $\mathbb{N}$  it is a minimum). Then  $n \leq k \leq T$  and no  $j < n$  visits  $\text{Stop}$ ; were  $\omega_n \in \text{Good}$ , the path would be a first-exit success at  $n$  — contradiction. So  $\omega_n \in \text{Stop} \setminus \text{Good}$ , and  $n$  is a first visit: the first disjunct holds. No measure, no measurability, no topology was consulted — the lemma is pure path-set algebra.  $\square$

**Remark R42.4** (where the content lives). The rule is sound for *any* events  $E_1, E_2$ ; every gram of content sits in the hypothesis  $\text{fail} \subseteq E_1 \cup E_2$ , which the citing rule must justify and the checker must see justified. Drop it and the conclusion is unrelated to the premises — one could “bound” any event by any two budgets. In the bounded-until battery's use, Lemma [R42.3](#) is the justification, at  $\text{Stop} := (A \cup R) \cup D^c$  and  $\text{Good} := R$ : failure of “stay safe until the target, by the deadline” splits into a bad first exit (bounded by the avoid certificate) and no exit by the horizon (bounded by the deadline certificate). The split covers every failure: a failing path either never visits  $\text{Stop}$  through the horizon (the second disjunct) or has a first  $\text{Stop}$ -visit, which either is or is not a good landing — and a good landing, since  $R \subseteq \text{Stop}$  in this instance, is a success, so no third thing remains.

**Remark R42.5** (measure-genericity, stated sibling-aware). Within this book's battery family — safety, deadline, until, and this rule — the other three carry conditional marks (now recording, for safety, deadline, and until, that their continuous-time martingale-problem theorems are cited rather than hash-bound; see each profile remark); this rule alone is warranted unconditionally, because its statement never mentions a kernel, a path construction, or a filtration. The prediction this remark has carried since its first

edition held: the continuous-time discharges of the siblings consume these same two theorems verbatim — nothing here needed to be touched. (The *début* lemma is the sequence-space instance of its decomposition; a continuous-time consumer would supply its own first-entry construction — a genuine theorem there, as the lemma's proof notes — through those same transports.)

*Remark* R42.6 (the second consumer). The same additivity is the slack-composition device in assume-guarantee composition: when interface contracts compose across a wiring, their slack budgets add, and the addition is this rule's theorem applied to the composed failure events. One lemma, two consumers — which is the reason the rule is kept as a separate node rather than inlined into the until battery: every consumer's soundness proof stays honest about where its additivity comes from.

*Remark* R42.7 (why core, and why this small). There is nothing to unpack into: subadditivity of measure is not derivable from the other battery rules — each of them concludes a bound on a single certificate-shaped event from kernel rows, and none mentions two arbitrary events, let alone their union — so the rule cannot be admissible: it is new semantic content, deliberately minimal. Its smallness is a design decision inherited from the base calculus's taste: keep kernel-level content atomic, let composite rules be admissible unpackings over the atoms.

| proof step                    | formal anchor                |                          |             |        |
|-------------------------------|------------------------------|--------------------------|-------------|--------|
| Theorem R42.1                 | sound_QUnionBound            |                          |             |        |
| Proposition R42.2             | sound_QUnionBound_ofReal     |                          |             |        |
| Lemma R42.3                   | compl_firstExit_subset       |                          |             |        |
| the until-battery consumption | the                          | sound_QUnionBound_ofReal | application | inside |
|                               | sound_QUntilBattery_firstHit |                          |             |        |



## Rule 43. QUNTILBATTERY (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — *QUntilBattery*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\begin{array}{c}
 E_{\text{bad}} := \{\text{the first exit from } D \setminus (A \cup R) \text{ lands in } A \cup D^c\}, \quad E_{\text{miss}} := \{A \cup R \text{ not reached by } T\} \\
 \llbracket \neg \varphi_s \rrbracket \subseteq A, \quad R \subseteq \llbracket \varphi \rrbracket \text{ (formula ties)} \\
 \mu_x(E_{\text{bad}}) \leq V_a(x) \text{ (the QSafetyBattery slice for } V_a \text{ at stop set } (A \cup R) \cup D^c \text{; floor on } A \text{ only)} \\
 \mu_x(E_{\text{miss}}) \leq \beta^T V_d(x) \text{ (the QDeadlineBattery slice for } V_d \text{ at target } A \cup R) \\
 \llbracket \neg(\varphi_s \cup_{\leq T} \varphi) \rrbracket \subseteq E_{\text{bad}} \cup E_{\text{miss}} \text{ (the debut decomposition; consumed by QUnionBound)} \\
 0 \leq \beta, \quad 0 \leq V_d(x), \quad \beta^T \leq c \text{ (sign gates; rate tie)} \\
 \hline
 \mathcal{M}, x_0 \in \{V_a + c V_d \leq 1 - p\} \models \text{SProbGE}(p, \text{PUntil}(T, \varphi_s, \varphi))
 \end{array}$$

rule sort baked in theory `qtsl` (*QUntilBattery*, avoidance and target certificates as separate family fields), the concluded affine region atom (*qraAffine*) baked beside it — the display is the realized form, as the discrete-profile assembly binds it. The two slice premises are the sibling batteries' kits re-aimed — the avoid slice is the safety kit at the composite stop set with its floor asked on the bad exit  $A$  only, the deadline slice drives into  $A \cup R$  by the horizon — and each premise is the measure bound that slice's rows conclude; the rows themselves are the sibling entries' premises. Catalogue instance of the rate tie:  $c = q^{-1}$  with  $q \leq e^{\theta T}$ ,  $\beta = e^{-\theta}$  — fully rational at the discrete profile, no transcendental leaf. The conclusion is the region-conditioned judgment in constructor vocabulary, its composite region the baked *qraAffine* atom over  $V_a + c V_d$  at threshold  $1 - p$ .

*QUNTILBATTERY* concludes the bounded-until judgment  $\text{SProbGE}(p, \text{PUntil}(T, \varphi_s, \varphi))$  (surface syntax  $\text{P}_{\geq p}[\varphi_s \cup_{\leq T} \varphi]$ ) on a composite *affine* region *qraAffine*( $[(1, V_a), (c, V_d)], 1 - p$ ) — the sublevel set  $\{V_a + c V_d \leq 1 - p\}$  — from two certificates doing different work: an avoid certificate  $V_a$  guarding the safe-side formula  $\varphi_s$  until the target is reached, and a deadline certificate  $V_d$  driving the process into the target by the horizon. It is this catalogue's exemplar admissible rule at the satisfaction stratum: one node unpacks into two battery slices and one *QUnionBound* application, and the mechanized soundness proof is that unpacking — it consumes the deadline battery's theorem as a black box, applies the union bound once, and never re-proves either battery.

The two floors do different work, and the asymmetry is the rule's design.  $V_a$ 's floor is asked on the bad-exit set  $A$  *only* — the avoid certificate is silent on  $R \setminus A$ , so a landing there costs no certificate budget: the rule does not charge for winning (a landing in  $A \cap R$ , when the formula ties permit one, succeeds but is still priced as avoid-territory).  $V_d$ 's floor is on all of the hull off the target, exactly as its parent deadline battery demands. Fail-closed: leaving the modelled domain joins the stop set on the *bad* side, while an  $R$ -landing succeeds wherever it lands (the target's formula tie is global).

What the checker recomputes: the side conditions inherited from both battery slices, plus the one arithmetic hypothesis binding the slices together — the rate tie  $\beta^T \leq c$  on the affine atom's second coefficient. The catalogued instance  $c = q^{-1}$  with  $q \leq e^{\theta T}$  and  $\beta = e^{-\theta}$  satisfies the tie; at this profile the whole certificate is exact rational arithmetic, with no transcendental leaf (one becomes available by the deadline battery's Taylor-leaf pattern when  $\theta$ -rates are wanted). Drop the tie and the affine atom under-budgets the miss mass; drop either slice's rows and that slice's own counterexamples apply.

Tier, argued: admissible, not alias — three premise sub-derivations, not a respelling of one. And deliberately not monolithic core: a third self-contained lemma would duplicate two batteries' soundness; the mechanized assembly vindicates the choice by factoring through the named machinery instead. The dashboard mark states the status: hash-bound at the discrete-kernel profile; proven at the continuous-time martingale-problem profile (`sound_GSHS_until_mp`, `sound_GSHS_until_battery_mp`, `QtslAux.LayerD.MPDDischarge`) — hash-bound below via the `qtsl-aux` package registration, so the row carries no conditional mark: each named profile is proven, and its theorems are bound.

At the general stratum the until acceptance atom is stated over the semantic universe's record carrier directly (the semantic-universe functor package, hash-bound below): at some grid index the reach leg is hit and the avoid leg is missed (*untilEvent*) — a countable Boolean combination of hit-events, measurable from per-component tickets alone (*measurableSet\_untilEvent*). The clocked instantiation shape is recorded at the corollary: reach legs closed (a  $\leq$ -cut of stages times a closed target), avoid legs open (a  $<$ -cut times an open unsafe region) — and miss-events of open sets are measurable with no compactness anywhere. As at the deadline battery, the until conclusion is a mark ( $\exists$ -face): persistent by the same window-calculus engine, with the forward judgment lemma and the same deliberate absence of a backward direction — the deadline semantics the two batteries share.

notes The concluded region is the affine region atom with rows  $[(1, V_a), (c, V_d)]$  and threshold  $1 - p$  — an affine sublevel region over certificate values, the region-vocabulary generalization whose  $k = 1$  case is the ordinary sublevel atom.

### Lean statement

`BernsteinLogicAux.QtslBattery . sound_QUntilBattery`

statement bound

```

/-- **`sound_QUntilBattery`, discrete profile** (T8): over the first-hit
form, the two formula ties – `A` over-approximates the falsity set of the
safe-side formula ( $\neg \text{sat}_s y \rightarrow y \in A$ , the T13 OUTWARD direction), `R`
under-approximates the truth set of the target formula ( $\forall y \in R, \text{sat } y$ ) –
weaken the concluded event to the truth of  $\phi_s U_{\leq T} \phi$  along the path.
No measurability is asked of either satisfaction predicate (measure_mono
on the measurable first-hit event). -/
theorem sound_QUntilBattery {sat_s sat : S → Prop}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar)
  (hA : MeasurableSet A) (hR : MeasurableSet R)
  (hVa : Measurable Va) (hVd : Measurable Vd)
  (hFs :  $\forall y, \neg \text{sat}_s y \rightarrow y \in A$ ) (hFt :  $\forall y \in R, \text{sat } y$ )
  (hβ :  $0 \leq \beta$ ) (hc :  $\beta \wedge T \leq c$ )
  (hDsub :  $D \subseteq Dbar$ )
  (hR3 :  $\forall y \in D, y \notin A \cup R \rightarrow K y \text{ Dbar} = 1$ )
  (hR4 :  $\forall y \in D, y \notin A \cup R \rightarrow K y (D \cup (A \cup R)) = 1$ )
  (hVaint :  $\forall y \in D, y \notin A \cup R \rightarrow \text{Integrable } Va (K y)$ )
  (hVadrift :  $\forall y \in D, y \notin A \cup R \rightarrow \int z, Va z \partial K y \leq Va y$ )
  (hVafloor :  $\forall y \in A, y \in Dbar \rightarrow 1 \leq Va y$ )
  (hVanonneg :  $\forall y \in Dbar, 0 \leq Va y$ )
  (hVdint :  $\forall y \in D, y \notin A \cup R \rightarrow \text{Integrable } Vd (K y)$ )
  (hVddecay :  $\forall y \in D, y \notin A \cup R \rightarrow \int z, Vd z \partial K y \leq \beta * Vd y$ )
  (hVdfloor :  $\forall y \in Dbar, y \notin A \cup R \rightarrow 1 \leq Vd y$ )
  (hVdnonneg :  $\forall y \in Dbar, 0 \leq Vd y$ )
  (hVdM :  $\forall y \in Dbar, Vd y \leq M$ )
  (hx :  $x \in D$ ) (hthr :  $Va x + c * Vd x \leq 1 - p$ ) :
  ENNReal.ofReal p ≤ pathMeasure K x
  {ω |  $\exists n \leq T, \text{sat } (\omega n) \wedge \forall j < n, \text{sat}_s (\omega j)$ }

```

BernsteinLogicAux.QtslBattery. **sound\_QUntilBattery\_firstHit**

statement bound

```

/-- **`sound_QUntilBattery`, discrete profile, first-hit form** (the
catalogue's T8 row, fail-closed reading). The rule is
admissible over {T5, T6, `QUnionBound`}, and this proof *is* that unpacking:

```

```

* the T5 slice is `measure_firstBad_le` on the composite stop set
  `Stop := (A ∪ R) ∪ D^c` with the clamped avoid certificate `V_a` – floor
  rows on `A` only, drift rows off `A ∪ R` (the process is stopped at the
  target, so `V_a` is silent on `R`);
* the T6 slice is `measure_missTarget_le` at target `A ∪ R` with the
  deadline certificate `V_d` – global floor off the target;
* the glue is one `sound_QUnionBound_ofReal` application whose event
  decomposition is `compl_firstExit_subset`.

```

The concluded region  $\{V_a + c \cdot V_d \leq 1 - p\}$  with  $\beta \wedge T \leq c$  is the `qraAffine` atom (row `[(1, V_a), (c, V_d)]`, threshold `1 - p`); the catalogue's `c = q-1` with `q ≤ e{0T}`, `β = e{-0}` satisfies  $\beta \wedge T \leq c$  – the tie is dischargeable by the same T7 leaf pattern as `sound_QDeadlineBattery`. Fail-closed: leaving the modeled domain `D` before reaching `R` counts as failure (`Dc` joins the stop set on the bad side); reaching `R` counts as success wherever it lands, since the formula tie for `R` is global.

```

System bridge: for any System whose chart compilation at
the discrete-kernel profile yields `K`, this is the concluded
`QRegionSat`'s reading on that System's records, fail-closed form. -/
theorem sound_QUntilBattery_firstHit

```

```

  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar)
  (hA : MeasurableSet A) (hR : MeasurableSet R)
  (hVa : Measurable Va) (hVd : Measurable Vd)
  (hβ :  $0 \leq \beta$ ) (hc :  $\beta \wedge T \leq c$ )
  (hDsub :  $D \subseteq Dbar$ )
  (hR3 :  $\forall y \in D, y \notin A \cup R \rightarrow K y \text{ Dbar} = 1$ )
  (hR4 :  $\forall y \in D, y \notin A \cup R \rightarrow K y (D \cup (A \cup R)) = 1$ )
  (hVaint :  $\forall y \in D, y \notin A \cup R \rightarrow \text{Integrable } Va (K y)$ )
  (hVadrift :  $\forall y \in D, y \notin A \cup R \rightarrow \int z, Va z \partial K y \leq Va y$ )

```

```

(hVafloor :  $\forall y \in A, y \in \text{Dbar} \rightarrow 1 \leq V_a y$ )
(hVanonneg :  $\forall y \in \text{Dbar}, 0 \leq V_a y$ )
(hVdint :  $\forall y \in D, y \notin A \cup R \rightarrow \text{Integrable } V_d (K y)$ )
(hVddecay :  $\forall y \in D, y \notin A \cup R \rightarrow \int z, V_d z \partial K y \leq \beta * V_d y$ )
(hVdfloor :  $\forall y \in \text{Dbar}, y \notin A \cup R \rightarrow 1 \leq V_d y$ )
(hVdnonneg :  $\forall y \in \text{Dbar}, 0 \leq V_d y$ )
(hVdM :  $\forall y \in \text{Dbar}, V_d y \leq M$ )
(hx :  $x \in D$ ) (hthr :  $\forall a x + c * V_d x \leq 1 - p$ ) :
ENNReal.ofReal p  $\leq$  pathMeasure K x
{ $\omega \mid \exists n \leq T, \omega n \in R \wedge \forall j < n, \omega j \notin (A \cup R) \cup D^c$ }

```

BernsteinLogicAux.QtslBattery .measure\_firstBad\_le

statement bound

```

/-- **The stopped avoid bound** (the until battery's T5 slice, stopped at
`τ_{AuR}`): a certificate `f`, bounded in `[0, C]`,
one-step non-increasing off the stop set `Stop`, and ` $\geq 1$ ` on a bad-exit set
`Bad`, bounds the probability that some *first* `Stop`-visit presents a
`Bad` state:  $\mu \{ \exists n, \omega n \in \text{Bad} \wedge \forall j < n, \omega j \notin \text{Stop} \} \leq f x$ .
`Supermartingale.ville_ineq` consumed on the `Battery.stopped` process at
barrier `1`. Unlike `measure_reach_le` (whose floor is on the stop set
itself), the floor here is asked only on `Bad` – which is what lets the
until battery's `V_a` be silent on the target region; `Bad  $\subseteq$  Stop` is not
required by the proof. -/
theorem measure_firstBad_le {K : Kernel S S} [IsMarkovKernel K]
  {f : S  $\rightarrow$   $\mathbb{R}$ } {Stop Bad : Set S} {C :  $\mathbb{R}$ }
  (hStop : MeasurableSet Stop) (hf : Measurable f)
  (hf0 :  $\forall y, 0 \leq f y$ ) (hfC :  $\forall y, f y \leq C$ )
  (hf1 :  $\forall y \in \text{Bad}, 1 \leq f y$ )
  (hdrift :  $\forall y, y \notin \text{Stop} \rightarrow \int z, f z \partial K y \leq f y$ ) (x : S) :
  pathMeasure K x { $\omega \mid \exists n, \omega n \in \text{Bad} \wedge \forall j < n, \omega j \notin \text{Stop}$ }
     $\leq$  ENNReal.ofReal (f x)

```

BernsteinLogicAux.QtslBattery .measure\_missTarget\_le

statement bound

```

/-- **The deadline miss bound** (the until battery's T6 slice, at target
`A  $\cup$  R`): under the deadline battery's rows at a target
`B`, the probability of missing `B` through time `T` is at most
 $\beta^T * V x$ . This is `sound_QDeadline_hit` consumed as a black box and
complemented once – the Lean proof structure mirroring the until rule's
admissible-tier unpacking, which cites the T6 battery rather than re-proving
it. The deliberately weak conclusion shape (a bound on the miss event) is
what the `QUnionBound` application consumes. -/
theorem measure_missTarget_le {K : Kernel S S} [IsMarkovKernel K]
  {D Dbar B : Set S} {V : S  $\rightarrow$   $\mathbb{R}$ } { $\beta$  :  $\mathbb{R}$ } {x : S} {T :  $\mathbb{N}$ } {M :  $\mathbb{R}$ }
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hB : MeasurableSet B)
  (hV : Measurable V) (h $\beta$  :  $0 \leq \beta$ )
  (hDsub :  $D \subseteq \text{Dbar}$ )
  (hR3 :  $\forall y \in D, y \notin B \rightarrow K y \text{ Dbar} = 1$ )
  (hR4 :  $\forall y \in D, y \notin B \rightarrow K y (D \cup B) = 1$ )
  (hint :  $\forall y \in D, y \notin B \rightarrow \text{Integrable } V (K y)$ )
  (hdecay :  $\forall y \in D, y \notin B \rightarrow \int z, V z \partial K y \leq \beta * V y$ )
  (hfloor :  $\forall y \in \text{Dbar}, y \notin B \rightarrow 1 \leq V y$ )
  (hnonneg :  $\forall y \in \text{Dbar}, 0 \leq V y$ )
  (hM :  $\forall y \in \text{Dbar}, V y \leq M$ )
  (hx :  $x \in D$ ) :
  pathMeasure K x { $\omega \mid \forall k \leq T, \omega k \notin B$ }  $\leq$  ENNReal.ofReal ( $\beta^T * V x$ )

```

**The general-stochastic instance, assembled.** The until-failure bound on the timed skeleton of the stopped presentation: failure decomposes into an unsafe presentation or a deadline overrun, both events of the one timed chain — the safety half runs the avoidance engine at zero discount (the clock drops out pointwise), the deadline half is the deadline instance below, and the glue is measure algebra. Headline and battery forms.

BernsteinLogicAux.LayerD.Until .sound\_GSHS\_until

statement bound

```

/-- **The GSHS until failure bound** (T8, headline form – the K4 event
decomposition on one chain). Hypotheses are exactly the union of the two

```

halves' (each inherited premise stated, never silent – the safety half's global bound `C`, the deadline half's `θ`-row and F7 floor `ε ≤ V\_d`): the probability that the timed skeleton of the stopped presentation ever presents an unsafe value **or** overruns the deadline is at most  $V_a(x) + e^{-\theta T} \cdot V_d(x) / \epsilon$ . -/

```

theorem sound_GSHS_until [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {Va Vd : State → ℝ} {Us : Set State} {θ ε : ℝ} {T : ℝ≥0}
  (hVa : Measurable Va) (hUs : MeasurableSet Us)
  (hSegDa : SegmentDynkin M θ Va) (hForcea : ForcedNonexpansion M Va)
  (hNonnega : NonnegV Va) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, Va y ≤ C)
  (hFloora : ∀ y ∈ Us, 1 ≤ Va y)
  (hVd : Measurable Vd) (hθ : 0 ≤ θ)
  (hSegDd : SegmentDynkin M θ Vd) (hForced : ForcedNonexpansion M Vd)
  (hNonnegd : NonnegV Vd) (hε : 0 < ε) (hFloord : ∀ y, ε ≤ Vd y)
  (x : State) :
  Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
    {ω | ∃ n, ω n ∈ tbadIn Us v T < tclock (ω n)}
    ≤ ENNReal.ofReal (Va x)
      + ENNReal.ofReal (Real.exp (-(θ * T)) * Vd x / ε)

```

BernsteinLogicAux.LayerD.Until .sound\_GSHS\_until\_battery

statement bound

/-- \*\*The until battery form\*\* (T8 as `QUntilBattery`'s GSHS instance, complement arithmetic): with the additive threshold accounting  $p + thr_a + e^{-\theta T} \cdot thr_d / \epsilon \leq 1$  and a start under both thresholds, the chain presents only safe values with the clock never past `T`, with probability at least `p`. On the stop-IS-target instantiation that event reads  $\phi_s$  held until the target, reached by the deadline – up to the stall residues (inside the event, named, emptied a.s. in-scope by W-Z-freedom). The composite-atom rendering  $\{V_a + (1/(q\epsilon)) \cdot V_d \leq 1-p\}$  of the catalogue's T8 entry maps onto ( $thr_a$ ,  $thr_d$ ) with  $q \leq e^{\theta T}$ ; the `graAffine` staging decision is schema-side and does not gate this lemma. -/

```

theorem sound_GSHS_until_battery [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {Va Vd : State → ℝ} {Us : Set State} {θ ε p thra thrd : ℝ} {T : ℝ≥0}
  (hVa : Measurable Va) (hUs : MeasurableSet Us)
  (hSegDa : SegmentDynkin M θ Va) (hForcea : ForcedNonexpansion M Va)
  (hNonnega : NonnegV Va) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, Va y ≤ C)
  (hFloora : ∀ y ∈ Us, 1 ≤ Va y)
  (hVd : Measurable Vd) (hθ : 0 ≤ θ)
  (hSegDd : SegmentDynkin M θ Vd) (hForced : ForcedNonexpansion M Vd)
  (hNonnegd : NonnegV Vd) (hε : 0 < ε) (hFloord : ∀ y, ε ≤ Vd y)
  (hθ : p + (thra + Real.exp (-(θ * T)) * thrd / ε) ≤ 1)
  (x : State) (hthra : Va x ≤ thra) (hthrd : Vd x ≤ thrd) :
  ENNReal.ofReal p ≤ Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
    {ω | ∀ n, ω n ∉ tbadIn Us ∧ tclock (ω n) ≤ T}

```

BernsteinLogicAux.LayerD.Until .sound\_GSHS\_avoid\_timed

statement bound

/-- \*\*The safety half on the timed chain\*\*: the probability that the timed skeleton ever presents a value in `Us` (alive entries **and** value-carrying halts – kills into `Us` included) is bounded by the initial safety certificate. This is `sound\_GSHS\_safety` transported to the timed chain – at  $\theta = 0$  the clock drops out of the certificate pointwise, so the same engine ( $Battery.measure\_reach\_le = K1$ ) runs unchanged. -/

```

theorem sound_GSHS_avoid_timed [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {Va : State → ℝ} {Us : Set State}
  (hVa : Measurable Va) (hUs : MeasurableSet Us)
  (hSegD : SegmentDynkin M 0 Va) (hForce : ForcedNonexpansion M Va)
  (hNonneg : NonnegV Va) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, Va y ≤ C)
  (hFloor : ∀ y ∈ Us, 1 ≤ Va y) (x : State) :
  Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
    {ω | ∃ n, ω n ∈ tbadIn Us}
    ≤ ENNReal.ofReal (Va x)

```

**The martingale-problem profile, end to end.** The general continuous-time theorems: both slices' formerly per-model constituents — the safety slice's stopping theory, the deadline slice's discounted supermartingale — are supplied by the same

martingale-problem discharge that serves the parents, composite stop set included, and the until conclusion follows for every system whose presentation satisfies the martingale-problem hypothesis.

`QtslAux.LayerD.MPDischarge . sound_GSHS_until_mp`

statement bound

```

/-- The timed until reach bound at the martingale-problem profile: both the
avoidance certificate (rate `0`) and the deadline certificate (rate `0`)
are presented by martingale problems. -/
theorem sound_GSHS_until_mp [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {Va Vd : State → ℝ} {Us : Set State} {θ ε : ℝ} {T : ℝ≥0}
  (hVa : Measurable Va) (hUs : MeasurableSet Us)
  (hmpa : ∀ x : State, Nonempty (MPFlow M θ Va x))
  (hForcea : ForcedNonexpansion M Va)
  (hNonnega : NonnegV Va) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, Va y ≤ C)
  (hFloora : ∀ y ∈ Us, 1 ≤ Va y)
  (hVd : Measurable Vd) (h0 : 0 ≤ θ)
  (hmpd : ∀ x : State, Nonempty (MPFlow M θ Vd x))
  (hForceD : ForcedNonexpansion M Vd)
  (hNonnegd : NonnegV Vd) (hε : 0 < ε) (hFloord : ∀ y, ε ≤ Vd y)
  (x : State) :
  Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
    {ω | ∃ n, ω n ∈ tbadIn Us v T < tclock (ω n)}
    ≤ ENNReal.ofReal (Va x)
      + ENNReal.ofReal (Real.exp (-(θ * T)) * Vd x / ε)

```

`QtslAux.LayerD.MPDischarge . sound_GSHS_until_battery_mp`

statement bound

```

/-- The until battery at the martingale-problem profile. -/
theorem sound_GSHS_until_battery_mp
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {Va Vd : State → ℝ} {Us : Set State} {θ ε p thra thrd : ℝ} {T : ℝ≥0}
  (hVa : Measurable Va) (hUs : MeasurableSet Us)
  (hmpa : ∀ x : State, Nonempty (MPFlow M θ Va x))
  (hForcea : ForcedNonexpansion M Va)
  (hNonnega : NonnegV Va) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, Va y ≤ C)
  (hFloora : ∀ y ∈ Us, 1 ≤ Va y)
  (hVd : Measurable Vd) (h0 : 0 ≤ θ)
  (hmpd : ∀ x : State, Nonempty (MPFlow M θ Vd x))
  (hForceD : ForcedNonexpansion M Vd)
  (hNonnegd : NonnegV Vd) (hε : 0 < ε) (hFloord : ∀ y, ε ≤ Vd y)
  (h0 : p + (thra + Real.exp (-(θ * T)) * thrd / ε) ≤ 1)
  (x : State) (hthra : Va x ≤ thra) (hthrd : Vd x ≤ thrd) :
  ENNReal.ofReal p ≤ Battery.pathMeasure (tskel M) (TSkelState.alive (0, x))
    {ω | ∀ n, ω n ∉ tbadIn Us ∧ tclock (ω n) ≤ T}

```

**The general stratum: the until atom on the record carrier.** The deadline-grid until event — some grid index at which the reach leg is hit and the avoid leg missed — as a countable Boolean combination of hit-events, measurable from per-component tickets; the clocked corollary takes closed reach legs and open avoid legs, with no compactness on the avoid side.

`SuFunctor.Site.FellAcceptance . untilEvent`

statement bound

```

/-- **The deadline-grid until event**: at some grid index, the reach
leg is hit and the avoid leg is missed. The countable-combination
skeleton of safety-until-a-deadline at the record tier: the clocked
instantiation takes `Reach n` = (stages ≤ nth cut) × target and
`Avoid n` = (stages < nth cut) × unsafe. -/
def untilEvent (Reach Avoid : ℕ → Set α) : Set (FellCloseds α)

```

`SuFunctor.Site.FellAcceptance . mem_untilEvent_iff`

statement bound

```

theorem mem_untilEvent_iff {Reach Avoid : ℕ → Set α}
  {F : FellCloseds α} :
  F ∈ untilEvent Reach Avoid ↔
  ∃ n, ((F : Set α) n Reach n).Nonempty ∧
  Disjoint (F : Set α) (Avoid n)

```

`SuFunctor.Site.FellAcceptance . measurableSet_untilEvent`

statement bound

```
theorem measurableSet_untilEvent {Reach Avoid :  $\mathbb{N} \rightarrow \text{Set } \alpha$ }
  (h1 :  $\forall n, \text{MeasurableSet } (\text{hitEvent } (\text{Reach } n))$ )
  (h2 :  $\forall n, \text{MeasurableSet } (\text{missEvent } (\text{Avoid } n))$ ) :
  MeasurableSet (untilEvent Reach Avoid)
```

SuFunctor.Site.FellAcceptance . measurableSet\_untilEvent\_of\_isClosed\_isOpen

statement bound

```
/-- The clocked until shape: closed reach legs (a ` $\leq$ `-cut of stages
times a closed target), open avoid legs (a ` $<$ `-cut times an open
unsafe region). The avoid legs' measurability costs nothing at all –
miss events of opens are Fell-closed at every stratum. -/
theorem measurableSet_untilEvent_of_isClosed_isOpen [SigmaCompactSpace  $\alpha$ ]
  {Reach Avoid :  $\mathbb{N} \rightarrow \text{Set } \alpha$ } (h1 :  $\forall n, \text{IsClosed } (\text{Reach } n)$ )
  (h2 :  $\forall n, \text{IsOpen } (\text{Avoid } n)$ ) :
  MeasurableSet (untilEvent Reach Avoid)
```

**Marks over the record carrier.** As at the deadline battery: the until conclusion's general-stratum spec family is persistent with the forward judgment lemma, and the deliberately absent backward direction is the shared deadline semantics.

SuFunctor.Site.FellBehaviorSpec . acceptanceSpecF

statement bound

```
/-- The spec family of a per-window acceptance event (deadline, until,
Streett): clip demands of the events' indicators. -/
def acceptanceSpecF (U :  $\iota \rightarrow \text{Set } T$ )
  (E :  $\forall i, \text{Set } (\text{FTraj } (U \ i) \ X)$ ) (j :  $\iota$ ) :
  Set (MeasPen (FTraj (U j) X))
```

SuFunctor.Site.FellBehaviorSpec . isPersistentMF\_acceptanceSpecF

statement bound

```
theorem isPersistentMF_acceptanceSpecF (U :  $\iota \rightarrow \text{Set } T$ )
  (hU :  $\forall i, \text{IsOpen } (U \ i)$ ) (E :  $\forall i, \text{Set } (\text{FTraj } (U \ i) \ X)$ ) :
  IsPersistentMF U (acceptanceSpecF U E)
```

SuFunctor.Site.FellBehaviorSpec . satAtM\_acceptanceSpecF\_ae

statement bound

```
/-- The forward judgment: satisfying an acceptance spec at a window
forces almost-sure acceptance there. The backward direction is
deliberately absent – liveness events do not clip-restrict, so
subwindow demands are genuinely stronger (a reach demand within a
subwindow is a deadline); the module docstring carries the reading. -/
theorem satAtM_acceptanceSpecF_ae (U :  $\iota \rightarrow \text{Set } T$ )
  (E :  $\forall i, \text{Set } (\text{FTraj } (U \ i) \ X)$ ) (j :  $\iota$ )
  { $\mu$  : MeasJoint (FTraj (U j) X)}
  (h : SatAtM  $\mu$  (acceptanceSpecF U E j)) :
  ( $\mu.1$  : Measure (FTraj (U j) X)) (E j)c = 0
```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Model frame and profile as at QSafetyBattery (discrete-kernel; continuous time at the martingale-problem presentation, hash-bound below per the profile remark). One structural honesty first: the display presents the two battery slices by their *conclusions* — the measure bounds  $\mu_x(E_{\text{bad}}) \leq V_a(x)$  and  $\mu_x(E_{\text{miss}}) \leq \beta^T V_d(x)$  — plus the début containment that QUnionBound consumes, while the certified statement sound\_QUntilBattery takes the two slices' row families *inline* (flattened hypothesis lists over one kernel) and derives those bounds internally (the witness names the waypoints: `measure_firstBad_le`, `measure_missTarget_le`, with the containment bound as `compl_firstExit_subset`); the admissible-tier unpacking is realized by the *proof*, which consumes the QDeadlineBattery theorem as a black box, not by the statement's shape.

- the concluded judgment  $\mathcal{M}, x_0 \in \{V_a + c V_d \leq 1 - p\} \models \text{SProbGE}(p, \text{PUntil}(T, \varphi_s, \varphi)) \mapsto$  the goal of `Real p ≤ pathMeasure K × { $\omega \mid \exists n \leq T, \text{sat } (\omega \ n) \wedge \forall j < n, \text{sat}_s (\omega \ j)$ }` — the until event verbatim, both satisfaction predicates measurability-free.



- the two formula ties, opposite polarities (the QClosure directions), displayed as the inclusions  $\llbracket \neg\varphi_s \rrbracket \subseteq A$  and  $R \subseteq \llbracket \varphi \rrbracket$ :  $A$  *over*-approximates the falsity set of  $\varphi_s$  (hFs) and  $R$  *under*-approximates the truth set of  $\varphi$  (hFt) — the displayed inclusions verbatim, safe direction on both sides.
- the QSAFETYBATTERY slice for  $V_a$  (rows behind the displayed bound  $\mu_x(E_{\text{bad}}) \leq V_a(x) \mapsto \text{hVadriфт/hVaint}$  (drift at the composite stop set  $(A \cup R) \cup D^c$ , encoded as  $y \in D, y \notin A \cup R$ ),  $\text{hVafloor}$  — on  $A$  *only*, the display's "floor on  $A$  only" — and  $\text{hVannonneg}$ .
- the QDEADLINEBATTERY slice for  $V_d$  (rows behind the displayed bound  $\mu_x(E_{\text{miss}}) \leq \beta^T V_d(x) \mapsto \text{hVdecay/hVdint}$  ( $\beta$ -decay off  $A \cup R$ ),  $\text{hVdfloor}$  (global off-target),  $\text{hVdnonneg}$ ,  $\text{hVdM}$ .
- the rate tie  $\beta^T \leq c$  and its displayed sign gates  $0 \leq \beta, 0 \leq V_d(x) \mapsto \text{hc}, \text{h}\beta$ , and the nonneg row read at the start point ( $\text{hVdnonneg}$  with  $\text{hx/hDsub}$ ) — the  $V_d(x)$  gate is what lets the outer assembly pass from the  $\beta^T V_d(x)$  miss bound to the affine row's  $c V_d(x)$ ; fully rational at this profile; the catalogue's  $c = q^{-1}$  is one instance.
- the concluded  $\text{qraAffine}$  atom  $[(1, V_a), (c, V_d)]$  at threshold  $1 - p \mapsto$  the start condition  $\text{hthr} : \forall a \ x + c \cdot V_d \ x \leq 1 - p$ , verbatim affine row.
- side conditions and regularity as at the siblings ( $\text{hR3/hR4}$  at the composite stop set; the measurability pack undisplayed).

Width note: beyond the two safe-direction ties, neither side is wider — every displayed row lands on a named hypothesis, and the statement's flattening loses nothing the display promises (each inline family is exactly its slice's row family, stopped where the display stops it).

## Lean verifier receipts

module BernsteinLogicAux.QtslBattery; bound build 5c75f9923f27... (bound from receipt 3fd830b4914bdeda, 2026-08-04T01:56:58Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 3fd830b4914bdeda | 2026-08-04T01:56:58Z | build+decorrelated | current |
| 96b18d8572451f65 | 2026-08-04T03:11:31Z | build+decorrelated | current |

13 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.LayerD.Until; bound build 5abcb1cb41f6... (bound from receipt dbbf5ae370de0748, 2026-08-04T01:57:07Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| dbbf5ae370de0748 | 2026-08-04T01:57:07Z | build+decorrelated | current |
| e2dab1e692fdab41 | 2026-08-04T03:11:31Z | build+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module QtslAux.LayerD.MPDIscharge; bound build bade74dd943... (bound from receipt 7228834dea10e5f9, 2026-07-24T11:04:04Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 7228834dea10e5f9 | 2026-07-24T11:04:04Z | build+decorrelated | current |

module SuFunctor.Site.FellAcceptance; bound build 651fb557bbfe... (bound from receipt aa2169560f8f571a, 2026-08-03T20:19:04Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 7ad85720de6e419e | 2026-08-03T10:13:50Z | build+decorrelated | current |
| 94a408109996d1bf | 2026-08-03T13:41:21Z | build+decorrelated | current |



|                  |                      |                    |         |
|------------------|----------------------|--------------------|---------|
| 580a7e551fab03b6 | 2026-08-03T15:39:44Z | build+decorrelated | current |
| aa2169560f8f571a | 2026-08-03T20:19:04Z | build+decorrelated | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module SuFunctor.Site.FellBehaviorSpec; bound build 345f9caf9453... (bound from receipt 72f2c3b62b6771fc, 2026-08-03T20:19:08Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| d7b846e785c1d7fb | 2026-08-03T10:13:53Z | build+decorrelated | current |
| c0e3dc15419765aa | 2026-08-03T13:41:24Z | build+decorrelated | current |
| 9a62b29c08b63b4f | 2026-08-03T15:39:47Z | build+decorrelated | current |
| 72f2c3b62b6771fc | 2026-08-03T20:19:08Z | build+decorrelated | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development ports the base calculus's checker chain (theory bern\_catalogue, developed side by side with the base catalogue sources, stopping below the proof-object layer); it contains no probability or measure theory. QUNTILBATTERY lives one theory-extension edge above the base calculus and its soundness content is path-space measure theory (a stopped-supermartingale slice, a discounted-supermartingale slice, and a union bound over a début decomposition), for which no counterpart exists anywhere in the ported development. A HOL statement of this rule would be a new measure-theoretic campaign, not an extension of the ported checker chain; until one exists, this box is the honest state of the edifice.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. The general admissible template (one node  $\rightarrow$  a derivation fragment over core and previously-justified rules, applied leaf-first) is stated in the NORM section, Recipe R16.1, and not restated here; this section gives the rule's instance. The verified theorems quoted above warrant the rule at the **discrete-kernel profile**; the profile remark at the end states what remains a hypothesis beyond it.

**Recipe R43.1** (the QUNTILBATTERY rewrite). A QUNTILBATTERY node carries: an avoid certificate  $V_a$  with its rows (drift off  $A \cup R$ , floor on  $A$  only, nonneg), a deadline certificate  $V_d$  with its rows (decay factor  $\beta$  off  $A \cup R$ , floor on all of the hull off the target, nonneg, bound), the shared side conditions (hull; fail-closed at the composite exit set  $A \cup R$ ), the sign gate  $\beta \geq 0$  with the rate tie  $\beta^T \leq c$ , and the threshold row  $V_a(x) + c V_d(x) \leq 1 - p$  on the concluded affine region. Here  $A$  over-approximates the falsity set of the safe-side formula  $\varphi_s$  and  $R$  under-approximates the truth set of the target  $\varphi$ . The node rewrites to a derivation fragment over three constituents:

$$\frac{V_a \text{ rows} \quad V_d \text{ rows} \quad \beta^T \leq c \quad V_a + c V_d \leq 1 - p \text{ on the region}}{\mathcal{M}, x_0 \in \{V_a + c V_d \leq 1 - p\} \models \text{SProbGE}(p, \text{PUntil}(T, \varphi_s, \varphi))} \sim \text{QUNTILBATTERY} \begin{array}{l} \text{(i) a QSafetyBattery slice: the battery pattern (QSAFETYBATTERY,} \\ \text{at the composite stop set } \textit{Stop} := (A \cup R) \cup D^c, \text{ certificate } V_a \\ \text{(ii) a QDeadlineBattery slice: the deadline miss bound (QDEADLINE} \\ \text{at target } A \cup R, \text{ certificate } V_d; \\ \text{(iii) one QUNIONBOUND node on the début decomposition (Ler} \end{array}$$

Honesty note, in the PCONSTEXACT tradition: the two slices are consumed through their *verified slice lemmas*, not by literal re-application of full battery nodes — the QSafetyBattery slice's floor family is deliberately weaker than a full safety node's (floor on the bad-exit set only; see [Remark R43.4](#)), and the QDeadlineBattery slice is consumed in complemented (miss-bound) form. The shipped verified object is therefore the assembled soundness lemma, whose Lean structure *is* this rewrite: it consumes the deadline theorem as a black box, applies the union bound once, and re-proves neither battery.

**Proposition R43.2** (semantic warrant, first-hit form). *On the substrate of the shared preamble, suppose: the  $V_a$  rows — drift off  $A \cup R$  within  $D$ , floor  $V_a \geq 1$  on  $A \cap \bar{D}$ , nonneg on  $\bar{D}$ ; the  $V_d$  rows — decay  $\int V_d dK y \leq \beta V_d(y)$  off  $A \cup R$  within  $D$ , floor  $V_d \geq 1$  on  $\bar{D} \setminus (A \cup R)$ , nonneg and bound  $M$  on  $\bar{D}$ ; hull and fail-closed side conditions at exit set  $A \cup R$ ;  $\beta \geq 0$  and the rate tie  $\beta^T \leq c$ . Then for every  $x \in D$  with  $V_a(x) + c V_d(x) \leq 1 - p$ ,*

$$\Pr_x[\exists n \leq T, \omega_n \in R \wedge \forall j < n, \omega_j \notin \text{Stop}] \geq p, \quad \text{Stop} = (A \cup R) \cup D^c.$$

*Proof.* Three bounds, then arithmetic. (i, the QSafetyBattery slice.) Clamp  $V_a$  as in the safety battery, with the floor forced only off  $D \cup R$  (the certificate is silent on the target away from  $A$ ). The stopped process at  $\text{Stop}$  is a nonnegative supermartingale,  $\geq 1$  whenever a first  $\text{Stop}$ -visit presents a bad state: a bad state lies in  $\text{Stop} \setminus R = (A \setminus R) \cup (D^c \setminus R)$ , and every reachable one is covered — in  $A \setminus R$  it lies in  $\bar{D}$  (a first visit is either the start, in  $D$ , or one kernel step from  $D \setminus \text{Stop}$ , confined to  $\bar{D}$  by the hull row), where the floor row on  $A \cap \bar{D}$  applies; in  $D^c \setminus R$ , which is the region off  $D \cup R$ , the forced clamp applies. Ville's inequality gives

$$E_1 := \{\text{some first Stop-visit is bad}\}, \quad \Pr_x[E_1] \leq V_a(x).$$

The floor is asked on  $A$  and off  $D \cup R$  only — in particular never on  $R \setminus A$ , which is what lets a landing there cost no  $V_a$ -budget (a landing in  $A \cap R$ , if the ties permit one, still succeeds, but the avoid floor prices it as avoid-territory; the no-charge-for-winning story is the  $R \setminus A$  story — [Remark R43.4](#)). (ii, the QDeadlineBattery slice.) The deadline battery's miss bound at target  $A \cup R$ , consumed as a black box and weakened from missing  $A \cup R$  to missing the larger  $\text{Stop}$  — avoiding  $\text{Stop} \supseteq A \cup R$  through  $T$  entails avoiding  $A \cup R$  through  $T$ , so  $E_2$  below is contained in the black box's miss event:

$$E_2 := \{\forall k \leq T, \omega_k \notin \text{Stop}\}, \quad \Pr_x[E_2] \leq \beta^T V_d(x) \leq c V_d(x),$$

the last step by the rate tie and  $V_d(x) \geq 0$ . (iii, the glue.) The d but decomposition (Lemma R42.3) at  $(\text{Stop}, R, T)$  says the complement of the concluded first-hit event is contained in  $E_1 \cup E_2$ ; one QUNIONBOUND application in real-valued form (Prop. R42.2, its nonnegativity riders supplied by the nonneg rows and by  $c \geq \beta^T \geq 0$ ) gives  $\Pr_x[\text{fail}] \leq V_a(x) + c V_d(x) \leq 1 - p$ . Complement — licit because the first-hit event is a region event over measurable regions (the profile's regularity pack), so its measure and its complement's sum to one; the weakening to possibly-nonmeasurable satisfaction events happens only in the next proposition, in the monotone direction.  $\square$

**Proposition R43.3** (soundness of QUNTILBATTERY). *With the two formula ties —  $A$  over-approximating where  $\varphi_s$  fails,  $R$  under-approximating where  $\varphi$  holds — the first-hit event weakens to the truth of the bounded-until formula along the path: for  $x$  in the affine region,  $\Pr_x[\exists n \leq T, \text{sat}(\omega_n) \wedge \forall j < n, \text{sat}_s(\omega_j)] \geq p$  — the concluded judgment  $\mathcal{M}$ ,  $x_0 \in \{V_a + c V_d \leq 1 - p\} \models \text{SProbGE}(p, \text{PUntil}(T, \varphi_s, \varphi))$ , its region the sublevel set  $\{V_a + c V_d \leq 1 - p\}$ .*

*Proof.* On the first-hit event: the landing state is in  $R$ , hence satisfies  $\varphi$ ; every earlier state avoided  $\text{Stop} \supseteq A$ , hence satisfies  $\varphi_s$  by the contrapositive of the safe-side tie. Monotonicity of the (outer) measure; neither satisfaction predicate is required measurable.  $\square$

**Remark R43.4** (two floors doing different work).  $V_a$ 's floor is on  $A$  only: the avoid certificate is silent on  $R \setminus A$ , so a landing there costs no certificate budget — the rule does not charge for winning (a state of  $A \cap R$ , when the ties permit one, is a successful landing that the avoid floor nonetheless prices as avoid-territory; the free wins are exactly the  $R \setminus A$  ones). Demand the floor on all of  $R$  too and the concluded region shrinks pointlessly: every reachable success state would need  $V_a \geq 1$ , pushing  $V_a(x) + c V_d(x)$  past any useful threshold.  $V_d$ 's floor is on all of  $\bar{D}$  off the target, exactly as its parent deadline battery requires — weaken it and [Remark R41.6](#)'s counterexample (the frozen chain certified to arrive) imports after one padding: take  $A = \emptyset$  and  $V_a \equiv 0$  (all avoid rows trivially satisfied), and the until node is its deadline slice, frozen chain and all. The asymmetry is not an optimization; it is the two slices' different jobs made visible in their price lists.

**Remark R43.5** (fail-closed, and the rate tie).  $D^c$  joins the stop set on the *bad* side: leaving the modelled domain before reaching the target counts as failure, while an  $R$ -landing succeeds wherever it lands (the target tie is global). Drop the rate tie  $\beta^T \leq c$  and step (ii)'s budget transfer fails — the affine atom would under-budget the *certified* miss bound by the gap  $(\beta^T - c) V_d(x)$  (the actual miss mass sits somewhere below the bound; it is the transfer, not the mass, that the tie protects). The catalogued instance  $c = q^{-1}$  with  $0 < q \leq e^{\theta T}$ ,  $\beta = e^{-\theta}$  satisfies the tie, and at this profile the whole certificate is exact rational arithmetic — no transcendental leaf anywhere; one becomes available through the deadline battery's Taylor-leaf pattern (Prop. R41.5) when  $\theta$ -rates are wanted.

**Remark R43.6** (tier, argued). Admissible, not alias: three premise sub-derivations (two battery slices and a union bound), not a respelling of one. And deliberately not monolithic core: a third self-contained soundness lemma would duplicate two batteries' arguments line by line — the assembled proof instead consumes the deadline theorem as a black box and the union bound as one application, which is this catalogue's admissibility discipline enforced at the Lean level. The concluded region is the affine atom with rows  $[(1, V_a), (c, V_d)]$  and threshold  $1 - p$  — the region-vocabulary generalization whose  $k = 1$  case is the ordinary sublevel atom.

**Remark R43.7** (profile — what is proven, at which presentations). Proven: the discrete-kernel profile, sorry-free (sound\_QUntilBattery\_firstHit, sound\_QUntilBattery), with both slice lemmas and the union bound as named constituents. Also proven: the continuous-time profile

at the martingale-problem presentation (`sound_GSHS_until_mp`, `sound_GSHS_until_battery_mp`, `module QtslAux.LayerD.MPDDischarge` of the semantic-universe strand's auxiliary package) — the recipe above is profile-independent in *shape*, and both parents' formerly filed constituents (the safety slice's stopping theory, the deadline slice's discounted supermartingale) are supplied by the same martingale-problem discharge that serves the parents, composite stop set included. The martingale-problem theorems are hash-bound by this catalogue's binder via the `qtsl-aux` package registration, family-wide, so the dashboard row carries no conditional mark — each named profile is proven, and its theorems are bound.

| proof step                                                | formal anchor                                                                                                                          |
|-----------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| Recipe R43.1, slice (i)                                   | <code>measure_firstBad_le</code> (Ville via Supermartingale.ville_ineq on the stopped process; floor asked on the bad-exit set only)   |
| Recipe R43.1, slice (ii)                                  | <code>measure_missTarget_le</code> (black-boxing <code>sound_QDeadline_hit</code> , complemented once)                                 |
| Recipe R43.1, glue (iii)                                  | <code>sound_QUnionBound_ofReal</code> on <code>compl_firstExit_subset</code>                                                           |
| Proposition R43.2: the assembly                           | <code>sound_QUntilBattery_firstHit</code>                                                                                              |
| Proposition R43.3: formula ties the concluded affine atom | <code>sound_QUntilBattery</code><br>region rows $[(1, V_a), (c, V_d)]$ , threshold $1 - p$ ; rate-tie hypothesis $hc : \beta^T \leq c$ |

## Rule 44. QSTREETTBATTERY (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — QStreettBattery. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\begin{array}{c}
 V_0, (W_i)_{i \leq k}, (V_i)_{i \in A} : \text{QCertFam}; \quad A \subseteq \{1, \dots, k\} \text{ (the pairs posting absorbing certificates); } \quad \varepsilon_0 \in \mathbb{Q}_{>0}; \quad (\gamma_i)_{i \in A} \in \mathbb{Q}_{>0} \\
 \text{the QSAFETYBATTERY rows for } V_0 \text{ at } U := \text{cmpl}(I) \text{ (interior, guard, floor on } U \cap \bar{D}, \text{ nonneg on } \bar{D}) \\
 \{S \setminus \llbracket \rho_i \rrbracket \vdash \mathcal{A}_m V_{i,m} \leq 0\}_{i \in A} \text{ (absorbing drift: off-request, on all of } S) \\
 \{S \vdash 0 \leq V_i \leq C_i^a\}_{i \in A} \text{ (absorbing bounds, on all of } S), \quad \{\llbracket \rho_i \rrbracket \vdash V_i \geq 1\}_{i \in A} \text{ (absorbing floors)} \\
 \{(\llbracket \varphi_i \rrbracket \cup J_i)^c \vdash \mathcal{A}_m W_{i,m} \leq -\varepsilon_0\}_{i \leq k} \text{ (ranking decay), } \quad J_i := \{V_i \leq 1 - \gamma_i\} \cap (I \setminus \llbracket \rho_i \rrbracket) \text{ for } i \in A; \quad J_i := \emptyset \text{ for } i \notin A \\
 W_i \equiv 0 \text{ on target cells (containment); } \quad \{\bar{D} \vdash W_i \geq 0\}_{i \leq k} \text{ (ranking nonnegs)} \\
 \{\bar{D} \vdash W_i \leq C_i\}_{i \leq k}, \quad C_i \in \mathbb{Q} \quad p + y \leq 1 \text{ (threshold tie)} \\
 \hline
 \mathcal{M}, x_0 \in \{V_0 \leq y\} \vdash \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\rho_1, \varphi_1), \dots, \text{PairBase}(\rho_k, \varphi_k)))
 \end{array}$$

a region written left of the turnstile scopes the row ( $R \vdash P$ :  $P$  at every point of  $R$ ); the certificate families enter as certificate data — shared references with no claim content; the rows impose every claim.  $J_i$  is *defined, not supplied*: the mixed soundness theorem defines it from the posted absorbing data, with the  $J_i := \emptyset$  case (absorbing family omitted,  $i \notin A$ ) the conservative scope — the baked node carries no absorbing-family fields yet, so this optional face is not yet wire-representable (a named schema divergence).  $V_0$ 's own nonnegativity rides the QSAFETYBATTERY kit row, stated once there. The margin premise  $\gamma_i > 0$  together with the absorbing floor forces  $J_i \cap \llbracket \rho_i \rrbracket = \emptyset$ ; no disjointness row is asked. The target-cell containment  $W_i \equiv 0$  is recomputed by the checker. The threshold tie  $p + y \leq 1$  is the soundness theorem's own hypothesis, displayed as a side condition: the baked Streett node carries no tie proof field yet (contrast the deadline and until nodes) — a named schema divergence, not a checked leaf. The conclusion is the region-conditioned judgment in constructor vocabulary, as at QSAFETYBATTERY; the base-exact request consumption is the PairBase constructor.

QSTREETTBATTERY concludes a probabilistic Streett-acceptance judgment: with probability at least  $p$ , a path of the modelled system satisfies every pair of the Streett condition  $\text{PDwellStreett}(\text{PairBase}(\rho_1, \varphi_1), \dots, \text{PairBase}(\rho_k, \varphi_k))$  (surface syntax  $\text{Streett}((\rho_1, \varphi_1), \dots, (\rho_k, \varphi_k))$ ) — for each pair  $i$ , if *request* states (those satisfying  $\rho_i$ ) occur infinitely often along the path, then *response* states (those satisfying  $\varphi_i$ ) also occur infinitely often. This is the shape of liveness-under-fairness obligations, and it subsumes recurrence and stability claims as degenerate pairs (the sub-rule sections that follow this one). A producer discharges such a specification by synthesizing the certificate family  $(V_0; (V_i, \gamma_i, W_i)_{i \leq k}; \varepsilon_0)$ : one safety certificate  $V_0$  confining probability mass to the operating region, and, per pair, a ranking certificate  $W_i$  that pays out a fixed rational  $\varepsilon_0 > 0$  at every un-responded request — plus, optionally, absorbing certificates  $(V_i, \gamma_i)$  licensing the sharper ranking scope discussed in the proof. Every premise row is an inherited base-calculus sequent (a Bernstein-cone-family polynomial inequality the base kernel replays), so the bundle adds *zero* kernel arithmetic: its entire content is the supermartingale assembly one judgment level up, which is where its soundness lemma lives.

What the checker recomputes rather than trusts: the target-cell containment behind the  $W_i \equiv 0$  convention (the collapse side-condition of the QSTREETTTARGETCELLCOLLAPSE section), and — when the absorbing face becomes wire-representable — the carved-out set  $J_i := \{V_i \leq 1 - \gamma_i\} \cap (I \setminus \llbracket \rho_i \rrbracket)$ ,  $I$  the display's confinement region  $\{V_0 < 1\}$ :  $J_i$  is *defined*, never supplied by the producer, but today it is defined by the soundness theorem from posted theorem data — the baked node carries no absorbing-family fields, so no enacted recomputation exists yet. Also recomputed: the region-hull mass side-conditions that make the certificate scopes fail-closed. The threshold tie  $p + y \leq 1$  is displayed as a side condition, not a checked leaf: it is the soundness theorem's own hypothesis, and the baked Streett node carries no tie proof field yet (contrast the deadline and until nodes) — a named schema divergence, as the rule's flavor text states. What would break without the hypotheses: the nonnegativity rows for all three certificate families are load-bearing (a signed certificate can subsidize probability-mass leakage — the recorded two-branch counterexample at the quantitative-safety rule); the strictness  $\varepsilon_0 > 0$  is load-bearing ( $W \equiv 1$  satisfies every row at  $\varepsilon_0 = 0$  while no response is ever forced); the floor and hull rows pair up fail-closed. Each is remarked, with its counterexample, in the proof.

Scope, stated plainly: the statements bound and receipt-credited in this edition are the five excursion-cluster theorems quoted below — the analytic engine of the typeset proof — together with the end-to-end assembly `sound_QStreettBattery` (the conservative-scope bundle theorem over `Kernel.traj`) and its discharged carved-scope corollary `sound_QStreettBattery_carved_discharged` (absorbing certificates supplied as kernel-level rows, composed through the absorption discharge), both in the auxiliary module `BernsteinLogicAux.QStreett`. These cover the rule at the *discrete-kernel profile* (a time-homogeneous Markov chain presented by a single step kernel); the complement-request tie `sound_QStreettBattery_tie` is likewise bound in this edition, under the `QSTREETTCMPLREQUEST` entry that carries its event-level kernel; the parameterized `sound_QStreettBattery_carved` and the absorption discharge `absorptionBound_of_kernel` remain machine-checked and cited, not bound. The continuous-time transport that used to sit beyond that — the residue this rule shared with its safety and deadline siblings, and this entry's former conditional mark — is discharged and bound: at the martingale-problem profile the whole bundle is proven, confinement and

zero-decrement ranking rows (`sound_GSHS_streett_bundle_mp`) together with the strict-decrement rows from strict presentations (`sound_GSHS_streett_bundle_mp_strict`, whose ranking-row input is produced by `rankingSegmentDynkin_of_mp_strict` from the pathwise progress row of `MPFlowStrict`), leaving no analytic per-model hypothesis in the bundle. All four are hash-bound below via the registered `qtsl-aux` package, so the dashboard row carries no conditional mark: each named profile is proven, and its theorems are bound.

notes Two design notes travel with the rule. First, schema: the bundle rule's rows are baked in the extension theory (per-pair rank certificates and nonneg proof lists); the source catalogue's old MISSING-ROW finding is discharged. Second, why Streett is the *primitive* acceptance condition here: a Büchi condition is exactly a one-pair Streett condition with the trivially-true request, and the degenerate-pair specialization erases the resulting tautological rows — so a Streett-primitive compile pays no premium on Büchi-shaped workloads, while a Büchi-primitive compile would pay an exponential-in- $k$  automaton re-encoding on Streett-shaped ones. The sub-rules of the following sections (degenerate request, complement request, target-cell collapse) are the compile-level faces of this design.

## Lean statement

BernsteinLogicAux.Excursions.**Supermartingale.exists\_ae\_tendsto\_of\_nonneg**

statement bound

```
/-- **Almost-everywhere convergence of nonnegative supermartingales** (K5a): a
nonnegative supermartingale is automatically bounded in  $L^1$  (its expectation is
nonincreasing), so Doob's convergence theorem applies with no further hypothesis. Dual
of MeasureTheory.Submartingale.exists_ae_tendsto_of_bdd` via Supermartingale.neg`. -/
theorem Supermartingale.exists_ae_tendsto_of_nonneg [IsFiniteMeasure  $\mu$ ] {f :  $\mathbb{N} \rightarrow \Omega \rightarrow \mathbb{R}$ }
  (hf : Supermartingale f  $\mathcal{G}$   $\mu$ ) (hnonneg :  $0 \leq f$ ) :
   $\forall^\mu \omega \partial \mu, \exists c, \text{Tendsto} (\text{fun } n \Rightarrow f \ n \ \omega) \text{ atTop } (\mathcal{N} \ c)$ 
```

BernsteinLogicAux.Excursions.**ranking\_hitting\_sum\_le**

statement bound

```
/-- **Ranking hitting-time bound** (K5b, layer form):  $\epsilon_0 \cdot \mathbb{E}[\tau_B] \leq \mathbb{E}[V \ 0]$ , the
expected hitting time decomposed by the layer-cake formula as
 $\mathbb{E}[\tau_B] = \sum' k, \mu \{\tau_B > k\}$  over the strict-survival events
 $\{\tau_B > k\} = \{\forall j < k + 1, X \ j \notin B\}$ . This is the certificate-facing form: the ranking
rows certify hdrift, the nonneg rows certify hnonneg, and the conclusion is a bound
on the expected number of steps before reaching B. -/
theorem ranking_hitting_sum_le (h $\epsilon$  :  $0 \leq \epsilon_0$ )
  (hint :  $\forall n, \text{Integrable} (V \ n) \ \mu$ ) (hnonneg :  $0 \leq V$ )
  (hmB :  $\forall n, \text{MeasurableSet}[\mathcal{G} \ n] \ \{\omega \mid X \ n \ \omega \in B\}$ )
  (hdrift :  $\forall n, \forall^\mu \omega \partial \mu, X \ n \ \omega \notin B \rightarrow (\mu[V \ (n + 1) \mid \mathcal{G} \ n]) \ \omega \leq V \ n \ \omega - \epsilon_0$ ) :
   $\text{ENNReal.ofReal } \epsilon_0 * \sum' k, \mu \{\omega \mid \forall j < k + 1, X \ j \ \omega \notin B\}$ 
   $\leq \text{ENNReal.ofReal} (\int \omega, V \ 0 \ \omega \partial \mu)$ 
```

BernsteinLogicAux.Excursions.**ae\_hits\_of\_ranking**

statement bound

```
/-- **Almost-sure hitting from a ranking certificate** (K5b, qualitative form): a
strictly positive ranking drift off B forces the state process to reach B almost
surely – the summability of the survival probabilities from ranking_hitting_sum_le`
sends them to zero. -/
theorem ae_hits_of_ranking (h $\epsilon$  :  $0 < \epsilon_0$ )
  (hint :  $\forall n, \text{Integrable} (V \ n) \ \mu$ ) (hnonneg :  $0 \leq V$ )
  (hmB :  $\forall n, \text{MeasurableSet}[\mathcal{G} \ n] \ \{\omega \mid X \ n \ \omega \in B\}$ )
  (hdrift :  $\forall n, \forall^\mu \omega \partial \mu, X \ n \ \omega \notin B \rightarrow (\mu[V \ (n + 1) \mid \mathcal{G} \ n]) \ \omega \leq V \ n \ \omega - \epsilon_0$ ) :
   $\forall^\mu \omega \partial \mu, \exists n, X \ n \ \omega \in B$ 
```

BernsteinLogicAux.Excursions.**streett\_window\_sum\_le**

statement bound

```
/-- **Summability of un-responded requests** (K5c, quantitative core): for each window
start N,  $\epsilon_0 \cdot \sum_{j \geq N} \mu(\text{no response in } [N, N+j] \wedge \text{request at } N+j) \leq \mathbb{E}[V \ N]$  – the form
the Borel–Cantelli lemma consumes. -/
theorem streett_window_sum_le (h $\epsilon$  :  $0 \leq \epsilon_0$ )
  (hint :  $\forall n, \text{Integrable} (V \ n) \ \mu$ ) (hnonneg :  $0 \leq V$ )
  (hmR :  $\forall n, \text{MeasurableSet}[\mathcal{G} \ n] \ \{\omega \mid X \ n \ \omega \in R\}$ )
  (hmG :  $\forall n, \text{MeasurableSet}[\mathcal{G} \ n] \ \{\omega \mid X \ n \ \omega \in G\}$ )
  (hdrift :  $\forall n, \forall^\mu \omega \partial \mu, X \ n \ \omega \notin G \rightarrow (\mu[V \ (n + 1) \mid \mathcal{G} \ n]) \ \omega \leq V \ n \ \omega$ )
  (hdriftR :  $\forall n, \forall^\mu \omega \partial \mu,$ 
     $X \ n \ \omega \in R \rightarrow X \ n \ \omega \notin G \rightarrow (\mu[V \ (n + 1) \mid \mathcal{G} \ n]) \ \omega \leq V \ n \ \omega - \epsilon_0$ )
  (N :  $\mathbb{N}$ ) :
   $\text{ENNReal.ofReal } \epsilon_0 * \sum' j : \mathbb{N},$ 
     $\mu (\{\omega \mid \forall k, N \leq k \rightarrow k < N + j + 1 \rightarrow X \ k \ \omega \notin G\} \cap \{\omega \mid X \ (N + j) \ \omega \in R\})$ 
   $\leq \text{ENNReal.ofReal} (\int \omega, V \ N \ \omega \partial \mu)$ 
```

BernsteinLogicAux.Excursions.streett\_ae\_frequently\_response

statement bound

```

/-- **Streett excursion assembly** (K5c): if `V` is nonincreasing in conditional
expectation off the response set `G` and decreases by  $\varepsilon_0 > 0$  on requests (`R`, off
`G`), then almost surely infinitely many requests imply infinitely many responses –
the probabilistic content of a Streett pair `(R, G)`. Proof: by
`streett_window_sum_le` the un-responded-request events of each window are summable, so
their limsup is null (`MeasureTheory.measure_limsup_atTop_eq_zero`, Borel–Cantelli);
a path with infinitely many requests and no response after `N` lies in window `N`'s
limsup, and the union over `N` is null. -/
theorem streett_ae_frequently_response (hε : 0 < ε₀)
  (hint : ∀ n, Integrable (V n) μ) (hnonneg : 0 ≤ V)
  (hmR : ∀ n, MeasurableSet[ℳ n] {ω | X n ω ∈ R})
  (hmG : ∀ n, MeasurableSet[ℳ n] {ω | X n ω ∈ G})
  (hdrift : ∀ n, ∀m ω ∂μ, X n ω ∉ G → (μ[V (n + 1) | ℳ n]) ω ≤ V n ω)
  (hdriftR : ∀ n, ∀m ω ∂μ,
    X n ω ∈ R → X n ω ∉ G → (μ[V (n + 1) | ℳ n]) ω ≤ V n ω - ε₀) :
  ∀m ω ∂μ, (∃f n in atTop, X n ω ∈ R) → ∃f n in atTop, X n ω ∈ G

```

BernsteinLogicAux.QStreett.sound\_QStreettBattery

statement bound

```

/-- **`sound_QStreettBattery`, discrete profile** (catalogue T9, rev 3
`thm:bundle-sound` transported; the assembly the receipt ledger carried as
FILED). Hypotheses: the `V₀` battery rows – quantitative safety of the
confinement region `D \ U` (the catalogue's `I := {V₀ < 1}` is the instance
`U := {y | 1 ≤ V₀ y}`, whose floor row is tautological) – and, per pair
`i : ι` (countable; a Streett bundle's finite `k` is an instance), the ranking
rows for `Wᵢ` at the **conservative scope** of the design note §4.1's ranking
row ("conservative on `I \ [φᵢ]`"): nonincrease off the response region,
`εᵢ`-decrease at requests, on the confinement region. Conclusion:
`Pr[(∏ᵢ (Fin(Rqᵢ) ∪ Inf(Gᵢ))) ≥ p]` from the start region `D n {V₀ ≤ thr}` with
`p + thr ≤ 1` – the compile's atom `{V₀ ≤ 1 - p}` is `thr = 1 - p`.

```

The absorbing certificates `(Vᵢ, γᵢ)` of the full bundle datum do not appear: at the conservative ranking scope they are not consumed (they license only the `Jᵢ` carve-out – see `sound\_QStreettBattery\_carved`). Presentation–semantics bridge and GSHS residue: module docstring. -/

```

theorem sound_QStreettBattery {Rq G : ι → Set S} {W : ι → S → ℝ} {ε C : ι → ℝ}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV₀ : Measurable V₀)
  (hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
  (hW : ∀ i, Measurable (W i))
  (hθ : p + thr ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ U → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ U → K y (D ∪ U) = 1)
  (hV₀int : ∀ y ∈ D, y ∉ U → Integrable V₀ (K y))
  (hV₀drift : ∀ y ∈ D, y ∉ U → ∫ z, V₀ z ∂K y ≤ V₀ y)
  (hV₀floor : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V₀ y)
  (hV₀nonneg : ∀ y ∈ Dbar, 0 ≤ V₀ y)
  (hε : ∀ i, 0 < εᵢ)
  (hWθ : ∀ i, ∀ y ∈ Dbar, 0 ≤ W i y) (hWC : ∀ i, ∀ y ∈ Dbar, W i y ≤ C i)
  (hWint : ∀ i, ∀ y ∈ D, y ∉ U → y ∉ G i → Integrable (W i) (K y))
  (hWdrift : ∀ i, ∀ y ∈ D, y ∉ U → y ∉ G i → ∫ z, W i z ∂K y ≤ W i y)
  (hWdriftR : ∀ i, ∀ y ∈ D, y ∉ U → y ∈ Rq i → y ∉ G i →
    ∫ z, W i z ∂K y ≤ W i y - εᵢ)
  (hx : x ∈ D) (hthr : V₀ x ≤ thr) :
  ENNReal.ofReal p ≤ pathMeasure K x
  {ω | ∀ i, (∃f n in atTop, ω n ∈ Rq i) → ∃f n in atTop, ω n ∈ G i}

```

BernsteinLogicAux.QStreett.sound\_QStreettBattery\_carved\_discharged

statement bound

```

/-- **`sound_QStreettBattery_carved`, discharged form** – the unconditional
corollary: the full bundle datum with the absorbing certificates `(Vᵢ, γᵢ)`
supplied as kernel-level rows (global scope, per the section docstring)
instead of the named hypothesis. The parameterized

```



```

`sound_QStreettBattery_carved` remains the citable assembly; this composes it
with `absorptionBound_of_kernel` per pair. -/
theorem sound_QStreettBattery_carved_discharged {D Dbar U : Set S} {V0 : S → ℝ}
  {p thr : ℝ} {x : S} {ι : Type*} [Countable ι] {Rq G J : ι → Set S}
  {W Va : ι → S → ℝ} {ε C γ Ca : ι → ℝ}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV0 : Measurable V0)
  (hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
  (hJ : ∀ i, MeasurableSet (J i)) (hW : ∀ i, Measurable (W i))
  (h0 : p + thr ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ U → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ U → K y (D ∪ U) = 1)
  (hV0int : ∀ y ∈ D, y ∉ U → Integrable V0 (K y))
  (hV0drift : ∀ y ∈ D, y ∉ U → ∫ z, V0 z ∂K y ≤ V0 y)
  (hV0floor : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V0 y)
  (hV0nonneg : ∀ y ∈ Dbar, 0 ≤ V0 y)
  (hε : ∀ i, 0 < ε i) (hγ : ∀ i, 0 < γ i)
  (hW0 : ∀ i, ∀ y ∈ Dbar, 0 ≤ W i y) (hWC : ∀ i, ∀ y ∈ Dbar, W i y ≤ C i)
  (hWint : ∀ i, ∀ y ∈ D, y ∉ U → y ∉ G i → y ∉ J i → Integrable (W i) (K y))
  (hWdrift : ∀ i, ∀ y ∈ D, y ∉ U → y ∉ G i → y ∉ J i →
    ∫ z, W i z ∂K y ≤ W i y)
  (hWdriftR : ∀ i, ∀ y ∈ D, y ∉ U → y ∈ Rq i → y ∉ G i → y ∉ J i →
    ∫ z, W i z ∂K y ≤ W i y - ε i)
  (hVa : ∀ i, Measurable (Va i))
  (hVa0 : ∀ i y, 0 ≤ Va i y) (hVaC : ∀ i y, Va i y ≤ Ca i)
  (hVaFloor : ∀ i, ∀ y ∈ Rq i, 1 ≤ Va i y)
  (hVadrift : ∀ i y, y ∉ Rq i → ∫ z, Va i z ∂K y ≤ Va i y)
  (hVaJ : ∀ i, ∀ y ∈ J i, Va i y ≤ 1 - γ i)
  (hx : x ∈ D) (hthr : V0 x ≤ thr) :
  ENNReal.ofReal p ≤ pathMeasure K x
  {ω | ∀ i, (∃f n in atTop, ω n ∈ Rq i) → ∃f n in atTop, ω n ∈ G i}

```

**The general-stochastic instance, assembled.** The bundle on the segment model's plain skeleton chain, obtained by instantiating the discrete bundle above at the skeleton kernel: alive-only region transport, scoped per-pair ranking fields with additive decrement (subtraction-free extended-nonnegative form), and the three fail-closed containment rows enter as per-model fields; the discrete engine's localization and windowed telescoping do the rest, restart-free. Plain and confined forms.

BernsteinLogicAux.LayerD.Streett. **sound\_GSHS\_streett\_bundle**

statement bound

```

/-- **`sound_QStreettBattery`, GSHS instance** (T9, conservative ranking
scope) – the discrete-statement-shaped corollary: the chain-level Streett
event alone. The `_confined` form is the semantically loaded one; this one
matches `QStreett.sound_QStreettBattery` line for line. -/
theorem sound_GSHS_streett_bundle
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV0 : Measurable V0)
  (hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
  (hW : ∀ i, Measurable (W i))
  (h0 : p + thr ≤ 1) (hDsub : D ⊆ Dbar)
  (hLive : NoVanish M (D \ U)) (hHalt : HaltsCaught M (D \ U) (U ∩ Dbar))
  (hReset : ResetsInto M (D \ U) ((D ∪ U) ∩ Dbar))
  (hSegD0 : RankingSegmentDynkin M (D \ U) V0 0)
  (hForce0 : ForcedNonexpansion M V0) (hNonneg0 : NonnegV V0)
  (hC00 : 0 ≤ C0) (hV0C : ∀ y, V0 y ≤ C0)
  (hFloor0 : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V0 y)
  (hε : ∀ i, 0 < ε i)
  (hSegDW : ∀ i, RankingSegmentDynkin M ((D \ U) \ G i) (W i) 0)
  (hSegDWR : ∀ i, RankingSegmentDynkin M (((D \ U) ∩ Rq i) \ G i) (W i) (ε i))
  (hForceW : ∀ i, ForcedNonexpansion M (W i)) (hNonnegW : ∀ i, NonnegV (W i))
  (hC0 : ∀ i, 0 ≤ C i) (hWC : ∀ i, ∀ y, W i y ≤ C i)
  (hx : x ∈ D) (hthr : V0 x ≤ thr) :
  ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
  {ω | ∀ i, (∃f n in atTop, ω n ∈ skelIn (Rq i)) →
    ∃f n in atTop, ω n ∈ skelIn (G i)}

```



BernsteinLogicAux.LayerD.Streett.sound\_GSHS\_streett\_bundle\_confined

statement bound

```

/-- **`sound_QStreettBattery`, GSHS instance, confined form** (T9; assault
doc §2.7 Theorem G at the conservative ranking scope, mechanized
restart-free – module docstring for why G1/G2's epoch machinery is not
consumed). Hypothesis fields are segment-level and per-model: the three
containment rows, the  $\theta = \theta$  confinement field for `V0`, and per pair the
scoped ranking fields for `W i` (nonincrease off the response region,
`ε i`-decrease at request entries – per-step shape, module docstring for
the rate-form conversion). Conclusion: with probability at least `p` the
skeleton chain of `Rep(P)` stays alive in `D \ U` forever **and** every
certified pair satisfies the chain-level Streett condition. Semantic
transport of the chain events: module docstring (T9b monotonicity for
requests, entry reads for responses, W-Z for time-cofinality). -/
theorem sound_GSHS_streett_bundle_confined
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV0 : Measurable V0)
  (hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
  (hW : ∀ i, Measurable (W i))
  (hθ : p + thr ≤ 1) (hDsub : D ⊆ Dbar)
  (hLive : NoVanish M (D \ U)) (hHalt : HaltsCaught M (D \ U) (U n Dbar))
  (hReset : ResetsInto M (D \ U) ((D ∪ U) n Dbar))
  (hSegD0 : RankingSegmentDynkin M (D \ U) V0 θ)
  (hForce0 : ForcedNonexpansion M V0) (hNonneg0 : NonnegV V0)
  (hC0θ : θ ≤ C0) (hV0C : ∀ y, V0 y ≤ C0)
  (hFloor0 : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V0 y)
  (hε : ∀ i, θ < ε i)
  (hSegDW : ∀ i, RankingSegmentDynkin M ((D \ U) \ G i) (W i) θ)
  (hSegDWR : ∀ i, RankingSegmentDynkin M (((D \ U) n Rq i) \ G i) (W i) (ε i))
  (hForceW : ∀ i, ForcedNonexpansion M (W i)) (hNonnegW : ∀ i, NonnegV (W i))
  (hC0 : ∀ i, θ ≤ C i) (hWC : ∀ i, ∀ y, W i y ≤ C i)
  (hx : x ∈ D) (hthr : V0 x ≤ thr) :
  ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
  {ω | (∀ n, ω n ∈ skelIn D \ skelBad U) ∧
    ∀ i, (∃f n in atTop, ω n ∈ skelIn (Rq i)) →
    ∃f n in atTop, ω n ∈ skelIn (G i))}

```

**The martingale-problem profile, end to end.** The general continuous-time theorems: the inter-jump motion enters through its generator —  $\mathcal{AV}$  as data, tied to the process only by the martingale hypothesis for the discounted compensated certificate process. The mixed-profile bundle discharges the confinement half and every zero-decrement ranking row; `MPFlowStrict` adds the one pathwise progress row (accumulated generator drift reaching  $-\varepsilon$  by a finite sample time), from which the strict ranking rows are derived at exactly the requested-but-ungranted states — and the strict bundle theorem then carries no analytic per-model hypothesis at all.

QtslAux.LayerD.MPDDischarge.sound\_GSHS\_streett\_bundle\_mp

statement bound

```

/-- The Streett/recurrence bundle at the martingale-problem profile,
mixed ranking scope: the confinement certificate `V0` and every pair's
ranking certificate `W i` are presented by martingale problems, which
discharges all the `ε = 0` ranking rows. The strict-decrement rows
`hSegDWR` (`ε i > 0`) stay hypotheses *of this theorem*, for models that
discharge them by other means; `sound_GSHS_streett_bundle_mp_strict` below
discharges them too, from strict presentations (`MPFlowStrict`) on the
requested-but-ungranted region. -/
theorem sound_GSHS_streett_bundle_mp
  {ι : Type*} [Countable ι]
  {D Dbar U : Set State} {V0 : State → ℝ} {Rq G : ι → Set State}
  {W : ι → State → ℝ} {p thr C0 : ℝ} {ε C : ι → ℝ} {x : State}
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV0 : Measurable V0)
  (hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
  (hW : ∀ i, Measurable (W i))
  (hθ : p + thr ≤ 1) (hDsub : D ⊆ Dbar)
  (hLive : NoVanish M (D \ U)) (hHalt : HaltsCaught M (D \ U) (U n Dbar))

```

```

(hReset : ResetsInto M (D \ U) ((D ∪ U) n Dbar))
(hmp0 : ∀ x : State, Nonempty (MPFlow M 0 V0 x))
(hForce0 : ForcedNonexpansion M V0) (hNonneg0 : NonnegV V0)
(hC0 : 0 ≤ C0) (hV0C : ∀ y, V0 y ≤ C0)
(hFloor0 : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V0 y)
(hε : ∀ i, 0 < ε i)
(hmpW : ∀ i, ∀ x : State, Nonempty (MPFlow M 0 (W i) x))
(hSegDWR : ∀ i, RankingSegmentDynkin M (((D \ U) n Rq i) \ G i) (W i) (ε i))
(hForceW : ∀ i, ForcedNonexpansion M (W i)) (hNonnegW : ∀ i, NonnegV (W i))
(hC0 : ∀ i, 0 ≤ C i) (hWC : ∀ i, ∀ y, W i y ≤ C i)
(hx : x ∈ D) (hthr : V0 x ≤ thr) :
ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
{ω | ∀ i, (∃f n in atTop, ω n ∈ skelIn (Rq i)) →
  ∃f n in atTop, ω n ∈ skelIn (G i)}

```

QtslAux.LayerD.MPDischarge . **MPFlowStrict**

statement bound

```

/-- **The strict martingale-problem presentation of a segment.** An
`MPFlow` together with the pathwise progress row: along every path, the
accumulated generator drift of the discounted certificate reaches `≤ -ε`
by some finite sample time. For exit paths the natural witness is the exit
time itself (`stopClip` caps the accumulation there), so the row reads
`∫0∞ T ω mpDrift ≤ -ε`; the intended certificate-side derivation is a
strict drift bound `≤ -δ` on the segment together with a minimum dwell
time `t_min`, giving `ε = δ · t_min`. -/
structure MPFlowStrict (M : SegmentModel State) (θ : ℝ) (V : State → ℝ)
  (x : State) (ε : ℝ) extends MPFlow M 0 V x where
/-- the pathwise progress row -/
hprog : ∀ ω, ∃ t : ℝ, 0 ≤ t ∧
  ∫ s in (0:ℝ)..(stopClip (T ω) (k ω) t),
    mpDrift 0 V γ r jr AV s ω ≤ -ε

```

QtslAux.LayerD.MPDischarge . **rankingSegmentDynkin\_of\_mp\_strict**

statement bound

```

/-- **The strict-decrement ranking rows at the martingale-problem
profile.** Entry states of the ranking region carrying strict
presentations at level `ε` satisfy `RankingSegmentDynkin M Sc V ε` – the
row family that `rankingSegmentDynkin_of_mp` could not produce. Note the
quantifier: only states in `Sc` need presentations (the `ε = 0` route
quantified over all states because it factored through `SegmentDynkin`). -/
theorem rankingSegmentDynkin_of_mp_strict (M : SegmentModel State)
  (V : State → ℝ) [IsSFiniteKernel M.jump] (hV : Measurable V)
  {Sc : Set State} {ε : ℝ} (hε : 0 ≤ ε)
  (h : ∀ x ∈ Sc, Nonempty (MPFlowStrict M 0 V x ε)) :
  RankingSegmentDynkin M Sc V ε

```

QtslAux.LayerD.MPDischarge . **sound\_GSHS\_streett\_bundle\_mp\_strict**

statement bound

```

/-- The Streett/recurrence bundle at the **fully martingale-problem
profile**: the confinement certificate `V0` and every pair's ranking
certificate `W i` are presented by martingale problems, with the
requested-but-ungranted region additionally carrying *strict*
presentations (`MPFlowStrict` at level `ε i`). This discharges the
`hSegDWR` hypothesis of `sound_GSHS_streett_bundle_mp` – at this profile
no analytic per-model hypothesis remains. -/
theorem sound_GSHS_streett_bundle_mp_strict
  {ι : Type*} [Countable ι]
  {D Dbar U : Set State} {V0 : State → ℝ} {Rq G : ι → Set State}
  {W : ι → State → ℝ} {p thr C0 : ℝ} {ε C : ι → ℝ} {x : State}
  [IsMarkovKernel M(seg)] [IsMarkovKernel M.jump]
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV0 : Measurable V0)
  (hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
  (hW : ∀ i, Measurable (W i))
  (h0 : p + thr ≤ 1) (hDsub : D ⊆ Dbar)
  (hLive : NoVanish M (D \ U)) (hHalt : HaltsCaught M (D \ U) (U n Dbar))

```

```

(hReset : ResetsInto M (D \ U) ((D ∪ U) n Dbar))
(hmp0 : ∀ x : State, Nonempty (MPFlow M 0 V0 x))
(hForce0 : ForcedNonexpansion M V0) (hNonneg0 : NonnegV V0)
(hC0 : 0 ≤ C0) (hV0C : ∀ y, V0 y ≤ C0)
(hFloor0 : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V0 y)
(hε : ∀ i, 0 < ε i)
(hmpW : ∀ i, ∀ x : State, Nonempty (MPFlow M 0 (W i) x))
(hmpWR : ∀ i, ∀ x ∈ ((D \ U) n Rq i) \ G i,
  Nonempty (MPFlowStrict M 0 (W i) x (ε i)))
(hForceW : ∀ i, ForcedNonexpansion M (W i)) (hNonnegW : ∀ i, NonnegV (W i))
(hC0 : ∀ i, 0 ≤ C i) (hWC : ∀ i, ∀ y, W i y ≤ C i)
(hx : x ∈ D) (hthr : V0 x ≤ thr) :
ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
{ω | ∀ i, (∃f n in atTop, ω n ∈ skelIn (Rq i)) →
  ∃f n in atTop, ω n ∈ skelIn (G i)}

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed judgment is concluded by `sound_QStreettBattery` (second block): from the  $V_0$  battery rows on the confinement region and, per pair, the conservative-scope ranking rows (nonincrease off the response region,  $\varepsilon_i$ -decay at requests), it concludes  $\Pr[\bigcap_i (\text{Fin}(Rq_i) \cup \text{Inf}(G_i))] \geq p$  from the start region at  $p + \text{thr} \leq 1$  — the display's conclusion at the discrete-kernel profile over `kernel.traj`; `_carved_discharged` adds the carved scope with the absorbing certificates consumed as kernel-level rows. (Notation:  $\text{Fin}(A) / \text{Inf}(A)$  denote the path events that a trajectory visits  $A$  finitely / infinitely often — the acceptance atoms of the Hanoi Omega-Automata format [11] and of Emerson-Lei conditions [68]; equivalently  $\text{Inf}(A) = \{\rho \mid \text{Inf}(\rho) \cap A \neq \emptyset\}$  in the infinity-set convention,  $\text{Inf}(\rho)$  being the set of states visited infinitely often. The upright  $\text{Fin}$  is distinct from the sans-serif type former  $\text{Fin}$  of the mechanization.) The five bound excursion theorems are the K5 analytic engine the assembly consumes, stated over an arbitrary filtered probability space  $(\Omega, \mathcal{G}, \mu)$  with an adapted state process  $X$  — *wider* than the discrete-kernel chain. What the display's rows reach in the engine's bound set, per pair:

- ranking interior rows  $\mathcal{A}_m W_{i,m} \leq -\varepsilon_0$  off  $\llbracket \varphi_i \rrbracket \cup J_i \mapsto$  the conditional-shadow pair in `streett_ae_frequently_response`: `hdrift` (conditional non-increase off the response set  $G$ ) and `hdriftR` (decay by  $\varepsilon_0$  at requests off  $G$ ) — per-time a.e. conditional-expectation clauses, the probabilistic sibling of the integrated semantic-shadow forms on the deterministic pages; the `BernCone` rows reach them through the kernel bridge in the unbound assembly. Conclusion:  $\forall^\mu \omega, (\exists^\mu n, X n \in \omega \in R) \rightarrow \exists^\mu n, X n \in \omega \in G$  — the probabilistic content of one Streett pair, exactly.
- `nonneg` rows  $\mapsto$  `hnonneg : 0 ≤ V` — and the roster's two-branch signed-certificate counterexample is why the binder is there.
- $\varepsilon_0 \in \mathbb{Q}_{>0} \mapsto$  `hε : 0 < ε0` (real, the  $\mathbb{Q}$  cast upstream); the quantitative window/hitting forms (`streett_window_sum_le`, `ranking_hitting_sum_le`) take the nonstrict  $0 \leq \varepsilon_0$  — strictness is consumed exactly where Borel-Cantelli or summability-to-hitting needs it (`ae_hits_of_ranking`, the frequently-response theorem).
- the measurability pack `hmR/hmG` (request/response events adapted at each time) is undisplayed regularity — the probabilistic sibling of the `PsdCert` evaluability class, as on the battery pages.
- `Supermartingale.exists_ae_tendsto_of_nonneg (K5a)` is engine stock: Doob convergence with the  $L^1$  bound derived from nonnegativity, no display row corresponds.

Width note: the  $V_0$  safety kit, the  $W_i \leq C_i$  upper-bound rows, the threshold tie  $p + y \leq 1$ , and the carved  $J_i$  with the absorbing families  $(V_i, \gamma_i)$  as kernel-level rows are all consumed by the two bound assembly statements; the  $G_5$  target-cell containment binds in the `QSTREETTTARGETCELLCOLLAPSE` entry, and the complement-request tie remains cited, not bound. The martingale-problem profile, stated at its true extent: the confinement half and all zero-decrement ranking rows are discharged by `sound_GSHS_streett_bundle_mp`, and the strict-decrement rows ( $\varepsilon_i > 0$ ) by strict presentations — `MPFlowStrict` extends `MPFlow` by one pathwise progress row (along every path the accumulated generator drift of the discounted certificate reaches  $-\varepsilon$  by a finite sample time; the certificate-side derivation is  $\varepsilon = \delta \cdot t_{\min}$  from a strict segment drift bound and a minimum dwell time), consumed by `rankingSegmentDynkin_of_mp_strict`, which needs presentations only at requested-but-ungranted states — so `sound_GSHS_streett_bundle_mp_strict` leaves no analytic per-model hypothesis in the bundle; all four hash-bound below. The honest scope note that replaces the retired residue sentence: the strict route's progress row is *pathwise* (per-path accumulation, not in-expectation), the per-state bound  $\varepsilon \leq V(x)$  is derived by the theorem rather than assumed as a row, and the strict theorem is a companion to the mixed-profile `_mp` form, not an in-place replacement — the landed  $\varepsilon = 0$  signature survives for citation stability. Width note, deliberate strengthening stated: the displayed ranking row asks strict  $-\varepsilon_0$  decay everywhere off  $\llbracket \varphi_i \rrbracket \cup J_i$ , where

the bound statements need nonincrease off the response and strict decrease only at requests — the displayed single-row form is stronger than consumed, safe direction, and the producer emits it as one family.

## Lean verifier receipts

module BernsteinLogicAux.Excursions; bound build ad62a846f92e... (hashed from the built olean at bind time). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 495e3739c82a865d | 2026-07-18T13:23:35Z | build+decorrelated                 | current |
| fd91ed9c49139bb3 | 2026-07-19T18:23:53Z | build+decorrelated                 | current |
| 40087fade64fc483 | 2026-07-20T01:21:05Z | build+decorrelated                 | current |
| 483b08e5a60a003d | 2026-07-20T20:46:29Z | build+decorrelated                 | current |
| 7142f9529d463930 | 2026-07-20T20:58:02Z | build+replay+hardened+decorrelated | current |
| a477dcd09a788edc | 2026-07-21T00:02:42Z | build+decorrelated                 | current |
| 30b972191d8805c2 | 2026-07-21T05:38:35Z | build+replay+hardened+decorrelated | current |
| c6df511b240a81db | 2026-07-22T18:26:52Z | build+decorrelated                 | current |
| 7d4ae5654cfa6cc8 | 2026-07-31T21:59:44Z | build+decorrelated                 | current |
| 50671021df35b6cc | 2026-07-31T22:28:38Z | build+decorrelated                 | current |
| 153a2e7124bfd8d7 | 2026-08-02T23:30:45Z | build+decorrelated                 | current |
| 069303d77fac0945 | 2026-08-03T16:26:49Z | build+decorrelated                 | current |
| b0dd60fee0daa2eb | 2026-08-03T22:45:33Z | build+decorrelated                 | current |

module BernsteinLogicAux.QStreutt; bound build 469881beb770... (bound from receipt 9d1d848e0ba1718a, 2026-07-29T01:03:31Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| e3990fe544d4708d | 2026-07-28T23:48:35Z | build+decorrelated                 | current |
| 68e9374fa1b418fe | 2026-07-29T00:00:34Z | build+decorrelated                 | current |
| e5895c9706854cd3 | 2026-07-29T00:49:04Z | build+replay+hardened+decorrelated | current |
| 9d1d848e0ba1718a | 2026-07-29T01:03:31Z | build+replay+hardened+decorrelated | current |
| 4a8d377b721b8523 | 2026-07-31T22:00:09Z | build+decorrelated                 | current |
| d251a122540ab741 | 2026-07-31T22:29:03Z | build+decorrelated                 | current |
| 333635b8e047113a | 2026-08-02T23:31:14Z | build+decorrelated                 | current |
| 9e96f80d63e4ad82 | 2026-08-03T16:27:18Z | build+decorrelated                 | current |
| 932b4b9e0b0fd323 | 2026-08-03T22:46:02Z | build+decorrelated                 | current |

9 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.LayerD.Streutt; bound build 7393065bbc7f... (bound from receipt abac471866cfcfd1, 2026-07-21T05:38:48Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt           | minted               | layers                             | status  |
|-------------------|----------------------|------------------------------------|---------|
| 0fb3b50b4d3f0d33c | 2026-07-20T22:52:50Z | build+decorrelated                 | current |
| 109669f848453ec2  | 2026-07-21T00:03:19Z | build+decorrelated                 | current |
| 7e17aa32b276623b  | 2026-07-21T04:51:49Z | build+decorrelated                 | current |
| abac471866cfcfd1  | 2026-07-21T05:38:48Z | build+replay+hardened+decorrelated | current |

6 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module QtslAux.LayerD.MPDIscharge; bound build bade74dd943... (bound from receipt 7228834dea10e5f9, 2026-07-24T11:04:04Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 7228834dea10e5f9 | 2026-07-24T11:04:04Z | build+decorrelated | current |

**gate-layer coverage of the bound build:**build (lake build (elaboration)) currentreplay (leanchecker (official kernel replay)) currenthardened (safe\_verify (hardened screen)) currentdecorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

**HOL4 statement****NO HOL STATEMENT YET**

Absent, with the gap located precisely: the HOL4 development ports the *base-calculus* checker cone (the `bern_catalogue` theory, built side-by-side with the Lean catalogue module) — its object is the wire-format certificate checker of the sequent stratum. The extension theory's satisfaction stratum has no HOL artefact at all, so there is no HOL-side wire object a checker could consume today. A HOL statement becomes meaningful once the extension stratum is ported; it would then join the proof-object layer the HOL port's own coordination notes name as successor territory. (Checked against the HOL sources: no Streett-, satisfaction-, or NNF-shaped definitions exist there.)

**HOL verifier receipts****NONE**

no HOL verification receipt exists for this rule.

**Proof**

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail, at the rule's *proven scope*: discrete-kernel profile, conservative ranking scope ([Remark R44.7](#) states the scope loudly, including what is and is not hash-bound in this edition). Every hypothesis is used and remarked on; the anchor table maps steps to the Lean declarations that carry them.

**Setting and claim.** The semantic object at the discrete-kernel profile is a time-homogeneous Markov chain on a measurable state space  $S$ , presented by a single step kernel  $K$  (modes, guards, and resets of the underlying hybrid presentation flattened into  $K$ ), with path measure  $\mu_x$  on  $S^{\mathbb{N}}$  from a start state  $x$  (the Ionescu–Tulcea trajectory construction). Region data: the operating region  $D$ , its checker-recomputed hull  $\bar{D} \supseteq D$ , and the exit set  $U$  for the confinement certificate; per Streett pair  $i$  (finitely or countably many), a certified request region  $R_i$  and response region  $G_i$ . Certificate data: the confinement certificate  $V_0 : S \rightarrow \mathbb{R}$  with threshold  $y$ , and per pair a ranking certificate  $W_i : S \rightarrow \mathbb{R}$  with decay rate  $\varepsilon_i \in \mathbb{Q}_{>0}$  and bound  $C_i$ . The premise rows, each an inherited base-calculus sequent:

- (H1) threshold arithmetic:  $p + y \leq 1$  (exact rational).
- (H2) hull and mass rows, for every  $z \in D \setminus U$ :  $K(z)(\bar{D}) = 1$  and  $K(z)(D \cup U) = 1$  — one-step mass stays in the hull, and stays in operating-or-exit.
- (H3) confinement rows: one-step generator row  $\int V_0 dK(z) \leq V_0(z)$  on  $D \setminus U$  (with one-step integrability); floor  $V_0 \geq 1$  on  $U \cap \bar{D}$ ; nonnegativity  $V_0 \geq 0$  on  $\bar{D}$ .
- (H4) ranking rows, per pair, on the confinement region  $D \setminus U$ : bounds  $0 \leq W_i \leq C_i$  on  $\bar{D}$ ; nonincrease  $\int W_i dK(z) \leq W_i(z)$  off the response region  $G_i$ ; request decay  $\int W_i dK(z) \leq W_i(z) - \varepsilon_i$  on  $R_i \setminus G_i$ .
- (H5) initialization:  $x \in D$  and  $V_0(x) \leq y$  — i.e.  $x$  lies in the concluded region atom `qraSublevel`( $V_0, y$ ).

The claim: under (H1)–(H5),

$$\mu_x \left\{ \omega \mid \forall i, (\exists^\infty n, \omega_n \in R_i) \Rightarrow (\exists^\infty n, \omega_n \in G_i) \right\} \geq p,$$

which is the Streett-acceptance reading of the concluded judgment  $\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\rho_1, \varphi_1), \dots, \text{PairBase}(\rho_n, \varphi_n)))$  once the certified regions are tied to the formulas ([Remark R44.8](#)).

**Lemma R44.1** (confinement).  $\mu_x \{ \omega \mid \forall n, \omega_n \in D \setminus U \} \geq 1 - V_0(x) \geq 1 - y \geq p$ .

*Proof.* This is the quantitative-safety rule consumed whole: by (H2)–(H3),  $V_0$  read along the path and stopped at the first  $U$ -visit is a nonnegative supermartingale with barrier  $\geq 1$  on the stopped set, so Ville’s maximal inequality bounds the exit probability by  $V_0(x)$ ; the mass rows (H2) keep every reachable state inside  $\bar{D}$ , where the nonnegativity and floor rows actually hold — the fail-closed pairing. The full argument, with the two-branch counterexample showing the nonnegativity row is not decoration, is the QSAFETYBATTERY section’s; the displayed intermediate bound is the chain inside that argument — Proposition R40.4 applied to the clamped certificate of Lemma R40.5 bounds the exit probability by  $g(x) = V_0(x)$  — consumed here through the same end-to-end statement (anchor table), whose packaged form Theorem R40.6 then states as the  $\geq p$  complement. The last two inequalities are (H5) and (H1).  $\square$

**Lemma R44.2** (windowed telescoping invariant). *Fix one pair  $(R, G)$  with certificate  $W$ , rate  $\varepsilon$ , and suppose the generator rows hold globally:  $\mathbb{E}[W_{n+1} \mid \mathcal{F}_n] \leq W_n$  a.e. off  $G$ , and  $\leq W_n - \varepsilon$  a.e. on requests off  $G$ , for the process  $W_n := W(X_n)$  adapted to a filtration  $(\mathcal{F}_n)$ , with  $W_n \geq 0$  and integrable. For a window start  $N$  let  $A_{N,n} := \{\text{no response in } [N, n)\}$ . Then for all  $n \geq N$ ,*

$$\int_{A_{N,n}} W_n d\mu + \varepsilon \sum_{k=N}^{n-1} \mu(A_{N,k+1} \cap \{X_k \in R\}) \leq \int W_N d\mu.$$

*Proof.* Induction on  $n$  from  $n = N$ , where  $A_{N,N}$  is everything and the sum is empty. For the step:  $A_{N,n+1}$  is  $\mathcal{F}_n$ -measurable (it constrains coordinates  $< n+1$ ), so the tower property converts  $\int_{A_{N,n+1}} W_{n+1}$  into  $\int_{A_{N,n+1}} \mathbb{E}[W_{n+1} \mid \mathcal{F}_n]$ . Split  $A_{N,n+1}$  at time  $n$  into its request and non-request parts: on the request part the decay row pays  $\varepsilon \cdot \mu(A_{N,n+1} \cap \{X_n \in R\})$  — exactly the new summand — and on the non-request part the nonincrease row holds (both parts lie off  $G$  at time  $n$ , since  $A_{N,n+1}$  forbids a response at  $n$ ). Finally  $A_{N,n+1} \subseteq A_{N,n}$  and  $W_n \geq 0$  shrink the integral domain monotonically, closing the induction. Two features deserve remark. First, *no sign condition on  $\varepsilon$  is used here*: the decrement paid at step  $k$  is exactly the summand recorded at index  $k$ , an exact cancellation — positivity of  $\varepsilon$  enters only in Lemma R44.3. Second, the two generator rows do genuinely different work and neither subsumes the other: nonincrease carries the (possibly long) non-request stretches; decay is charged only at requests.  $\square$

**Lemma R44.3** (summability and Borel–Cantelli). *Under the hypotheses of Lemma R44.2 with  $\varepsilon > 0$  and  $W$  bounded: almost surely, infinitely many requests imply infinitely many responses.*

*Proof.* Fix a window start  $N$ . Lemma R44.2 bounds every partial sum of the series below by  $\mathbb{E}[W_N]$  uniformly (the leading integral it retains is nonnegative), and a nonnegative series is the supremum of its partial sums, so

$$\varepsilon \sum_{j \geq 0} \mu(\{\text{no response in } [N, N+j]\} \cap \{X_{N+j} \in R\}) \leq \mathbb{E}[W_N] < \infty,$$

so the un-responded-request events of window  $N$  are summable, and by the Borel–Cantelli lemma their limsup is null. A path with infinitely many requests but only finitely many responses has a finite (possibly empty) set of response times; for any window start  $N$  beyond all of them ( $N = 0$  when there are none), the path lies in infinitely many of window  $N$ ’s un-responded-request events — i.e. in that null limsup. The union over the countably many window starts is null, which is the claim. (Here  $\varepsilon > 0$  is finally consumed, dividing the bound; boundedness of  $W$  supplies the finite right-hand side and, upstream, integrability of every  $W_n$ .)  $\square$

**Lemma R44.4** (localization to the confinement region). *The bundle’s ranking rows (H4) hold only on  $D \setminus U$ , while Lemma R44.2 wants global rows. Define the localized pair process: flag the state with the survival indicator of the event “no visit to  $U \cup D^c$  so far”, zero the certificate at the first exit, and count exited states as responded. Then the flagged process satisfies the global hypotheses of Lemmas R44.2–R44.3, and the conclusion, restricted to paths confined forever, descends to the raw process: almost surely on  $\{\forall n, \omega_n \in D \setminus U\}$ , infinitely many  $R_i$ -visits imply infinitely many  $G_i$ -visits.*

*Proof.* Write it out. Let  $\sigma := \inf\{n : \omega_n \in U \cup D^c\}$  (a stopping time — the regions are measurable), and localize by

$$W'_n := W(\omega_n) \mathbf{1}[\sigma > n], \quad G' \text{ holds at time } n : \iff \omega_n \in G \text{ or } \sigma \leq n,$$

the zeroed certificate and the enlarged (exited-counts-as-responded) response. Check the global hypotheses of Lemma R44.2 for  $(W', G')$  by cases on the state class at time  $n$ :

(*surviving, non-request*)  $\sigma > n$  and  $\omega_n \notin R$ : off  $G'$  means also  $\omega_n \notin G$ , so  $\omega_n \in (D \setminus U) \setminus G$ ; the mass row (H2) carries the next step into  $\bar{D}$ , where  $0 \leq W \leq C$  (H4), so  $W'_{n+1} = W(\omega_{n+1}) \mathbf{1}[\sigma > n+1] \leq W(\omega_{n+1})$  almost surely on this branch — the masked certificate is dominated by the unmasked one — and  $\mathbb{E}[W'_{n+1} \mid \mathcal{F}_n] \leq \int W dK(\omega_n) \leq W(\omega_n) = W'_n$  by the nonincrease row.

(*surviving, request*) the same domination, closed by the decay row:  $\mathbb{E}[W'_{n+1} \mid \mathcal{F}_n] \leq W'_n - \varepsilon$ .

(*exited*)  $\sigma \leq n$ : the state is in the enlarged response  $G'$  by construction, so no generator row is demanded of it — the global hypotheses hold vacuously, and indeed  $W'_n = 0$  there.

Bounds and adaptedness:  $0 \leq W'_n \leq C$  everywhere (H4 on  $\bar{D}$  while surviving — reachable surviving states lie in  $\bar{D}$  by (H2) and the deterministic start — zero after), and  $W'$  is adapted since  $\sigma$  is a stopping time. (To consume Lemma R44.2 literally in its certificate reading, run the chain on the flagged state space  $S \times \{0, 1\}$ : the flag is a running conjunction updated by membership of each new coordinate in  $U \cup D^c$ , so the flagged chain is again time-homogeneous Markov,  $W'$  and the enlarged response are state functions there —  $W'(z, f) = f \cdot W(z)$ , response  $\iff f = 0$  or  $z \in G$  — and the flagged trajectory law projects to  $\mu_x$ .) Lemmas R44.2–R44.3 then give: almost surely, infinitely many requests-while-surviving imply infinitely many  $G'$ -responses. Restricted to confined paths the flag never trips ( $\sigma = \infty$ ),  $W' = W$ , the enlarged response set coincides with  $G_i$ , and the conclusion is the raw one. The exited-counts-as-responded device is not a weakening of the claim: the theorem below asserts the pair implications only on the confinement event of Lemma R44.1, exactly where the device is invisible.  $\square$



**Theorem R44.5** (soundness of QSTRETTBATTERY, discrete profile, conservative ranking scope). *Under (H1)–(H5),  $\mu_x\{\omega \mid \forall i, (\exists^\infty n, \omega_n \in R_i) \Rightarrow (\exists^\infty n, \omega_n \in G_i)\} \geq p$ .*

*Proof.* Per pair  $i$ , Lemmas R44.2–R44.4 give the pair implication almost surely on the confinement event. The pairs are countably many, so the almost-sure statements intersect (countable intersection of co-null sets); the Streett event therefore contains the confinement event up to a null set, and Lemma R44.1 bounds its measure below by  $p$ .  $\square$

*Remark R44.6* (every hypothesis earns its seat). (H1): at  $p + y > 1$  the concluded atom admits start states with  $1 - V_0(x) < p$  and Lemma R44.1 yields less than  $p$  — the sublevel atom and the probability bound are coupled by exactly this row. (H2): drop the hull row and probability mass can reach  $U \setminus \bar{D}$ , where no floor row holds, voiding Ville’s barrier in Lemma R44.1; drop the operating-or-exit row and the stopped-process reading of the confinement certificate is unsound (mass escapes the region where a generator row was ever asserted) — the two recomputed side-conditions are the fail-closed pairing. (H3) *nonnegativity*: load-bearing, not hygiene — a signed confinement certificate can go negative on one branch and thereby subsidize exit mass on another; the recorded two-branch counterexample lives in the QSAFETYBATTERY section. (H4) *strictness*: at  $\varepsilon_i = 0$ , take  $W_i \equiv 1$  on a one-state chain with  $K(s) = \delta_s$ ,  $R_i = \{s\}$ ,  $G_i = \emptyset$ ,  $V_0 \equiv 0$ : every row holds, the path requests forever, and no response ever occurs — the conclusion fails at any  $p > 0$ . (H4) *bound rows*: the constant  $C_i$  gives integrability of the localized certificate (Lemma R44.4) and the finite right-hand side of Lemma R44.3; nonnegativity of  $W_i$  additionally powers the monotone domain-shrink step inside Lemma R44.2. (H5): initialization is what makes the region atom an *under*-approximation of the satisfaction set — the workhorse orientation of the satisfaction judgment.

*Remark R44.7* (scope, loudly). Three ledger entries: the discrete-profile warrant, the cited testimony, and the continuous-time profile. (i) *Bound in this edition*: the five excursion-cluster declarations (anchor table) — the windowed telescoping engine (Lemmas R44.2–R44.3), its additive-drift sibling for plain reachability (the ranking hitting-time bound and almost-sure hitting), and the nonnegative-supermartingale convergence theorem (part of the cluster, available for the general excursion picture; the presented proof path does not consume it) — together with the end-to-end assembly at the conservative scope (sound\_QStreettBattery — Lemmas R44.1, R44.4 and Theorem R44.5 as one statement) and the discharged carved-scope corollary (sound\_QStreettBattery\_carved\_discharged), both in the auxiliary module BernsteinLogicAux.QStreett. These warrant the rule at the discrete-kernel profile. Also bound in this edition, under the QSTRETTCMPLREQUEST entry whose section carries its event-level kernel: the formula tie (sound\_QStreettBattery\_tie, with streett\_event\_mono). (ii) *Machine-checked and cited, not bound*: the localization lemma under its own name (streett\_pair\_localized), and the parameterized route to the sharper carved ranking scope, where decay is demanded only off  $G_i \cup J_i$  with  $J_i := \{V_i \leq 1 - \gamma_i\} \cap (I \setminus \llbracket \rho_i \rrbracket)$  —  $I$  is the display’s confinement region  $\{V_0 < 1\}$ ; this section’s exit set instances it as  $U = \{1 \leq V_0\}$  — and the absorbing certificates  $(V_i, \gamma_i)$  repay the carve-out: a Lévy upward-convergence argument shows frequent  $J_i$ -visits preclude frequent requests, the absorption bound being discharged from kernel-level rows by a time-shifted supermartingale bridge plus a conditional Ville inequality (sound\_QStreettBattery\_carved, absorptionBound\_of\_kernel). Binding those in a later edition is the named follow-up; until then this page claims them as testimony, not as hash-bound warrant. (iii) *The martingale-problem profile, discharged whole*: the confinement half and every zero-decrement ranking row ( $\varepsilon_i = 0$ : nonincrease off the response region) are discharged by sound\_GSHS\_streett\_bundle\_mp, and the strict-decrement rows ( $\varepsilon_i > 0$ ) by strict presentations — MPFlowStrict extends the martingale-problem presentation by one pathwise progress row (along every path the accumulated generator drift of the discounted certificate reaches  $-\varepsilon$  by a finite sample time; the intended certificate-side derivation is  $\varepsilon = \delta \cdot t_{\min}$  from a strict segment drift bound and a minimum dwell time), and rankingSegmentDynkin\_of\_mp\_strict derives from it the ranking row family the  $\varepsilon = 0$  route could not produce, needing presentations only at requested-but-ungranted states. The proof mechanism, for the record, is *not* the lower-bound optional-sampling inequality an earlier edition of this remark predicted: the stopped certificate splits as martingale plus drift accumulation, the existing upper-bound optional sampling serves the martingale part, and dominated convergence on  $\varepsilon$ -truncated drift accumulations supplies the strict term (the lower-bound lemma was built anyway, completing the optional-sampling module). The assembled sound\_GSHS\_streett\_bundle\_mp\_strict therefore carries no analytic per-model hypothesis — all four statements live in module QtsLAux.LayerD.MPDIscharge and are hash-bound in this edition via the registered qtsl-aux package, so, like its safety and deadline siblings, this rule carries no conditional mark: each named profile is proven, and its theorems are bound.

*Remark R44.8* (the formula tie). The theorem is stated over certified regions  $R_i, G_i$ ; the concluded judgment speaks of formulas  $\rho_i, \varphi_i$ . The sound directions are asymmetric: certified *request* regions may *over*-approximate the request denotation, certified *response* regions must *under*-approximate the response denotation — weakening the hypothesis and strengthening the conclusion of each pair implication. This is the content of the complement-request sub-rule (QSTRETTCMPLREQUEST, two sections on), whose event-level kernel is two monotonicity steps.

*Remark R44.9* (degenerate pairs, and why Streett is primitive). A Büchi acceptance condition is the one-pair Streett condition with trivially-true request; on that pair the bundle degenerates to the classical recurrence certificate (the QSTRETTDEGENERATE section). The converse embedding does not exist at parity of cost: deterministic Büchi automata recognize only limit languages [150], and re-encoding a  $k$ -pair Streett condition Büchi-primitively costs an exponential-in- $k$  automaton product, where the bundle pays  $2k + 1$  certificates. Streett conditions enter the literature as the complemented-pairs acceptance of [226], the linear-time face of Rabin’s pairs [205]; the standard survey is [232].

*Remark R44.10* (relation to the literature). The ranking row is a Foster–Lyapunov drift condition [81, 180] in certificate form; its use to force almost-sure hitting is the ranking-supermartingale argument of the probabilistic-programs tradition [41]. The confinement certificate is Ville’s inequality [234]; the convergence item of the bound cluster is Doob’s supermartingale convergence theorem [64]; the summability-to-limsup step is Borel–Cantelli (see e.g. [65, Thm. 2.3.1]). As with the polynomial rules of the base calculus, the certificate form moves all analytic content into the *choice* of  $(V_0, W_i, \varepsilon_0)$ : what the checker replays is row



arithmetic, and what this proof adds is only the assembly. No completeness is claimed or needed here; the source catalogue records a completeness theorem for the bundle relative to its termination seam, outside this book’s kernel-facing scope.

| proof step                                    | formal anchor                                                                                                                                                                                                        |
|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lemma R44.1                                   | sound_QSafetyBattery_failClosed (bound under the QSafetyBattery entry; consumed whole)                                                                                                                               |
| Lemma R44.2 + no-sign-condition remark        | streett_window_invariant (the engine’s inner induction; subsumed by the bound streett_window_sum_le)                                                                                                                 |
| Lemma R44.3, summability form                 | streett_window_sum_le (bound)                                                                                                                                                                                        |
| Lemma R44.3, Borel–Cantelli assembly          | streett_ae_frequently_response (bound)                                                                                                                                                                               |
| additive-drift sibling (reachability)         | ranking_hitting_sum_le, ae_hits_of_ranking (bound)                                                                                                                                                                   |
| supermartingale convergence (cluster item)    | Supermartingale.exists_ae_tendsto_of_nonneg (bound; not consumed by the presented path)                                                                                                                              |
| Lemma R44.4                                   | streett_pair_localized (BernsteinLogicAux.QStreett; machine-checked, binding pending)                                                                                                                                |
| Theorem R44.5                                 | sound_QStreettBattery (same module, <b>bound in this edition</b> )                                                                                                                                                   |
| Remark R44.8                                  | streett_event_mono, sound_QStreettBattery_tie (same module; <b>bound in this edition</b> under the QStreettCmplRequest entry)                                                                                        |
| carved scope, discharged corollary            | sound_QStreettBattery_carved_discharged (same module, <b>bound in this edition</b> )                                                                                                                                 |
| Remark R44.7(ii), carved scope route          | AbsorptionBound, ae_not_frequently_of_absorption, supermartingale_stopped_shift, measure_reach_after_le, absorptionBound_of_kernel, sound_QStreettBattery_carved (same module, binding pending)                      |
| Remark R44.7(iii), martingale–problem profile | sound_GSHS_streett_bundle_mp, MPFlowStrict, rankingSegmentDynkin_of_mp_strict, sound_GSHS_streett_bundle_mp_strict (QtslAux.LayerD.MPDDischarge; <b>bound in this edition</b> via the qtsl-aux package registration) |

## Rule 45. QAVGBATTERY (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — *QAvgBattery*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{\text{drift}_m : R_m \vdash g - r_m - \mathcal{A}_m B_m \geq 0 \quad \overline{\text{guard}_j} \quad \overline{\text{floor}_m} \quad \overline{\text{nonneg}_{g_m}} \quad \text{tie} : \vdash g \bowtie c}{\mathcal{M}, x_0 \in R \models \text{SReturn} \bowtie (c, \text{poAvg}(o))}$$

**QAVGBATTERY** concludes long-run-average claims from the certificate pair of the Poisson equation: a rational *gain*  $g$  and a *bias* certificate family  $B$ , with the drift rows carrying the constant term inside each row's sequent polynomial — so the discharge species is the same sign-sequent genre as every other battery, with no new checker architecture. On the upper face the drifts certify gain minus reward minus generator-of-bias nonnegative on scope, and the tie bounds the gain above the claimed constant; on the lower face the orientations flip, and the floors bound the bias above so the correction term dies in the limit. The walk recomputes the required orientation from the concluded root — a producer never declares a face.

**notes** Entered at the roster census: the node was baked — including its tie proof field, which the safety and acceptance batteries still lack — but had never been catalogued. No Lean statement binds it yet; the display is the schema-faithful schematic form, and the soundness obligation (the Poisson-pair argument: telescoping the bias along the trajectory, dividing by the horizon, the floors killing the correction term) is stated in the proof section as an obligation, not discharged.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: *chase* = the engine derives; *report* = a violation is a loud judgment; *refuse* = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

theory `Qtsl.QAvgBattery`

quoted from the theory source at render

| `QAvgBattery` : Set;

theory `Qtsl.QAvgBattery/*`

quoted from the theory source at render

```
QAvgBattery/proof : QAvgBattery -> QProof;
QAvgBattery/gain : QAvgBattery -> Q;
QAvgBattery/arg : QAvgBattery -> StatePred;
QAvgBattery/cert : QAvgBattery -> QCertFam;
QAvgBattery/drifts : QAvgBattery -> QProofList;
QAvgBattery/guards : QAvgBattery -> QProofList;
QAvgBattery/lowbounds : QAvgBattery -> QProofList;
QAvgBattery/nonnegs : QAvgBattery -> QProofList;
QAvgBattery/tie : QAvgBattery -> Proof;
QAvgBattery/scope_warrant : QAvgBattery -> QInvScope;
```

### Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

### HOL4 statement

#### NO HOL STATEMENT YET

No HOL development of the extension theory exists.

### HOL verifier receipts

#### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), battery group. This section states the rule's soundness obligation exactly, per the catalogue's completeness-over-polish charter (a rule that can only be named-and-gapped is still a row); the discharge is open debt on the mechanization column, and the schema quoted in the entry is the baked ground truth.

**Soundness obligation.** The Poisson-equation pair argument: with drift rows certifying  $g - r_m - \mathcal{A}_m B_m \geq 0$  on scope per mode (upper face), guards at the seams, floors and nonnegativity on the bias family, telescoping the compensated bias along any admitted trajectory bounds the accumulated reward's time average by the gain plus a correction term of order bias-over-horizon; the floors kill the correction in the limit, and the tie transfers the gain bound to the claimed constant. The lower face is the same argument with orientations flipped. The obligation is stated over the same behavioural models as the safety battery's, with the walk recomputing the required orientation from the concluded root.

*Remark* R45.1 (status). No Lean statement binds this node. The discharge species is the landed sign-sequent genre — the expected mechanization reuses the battery kit's supermartingale machinery with the compensated bias in the certificate slot — and the node's tie field is already baked, so no schema motion is expected at mechanization time.

## Rule 46. QRATEFROMCLOCK (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QRateFromClock*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{\text{TrMeter}(t, d) \quad \rho \cdot d \geq 1 \quad \sigma \geq 1 \quad \text{qtsl\_rate\_clock}(n)}{\mathcal{M} \models \text{SRateLE}(\{t\}, \rho, \sigma)}$$

QRATEFROMCLOCK discharges the rate-envelope specification former from clock data alone: a transition whose clock carries a dwell floor  $d$  (demonic, forced, or carved — never the instantaneous race clock, which has no floor) satisfies  $\text{SRateLE}(\{t\}, \rho, \sigma)$  for any  $\rho \cdot d \geq 1$  and  $\sigma \geq 1$ . The rate lane's other feed is the supermartingale kit — a different evidence family, deliberately not conflated; this rule is the clock seam.

The premise is a premise *class*, not a node:  $\text{TrMeter}(t, d)$  names three things together — the transition's clock-content conformance (exactly-one content presentation, wing positivity; already checker-discharged rows), a POSITIVE lower witness  $0 < d \leq d'$  below the declared floor  $d'$  (positivity is the class's own formation: a nonpositive witness would inhabit the class while its division law is vacuous), and the meter law as a *named obligation* — the conformance rows alone do not prove elapsed-time separation. The law: under the declared meter semantics (dwell accrues per transition; guard exit and own firing reset it), successive firings of  $t$  are separated by at least  $d$  of elapsed time, hence  $N_t(T) \leq T/d + 1$  on any window under every resolution. The checker's trusted contribution is two exact rational inequalities ( $\rho \cdot d \geq 1$ ;  $\sigma \geq 1$ ) over conformance rows it already checks — with  $d$  resolved deterministically as the transition's declared finite floor (no producer freedom; the class is monotone in  $d$ , so the declared floor is the tightest lawful witness).

The concluded envelope is an upper-prevision claim; the obligation actually proved is pathwise and surely — strictly stronger — and the entry records that strength so a future pathwise rate former can strengthen the conclusion without touching the node. Singleton rosters compose by the rate lane's documented disjoint-roster sum law (envelopes add on disjoint rosters); this rule is that law's first in-catalogue consumer.

**notes** The schema is baked in the extension theory: the node, its walk, the race-refusal, and the per-wing meter-demand rows are quoted below from the theory source; the per-transition conformance rows the premise class reads were checker-discharged before the node landed. What remains owed is the meter law's first formalization (the shared keystone of this rule family's continuous-time obligations — the discrete-profile induction is elementary). No Lean statement binds the node yet; the mechanization column is honest open debt.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The baked rows: the node with its two exact-rational envelope reports, the race-refusal (the instantaneous wing carries no floor and is refused as subject), the walk, and the per-wing meter-demand rows that mint the premise class's conformance-plus-floor reading at the declared dwell floor.

theory Qtsl . QRATEFROMCLOCK

quoted from the theory source at render

| QRATEFROMCLOCK : Set;

theory Qtsl . QRATEFROMCLOCK/\*

quoted from the theory source at render

| QRATEFROMCLOCK/proof : QRATEFROMCLOCK -> QPROOF;  
| QRATEFROMCLOCK/tr : QRATEFROMCLOCK -> TRANSITION;  
| QRATEFROMCLOCK/rho : QRATEFROMCLOCK -> Q;  
| QRATEFROMCLOCK/sigma : QRATEFROMCLOCK -> Q;

theory Qtsl . ax/wf/qrfc1\_rho\_pos

quoted from the theory source at render

| report ax/wf/qrfc1\_rho\_pos : forall n : QRATEFROMCLOCK, x : Q.  
| n QRATEFROMCLOCK/rho = x  
| |- {l: 0, r: x} q\_lt;

theory Qtsl.ax/wf/qrfc1\_sigma\_ge\_one

quoted from the theory source at render

```
report ax/wf/qrfc1_sigma_ge_one : forall n : QRateFromClock, x : Q.
  n QRateFromClock/sigma = x
|- {l: 1, r: x} q_le;
```

theory Qtsl.ax/qtsl/rate\_clock\_race

quoted from the theory source at render

```
refuse ax/qtsl/rate_clock_race : forall n : QRateFromClock,
  t : Transition, e : EnnRat, k : TrClock.
  n QRateFromClock/tr = t,
  t Transition/clock = k,
  k = {majorant: e} tkRace
|- false;
```

theory Qtsl.qtsl\_rate\_clock

quoted from the theory source at render

```
qtsl_rate_clock : {node: QRateFromClock} -> Prop;
```

theory Qtsl.ax/tf/qrfc1\_walk

quoted from the theory source at render

```
report ax/tf/qrfc1_walk : forall n : QRateFromClock.
|- {node: n} qtsl_rate_clock;
```

theory Qtsl.ax/tf/qrfc1\_meter\_forced

quoted from the theory source at render

```
report ax/tf/qrfc1_meter_forced : forall n : QRateFromClock,
  t : Transition, d : Q, k : TrClock.
  n QRateFromClock/tr = t,
  t Transition/clock = k,
  k = {dwell: d} tkForced
|- {tr: t, d: d} qtsl_tr_meter;
```

theory Qtsl.ax/tf/qrfc1\_meter\_demonic

quoted from the theory source at render

```
report ax/tf/qrfc1_meter_demonic : forall n : QRateFromClock,
  t : Transition, d : Q, e : EnnRat, k : TrClock.
  n QRateFromClock/tr = t,
  t Transition/clock = k,
  k = {dwellMin: e} tkDemonic,
  e = {q: d} enFin
|- {tr: t, d: d} qtsl_tr_meter;
```

theory Qtsl.ax/tf/qrfc1\_meter\_carved

quoted from the theory source at render

```
report ax/tf/qrfc1_meter_carved : forall n : QRateFromClock,
  t : Transition, fl : Q, ce : EnnRat, k : TrClock.
  n QRateFromClock/tr = t,
  t Transition/clock = k,
  k = {floor: fl, ceil: ce} tkCarved
|- {tr: t, d: fl} qtsl_tr_meter;
```

theory Qtsl.qtsl\_tr\_meter

quoted from the theory source at render

```
qtsl_tr_meter : {tr: Transition, d: Q} -> Prop;
```

theory Qtsl.qtsl\_tr\_content

quoted from the theory source at render

```
qtsl_tr_content : {tr: Transition} -> Prop;           // side-condition
```

## Lean statement

not yet mechanized

this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

## HOL4 statement

### NO HOL STATEMENT YET

No HOL development of the extension theory exists; this node's content is walk structure over baked rows.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), the timers family (the rate-envelope former's clock seam). This section states the rule's soundness obligation exactly; the rule's schema and walk are baked in the extension theory (rows quoted in this section), and the meter law's formalization is the named obligation of the mechanization column.

**Soundness obligation (meter law + pathwise domination).** Under the declared meter semantics (dwell accrues per transition; guard exit and own firing reset it), a transition  $t$  whose clock carries floor  $d > 0$  fires with successive firings separated by at least  $d$  of elapsed time, hence  $N_t(T) \leq T/d + 1$  on any window of length  $T$ , under every resolution. The witness  $d$  is the transition's declared finite floor, resolved deterministically by the walk (the premise class demands a POSITIVE witness at most the declared floor, and is monotone, so the declared floor is the tightest lawful instance; no producer freedom enters). With the checker-verified exact inequalities  $\rho \cdot d \geq 1$  and  $\sigma \geq 1$ ,  $N_t(T) \leq T/d + 1 \leq \rho T + \sigma$  pathwise and surely; the upper prevision of the roster firing count therefore satisfies the envelope — pathwise domination, strictly stronger than the concluded prevision-level claim. The instantaneous race clock is refused as subject: it carries no floor, and its rate story is a different evidence family (the thinning majorant), deliberately not conflated with this seam.

*Remark R46.1 (status).* The schema is baked: the node, its walk, the race-refusal, and the per-wing meter-demand rows are landed theory rows (quoted in this section), and the premise class's per-transition conformance rows were checker-discharged before the node landed. No Lean statement binds the node yet; the meter LAW is the named obligation — elementary at the discrete profile, and the shared keystone of this rule family's continuous-time lifts.

## Rule 47. QINVRELATIVIZE (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QInvRelativize*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{BoxInv}(B, \nu, x)}{\text{AService}(B, T, \chi \cap \nu, x) \leftrightarrow \text{AService}(B, T, \chi, x)}$$

$B$  an *arbitrary* behaviour — no closure field consumed;  $\text{AService}$  is the début-as-infimum reading unpacked without infima (a visit at some  $t < T + \varepsilon$  for every  $\varepsilon > 0$ ),  $\text{BoxInv}$  the  $\square$  reading (every point of every solution from  $x$ ).

certificate content out of scope of the current baked theory — realized in Lean over the behavioural floor.

One of the three timed-core basis schemata proved non-admissible in the gñand completeness development (machine independence witnesses; see the proof section’s anchors), entering as tier-1 core: its soundness obligation is semantic, discharged at Putnam-exam-solution detail in the proof section and mechanized in `BernsteinLogic.QtslTimedCore`. Sound over *all* behavioural models — no side condition; the proof consumes none of the closure laws.

### Lean statement

`BernsteinLogic.QtslTimedCore.sound_QInvRelativize`

statement bound

```
/-- **QInvRelativize** (obligation relativization through invariants), the
biconditional core: under  $\square \nu$ , the visit sets of  $\chi \cap \nu$  and  $\chi$ 
coincide along every solution from  $x$ , so the two service claims
agree. Sound for ARBITRARY behaviors — none of B1–B3 is consumed.
Catalogue: tier-1 core; independence machine-checked — a 3-state
deadline-blind structure validates the remaining basis and every rule
while refuting this schema, so it is underivable. -/
theorem sound_QInvRelativize (B : Behavior X) (T : ℝ≥0) (ν χ : Set X) (x : X)
  (hbox : B.BoxInv ν x) :
    B.AService T (χ ∩ ν) x ↔ B.AService T χ x
```

### Correspondence (Steele display ↔ Lean)

#### NO CORRESPONDENCE DICTIONARY YET

no reviewer-certified dictionary maps the displayed rule onto the mechanized statements above; until one is written and certified, that mapping rests on this section's editorial care alone.

### Lean verifier receipts

module `BernsteinLogic.QtslTimedCore`; bound build `e6370ca0a2ac...` (bound from receipt `6460a6dd4d227c9a`, 2026-08-04T01:16:03Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 6460a6dd4d227c9a | 2026-08-04T01:16:03Z | build+decorrelated | current |
| 659fe3396a48b674 | 2026-08-04T02:23:12Z | build+decorrelated | current |

4 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only



hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_t5qdyn, HOL4 development script bern\_tr\_t5qdynScript.sml script bound

### Theorem sound\_QInvRelativize

```
Theorem sound_QInvRelativize:
  ∀fwd bud nu chi (x:'x).
    qdBoxInv fwd nu x ⇒
      (qdAService fwd bud (chi INTER nu) x ⇔ qdAService fwd bud chi x)
```

## HOL verifier receipts

documentary hol-strict CLEAN: theory bern\_tr\_t5qdyn, 22 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob d64e1954...); an independent acceptance host ran the same build green at the porting seating (2026-07-21). The timed core this statement quantifies over is ported with an inventoried seam: trajectories ride the full real line with  $0 \leq t$  guards at exactly the positions where the nonnegative-time binders sit, and the behaviour structure's closure proof fields flatten to an explicit wellformedness conjunction. This rule's twin is closure-free exactly as its Lean statement is — stated over a bare forward-solution set, strictly stronger than a hypothesis-for-hypothesis transliteration.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail. This section also fixes the semantic setting shared by the three timed-core sections (QINVRELATIVIZE, QFIRSTSERVICE, QFIRSTSERVICEPLIT); the later two abbreviate it as “the timed-core setting”.

**Setting (shared by the timed-core sections).** A *behavioural model* is a state set  $X$  with atom denotations  $\llbracket \cdot \rrbracket \subseteq X$ , extended over  $\neg, \wedge, \vee$  as set operations (so  $\llbracket \chi \wedge \nu \rrbracket = \llbracket \chi \rrbracket \cap \llbracket \nu \rrbracket$ , etc.), and, for each  $x \in X$ , a set  $B_x$  of trajectories  $\tau : \mathbb{R}_{\geq 0} \rightarrow X$ , subject to three closure conditions and nothing else: **(B1)**  $\tau \in B_x \Rightarrow \tau(0) = x$ ; **(B2)**  $\tau \in B_x \Rightarrow (s \mapsto \tau(t+s)) \in B_{\tau(t)}$ ; **(B3)**  $\tau \in B_x$  and  $\tau' \in B_{\tau(t)}$  imply  $\tau \cdot_t \tau' \in B_x$ , where  $\tau \cdot_t \tau'$  follows  $\tau$  on  $[0, t)$  and  $\tau'$  (re-anchored) from  $t$  on. Empty  $B_x$  is permitted. The *début* of  $\tau$  into  $E \subseteq X$  is  $\tau_E := \inf\{t : \tau(t) \in E\}$  ( $\inf \emptyset = \infty$ ); worst-case service and invariance are

$$x \models A_T \varphi \iff \forall \tau \in B_x : \tau_{\llbracket \varphi \rrbracket} \leq T, \quad x \models \Box \varphi \iff \forall \tau \in B_x \forall t : \tau(t) \in \llbracket \varphi \rrbracket,$$

where deadlines  $T$  range over *finite* nonnegative reals (the mechanization types them so); in particular  $\tau_E \leq T$  forces the visit set nonempty. For  $S \subseteq \mathbb{R}_{\geq 0}$  and finite  $T \geq 0$ ,  $\inf S \leq T$  holds iff for every  $\varepsilon > 0$  some  $s \in S$  has  $s < T + \varepsilon$  — the infimum-free reading used throughout, and the form the mechanized statements take. A début need *not* be attained: nothing in B1–B3 forces a visit set to contain its infimum. The three sections split exactly along that line: QINVRELATIVIZE is sound over all behavioural models; the two first-service laws demand *attained débuts* for their obligation formulas (their sections state the attainment lemma and the countermodel making the demand honest).

**Claim (sound\_QInvRelativize).**  $\Box \nu \wedge A_T(\chi \wedge \nu) \leftrightarrow \Box \nu \wedge A_T \chi$  is valid in every behavioural model.

**Theorem R47.1** (soundness of QINVRELATIVIZE). *At every state  $x$  of every behavioural model:  $x \models \Box \nu \wedge A_T(\chi \wedge \nu)$  iff  $x \models \Box \nu \wedge A_T \chi$ .*

*Proof.* It suffices to show, for  $x \models \Box \nu$ , that  $x \models A_T(\chi \wedge \nu) \iff x \models A_T \chi$ ; conjoining  $\Box \nu$  on both sides then gives the displayed biconditional (both sides carry it as a conjunct).

Let  $\tau \in B_x$ . By  $\Box \nu$ , every point of  $\tau$  lies in  $\llbracket \nu \rrbracket$ , so

$$\{t : \tau(t) \in \llbracket \chi \wedge \nu \rrbracket\} = \{t : \tau(t) \in \llbracket \chi \rrbracket\} \cap \{t : \tau(t) \in \llbracket \nu \rrbracket\} = \{t : \tau(t) \in \llbracket \chi \rrbracket\},$$

the second equality because the second intersecting set is all of  $\mathbb{R}_{\geq 0}$ . Equal visit sets have equal infima:  $\tau_{\llbracket \chi \wedge \nu \rrbracket} = \tau_{\llbracket \chi \rrbracket}$ . So the two service conditions agree trajectory by trajectory, hence at  $x$ .  $\square$

*Remark R47.2* (why the invariant conjunct is everything). Drop  $\Box \nu$  and the right-to-left direction fails in a two-location clocked model whose  $\chi$ -location lies outside  $\nu$ : the service fact about  $\chi$  says nothing about  $\chi \wedge \nu$ . (Left-to-right needs no invariant — every  $\chi \wedge \nu$ -visit is a  $\chi$ -visit, so that direction is valid outright.) The proof uses  $\Box \nu$  once, exactly to collapse the two visit sets.

*Remark R47.3* (no closure is consumed). The proof never builds a trajectory: B1–B3 are not used, so the schema is sound over arbitrary trajectory assignments. Its independence witness must therefore deviate in the *operator* rather than the trajectory space, and does: the independence witness is a machine-checked, deadline-blind three-state structure whose service-operator interpretation deviates at an unreachable state; it validates the seed basis — the axiom set prior to the corrections these sections record — while refuting QINVRELATIVIZE directly. Nothing else in the basis ties finite worst-case débuts to reachability, which is precisely what this axiom asserts.

| proof step                    | formal anchor                                                                                                          |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------|
| Theorem <a href="#">R47.1</a> | <code>sound_QInvRelativize</code> (BernsteinLogic/QtsLTimedCore.lean); semantics<br>Behavior.AService, Behavior.BoxInv |
| independence witness          | machine-checked three-state deviant structure (see the remark above)                                                   |

## Rule 48. QFIRSTSERVICE (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QFirstService*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{B.\text{FusionClosed} \quad 0 < n \quad \forall i. \text{FirstVisits}(B, \alpha_i) \quad \forall i. \text{AService}(B, T_i, \alpha_i, x)}{\text{AService}(B, \min_i T_i, \text{fsTarget}(\alpha, T), x)}$$

$\text{fsTarget}(\alpha, T) = \bigcup_i (\alpha_i \cap \{y \mid \forall j \neq i. \text{AService}(B, T_j, \alpha_j, y)\})$  — some obligation served *right here* while every other still binds at its ORIGINAL budget; FirstVisits = nonempty visit sets have least elements (attained débuts).

certificate content out of scope of the current baked theory — realized in Lean over fusion-closed behaviours with attained débuts.

One of the three timed-core basis schemata proved non-admissible in the gmand completeness development (machine independence witnesses; see the proof section's anchors), entering as tier-1 core: its soundness obligation is semantic, discharged at Putnam-exam-solution detail in the proof section and mechanized in BernsteinLogic.QtslTimedCore. Sound over behavioural models with *attained débuts* for the obligation formulas — a real side condition: the proof section's closing remark carries the countermodel (mechanized) witnessing failure without it.

### Lean statement

BernsteinLogic.QtslTimedCore . sound\_QFirstService

statement bound

```
/-- **FS_n (first-service, EQ-FS)** – mechanized soundness. Over a
fusion-closed behavior with attained débuts for the obligations: if
every obligation binds at `x`, then within the tightest budget every
solution reaches a state where some obligation is served and every
other still binds at its ORIGINAL budget. Catalogue: tier-1 core
(schema family, n ≥ 2; the n = 1 instance is degenerate-derivable);
independence machine-checked – a blind 3-state structure validates the
FS-less basis over all definable sets while refuting a 2-obligation
instance of FS. Class boundary:
`qFirstService_attainment_necessary` below. -/
theorem sound_QFirstService (hB3 : B.FusionClosed) (hn : 0 < n)
  (α : Fin n → Set X) (T : Fin n → ℝ≥0) (x : X)
  (hatt : ∀ i, B.FirstVisits (α i))
  (hx : ∀ i, B.AService (T i) (α i) x) :
  B.AService (Finset.univ.inf' (Finset.univ_nonempty_iff.mpr {(0, hn)})) T)
  (B.fsTarget α T) x
```

BernsteinLogic.QtslTimedCore . qFirstService\_attainment\_necessary

statement bound

```
/-- **The theorem the master doc's countermodel E certifies** : there is
a fusion-closed behavior and a state satisfying the FS2 premise at
budgets (2, 2) whose FS-target conclusion FAILS – so the attainment
hypothesis of `sound_QFirstService` is not removable, and FS_n is
invalid over raw C0. Witness trajectory for the failure: `stay 0`;
along it every point of the FS-target would be a spine point at some
parameter in (1, 2) at which the OTHER atom still binds – but from a
rational point the escape `esc t t` never meets `Eb`, and from an
irrational point it never meets `Ea`. -/
theorem qFirstService_attainment_necessary :
  ∃ (S : Type) (B : Behavior S) (α : Fin 2 → Set S) (x : S),
  B.FusionClosed ∧
  (∀ i, B.AService 2 (α i) x) ∧
  ¬ B.AService 2 (B.fsTarget α (fun _ => 2)) x
```

### Correspondence (Steele display ↔ Lean)

**NO CORRESPONDENCE DICTIONARY YET**

no reviewer-certified dictionary maps the displayed rule onto the mechanized statements above; until one is written and certified, that mapping rests on this section's editorial care alone.

**Lean verifier receipts**

module BernsteinLogic.QtsLTimedCore; bound build e6370ca0a2ac... (bound from receipt 6460a6dd4d227c9a, 2026-08-04T01:16:03Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 6460a6dd4d227c9a | 2026-08-04T01:16:03Z | build+decorrelated | current |
| 659fe3396a48b674 | 2026-08-04T02:23:12Z | build+decorrelated | current |

4 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

**gate-layer coverage of the bound build:**

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

**HOL4 statement**

theory bern\_tr\_t5qdyn, HOL4 development script bern\_tr\_t5qdynScript.sml script bound

**Theorem sound\_QFirstService**

```
Theorem sound_QFirstService:
  ∀fwd al (Tb:num->real) n (x:'x).
    qdFusionClosed fwd ∧ 0 < n ∧
    (∀i. i < n ⇒ qdFirstVisits fwd (al i)) ∧
    (∀i. i < n ⇒ qdAService fwd (Tb i) (al i) x) ⇒
    qdAService fwd (rminN Tb n) (qdFsTarget fwd al Tb n) x
```

**HOL verifier receipts**

documentary hol-strict CLEAN: theory bern\_tr\_t5qdyn, 22 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob d64e1954...); an independent acceptance host ran the same build green at the porting seating (2026-07-21). The timed core this statement quantifies over is ported with an inventoried seam: trajectories ride the full real line with  $0 \leq t$  guards at exactly the positions where the nonnegative-time binders sit, and the behaviour structure's closure proof fields flatten to an explicit wellformedness conjunction; the fusion-closure hypothesis is consumed exactly as displayed. The début/attainment engine lemmas behind the statement (first-visit and served-point existence) are ported alongside as local instruments, mirroring their private status on the Lean side.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

**Proof**

Genre: core rule (tier 1), in the timed-core setting fixed at the QINVRELATIVIZE section. This section proves the attainment lemma shared by the timed-core sections; the class boundary (attained débuts are a genuine side condition) is certified by an explicit countermodel, sketched in the final remark and mechanized alongside the soundness lemma.

**Claim (sound\_QFirstService).** For  $n \geq 2$  obligations,

$$\bigwedge_{i \leq n} A_{T_i} \alpha_i \sqsubseteq A_{\min_i T_i} \left( \underbrace{\bigvee_i (\alpha_i \wedge \bigwedge_{i' \neq i} A_{T_{i'}} \alpha_{i'})}_{=: \Phi, \text{ the FS-target}} \right)$$

is valid in every behavioural model with *attained débuts* for the  $\alpha_i$ : along every trajectory, a nonempty  $\alpha_i$ -visit set has a least element.

**Lemma R48.1** (attainment). *Attained débuts hold: (i) for closed  $\llbracket \alpha_i \rrbracket$  in any model whose trajectories are right-continuous into a topological state space; (ii) in every clocked model, for every formula; (iii) in discrete time, for every formula.*

*Proof.* (i) Let  $E$  be closed,  $\tau$  right-continuous,  $V := \{t : \tau(t) \in E\}$  nonempty,  $t_0 := \inf V$ . If  $t_0 \notin V$ , pick visits  $v_n \downarrow t_0$  with  $v_n > t_0$ ; right-continuity at  $t_0$  gives  $\tau(v_n) \rightarrow \tau(t_0)$ , and since each  $\tau(v_n) \in E$ , every neighbourhood of  $\tau(t_0)$  meets  $E$ , so  $\tau(t_0) \in \overline{E} = E$  — contradiction. (No Hausdorff hypothesis: only “closed” and “converges” are used.) (ii) A clocked trajectory is a right-continuous step function through finitely many locations on any bounded window, with the post-jump location taken at the jump time, and every formula denotation is a union of locations — a subset of the finite discrete location set, hence closed; so (i) applies with the discrete topology. Concretely: given any visit at time  $R$ , the visit set restricted to  $[0, R]$  is a finite union of left-closed intervals, whose least point is the global least point. (iii) Nonempty subsets of  $\mathbb{N}$  have least elements.  $\square$

**Theorem R48.2** (soundness of QFIRSTSERVICE). *In every behavioural model with attained débuts for  $\alpha_1, \dots, \alpha_n$ , at every state  $x$ : if  $x \models A_{T_i} \alpha_i$  for all  $i$ , then  $x \models A_{\min_i T_i} \Phi$ .*

*Proof.* Let  $\tau \in B_x$ .

*First services exist.* For each  $i$ ,  $\tau_{\llbracket \alpha_i \rrbracket} \leq T_i < \infty$  forces the visit set nonempty; attainment gives its least element  $t_i = \tau_{\llbracket \alpha_i \rrbracket} \leq T_i$ .

*The first service overall.* Put  $t^* := \min_i t_i$ , attained at some  $i_0$ , and  $y := \tau(t^*) \in \llbracket \alpha_{i_0} \rrbracket$ . Choosing  $j^* \in \arg \min_i T_i$ :  $t^* \leq t_{j^*} \leq T_{j^*} = \min_i T_i$ .

*Co-obligations persist at  $y$ , at original budgets.* Fix  $i' \neq i_0$ ,  $\sigma \in B_y$ , and  $\varepsilon > 0$ . The fusion  $\rho := \tau \cdot t^* \sigma$  lies in  $B_x$  by B3 (typed by B1:  $\sigma(0) = y = \tau(t^*)$ ), so  $\rho_{\llbracket \alpha_{i'} \rrbracket} \leq T_{i'}$ , giving a  $\rho$ -visit of  $\alpha_{i'}$  at some  $s < T_{i'} + \varepsilon$ . If  $s < t^*$  then  $\rho(s) = \tau(s)$ , so  $t_{i'} \leq s < t^* \leq t_{i'}$  — impossible. Hence  $s \geq t^*$  and  $\rho(s) = \sigma(s - t^*)$ : a  $\sigma$ -visit at time  $s - t^* \leq s < T_{i'} + \varepsilon$ . As  $\varepsilon$  was arbitrary,  $\sigma_{\llbracket \alpha_{i'} \rrbracket} \leq T_{i'}$ . (Indeed  $\leq T_{i'} - t^*$ ; the schema states the weaker original-budget bound, which is what survives grid transport — the residual-split law QFIRSTSERVICE SPLIT is where the true residual is recovered.)

*Assembly.*  $\sigma$  was arbitrary, so  $y \models A_{T_{i'}} \alpha_{i'}$  for every  $i' \neq i_0$ ; with  $y \in \llbracket \alpha_{i_0} \rrbracket$ ,  $y \in \llbracket \Phi \rrbracket$ , and  $\tau$  visits it at  $t^* \leq \min_i T_i$ .  $\square$

**Remark R48.3** (each hypothesis, remarked). B1 types the fusion; B3 is the engine (facts about all of  $B_y$  reach back to the premise at  $x$  only through fusion); B2 is not consumed. Attainment is consumed exactly once, and essentially — to produce the service point  $y = \tau(t^*)$ : an infimum is not a visit. (The placement of  $s$  then runs on the ordinary lower-bound property of  $t_{i'}$ .) It is not removable: see the next remark. At  $n = 1$  the instance degenerates and is derivable; the schema family starts at  $n = 2$ .

**Remark R48.4** (the class boundary is real). Over bare B1–B3 models the schema is *false*. Countermodel (mechanized as `qFirstService_attainment_necessary`): a spine  $\{y_p : p \geq 0\}$  with  $\llbracket a \rrbracket$  the rational  $p \in (1, 2)$  and  $\llbracket b \rrbracket$  the irrational ones; from every  $y_p$ , the trajectories are the spine-ride and, for each  $s \geq p$  with  $s > 1$ , a ride-to- $s$ -then-exit to an atom-free sink. The family is fusion-closed; every trajectory from  $y_0$  meets both atoms before time 2 (density), so the premise  $A_2 a \wedge A_2 b$  holds; but at every  $y_q \in \llbracket a \rrbracket$  (so  $q \in (1, 2)$ , making the immediate exit legal) that exit kills  $A_2 b$ , at every  $y_i \in \llbracket b \rrbracket$  it kills  $A_2 a$ , and states outside the two atoms fail both disjuncts outright, so the FS-target denotes  $\emptyset$  and the conclusion fails. Dense interleaving of disjoint atoms along one trajectory is exactly what right-continuity into closed denotations forbids (Lemma R48.1(i)) — the countermodel lives strictly outside the attained class, as it must.

| proof step                           | formal anchor                                                                                                                                                                                        |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Theorem R48.2                        | <code>sound_QFirstService;</code> <code>class</code> <code>mixin</code> <code>Behavior.FusionClosed,</code><br><code>Behavior.FirstVisits;</code> <code>target Behavior.fsTarget</code>              |
| Lemma R48.1                          | <code>prose</code> (this section); clocked/discrete instances discharge <code>FirstVisits</code> per model                                                                                           |
| Remark R48.4<br>independence witness | <code>qFirstService_attainment_necessary</code> (the countermodel, mechanized)<br>machine-checked structure validating the corrected basis with this schema removed while refuting the schema itself |

## Rule 49. QFIRSTSERVICE SPLIT (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QFirstServiceSplit*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{B.\text{FusionClosed} \quad 0 < n \quad \forall i. c < T_i \quad \forall i. \text{FirstVisits}(B, \alpha_i) \quad \forall i. \text{AService}(B, T_i, \alpha_i, x)}{\text{AService}(B, c, \text{fsTarget}(\alpha, T) \cup \text{residualCell}(\alpha, T, c), x)}$$

$\text{residualCell}(\alpha, T, c) = \{y \mid \forall i. \text{AService}(B, T_i - c, \alpha_i, y)\}$  — the no-service-by- $c$  branch pins elapsed time by the cutoff, recovering TRUE residuals  $T_i - c$  where  $\text{FS}_n$  can only re-issue original budgets.

certificate content out of scope of the current baked theory — realized in Lean over fusion-closed behaviours with attained débuts.

One of the three timed-core basis schemata proved non-admissible in the gmand completeness development — for this law the machine independence witnesses cover integer and half-integer deadline menus, with finer denominators open (see the proof section's closing remark and anchors) — entering as tier-1 core: its soundness obligation is semantic, discharged at Putnam-exam-solution detail in the proof section and mechanized in *BernsteinLogic.QtslTimedCore*. Sound over behavioural models with *attained débuts* for the obligation formulas — a real side condition; its witnessing countermodel is prose-only (not yet mechanized; the *QFirstService* countermodel is the mechanized one).

### Lean statement

BernsteinLogic.QtslTimedCore . sound\_QFirstServiceSplit

statement bound

```
/-- **FS-R (residual-split first-service)** – mechanized soundness,
same class as FS_n. For any cutoff `c` strictly below every budget:
within `c`, every solution reaches EITHER a first-service state
(original co-budgets – the FS-target) OR the residual cell, where
every obligation holds at its true residual `T i - c`. Genuinely new
on top of FS_n; independence machine-checked – a minimal 6-state
witness settles every all-integer deadline menu, and two further
certified 6-state witnesses extend the underivability to all
half-integer menus. -/
theorem sound_QFirstServiceSplit (hB3 : B.FusionClosed) (hn : 0 < n)
  (α : Fin n → Set X) (T : Fin n → ℝ≥0) (c : ℝ≥0) (x : X)
  (hc : ∀ i, c < T i)
  (hatt : ∀ i, B.FirstVisits (α i))
  (hx : ∀ i, B.AService (T i) (α i) x) :
  B.AService c (B.fsTarget α T ∪ B.residualCell α T c) x
```

### Correspondence (Steele display ↔ Lean)

#### NO CORRESPONDENCE DICTIONARY YET

no reviewer-certified dictionary maps the displayed rule onto the mechanized statements above; until one is written and certified, that mapping rests on this section's editorial care alone.

### Lean verifier receipts

module BernsteinLogic.QtslTimedCore; bound build e6370ca0a2ac... (bound from receipt 6460a6dd4d227c9a, 2026-08-04T01:16:03Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 6460a6dd4d227c9a | 2026-08-04T01:16:03Z | build+decorrelated | current |

4 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_t5qdyn, HOL4 development script bern\_tr\_t5qdynScript.sml script bound

### Theorem sound\_QFirstServiceSplit

```
Theorem sound_QFirstServiceSplit:
  ∀fwd al (Tb:num->real) n c (x:'x).
    qdFusionClosed fwd ∧ 0 < n ∧ 0 ≤ c ∧
    (∀i. i < n ⇒ c < Tb i) ∧
    (∀i. i < n ⇒ qdFirstVisits fwd (al i)) ∧
    (∀i. i < n ⇒ qdAService fwd (Tb i) (al i) x) ⇒
    qdAService fwd c
    (qdFsTarget fwd al Tb n UNION qdResidualCell fwd al Tb n c) x
```

## HOL verifier receipts

documentary hol-strict CLEAN: theory bern\_tr\_t5qdyn, 22 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob d64e1954...); an independent acceptance host ran the same build green at the porting seating (2026-07-21). The timed core this statement quantifies over is ported with an inventoried seam: trajectories ride the full real line with  $0 \leq t$  guards at exactly the positions where the nonnegative-time binders sit, and the behaviour structure's closure proof fields flatten to an explicit wellformedness conjunction. The residual budget's truncated subtraction is rendered as the max-0 coercion — agreeing with the nonnegative-reals reading on the whole consulted domain, not only under the split theorem's guard.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1), in the timed-core setting fixed at the QINVRELATIVIZE section, with the attainment lemma of the QFIRSTSERVICE section. This is the fourth and final basis correction from the gnand completeness analysis: the law that makes the decision procedure's deadline-tightening step derivable.

**Claim (sound\_QFirstServiceSplit).** For  $n \geq 1$  obligations and any cutoff  $0 < c < \min_i T_i$ ,

$$\bigwedge_{i \leq n} A_{T_i} \alpha_i \sqsubseteq A_c \left( \Phi \vee \underbrace{\bigwedge_i A_{T_i - c} \alpha_i}_{=: P, \text{ the residual cell}} \right)$$

is valid in every behavioural model with attained débuts for the  $\alpha_i$ , where  $\Phi$  is the FS-target of QFIRSTSERVICE (read at  $n = 1$  with the empty co-obligation conjunction as  $\top$ , so  $\Phi = \alpha_1$ ). Reading: within the cutoff, either some first service has happened (and the co-obligations persist at original budgets), or nothing has been served and every obligation holds at its *true* residual.

**Theorem R49.1** (soundness of QFIRSTSERVICE SPLIT). *In every behavioural model with attained débuts for  $\alpha_1, \dots, \alpha_n$ , at every state  $x$ : if  $x \models A_{T_i} \alpha_i$  for all  $i$  and  $0 < c < \min_i T_i$ , then  $x \models A_c (\Phi \vee P)$ .*

*Proof.* Let  $\tau \in B_x$ . As in the QFIRSTSERVICE proof, attainment gives each obligation a least visit time  $t_i \leq T_i$  along  $\tau$ ; put  $t^* := \min_i t_i$ , attained at  $i_0$ . Split on the cutoff.

*Case  $t^* \leq c$ .* The QFIRSTSERVICE argument, verbatim, puts  $y := \tau(t^*)$  in  $\llbracket \Phi \rrbracket$ : it serves  $\alpha_{i_0}$ , and for  $i' \neq i_0$  every  $\sigma \in B_y$  inherits  $\sigma \llbracket \alpha_{i'} \rrbracket \leq T_{i'}$  through the fusion  $\tau \cdot_{t^*} \sigma \in B_x$ , the prefix being  $\alpha_{i'}$ -free below  $t^*$  by minimality. So  $\tau$  visits  $\llbracket \Phi \vee P \rrbracket$  at  $t^* \leq c$ .

*Case  $t^* > c$ .* Let  $y := \tau(c)$ ; we show  $y \in \llbracket P \rrbracket$ , so that  $\tau$  visits the disjunction at time  $c$ . Fix  $i, \sigma \in B_y, \varepsilon > 0$ . Fuse:  $\rho := \tau \cdot_c \sigma \in B_x$  by B3, so there is a  $\rho$ -visit of  $\alpha_i$  at some  $s < T_i + \varepsilon$ . The prefix is service-free through  $c$ : for  $s' < c$ ,  $\rho(s') = \tau(s')$ , and a visit



there would give  $t_i \leq s' < c < t^* \leq t_i$  — impossible; and the point at  $c$  itself is  $\rho(c) = \sigma(0) = \tau(c)$ , likewise unvisited (a visit at  $c$  would give  $t_i \leq c < t^*$ ). So  $s \geq c$  and  $\rho(s) = \sigma(s - c)$ : a  $\sigma$ -visit at time  $s - c < (T_i - c) + \varepsilon$  (subtract  $c$  across  $s < T_i + \varepsilon$ , using  $c < T_i$  to keep the residual budget a genuine nonnegative quantity). As  $\varepsilon$  was arbitrary,  $\sigma_{\llbracket \alpha_i \rrbracket} \leq T_i - c$ .  $\square$

*Remark R49.2* (each hypothesis, remarked).  $c < \min_i T_i$  keeps every residual budget strictly positive — the regime the law is stated for; at  $c = \min_i T_i$  the split degenerates (under attained débuts the served case always fires, since  $t^* \leq \min_i T_i$ ) and the residual row would carry  $A_0$ . Attainment is consumed only through  $t^*$  — and only the *served* case consumes it: the no-service case uses the visit  $s$  directly, never its infimum. That asymmetry is honest and sharp: over bare B1–B3 models the residual branch alone survives, and the schema as a whole fails — the countermodel (prose; not mechanized) retunes the QFIRSTSERVICE countermodel — the dense cluster moved, the escape-legality threshold lowered to match — and adds “lazy wings”: branches postponing every service to just inside the original budgets, legal precisely because the premise pins only original budgets before the cluster, defeating target and residual simultaneously along the spine. Attained débuts restore the dichotomy the case split runs on.

*Remark R49.3* (what the law buys). QFIRSTSERVICE can only re-issue original budgets at the service point; QFIRSTSERVICE SPLIT recovers the true residual  $T_i - c$  on the no-service branch, where the elapsed time is pinned by the cutoff rather than by an unknowable service instant. Iterated at successive cutoffs this derives the decision procedure’s deadline-tightening step exactly (its descent lemma), which is what closed the derivation side of the g<sub>nd</sub> completeness theorem. Independence: a six-state witness proves the law underivable from the FS-corrected basis for integer deadline menus, extended to half-integer menus by two certified witnesses; independence at finer denominators is open, and citation is bounded accordingly.

**proof step**

**formal anchor**

Theorem R49.1  
countermodel (class boundary)  
independence witnesses

sound\_QFirstServiceSplit; residual Behavior.residualCell  
prose only (not yet mechanized); sketched in the first remark  
six-state minimal witness + two half-integer-menu witnesses, machine-  
checked (see the closing remark for the citation bound)

## 15.6 Streett Rules and the Specification Surface

### Rule 50. QSTREETTDEGENERATE (alias for QSTREETTBATTERY)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QStreettDegenerate*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{one QSTREETTBATTERY node at } \rho_i := \text{STrue for every pair} \\ (\text{absorbing families and } J_i \text{ syntactically empty; ranking decay off } \llbracket \varphi_i \rrbracket \text{ alone})}{\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\text{STrue}, \varphi_1), \dots, \text{PairBase}(\text{STrue}, \varphi_k)))}$$

the pair at request  $\top$  means  $\text{Inf}(\llbracket \varphi_i \rrbracket)$  — the classical recurrence certificate; the degenerate-pair specialization erases the tautological rows before emission, so the emitted kit is the recurrence kit row for row.

QSTREETTDEGENERATE is the recurrence rule in its compact spelling: to conclude, with probability at least  $p$ , that every response region is visited infinitely often (GF-shaped claims, the classical Büchi/recurrence certificate), instantiate the Streett bundle at the trivially-true request. It is an *alias* (né StreettDegenerate — the first sub-rule of the battery row): one QSTREETTBATTERY application, at request regions equal to the whole state space, is the entire unpacking — the emptiness it exploits is syntactic, not semantic. When every state requests, the finitely-many-requests disjunct of each pair is empty along the tail filter, the absorbing-certificate family and its carved-out sets  $J_i$  are vacuous, and the ranking decay row is demanded off the response region alone — so the degenerate-pair specialization erases exactly the tautological rows, and what the checker replays is the recurrence certificate, row for row. A producer uses it wherever the specification is recurrence rather than request-response fairness; nothing about the certificate synthesis changes except that the request machinery never materializes.

Scope, stated plainly: the bound statement below is the degenerate-instance witness `sound_QStreettDegenerate` (module `BernsteinLogicAux.QStreett` — one application of `sound_QStreettBattery` at request set `univ`, whose finitely-many-requests disjunct is empty along the tail filter; its conclusion is the recurrence event  $\Pr[\forall i, \text{Inf}(G_i)] \geq p$ , exactly the amended display's claim). The witness covers the rule at the *discrete-kernel profile* (a time-homogeneous Markov chain presented by a single step kernel); this entry once carried a conditional mark inherited from the bundle's continuous-time residue, and it goes the way the bundle's went: at the martingale-problem profile the bundle is discharged whole — strict-decrement rows included, hash-bound under the QSTREETTBATTERY entry — and the alias inherits that discharge the same way it inherits everything else, as one bundle application at request set `univ`, per the bundle section's profile remark. The payload-level unpack function is not among this edition's bound artefacts: the alias deliberately has no wire sort of its own (producers elaborate before emission), and the transport table in the proof is stated at the catalogued-field level.

**notes** The degenerate pair is also the honest cost anchor of the Streett-primitive design: on recurrence-shaped workloads the bundle emits the same rows a dedicated recurrence rule would, so the general rule costs nothing where the special case suffices — while the reverse specialization (recurrence-primitive, Streett by automaton product) would pay exponentially in the number of pairs.

#### Lean statement

BernsteinLogicAux.QStreett . **sound\_QStreettDegenerate**

statement bound

```
-- **`QStreettDegenerate` (T9a)**: request `= T` degenerates the pair to
`Inf(G_i)` — the classical recurrence certificate, one `sound_QStreettBattery`
application with the request set `univ` (whose `Fin` is empty along `atTop`).
Alias tier: the statement is the bundle's, specialized and simplified. -/
theorem sound_QStreettDegenerate {G : ι → Set S} {W : ι → S → ℝ} {ε C : ι → ℝ}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV₀ : Measurable V₀)
  (hG : ∀ i, MeasurableSet (G i)) (hW : ∀ i, Measurable (W i))
  (hθ : p + thr ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ U → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ U → K y (D ∪ U) = 1)
  (hV₀int : ∀ y ∈ D, y ∉ U → Integrable V₀ (K y))
  (hV₀drift : ∀ y ∈ D, y ∉ U → ∫ z, V₀ z ∂K y ≤ V₀ y)
  (hV₀floor : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V₀ y)
```

```

(hV0nonneg :  $\forall y \in \text{Dbar}, 0 \leq V_0 y$ )
(h $\epsilon$  :  $\forall i, 0 < \epsilon i$ )
(hW $\emptyset$  :  $\forall i, \forall y \in \text{Dbar}, 0 \leq W i y$ ) (hWC :  $\forall i, \forall y \in \text{Dbar}, W i y \leq C i$ )
(hWint :  $\forall i, \forall y \in D, y \notin U \rightarrow y \notin G i \rightarrow \text{Integrable } (W i) (K y)$ )
(hWdriftR :  $\forall i, \forall y \in D, y \notin U \rightarrow y \notin G i \rightarrow \int z, W i z \partial K y \leq W i y - \epsilon i$ )
(hx :  $x \in D$ ) (hthr :  $V_0 x \leq \text{thr}$ ) :
ENNReal.ofReal p  $\leq$  pathMeasure K x
{ $\omega \mid \forall i, \exists^f n \text{ in atTop}, \omega n \in G i$ }

```

**The general-stochastic instance.** The degenerate reading needs no separate statement at the segment-model tier: it rides the confined form of the bundle instance below with the vacuous premise sets — the same instantiation-at-the-skeleton the bundle section's general-stochastic block describes.

BernsteinLogicAux.LayerD.Streett.sound\_GSHS\_streett\_bundle\_confined

statement bound

```

/-- **`sound_QStreettBattery`, GSHS instance, confined form** (T9; assault
doc §2.7 Theorem G at the conservative ranking scope, mechanized
restart-free – module docstring for why G1/G2's epoch machinery is not
consumed). Hypothesis fields are segment-level and per-model: the three
containment rows, the  $\theta = \emptyset$  confinement field for `V0`, and per pair the
scoped ranking fields for `W i` (nonincrease off the response region,
` $\epsilon i$ `-decrease at request entries – per-step shape, module docstring for
the rate-form conversion). Conclusion: with probability at least `p` the
skeleton chain of `Rep(P)` stays alive in `D \ U` forever **and** every
certified pair satisfies the chain-level Streett condition. Semantic
transport of the chain events: module docstring (T9b monotonicity for
requests, entry reads for responses, W-Z for time-cofinality). -/
theorem sound_GSHS_streett_bundle_confined
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV0 : Measurable V0)
  (hRq :  $\forall i, \text{MeasurableSet } (Rq i)$ ) (hG :  $\forall i, \text{MeasurableSet } (G i)$ )
  (hW :  $\forall i, \text{Measurable } (W i)$ )
  (h $\theta$  :  $p + \text{thr} \leq 1$ ) (hDsub :  $D \subseteq \text{Dbar}$ )
  (hLive : NoVanish M (D \ U)) (hHalt : HaltsCaught M (D \ U) (U n Dbar))
  (hReset : ResetsInto M (D \ U) ((D u U) n Dbar))
  (hSegD0 : RankingSegmentDynkin M (D \ U) V0  $\emptyset$ )
  (hForce0 : ForcedNonexpansion M V0) (hNonneg0 : NonnegV V0)
  (hC0 $\emptyset$  :  $\emptyset \leq C_0$ ) (hV0C :  $\forall y, V_0 y \leq C_0$ )
  (hFloor0 :  $\forall y \in U, y \in \text{Dbar} \rightarrow 1 \leq V_0 y$ )
  (h $\epsilon$  :  $\forall i, 0 < \epsilon i$ )
  (hSegDW :  $\forall i, \text{RankingSegmentDynkin } M ((D \ U) \setminus G i) (W i) \emptyset$ )
  (hSegDWR :  $\forall i, \text{RankingSegmentDynkin } M (((D \ U) \setminus Rq i) \setminus G i) (W i) (\epsilon i)$ )
  (hForceW :  $\forall i, \text{ForcedNonexpansion } M (W i)$ ) (hNonnegW :  $\forall i, \text{NonnegV } (W i)$ )
  (hC $\emptyset$  :  $\forall i, \emptyset \leq C i$ ) (hWC :  $\forall i, \forall y, W i y \leq C i$ )
  (hx :  $x \in D$ ) (hthr :  $V_0 x \leq \text{thr}$ ) :
ENNReal.ofReal p  $\leq$  Battery.pathMeasure (skel M) (SkelState.alive x)
  { $\omega \mid (\forall n, \omega n \in \text{skelIn } D \setminus \text{skelBad } U) \wedge$ 
     $\forall i, (\exists^f n \text{ in atTop}, \omega n \in \text{skelIn } (Rq i)) \rightarrow$ 
     $\exists^f n \text{ in atTop}, \omega n \in \text{skelIn } (G i)$ }

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed conclusion  $\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\text{STrue}, \varphi_1), \dots, \text{PairBase}(\text{STrue}, \varphi_k)))$   $\mapsto$   $\text{ofReal } p \leq \text{pathMeasure } K \times \{\omega \mid \forall i, \exists^f n \text{ in atTop}, \omega n \in G i\}$  for every start with  $x \in D$  and  $V_0 x \leq \text{thr}$  (the Lean source's identifier for the display's  $y$ ) — the recurrence event, with the  $\text{STrue}$ -requests *erased at the statement level*, not merely in the proof: no request binder exists in `sound_QStreettDegenerate` at all, which is the display's “absorbing families and  $J_i$  syntactically empty” carried into the type. The alias tier is visible in the proof term: one `sound_QStreettBattery` application at `Rq := fun _ => Set.univ`, whose frequently-clause is discharged by `univ-membership`.

- the inherited  $V_0$  QSafetyBattery kit  $\mapsto$  the four kernel-integral rows verbatim:  $hV_{\circ}drift$  (interior),  $hV_{\circ}floor$  (floor on  $U \cap \bar{D}$ ),  $hV_{\circ}nonneg$  (on  $\bar{D}$ ), with  $hV_{\circ}int$  the undisplayed integrability rider, plus the hull/fail-closed side conditions  $hR3/hR4$  and the threshold  $h\theta : p + thr \leq 1$  — the battery pattern, unchanged.
- “ranking decay off  $\llbracket \varphi_i \rrbracket$  alone”  $\mapsto hWdriftR : \forall i, \forall y \in D, y \notin U \rightarrow y \notin G_i \rightarrow \int z, W i z \partial K y \leq W i y - \varepsilon_i$  — one generator row where the bundle carries two: the request-conditioned decay and the off-response non-increase coincide when every state is a request, and the statement asks only the merged row (the proof re-splits it for the bundle).
- $nonneg + upper-bound$  rows  $\mapsto hW0/hWC$  on  $\bar{D}$ ; strictness  $\mapsto h\varepsilon : \forall i, 0 < \varepsilon_i$  — note the specialized statement takes per-pair  $\varepsilon_i$ , mildly wider than the display's single  $\varepsilon_0$ .
- the measurability pack ( $hD/hDbar/hU/hV_{\circ}/hG/hW$ ) is undisplayed profile regularity, per the battery precedent.

Width note: this alias's warrant is inherited from the bundle at both profiles (hash-bound at the discrete-kernel profile; the martingale-problem discharge, strict rows included, hash-bound under the bundle entry); the producer-side erasure of tautological rows before emission is compile behaviour with no Lean counterpart — correctly, since the certified statement never receives the tautological rows to begin with.

## Lean verifier receipts

module BernsteinLogicAux.QStreett; bound build 469881beb770... (bound from receipt 9d1d848e0ba1718a, 2026-07-29T01:03:31Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| e3990fe544d4708d | 2026-07-28T23:48:35Z | build+decorrelated                 | current |
| 68e9374fa1b418fe | 2026-07-29T00:00:34Z | build+decorrelated                 | current |
| e5895c9706854cd3 | 2026-07-29T00:49:04Z | build+replay+hardened+decorrelated | current |
| 9d1d848e0ba1718a | 2026-07-29T01:03:31Z | build+replay+hardened+decorrelated | current |
| 4a8d377b721b8523 | 2026-07-31T22:00:09Z | build+decorrelated                 | current |
| d251a122540ab741 | 2026-07-31T22:29:03Z | build+decorrelated                 | current |
| 333635b8e047113a | 2026-08-02T23:31:14Z | build+decorrelated                 | current |
| 9e96f80d63e4ad82 | 2026-08-03T16:27:18Z | build+decorrelated                 | current |
| 932b4b9e0b0fd323 | 2026-08-03T22:46:02Z | build+decorrelated                 | current |

9 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.LayerD.Streett; bound build 7393065bbc7f... (bound from receipt abac471866cfcfd1, 2026-07-21T05:38:48Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt           | minted               | layers                             | status  |
|-------------------|----------------------|------------------------------------|---------|
| 0fb3b50b4d3f0d33c | 2026-07-20T22:52:50Z | build+decorrelated                 | current |
| 109669f848453ec2  | 2026-07-21T00:03:19Z | build+decorrelated                 | current |
| 7e17aa32b276623b  | 2026-07-21T04:51:49Z | build+decorrelated                 | current |
| abac471866cfcfd1  | 2026-07-21T05:38:48Z | build+replay+hardened+decorrelated | current |

6 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*, not by backlog: the rule is a producer-side respelling that unpacks to a single QSTREETTBATTERY node before any certificate is emitted, so no wire object can ever contain a QSTREETTDEGENERATE node — the HOL statement that would verify a degenerate-request bundle is the bundle's own (itself absent today, for the reason given in that section: the satisfaction stratum has no HOL artefact). Checked against the HOL sources: the bern\_catalogue theory is the base-calculus checker cone and carries nothing Streett-shaped.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template. This section additionally states, once for the whole batch of satisfaction-level aliases, the general template those sections share; the QSTREETTCMPLREQUEST, QSTREETTTARGETCELLCOLLAPSE, QNNFREWRITE, QSTREETTREQUESTNNF, QAVOID, QSTREETTGF, QSTREETTFG, and QNEGFALIAS sections reference it by this section's rule name instead of restating it.

**Recipe R50.1** (the satisfaction-level alias template). An alias rule  $A$  at the satisfaction stratum is warranted by a  $1 \rightarrow 1$  unpack into a single application of a target rule  $T$  (or, for the spec-surface rewrites, into a single definitional denotation equality): the  $A$ -node and its image conclude the same judgment from the same premise data, payload transported, nothing derived. Soundness is then a corollary,  $\text{sound}_A = \text{unpack} \circ \text{sound}_T$ , and the certificate format need never carry  $A$ : producers elaborate before emission. The five artefacts an alias owes this book are the same as at the base stratum (the SMALLGAIN section is the model): the unpack, the filled-in target application, the payload-transport table, the exactness identity, and the soundness corollary. One honest divergence from the base-stratum model, stated once here: no alias of this batch has a *verified payload function* among this edition's bound artefacts. The transport tables of this batch are therefore stated at the catalogued-field level, and the unpack-function slot of each alias says so plainly; where a discrete-profile *semantic* witness exists in the auxiliary Lean development it is named (the soundness content is machine-checked even though the payload-level function is not), with its statement pin riding the bindings record wherever the per-section text says hash-bound, and where none exists the status is stated as filed. In every case it is the payload-level slot, never the semantic witness, that remains outside the bound warrant; the per-section text says which case holds.

**Recipe R50.2** (the filled-in application — QSTREETTDEGENERATE rewritten into its QSTREETTBATTERY instance). A degenerate node claims, from the confinement certificate  $V_0$  and per-region ranking certificates  $(W_i, \varepsilon_i)$ , that with probability  $\geq p$  every response region  $G_i$  is visited infinitely often. Its image is the single bundle node at request regions  $R_i := S$  (every state requests):

$$\frac{\text{confinement rows for } V_0 \quad \{\vdash \text{nonincrease of } W_i \text{ off } G_i\} \quad \{\vdash \varepsilon_i\text{-decay of } W_i \text{ off } G_i\} \quad (\text{the request row at } R_i = S) \quad p + y \leq 1 \text{ (threshold tie)}}{\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\text{STrue}, \varphi_1), \dots, \text{PairBase}(\text{STrue}, \varphi_k)))} \quad \text{QSTREETTBATTERY}$$

Discharging the bundle's obligations at this instance: the request regions are measurable trivially; the decay row *subsumes* the nonincrease row (off  $G_i$ , every state is a request state, so nonincrease follows from decay with the kit's  $\varepsilon_i > 0$  — nonnegativity is all the subsumption spends); the absorbing families  $(V_i, \gamma_i)$  and carved sets  $J_i$  are vacuous — each  $J_i$  lives inside  $I \setminus \llbracket \rho_i \rrbracket$ , with  $I$  the confinement region of the bundle's carved-scope formula, and at  $\rho_i = \top$  that base is  $I \setminus S = \emptyset$ , so  $J_i = \emptyset$  syntactically, the rows carved to  $J_i$  and to off-request regions are rows over the empty region, and the specialization targets the bundle's conservative (absorber-free) scope — the scope its ranking-route soundness theorem consumes — so no request-region absorber obligation survives. The dead premise slots are honestly dead, and the degenerate-pair specialization pass erases them before emission.

### Recipe R50.3 (payload transport, catalogued-field level).

#### degenerate node

$V_0$  rows, threshold, init  
response regions  $G_i$ , certificates  $(W_i, \varepsilon_i)$   
shared side conditions (hull, fail-closed) + measurability pack  
—  
—  
decay rows off  $G_i$

#### QSTREETTBATTERY node

verbatim  
verbatim (bound rows  $0 \leq W_i \leq C_i$  and strict  $\varepsilon_i > 0$  included)  
verbatim (the battery pattern's)  
request regions  $:= S$  (each pair; measurable trivially)  
absorbing families: empty;  $J_i = \emptyset$  (syntactic)  
request-decay rows (at  $R_i = S$ ); nonincrease subsumed

Unpack-function slot: no verified payload function in this edition (Recipe R50.1); the semantic witness exists — `sound_QStreettDegenerate`, one `sound_QStreettBattery` application at request set `univ` (auxiliary module; hash-bound in this edition).

**Proposition R50.4** (the exactness identity). *For every path  $\omega$  and pair  $i$ , at  $R_i = S$ :*

$$(\exists^\infty n, \omega_n \in S) \Rightarrow (\exists^\infty n, \omega_n \in G_i) \iff \exists^\infty n, \omega_n \in G_i.$$

*Proof.* The hypothesis  $\exists^\infty n, \omega_n \in S$  holds of every path (every time is a request time), so the implication and its conclusion are equivalent — one line, which is the point. Equivalently, in acceptance vocabulary: the finitely-many-requests disjunct  $\text{Fin}(S)$  is empty along the tail filter, so the pair condition  $\text{Fin}(R_i) \vee \text{Inf}(G_i)$  collapses to  $\text{Inf}(G_i)$ .  $\square$

**Proposition R50.5** (soundness corollary). *Under the degenerate node's rows,  $\mu_x\{\omega \mid \forall i, \exists^\infty n, \omega_n \in G_i\} \geq p$ .*

*Proof.* Apply the bundle's soundness theorem (the QSTREETTBATTERY section's — at its proven scope, the discrete-kernel profile and conservative ranking scope; the bundle's martingale-problem discharge transfers to this alias the same way, one application at the trivially-true request) at  $R_i = S$  and rewrite the event by Proposition R50.4. There is no other content — the alias genre working as intended. (Machine-checked as sound\_QStreettDegenerate; hash-bound in this edition.)  $\square$

*Remark R50.6* (this is the Büchi case). A Büchi condition *is* the one-pair Streett condition with trivially-true request, so this alias is where the classical recurrence certificate lives inside the Streett-primitive design. Deterministic Büchi automata recognize only limit languages [150], so the reverse direction — expressing a general Streett condition Büchi-primitively — is not a respelling at all but an automaton re-encoding, exponential in the number of pairs for the classical translations; the asymmetry is why the bundle, not the recurrence rule, is the primitive ([226, 232]; see the bundle section's remark).

| proof step             | formal anchor                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------|
| Recipe R50.2 target    | the QSTREETTBATTERY section, Theorem & hypotheses (H1)–(H5)                                        |
| Proposition R50.4      | syntactic; no formal anchor needed (one line of filter algebra)                                    |
| Proposition R50.5      | sound_QStreettDegenerate (BernsteinLogicAux.QStreett; machine-checked, hash-bound in this edition) |
| no wire tag, by design | producers elaborate before emission (Recipe R50.1)                                                 |

## Rule 51. QSTREETTCMPLREQUEST (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QStreettCmplRequest*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{a QSTREETTBATTERY kit at certified regions } (R_i, G_i) \quad \{\llbracket \text{cmpl } \rho_i \rrbracket \subseteq R_i\} \text{ (requests over-approximated)} \quad \{G_i \subseteq \llbracket \varphi_i \rrbracket\} \text{ (responses under-approximated)}}{\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairCmpl}(\rho_1, \varphi_1), \dots))}$$

requests sit on the hypothesis side of each pair implication, so the orientation swap of the complement (the workhorse judgment certifies under-approximations; complements arrive as over-approximations) is exactly the sound direction, for free.

QSTREETTCMPLREQUEST lets a Streett pair carry its request as a *complement*: the pair  $(\text{cmpl } \rho_i, \varphi_i)$  asks that if states *outside*  $\llbracket \rho_i \rrbracket$  recur forever, responses recur forever. The satisfaction judgment's workhorse orientation certifies regions that *under-approximate* a formula's denotation — one orientation, deliberately: the complement of an under-approximation is an *over-approximation*, so a complement-typed request arrives with the opposite orientation for free. The rule (né StreettCmplRequest) is *admissible*, not *alias*: its unpacking is one QSTREETTBATTERY node *plus* region-weakening premises — the certified request regions over-approximate the complement denotations, the certified response regions under-approximate the response denotations — and the direction check that makes this the sound weakening is genuine content, not transport. Requests sit on the hypothesis side of each pair implication and responses on the conclusion side, so over-approximating requests and under-approximating responses each *weaken the claimed event*: any path satisfying the certified pair satisfies the semantic pair. What the checker recomputes: nothing new in kind — the weakening premises are ordinary region-containment rows it already knows how to verify, and the complement itself is consumed at the region-atom level (the atom dualities of the QNNFREWRITE section), never as a formula node.

Scope, stated plainly (this entry is marked *conditional*): the bound statements below are the discrete-profile witnesses — the event-level direction lemma `streett_event_mono` (two monotonicity steps, exactly the two lines the recipe below typesets) and the assembled bundle-through-formula-tie `sound_QStreettBattery_tie` (requests over-approximated, responses under-approximated, concluded over the semantic satisfaction predicates), both in module `BernsteinLogicAux.QStreett`, plus the atom-level complement step `denot_elimNeg` in `BernsteinLogicAux.NNFDuality`. One named residue keeps the flag — and it is this entry's own, not the bundle's (whose continuous-time residue is discharged whole at the martingale-problem profile, strict rows included, hash-bound under the QSTREETTBATTERY entry): the continuous-time face of the complement step remains prose. At the general-stochastic-hybrid profile, consuming a complemented request requires fine-topology regularity of the region (that the region is nearly Borel and contains its regular points), which is argued in the design record.

**notes** Why one orientation of the workhorse judgment suffices, in one line: an under-approximating region calculus is closed under the connectives the rules consume, and where a complement appears, the swap to over-approximation is exactly what the hypothesis side of an implication wants — demanding a second, over-approximating judgment kind would duplicate every rule for zero content.

### Lean statement

`BernsteinLogicAux.QStreett.streett_event_mono`

statement bound

```
-- **`QStreettCmplRequest` (T9b), the event-level kernel** – the discrete face
of request/response event monotonicity: requests over-approximated and
responses under-approximated weaken the per-pair Streett event in the
sound direction. Two `Frequently.mono`s – the "two lines" the catalogue prices. -/
theorem streett_event_mono {Rq Rqhat G Ghat : Set S} (hR : Rq ≤ Rqhat)
  (hG : Ghat ≤ G) (ω : ℕ → S)
  (h : (∃ n in atTop, ω n ∈ Rqhat) → ∃ n in atTop, ω n ∈ Ghat) :
  (∃ n in atTop, ω n ∈ Rq) → ∃ n in atTop, ω n ∈ G
```

`BernsteinLogicAux.QStreett.sound_QStreettBattery_tie`

statement bound

```
-- **`QStreettCmplRequest` (T9b), assembled** – the bundle consumed through the
formula tie, direction-checked by request/response event monotonicity and the
antitone region-complement kernel: the certified request region
`Rq i` over-approximates the truth set of the semantic request `reqSat i` (for
`cmpl`-typed requests this is atom-level complement elimination's
`llbracket cmpl φ llbracket ≥ llbracket φ llbracket^c = llbracket ¬φ llbracket`), and the certified response region
under-approximates the semantic response. Conclusion over the semantic
```



```

satisfaction predicates; no measurability is asked of them. -/
theorem sound_QStreettBattery_tie {Rq G :  $\iota \rightarrow \text{Set } S$ } {W :  $\iota \rightarrow S \rightarrow \mathbb{R}$ } { $\varepsilon$  C :  $\iota \rightarrow \mathbb{R}$ }
  {reqSat respSat :  $\iota \rightarrow S \rightarrow \text{Prop}$ }
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV0 : Measurable V0)
  (hRq :  $\forall i$ , MeasurableSet (Rq i)) (hG :  $\forall i$ , MeasurableSet (G i))
  (hW :  $\forall i$ , Measurable (W i))
  (hreq :  $\forall i y$ , reqSat i y  $\rightarrow y \in \text{Rq } i$ )
  (hresp :  $\forall i$ ,  $\forall y \in G i$ , respSat i y)
  (h $\theta$  :  $p + \text{thr} \leq 1$ )
  (hDsub :  $D \subseteq \text{Dbar}$ )
  (hR3 :  $\forall y \in D$ ,  $y \notin U \rightarrow K y \text{ Dbar} = 1$ )
  (hR4 :  $\forall y \in D$ ,  $y \notin U \rightarrow K y (D \cup U) = 1$ )
  (hV0int :  $\forall y \in D$ ,  $y \notin U \rightarrow \text{Integrable } V_0 (K y)$ )
  (hV0drift :  $\forall y \in D$ ,  $y \notin U \rightarrow \int z, V_0 z \partial K y \leq V_0 y$ )
  (hV0floor :  $\forall y \in U$ ,  $y \in \text{Dbar} \rightarrow 1 \leq V_0 y$ )
  (hV0nonneg :  $\forall y \in \text{Dbar}$ ,  $0 \leq V_0 y$ )
  (h $\varepsilon$  :  $\forall i$ ,  $0 < \varepsilon i$ )
  (hW $\theta$  :  $\forall i$ ,  $\forall y \in \text{Dbar}$ ,  $0 \leq W i y$ ) (hWC :  $\forall i$ ,  $\forall y \in \text{Dbar}$ ,  $W i y \leq C i$ )
  (hWint :  $\forall i$ ,  $\forall y \in D$ ,  $y \notin U \rightarrow y \notin G i \rightarrow \text{Integrable } (W i) (K y)$ )
  (hWdrift :  $\forall i$ ,  $\forall y \in D$ ,  $y \notin U \rightarrow y \notin G i \rightarrow \int z, W i z \partial K y \leq W i y$ )
  (hWdriftR :  $\forall i$ ,  $\forall y \in D$ ,  $y \notin U \rightarrow y \in \text{Rq } i \rightarrow y \notin G i \rightarrow$ 
     $\int z, W i z \partial K y \leq W i y - \varepsilon i$ )
  (hx :  $x \in D$ ) (hthr :  $V_0 x \leq \text{thr}$ ) :
  ENNReal.ofReal p  $\leq \text{pathMeasure } K x$ 
  { $\omega \mid \forall i$ , ( $\exists' n \text{ in atTop}$ , reqSat i ( $\omega n$ ))  $\rightarrow \exists' n \text{ in atTop}$ , respSat i ( $\omega n$ )}
```

BernsteinLogicAux.NNFDuality.denot\_elimNeg

statement bound

```

/-- **Denotation preservation of dual-atom elimination**: where each dual atom
denotes the complement (`hd` – the per-duality definitional axioms, e.g. the
`compl_le_atom` family above), elimination preserves the region. This is the
lawfulness of consuming a `cmpl`-typed request through swapped atoms
(atom-level complement elimination's atom step, the T9b input shape). -/
theorem denot_elimNeg { $\sigma$  : Type*} (den :  $\alpha \rightarrow \text{Set } \sigma$ ) (d :  $\alpha \rightarrow \alpha$ )
  (hd :  $\forall a$ , den (d a) = (den a)c) :
   $\forall \phi$  : NForm  $\alpha$ , ( $\phi.\text{elimNeg } d$ ).denot den =  $\phi.\text{denot den}$ 
```

**The general-stochastic instance, carved requests discharged.** The complemented-request form at the segment-model tier: the carved variants below instantiate the bundle on the skeleton chain with the complement-carved request regions supplied and their side conditions discharged — plain and confined forms.

BernsteinLogicAux.LayerD.Streett.sound\_GSHS\_streett\_bundle\_carved\_discharged

statement bound

```

/-- **The full bundle datum, GSHS instance** (T9, carved scope, absorbing
certificates as segment-level rows) – the discrete-statement-shaped
corollary of the `_confined` form; matches
`QStreett.sound_QStreettBattery_carved_discharged` line for line. -/
theorem sound_GSHS_streett_bundle_carved_discharged
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV0 : Measurable V0)
  (hRq :  $\forall i$ , MeasurableSet (Rq i)) (hG :  $\forall i$ , MeasurableSet (G i))
  (hJ :  $\forall i$ , MeasurableSet (J i)) (hW :  $\forall i$ , Measurable (W i))
  (h $\theta$  :  $p + \text{thr} \leq 1$ ) (hDsub :  $D \subseteq \text{Dbar}$ )
  (hLive : NoVanish M (D \ U)) (hHalt : HaltsCaught M (D \ U) (U n Dbar))
  (hReset : ResetsInto M (D \ U) ((D \ U) n Dbar))
  (hSegD0 : RankingSegmentDynkin M (D \ U) V0  $\theta$ )
  (hForce0 : ForcedNonexpansion M V0) (hNonneg0 : NonnegV V0)
  (hC0 $\theta$  :  $0 \leq C_0$ ) (hV0C :  $\forall y$ ,  $V_0 y \leq C_0$ )
  (hFloor0 :  $\forall y \in U$ ,  $y \in \text{Dbar} \rightarrow 1 \leq V_0 y$ )
  (h $\varepsilon$  :  $\forall i$ ,  $0 < \varepsilon i$ ) (h $\gamma$  :  $\forall i$ ,  $0 < \gamma i$ )
  (hSegDW :  $\forall i$ , RankingSegmentDynkin M ((D \ U) \ (G i \cup J i)) (W i)  $\theta$ )
  (hSegDWR :  $\forall i$ ,
    RankingSegmentDynkin M (((D \ U) n Rq i) \ (G i \cup J i)) (W i) ( $\varepsilon i$ ))
  (hForceW :  $\forall i$ , ForcedNonexpansion M (W i)) (hNonnegW :  $\forall i$ , NonnegV (W i))
```

```

(hC0 : ∀ i, 0 ≤ C i) (hWC : ∀ i, ∀ y, W i y ≤ C i)
(hVa : ∀ i, Measurable (Va i)) (hNonnegVa : ∀ i, NonnegV (Va i))
(hCa0 : ∀ i, 0 ≤ Ca i) (hVaC : ∀ i, ∀ y, Va i y ≤ Ca i)
(hVaFloor : ∀ i, ∀ y ∈ Rq i, 1 ≤ Va i y)
(hSegDVA : ∀ i, RankingSegmentDynkin M ((Rq i)c) (Va i) 0)
(hForceVa : ∀ i, ForcedNonexpansion M (Va i))
(hVaJ : ∀ i, ∀ y ∈ J i, Va i y ≤ 1 - γ i)
(hx : x ∈ D) (hthr : V0 x ≤ thr) :
ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
{ω | ∀ i, (∃f n in atTop, ω n ∈ skelIn (Rq i)) →
  ∃f n in atTop, ω n ∈ skelIn (G i)}

```

BernsteinLogicAux.LayerD.Streett.sound\_GSHS\_streett\_bundle\_carved\_discharged\_confined

statement bound

```

/-- **The full bundle datum, GSHS instance, confined form** (T9 with the
absorbing certificates; assault doc §2.7 Theorem G proper, at the
hypothesis-carrying ranking scope `I \ (⋈i Ji)`: the ranking rows are
demanded only off the absorbed region `J i`, and in exchange each pair
carries an absorbing certificate `Va i` – nonnegative, bounded, `≥ 1` on
the request region, segment-nonincreasing off it, `≤ 1 - γ i` on `J i`.
The absorption bound is discharged on the skeleton by
`QStreett.absorptionBound_of_kernel` (the shifted supermartingale bridge +
conditional Ville – the mechanized replacement for prose-G2's epoch
argument); its off-request drift row is demanded at every alive state
(scope `(Rq i)c`), the global-sharp reading of the discrete discharge –
frozen states satisfy it for free. -/
theorem sound_GSHS_streett_bundle_carved_discharged_confined
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV0 : Measurable V0)
  (hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
  (hJ : ∀ i, MeasurableSet (J i)) (hW : ∀ i, Measurable (W i))
  (h0 : p + thr ≤ 1) (hDsub : D ⊆ Dbar)
  (hLive : NoVanish M (D \ U)) (hHalt : HaltsCaught M (D \ U) (U n Dbar))
  (hReset : ResetsInto M (D \ U) ((D ∪ U) n Dbar))
  (hSegD0 : RankingSegmentDynkin M (D \ U) V0 0)
  (hForce0 : ForcedNonexpansion M V0) (hNonneg0 : NonnegV V0)
  (hC0 : 0 ≤ C0) (hV0C : ∀ y, V0 y ≤ C0)
  (hFloor0 : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V0 y)
  (hε : ∀ i, 0 < ε i) (hγ : ∀ i, 0 < γ i)
  (hSegDW : ∀ i, RankingSegmentDynkin M ((D \ U) \ (G i ∪ J i)) (W i) 0)
  (hSegDWR : ∀ i,
    RankingSegmentDynkin M (((D \ U) n Rq i) \ (G i ∪ J i)) (W i) (ε i))
  (hForceW : ∀ i, ForcedNonexpansion M (W i)) (hNonnegW : ∀ i, NonnegV (W i))
  (hC0 : ∀ i, 0 ≤ C i) (hWC : ∀ i, ∀ y, W i y ≤ C i)
  (hVa : ∀ i, Measurable (Va i)) (hNonnegVa : ∀ i, NonnegV (Va i))
  (hCa0 : ∀ i, 0 ≤ Ca i) (hVaC : ∀ i, ∀ y, Va i y ≤ Ca i)
  (hVaFloor : ∀ i, ∀ y ∈ Rq i, 1 ≤ Va i y)
  (hSegDVA : ∀ i, RankingSegmentDynkin M ((Rq i)c) (Va i) 0)
  (hForceVa : ∀ i, ForcedNonexpansion M (Va i))
  (hVaJ : ∀ i, ∀ y ∈ J i, Va i y ≤ 1 - γ i)
  (hx : x ∈ D) (hthr : V0 x ≤ thr) :
ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
{ω | (∀ n, ω n ∈ skelIn D \ skelBad U) ∧
  ∀ i, (∃f n in atTop, ω n ∈ skelIn (Rq i)) →
  ∃f n in atTop, ω n ∈ skelIn (G i)}

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Three bound statements, three display constituents. The orientation-swap kernel — the flavor text's “exactly the sound direction, for free” —  $\mapsto \text{streett\_event\_mono}$ :  $Rq \sqsubseteq Rqhat$  (requests over-approximated) and  $Ghat \sqsubseteq G$  (responses under-approximated) transport the per-pair Streett implication pathwise; two `Frequently.monos`, the priced two lines, axiom-light and *deterministic* — no measure appears.

- the assembled rule  $\mapsto \text{sound\_QStreettBattery\_tie}$ : the full bundle kit (the battery-pattern binder block, as on the degenerate-request page) plus the two tie hypotheses, and the polarity pair is carried exactly —  $hreq : \forall i y, reqSat\ i\ y \rightarrow y \in Rq\ i$  (the certified region *over*-approximates the semantic request truth set: for `cmpl`-typed requests this is the atom-level complement-elimination inclusion  $\llbracket \text{cmpl } \varphi \rrbracket \supseteq \llbracket \varphi \rrbracket^c$ ) and  $hresp : \forall i, \forall y \in G\ i, respSat\ i\ y$  (responses *under*-approximated). The conclusion event quantifies over the semantic predicates `reqSat/respSat`, of which *no measurability is asked* (the measurable objects are the certified regions — the `QSafetyBattery sat` pattern), and both tie hypotheses are safe-direction widenings in the `QClosure` polarity sense.
- the atom-level complement step  $\mapsto \text{denot\_elimNeg}$  (the NNF module): where each dual atom denotes the complement, consuming a `cmpl`-typed request through swapped atoms preserves the region — the lawfulness clause cited at `QStreettRequestNnf`'s consumption boundary (its flavor text).

Width note: the conditional mark's one remaining residue, exactly as the roster carries it — the fine-topology content of the complement step at the general-stochastic-hybrid profile (nearly-Borel regions,  $E' \subseteq E$ ), which remains design-note prose with no bound counterpart; the bundle side's former continuous-time residue is discharged and hash-bound under the bundle entry. `streett_event_mono` is itself profile-free (pure path logic); the profile enters only with the assembled tie.

## Lean verifier receipts

module `BernsteinLogicAux.QStreett`; bound build 469881beb770... (bound from receipt 9d1d848e0ba1718a, 2026-07-29T01:03:31Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| e3990fe544d4708d | 2026-07-28T23:48:35Z | build+decorrelated                 | current |
| 68e9374fa1b418fe | 2026-07-29T00:00:34Z | build+decorrelated                 | current |
| e5895c9706854cd3 | 2026-07-29T00:49:04Z | build+replay+hardened+decorrelated | current |
| 9d1d848e0ba1718a | 2026-07-29T01:03:31Z | build+replay+hardened+decorrelated | current |
| 4a8d377b721b8523 | 2026-07-31T22:00:09Z | build+decorrelated                 | current |
| d251a122540ab741 | 2026-07-31T22:29:03Z | build+decorrelated                 | current |
| 333635b8e047113a | 2026-08-02T23:31:14Z | build+decorrelated                 | current |
| 9e96f80d63e4ad82 | 2026-08-03T16:27:18Z | build+decorrelated                 | current |
| 932b4b9e0b0fd323 | 2026-08-03T22:46:02Z | build+decorrelated                 | current |

9 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogicAux.NNFDuality`; bound build 97564f8e3950... (bound from receipt 21f04961748c0302, 2026-07-29T01:03:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 1417cdf27f69842a | 2026-07-28T23:48:35Z | build+decorrelated                 | current |
| 65fc1b51055d38bd | 2026-07-29T00:00:33Z | build+decorrelated                 | current |
| 2abbbc61b6065811 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| 21f04961748c0302 | 2026-07-29T01:03:33Z | build+replay+hardened+decorrelated | current |
| 8879c5b9721acf0c | 2026-07-31T22:00:03Z | build+decorrelated                 | current |
| e47a2c2d7f152711 | 2026-07-31T22:28:57Z | build+decorrelated                 | current |
| ce1bb97f7cbcaf4f | 2026-08-02T23:31:07Z | build+decorrelated                 | current |
| 8755b28f77990ba0 | 2026-08-03T16:27:11Z | build+decorrelated                 | current |
| 64948e60a56ae3bc | 2026-08-03T22:45:55Z | build+decorrelated                 | current |

8 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogicAux.LayerD.Streett`; bound build 7393065bbc7f... (bound from receipt abac471866cfcfd1, 2026-07-21T05:38:48Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to

reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0f3b50b4d3f0d33c | 2026-07-20T22:52:50Z | build+decorrelated                 | current |
| 109669f848453ec2 | 2026-07-21T00:03:19Z | build+decorrelated                 | current |
| 7e17aa32b276623b | 2026-07-21T04:51:49Z | build+decorrelated                 | current |
| abac471866cfcfd1 | 2026-07-21T05:38:48Z | build+replay+hardened+decorrelated | current |

6 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

Absent, precisely located: the rule's realization rides the typed request field of the Streett bundle — and the HOL4 development (bern\_catalogue, the base-calculus checker cone) carries nothing at the satisfaction stratum, so there is no HOL-side wire object for a checker to consume. A HOL statement becomes meaningful only after the extension stratum is ported; the weakening rows themselves are ordinary containment sequents the base checker already covers. Checked against the HOL sources this edition.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe (the general admissible template is stated in the PCONSTEXACT section; the batch-level honesty clause about absent payload functions is Recipe R50.1 of the QSTREETTDEGENERATE section and applies here verbatim).

**Recipe R51.1** (the QSTREETTCMPLREQUEST instance — one node, one bundle, per-pair weakening premises). A node concluding  $\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairCmpl}(\rho_1, \varphi_1), \dots))$  rewrites to the derivation fragment

$$\frac{\begin{array}{c} \text{the bundle kit at certified regions } (R_i, G_i) \\ \{ \llbracket \text{cmpl } \rho_i \rrbracket \subseteq R_i \}_i \text{ (request weakening)} \quad \{ G_i \subseteq \llbracket \varphi_i \rrbracket \}_i \text{ (response weakening)} \end{array}}{\mathcal{M}, x_0 \in \{V_0 \leq y\} \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairCmpl}(\rho_1, \varphi_1), \dots))} \text{QSTREETTBATTERY}$$

where the request-weakening premises are supplied for free by the orientation swap: a certified under-approximation  $A_i \subseteq \llbracket \rho_i \rrbracket$  yields the over-approximation  $A_i^c \supseteq \llbracket \rho_i \rrbracket^c$ , and  $R_i := A_i^c$  is a region atom by the atom dualities (QNNFREWRIE section) wherever  $A_i$ 's atom class carries a dual — the ordered-comparison classes, which are the shapes normalized request bodies produce; an over-approximation of any other shape is simply supplied as its own certified region — and the complement is consumed at the atom level, never as a formula node. The weakening premises are ordinary containment rows the checker already knows how to verify; nothing new *in kind* is recomputed.

**Proposition R51.2** (the direction lemma — two lines). *Fix a path  $\omega$  and one pair. If the certified regions satisfy  $\text{Rq} \subseteq \widehat{\text{Rq}}$  (requests over-approximated when read semantic-to-certified) and  $\widehat{G} \subseteq G$  (responses under-approximated), then*

$$\left[ (\exists^\infty n, \omega_n \in \widehat{\text{Rq}}) \Rightarrow (\exists^\infty n, \omega_n \in \widehat{G}) \right] \Rightarrow \left[ (\exists^\infty n, \omega_n \in \text{Rq}) \Rightarrow (\exists^\infty n, \omega_n \in G) \right].$$

*Proof.* Frequent membership is monotone in the set: frequent  $\text{Rq}$ -visits are frequent  $\widehat{\text{Rq}}$ -visits (hypothesis side, first monotonicity), and frequent  $\widehat{G}$ -visits are frequent  $G$ -visits (conclusion side, second monotonicity). Two applications of monotonicity — which is the entire lemma, and why the weakening premises are the only new obligations of the rewrite. (Machine-checked as `streett_event_mono`; hash-bound in this edition.)  $\square$

**Proposition R51.3** (semantic warrant of the rewrite). *Under the bundle kit's rows and the weakening premises of Recipe R51.1, the concluded judgment holds: with probability  $\geq p$ , every semantic pair implication  $(\exists^\infty \text{ states outside } \llbracket \rho_i \rrbracket) \Rightarrow (\exists^\infty \text{ states in } \llbracket \varphi_i \rrbracket)$  is satisfied.*

*Proof.* The bundle theorem gives the certified-region Streett event probability  $\geq p$ ; Proposition R51.2, applied per pair with  $R_q := \llbracket \text{cmpl } \rho_i \rrbracket \subseteq R_i$  and  $G_i \subseteq \llbracket \varphi_i \rrbracket$ , includes that event in the semantic one; the certified event is measurable (its regions are the kit's), and the path measure, read as outer measure on the possibly-nonmeasurable semantic event, is monotone — the lower bound transfers. (Machine-checked, assembled, as `sound_QStreettBattery_tie`; hash-bound in this edition.)  $\square$

*Remark R51.4* (why the orientations are not symmetric, and not optional). Swap either direction and the rule is unsound. If a certified request region *under*-approximated the complement, a path could request infinitely often in the semantic sense while the certificate sees only finitely many requests — the certified pair implication holds vacuously, the semantic one fails. Dually, a response region *over*-approximating  $\llbracket \varphi_i \rrbracket$  lets certified “responses” land outside the actual response set. The hypothesis-side / conclusion-side asymmetry of implication is the entire content of the direction check, and it is why the complement orientation swap — which produces exactly over-approximations — is fitted to requests and to nothing else.

*Remark R51.5* (profile residue). At the discrete-kernel profile the lemma chain above is complete, machine-checked, and hash-bound in this edition. At the continuous-time martingale-problem profile the bundle side is discharged whole per the bundle section's profile remark (confinement, zero-decrement, and strict-decrement rows; hash-bound there); what keeps this entry's conditional mark is the complement consumption's own fine-topology content (regular points of the complement region), which remains prose in the design record — a residue of this entry's own, not of the bundle's.

| proof step                                          | formal anchor                                                                                                                                     |
|-----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| Recipe R51.1 target<br>atom-level complement step   | the <code>QStreettBattery</code> section<br><code>denot_elimNeg</code> ( <code>BernsteinLogicAux.NNFDuality</code> ; machine-checked, hash-bound) |
| Proposition R51.2                                   | <code>streett_event_mono</code> ( <code>BernsteinLogicAux.QStreett</code> ; machine-checked, hash-bound)                                          |
| Proposition R51.3<br>no wire object in this edition | <code>sound_QStreettBattery_tie</code> (same module, hash-bound)<br>producers rewrite before emission (Recipe R51.1)                              |

## Rule 52. QSTREETTTARGETCELLCOLLAPSE (alias for QSTREETTBATTERY)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QStreettTargetCellCollapse*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\mathcal{K}, \text{slab} \vdash W_i \geq 0}{\mathcal{K}, \text{slab} \vdash g \circ W_i \leq W_i} g(\text{slab}) \subseteq T, W_i \equiv 0 \text{ ON } T \text{ (G5, RECOMPUTED)}$$

$\mathcal{K}$  ranges over QSTREETTBATTERY kits; the slab region left of the turnstile scopes both sequents.

a deduction rule between sequents: the kits demand the nonnegativity premise anyway, and under the side conditions (checker-recomputed, never a trusted annotation) the reset lands where the ranking certificate is identically zero, so the conclusion follows pointwise from the premise. On slabs whose resets leave the zero region the side condition fails and the full guard row is owed as usual.

QSTREETTTARGETCELLCOLLAPSE is a deduction rule between sequents of one QSTREETTBATTERY kit context (né StreettTargetCellCollapse): with  $\mathcal{K}$  the premise metavariable ranging over bundle kits, it derives the ranking-guard row  $g \circ W_i \leq W_i$  on a guard slab from the nonnegativity row  $W_i \geq 0$  there — a row the kit demands anyway — under two side conditions: the containment of the reset image in the zero-convention region ( $g(\text{slab}) \subseteq T$ , with  $W_i \equiv 0$  on  $T$ ). The rule is sound because under the side conditions the conclusion's left side is literally zero on the slab. Both side conditions are recomputed geometric checks, never trusted annotations — which is also why the rule cannot be abused: on a slab whose resets leave the zero region, the side condition fails and the full guard row is owed as usual.

Mechanization state, plainly: the bound statement below is the one-line witness `ranking_guard_collapse` (module `BernsteinLogicAux.QStreett`) — under the zero-on-target convention and the recomputed containment, the guard row follows from the nonneg row on the slab, which is the entirety of the rule's semantic claim (pure state-space set algebra: no kernel, no profile, no probability enters, so no profile residue attaches and the entry carries no conditional flag). The payload-level unpack owes no sort of its own — the extension certificate schema's adjudication of the batch family (QSTREETTDEGENERATE section, Recipe R50.1): the collapse is a citation (the guard row cites the nonneg row's proof), and the containment is a recomputed side condition of the battery walk, never a stored row — so there is nothing to elaborate, by design.

**notes** The checker's implementation is citation-shaped, deliberately decoupled from the deduction-rule presentation: no wire object ever carries a collapse node — a producer that has already discharged the nonneg row cites it where the guard row was owed and emits nothing new, one row to one row. Tier flag, carried honestly from the source catalogue: that one-to-one citation shape is the alias tiering, but the licensing side condition is a recomputed containment — if side-condition weight is ever ruled heavier than the alias criterion intends, the rule re-tiers admissible with machinery unchanged. Flagged, not forced.

### Lean statement

BernsteinLogicAux.QStreett . `ranking_guard_collapse`

statement bound

```
/-- **`QStreettTargetCellCollapse` (T9c)**: when the reset of a guard transition
lands inside a region where the ranking certificate is identically zero (the
`W_i ≡ 0`-on-target-cells convention, G5), the ranking-guard row
`W (g y) ≤ W y` collapses to the nonneg row `0 ≤ W y` — one row to one row,
under the recomputed containment side-condition `g '' slab ⊆ T`. -/
theorem ranking_guard_collapse {W : S → ℝ} {T slab : Set S} {g : S → S}
  (hzero : ∀ y ∈ T, W y = 0) (himg : ∀ y ∈ slab, g y ∈ T) :
  ∀ y ∈ slab, 0 ≤ W y → W (g y) ≤ W y
```

### Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

One bound statement, and the displayed deduction rule is its literal shape: `ranking_guard_collapse` concludes  $\forall y \in \text{slab}, \emptyset \leq W y \rightarrow W (g y) \leq W y$  — the premise sequent as the antecedent, the concluded ranking-guard row as the consequent, per point of the slab. The two licensing conditions map as:  $W_i \equiv 0$  on target cells  $\mapsto \text{hzero} : \forall y \in T, W y = \emptyset$  (the G5 convention as a hypothesis), and the recomputed containment  $g(\text{slab}) \subseteq T \mapsto \text{himg} : \forall y \in \text{slab}, g y \in T$  — consumed as a hypothesis, recomputed checker-side, which is the tier-ambiguity flag's exact locus rendered as a binder. Width note: the statement is one line over arbitrary sets and an arbitrary reset map  $g : S \rightarrow S$  — no slab geometry, no polynomiality of  $W$ , no guard structure — strictly wider than the displayed instance, safe direction; the failure branch the flavor text names (resets leaving the zero region owe the full guard row) is representable precisely as `himg` failing, whereupon the theorem offers nothing — refusal-shaped, as it should be.

## Lean verifier receipts

module BernsteinLogicAux.QStreett; bound build 469881beb770... (bound from receipt 9d1d848e0ba1718a, 2026-07-29T01:03:31Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| e3990fe544d4708d | 2026-07-28T23:48:35Z | build+decorrelated                 | current |
| 68e9374fa1b418fe | 2026-07-29T00:00:34Z | build+decorrelated                 | current |
| e5895c9706854cd3 | 2026-07-29T00:49:04Z | build+replay+hardened+decorrelated | current |
| 9d1d848e0ba1718a | 2026-07-29T01:03:31Z | build+replay+hardened+decorrelated | current |
| 4a8d377b721b8523 | 2026-07-31T22:00:09Z | build+decorrelated                 | current |
| d251a122540ab741 | 2026-07-31T22:29:03Z | build+decorrelated                 | current |
| 333635b8e047113a | 2026-08-02T23:31:14Z | build+decorrelated                 | current |
| 9e96f80d63e4ad82 | 2026-08-03T16:27:18Z | build+decorrelated                 | current |
| 932b4b9e0b0fd323 | 2026-08-03T22:46:02Z | build+decorrelated                 | current |

9 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

- build (lake build (elaboration)) current
- replay (leanchecker (official kernel replay)) current
- hardened (safe\_verify (hardened screen)) current
- decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*: the rule rewrites one premise row of a bundle kit into another *before* emission — no wire object ever contains a collapse node, and the surviving row is an ordinary base-calculus sequent the base checker already covers. The HOL statement that would matter is the bundle's own, absent for the reasons given in that section. Checked against the HOL sources: the `bern_catalogue` theory carries nothing Streett-shaped.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3); the batch's shared template, including the honesty clause about absent payload functions, is Recipe [R50.1](#) of the `QSTREETTDEGENERATE` section. This section's rule is stated in deduction-rule form — the respelling content is carried by the payload-transport recipe and the checker's citation-shaped implementation, noted there.

**Recipe R52.1** (the rule — a deduction between kit sequents). Fix a `QSTREETTBATTERY` kit context  $\mathcal{K}$ , a pair  $i$ , a guard transition with reset map  $g$  and guard slab  $\text{slab}$ , and let  $T$  be a region on which the kit's convention  $W_i \equiv 0$  holds (the target cells). The rule derives the ranking-guard row from the nonnegativity row the kit demands anyway — the kit's bound rows live on the hull  $\bar{D}$ , so



on guard slabs inside the hull the premise below is a row already owed, while a slab outside them simply owes its premise sequent separately; the deduction as displayed takes the on-slab sequent itself as premise either way:

$$\frac{\mathcal{K}, \text{slab} \vdash W_i \geq 0}{\mathcal{K}, \text{slab} \vdash g \circ W_i \leq W_i} \quad g(\text{slab}) \subseteq T, W_i \equiv 0 \text{ ON } T \text{ (G5, RECOMPUTED)}$$

Both side conditions are checker-recomputed (the G5-convention check), never producer-supplied. The premise metavariable does the honest work: premise and conclusion are sequents of the *same* kit context, so admitting the rule claims exactly that any kit admitting the premise admits the conclusion — Proposition R52.4, and nothing else.

**Recipe R52.2** (payload transport, catalogued-field level).

| collapse node                                | bundle kit row                                                                                                         |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| guard row $(W_i, g, \text{slab})$            | the kit's nonneg row for $W_i$ on slab, cited (licit where the slab lies in the cited row's region)                    |
| containment $g(\text{slab}) \subseteq T$     | recomputed side-condition (never producer-supplied)                                                                    |
| zero convention $W_i \equiv 0 \text{ on } T$ | the kit's target-cell convention, verbatim (that it holds on $T$ is the recomputed G5 check, not a trusted annotation) |

The checker's implementation is citation-shaped: a producer that has already discharged the nonnegativity row cites it where the guard row was owed, and emits nothing new — one row to one row, which is the alias tiering. Presentation and implementation are deliberately decoupled: the rule *is* the deduction above; the citation is how a certificate spells an application of it. Unpack-function slot: nothing to elaborate, by design — the collapse is a citation (the guard row cites the nonneg row's proof) and the containment is a recomputed side condition, never a stored row; the semantic witness `ranking_guard_collapse` (auxiliary module) is hash-bound in this edition.

**Proposition R52.3** (the exactness identity — one line). *For every  $y \in \text{slab}$ : if  $g(y) \in T$  and  $W_i \equiv 0 \text{ on } T$  and  $0 \leq W_i(y)$ , then  $W_i(g(y)) = 0 \leq W_i(y)$ .*

*Proof.* Substitution — the displayed chain is the proof. The guard row's left side is identically zero on the slab, so the guard row and the nonneg row assert the same inequality there.  $\square$

**Proposition R52.4** (soundness corollary — admissibility of the rule). *Any bundle kit admitting the nonneg row on the slab admits the ranking-guard row on the slab — the displayed rule, read as a claim; hence every conclusion drawn through `QSTREETTBATTERY` from the collapsed kit is drawn from the full kit, with the bundle's own soundness theorem as warrant.*

*Proof.* Proposition R52.3 produces the guard row pointwise from the nonneg row under the side conditions — that collapse is the machine-checked content (`ranking_guard_collapse`; hash-bound in this edition), and it is the whole of what this section proves. The guard row is consumed *upstream* of the flattened kernel: for a hybrid presentation it is one of the base-calculus sequents from which the kernel-integral ranking rows — the bundle kit's H4, which the bundle theorem is what actually consumes — are assembled by the presentation-to-kernel compile bridge (the safety battery's staged form carries that bridge as its explicit `hmix` hypothesis; the bundle theorem takes the compiled kernel and its H4 rows directly), never re-proven here. The corollary therefore lives at the kit stratum: the collapsed kit discharges every base-calculus obligation the full kit's guard row was owed for, and every conclusion drawn through `QSTREETTBATTERY` whose compile bridge owed that guard row is drawn from the full kit, the bundle's own soundness theorem — its assembly hypothesis unchanged — supplying the rest.  $\square$

*Remark R52.5* (why the side-condition is not decoration). Drop the containment and the collapse is unsound: a reset landing at a state with  $W_i(g(y)) > W_i(y)$  (outside the zero region) breaks the upstream derivation of the ranking generator rows that the bundle's telescoping lemma consumes — the certificate could be pumped back up at each guard crossing, and the summability of un-responded requests (`QSTREETTBATTERY` section) fails. The containment is checker-recomputed for precisely this reason; its weight is the tier flag recorded in the commentary.

| proof step                                 | formal anchor                                                                                                |
|--------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| Recipe R52.1 premise context $\mathcal{K}$ | the <code>QSTREETTBATTERY</code> section's kit (H4 rows)                                                     |
| Proposition R52.3                          | <code>ranking_guard_collapse</code> ( <code>BernsteinLogicAux.QStreett</code> ; machine-checked, hash-bound) |
| side conditions                            | the zero-on-target containment, recomputed (the bundle's G5-convention check)                                |

## Rule 53. QN<sub>NF</sub>REWRITE (alias rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTS<sub>L</sub>-rules.toml — *QN<sub>NF</sub>Rewrite*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\begin{array}{llll} \overline{[\neg\neg\varphi]} = [\varphi] & \overline{[\neg(\varphi \wedge \psi)]} = [\neg\varphi \vee \neg\psi] & \overline{[\neg(\varphi \vee \psi)]} = [\neg\varphi \wedge \neg\psi] & \overline{[\neg\{f \leq c\}]} = \{f > c\} \\ \overline{[\neg\{f < c\}]} = \{f \geq c\} & \overline{[\neg\{f \geq c\}]} = \{f < c\} & \overline{[\neg\{f > c\}]} = \{f \leq c\} \end{array}$$

seven axioms — deduction rules with nothing above the line — at the spec surface, each concluding its region-denotation equality; a satisfaction judgment rewrites across any of them in either direction with certified region and rows transported verbatim, and no negation node survives to the baked representation (surviving negative literals are the typing rejection's business, not a rule's).

QN<sub>NF</sub>REWRITE is the spec-surface negation calculus (né NNFDuality): double negation, De Morgan through  $\wedge/\vee$ , and the four atom/strict swaps  $\neg\{f \leq c\} \leftrightarrow \{f > c\}$ ,  $\neg\{f < c\} \leftrightarrow \{f \geq c\}$  — both orientations of both comparison directions. The seven dualities are *axioms*: deduction rules with nothing above the line, each concluding its region-denotation equality, stated once per duality. Each licenses a judgment-level  $1 \rightarrow 1$  rewrite of a formula inside a satisfaction judgment: the rewritten formula denotes the same set of states, so any region under-approximating one denotation under-approximates the other, and the judgment transports unchanged. A producer uses the dualities to push negations down to atoms (the negation-normal-form pass of the QSTREETTREQUESTNNF section) so that every negative position is consumed either by an atom swap or by the typing pass's rejection — never by a negation node in the certificate. The checker recomputes nothing: the dualities are definitional set identities over the state space, and the strict-inequality sides interact with the closure discipline (rounding of strict literals at positive positions) only through the separate closure rules, deliberately out of this rule's scope.

There is no single target rule, and hence no *alias-for* header here: each axiom stands on its own definitional equality. Mechanization state, plainly: the bound statements below are the four atom swaps, machine-checked as set-level denotation equalities in both orientations of both comparison directions (module BernsteinLogicAux.NNFDuality: `compl_le_atom`, `compl_lt_atom`, `compl_ge_atom`, `compl_gt_atom`), together with the normalization pass's denotation-preservation theorem `denot_nnfAux`, inside which double negation and De Morgan are consumed from the ambient set library — each applied once per duality, so the displayed equalities are covered jointly by the swaps (atoms) and the pass theorem (connectives). These are definitional set identities over the state space, warranted in full; the entry carries no conditional flag.

*notes* Both orientations of both directions are stated deliberately: an early one-sided draft of the atom swaps was caught in review, and the repaired statement is the four-lemma family. The strict-atom sides are stated here as *denotation* facts only; whether a strict literal may appear in a concluded region is the closure discipline's question, answered by polarity-directed rounding in its own catalogue rows.

### Lean statement

BernsteinLogicAux.NNFDuality.`compl_le_atom`

statement bound

```
-- T12 atom swap: `¬{f ≤ c} ↔ {f > c}`. -/
theorem compl_le_atom : {x | f x ≤ c}ᶜ = {x | c < f x}
```

BernsteinLogicAux.NNFDuality.`compl_lt_atom`

statement bound

```
-- T12 atom swap: `¬{f < c} ↔ {f ≥ c}`. -/
theorem compl_lt_atom : {x | f x < c}ᶜ = {x | c ≤ f x}
```

BernsteinLogicAux.NNFDuality.`compl_ge_atom`

statement bound

```
-- T12 atom swap, the other orientation: `¬{f ≥ c} ↔ {f < c}`. -/
theorem compl_ge_atom : {x | c ≤ f x}ᶜ = {x | f x < c}
```

BernsteinLogicAux.NNFDuality.`compl_gt_atom`

statement bound

```
-- T12 atom swap, the other orientation: `¬{f > c} ↔ {f ≤ c}`. -/
theorem compl_gt_atom : {x | c < f x}ᶜ = {x | f x ≤ c}
```

BernsteinLogicAux.NNFDuality.`denot_nnfAux`

statement bound

```

/-- **Denotation preservation of the NNF pass, both polarities** – the T12
dualities (double negation `compl_compl`, De Morgan `Set.compl_inter`/
`Set.compl_union`) are exactly the cases of this induction; each is applied
once, which is the catalogue's "stated once as a definitional axiom per
duality" realized as proof structure. -/
theorem denot_nnfAux : ∀ (φ : Form α) (b : Bool),
  (φ.nnfAux b).denot den = if b then φ.denot den else (φ.denot den)ᶜ

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed atom/strict swaps map one-to-one onto the four bound equalities, each a region-denotation identity:  $\llbracket \neg\{f \leq c\} \rrbracket = \{f > c\} \mapsto \text{compl\_le\_atom} : \{x \mid f\ x \leq c\}^c = \{x \mid c < f\ x\}$ , and likewise  $\text{compl\_lt\_atom} (\neg\{f < c\} = \{f \geq c\})$ ,  $\text{compl\_ge\_atom}$ ,  $\text{compl\_gt\_atom}$  (the reversed orientations — both directions bound separately, honouring the recorded one-sidedness catch). Binder note, wider-safe:  $f : \sigma \rightarrow \mathbb{R}$  is an arbitrary function on an arbitrary carrier — the polynomial atom structure of the certificate format is erased, so the certified equalities cover every instance the rule will ever be applied to. The propositional dualities ( $\neg\neg\varphi$ , De Morgan both ways)  $\mapsto$  the neg/and/or cases of  $\text{denot\_nnfAux}$ 's both-polarity induction, where  $\text{compl\_compl}$  /  $\text{Set.compl\_inter}$  /  $\text{Set.compl\_union}$  are each consumed from `mathlib` once — the entry's “stated once per duality” realized as proof structure rather than as four more standalone statements. Width note: the entry is unconditional — set-level witnesses cover the catalogued claim in full (axioms: `propext` alone) — and the grammar-level guarantee the flavor text mentions (no negation node survives to the baked representation) is certified not here but by the `QStreettRequestNnf` entry's normal-form type; this page's statements are the denotation half only, as its note already rules.

## Lean verifier receipts

module `BernsteinLogicAux.NNFDuality`; bound build 97564f8e3950... (bound from receipt 21f04961748c0302, 2026-07-29T01:03:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 1417cdf27f69842a | 2026-07-28T23:48:35Z | build+decorrelated                 | current |
| 65fc1b51055d38bd | 2026-07-29T00:00:33Z | build+decorrelated                 | current |
| 2abbb61b6065811  | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| 21f04961748c0302 | 2026-07-29T01:03:33Z | build+replay+hardened+decorrelated | current |
| 8879c5b9721acf0c | 2026-07-31T22:00:03Z | build+decorrelated                 | current |
| e47a2c2d7f152711 | 2026-07-31T22:28:57Z | build+decorrelated                 | current |
| ce1bb97f7cbcaf4f | 2026-08-02T23:31:07Z | build+decorrelated                 | current |
| 8755b28f77990ba0 | 2026-08-03T16:27:11Z | build+decorrelated                 | current |
| 64948e60a56ae3bc | 2026-08-03T22:45:55Z | build+decorrelated                 | current |

8 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*: the dualities are producer-side formula rewrites that unpack to definitional denotation equalities before any certificate is emitted — no wire object carries a negation node (the typing discipline rejects surviving negative literals), so there is nothing for a wire checker, HOL or otherwise, to verify. Checked against the HOL sources: the `bern_catalogue` theory is the base-calculus checker cone; formula-level negation of the extension theory's spec surface does not appear in it.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: axioms at the spec surface (roster tier 3). The seven dualities are deduction rules with nothing above the line, each concluding a definitional region-denotation equality — there is no single target rule being respelled, and no premise to carry: the equality is the entire content. The payload-transport honesty clause of the batch is Recipe [R50.1](#) of the `QSTREETTDEGENERATE` section, and applies here vacuously (no payload ever exists to transport — the dualities are applied before emission).

**Recipe R53.1** (the seven axioms). Over a state space  $S$ , with  $\llbracket \cdot \rrbracket$  the region denotation of spec-surface formulas and  $f : S \rightarrow \mathbb{R}$  ranging over atom left-hand sides:

$$\begin{array}{llll} \overline{\llbracket \neg \neg \varphi \rrbracket} = \overline{\llbracket \varphi \rrbracket} & \overline{\llbracket \neg(\varphi \wedge \psi) \rrbracket} = \overline{\llbracket \neg \varphi \vee \neg \psi \rrbracket} & \overline{\llbracket \neg(\varphi \vee \psi) \rrbracket} = \overline{\llbracket \neg \varphi \wedge \neg \psi \rrbracket} & \overline{\llbracket \neg\{f \leq c\} \rrbracket} = \overline{\{f > c\}} \\ \overline{\llbracket \neg\{f < c\} \rrbracket} = \overline{\{f \geq c\}} & \overline{\llbracket \neg\{f \geq c\} \rrbracket} = \overline{\{f < c\}} & \overline{\llbracket \neg\{f > c\} \rrbracket} = \overline{\{f \leq c\}} \end{array}$$

Each axiom licenses a judgment-level  $1 \rightarrow 1$  rewrite in either direction: a satisfaction judgment whose formula contains one side concludes identically with the other side substituted, the certified region unchanged.

|                                          | rewritten node                                                                             | image                                                |
|------------------------------------------|--------------------------------------------------------------------------------------------|------------------------------------------------------|
| <b>Recipe R53.2</b> (payload transport). | formula (one duality applied at one position)<br>certified region, judgment kind, all rows | formula (the dual form at that position)<br>verbatim |

Unpack-function slot: no verified payload function in this edition (batch template) — and here the slot is moot by design: the typing discipline rejects surviving negative literals, so the dualities exist to be applied *before* any payload exists.

**Proposition R53.3** (soundness of the axioms — two lines each). *All seven concluded equalities hold.*

*Proof.* The propositional three are complement involution and De Morgan for sets. The atom four: pointwise over  $S$ , the negation of  $f(y) \leq c$  is  $c < f(y)$  and the negation of  $f(y) < c$  is  $c \leq f(y)$  — the total order on  $\mathbb{R}$  — and the remaining two are the same facts read in the other orientation. (Machine-checked: the four atom swaps as set-level equalities, and the propositional three consumed from the ambient set library inside the normalization pass's denotation theorem; hash-bound in this edition.)  $\square$

**Proposition R53.4** (soundness corollary). *If  $R \subseteq \llbracket \varphi \rrbracket$  and  $\varphi'$  arises from  $\varphi$  by one duality, then  $R \subseteq \llbracket \varphi' \rrbracket$ ; hence every satisfaction judgment transports across a duality unchanged, in both directions.*

*Proof.* Denotations are equal (Proposition [R53.3](#)), and substitution preserves that: region denotation is compositional — each connective's denotation is a fixed set operation on its children's — so by induction on the surrounding formula, replacing a subformula by one of equal denotation leaves every enclosing denotation unchanged. The machine-checked cousin of this one-position congruence is the normalization pass's denotation-preservation theorem, which pushes negation through *all* eligible positions in one traversal. The pass consumes the propositional facts — the complement of a conjunction or disjunction, once per constructor case — and treats atoms opaquely: a negatively-occurring atom survives as a negated literal whose denotation is the complement, and the four atom dualities are separate bound facts that swap that complement for its closed-form partner. That division realizes what the display means by “stated once per duality”: each equality is one definitional fact, cited where it is used, never re-derived.  $\square$

*Remark R53.5* (what this rule deliberately does not say). The strict-atom swaps are *denotation* facts; whether a strict inequality may appear in a concluded region atom is a different question, answered by the closure discipline (polarity-directed rounding) in its own catalogue rows. And the probability operator is absent from the display on purpose: a negated probability bound at even polarity is not rewritten but *rejected* by the typing pass — the polarity theorem of the `QSTREETTREQUESTNNF` section is what makes that rejection a surface-level check.

proof step

formal anchor

atom swaps, both orientations

Proposition **R53.4** induction

no wire object, by design

`compl_le_atom, compl_lt_atom, compl_ge_atom, compl_gt_atom`  
 (BernsteinLogicAux.NNFDuality; machine-checked, hash-bound)  
`denot_nnfAux` (same module, hash-bound; the whole-pass form — the one-  
 position congruence is the compositionality induction in the proof)  
 negation nodes are rejected by typing; producers rewrite before emission

## Rule 54. QSTREETTREQUESTNNF (alias for QSTREETTBATTERY)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QStreettRequestNnf*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PDwellStreett}(\dots, \text{PairCmpl}(\text{elimNeg}(\text{nnf}(\rho_i)), \varphi_i), \dots))}{\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PDwellStreett}(\dots, \text{PairCmpl}(\rho_i, \varphi_i), \dots))} \quad \overline{\llbracket \text{elimNeg}(\text{nnf}(\rho)) \rrbracket = \llbracket \rho \rrbracket}$$

per pair  $(\rho_i, \varphi_i)$ , applied at each cmpl-typed pair while base-typed requests pass unchanged: the request *body* is rewritten to literal normal form by the *QNNfRewrite* dualities — negation-freedom of the output holds *by construction* of the normal-form grammar (no negation node exists to build) — with the cmpl tag **RETAINED**: the complement applies at the consumption boundary (*QStreettCmplRequest*), never as a literal. The displayed axiom is the warrant: complements of equal denotations are equal, so the retained tag consumes identically. Polarity theorem: a surviving negative literal is exactly an odd-polarity atom occurrence of the input — so the neg-P rejection is a surface polarity check, stable under normalization.

QSTREETTREQUESTNNF normalizes each Streett pair into the bundle's expected input shape (né *StreettRequestNNF*): base-typed requests pass through unchanged, and a complement-typed request has its formula rewritten to negation normal form — negations pushed to the atoms by the *QNNfRewrite* rewrites — so that the complement is applied at the *consumption boundary* (the atom swaps, or the complement-request rule), never carried as a formula literal. It is an alias: the rewrite is a  $1 \rightarrow 1$  respelling of one pair inside one *QSTREETTBATTERY* node, and its warrant is denotation preservation of the normalization pass — the region the pair constrains is unchanged, so the judgment transports verbatim.

The load-bearing extra is the *polarity theorem*: the surviving negative literals of the normal form are exactly the odd-polarity atom occurrences of the input formula. Its consequence is that the typing pass's rejection of negated probability literals — which guards against smuggling a probability *upper* bound through an even number of negations — is equivalent to a surface-level polarity check on the un-normalized formula: normalization can never manufacture a rejected literal that was not already there at odd polarity, and never loses one. What would break without it: a normalization pass without the polarity invariant could present a well-typed surface whose normal form hides a rejected literal, and the consumption boundary would silently consume a bound in the wrong direction.

Mechanization state, plainly: the pass and both theorems are machine-checked structurally, and the bound statements below are their statements (module *BernsteinLogicAux.NNFDuality*): the normalizer *Form.nnf* lands in a literal normal form whose *type* has no negation node — negation-freedom by construction, the strongest form of the well-formedness claim — with denotation preservation *denot\_nnf* (the displayed  $\llbracket \text{nnf}(\rho_i) \rrbracket = \llbracket \rho_i \rrbracket$ ), the polarity theorem *negLits\_nnf* (surviving negative literals = odd-polarity atom occurrences) and its rejection face *nnf\_no\_negLit\_iff*, and the consumption-boundary elimination *denot\_elimNeg*. The rule's scope is the propositional skeleton, deliberately — the path-operator nodes of the spec surface are handled by their own ruled aliases and typing rejections, never by this pass — and the witnesses cover that scope in full, so the entry carries no conditional flag.

**notes** The pass is a reference implementation on the producer side; nothing checker-facing depends on running it. What the certificate format sees is only the normalized result, and the typing pass independently rejects any surviving negative probability literal — so a defective producer normalization can produce rejections, never unsoundness.

### Lean statement

*BernsteinLogicAux.NNFDuality*. **denot\_nnf**

statement bound

```
-- Denotation preservation of `Form.nnf` (T22's soundness face). -/
theorem denot_nnf (φ : Form α) : (φ.nnf).denot den = φ.denot den
```

*BernsteinLogicAux.NNFDuality*. **negLits\_nnf**

statement bound

```
-- **The T22 polarity theorem** ("a surviving neg-P literal is ALWAYS the
-- rejected even-polarity form", catalogue T22 – the wf theorem): the surviving
-- negative literals of the normal form are exactly the **odd-polarity** atom
-- occurrences of the input. Consequently the typing pass's neg-literal rejection
-- (`qtsl_typing/v1`, catalogue R6) is equivalent to a surface-level polarity
-- check on the un-normalized formula. -/
theorem negLits_nnf (φ : Form α) : (φ.nnf).negLits = φ.occs false
```

*BernsteinLogicAux.NNFDuality*. **nnf\_no\_negLit\_iff**

statement bound

```

/-- The rejection face of the polarity theorem: the normal form is free of
`P`-flagged negative literals iff no `P`-atom occurs at odd polarity. -/
theorem nnf_no_negLit_iff (φ : Form α) (P : α → Prop) :
  (∀ a ∈ (φ.nnf).negLits, ¬ P a) ↔ ∀ a ∈ φ.occs false, ¬ P a

```

BernsteinLogicAux.NNFDuality.denot\_elimNeg

statement bound

```

/-- **Denotation preservation of dual-atom elimination**: where each dual atom
denotes the complement (`hd` – the per-duality definitional axioms, e.g. the
`compl_le_atom` family above), elimination preserves the region. This is the
lawfulness of consuming a `cmpl`-typed request through swapped atoms
(atom-level complement elimination's atom step, the T9b input shape). -/
theorem denot_elimNeg {σ : Type*} (den : α → Set σ) (d : α → α)
  (hd : ∀ a, den (d a) = (den a)c) :
  ∀ φ : NForm α, (φ.elimNeg d).denot den = φ.denot den

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The flavor text's “negation-freedom holds by construction of the normal-form grammar” is certified *by type*: the target grammar `NForm` (positive/negative literals under  $\wedge/\vee$ ) has no negation constructor, so no theorem about negation-freedom exists to state — the inductive definition is the certificate. The displayed constituents — the judgment-transport rule and its warranting axiom — map onto the four bound statements:

- $\llbracket \text{elimNeg}(\text{nnf}(\rho_i)) \rrbracket = \llbracket \rho_i \rrbracket \mapsto$  the composition of `denot_nnf` ( $(\phi.\text{nnf}).\text{denot den} = \phi.\text{denot den}$  — the normalizer preserves the region, `QNnfRewrite`'s dualities consumed as the induction) with `denot_elimNeg`, whose hypothesis `hd : ∀ a, den (d a) = (den a)c` is the per-duality definitional-axiom pack: elimination is lawful exactly where a dual atom exists, which is the flavor text's “the complement applies at the consumption boundary, never as a literal” — the retained `cmpl` tag consumes at `QStreettCmplRequest` through this same lemma, and dropping the tag would change the pair's truth value (a one-state counterexample witnesses it), which is why the rewrite retains it.
- the polarity theorem  $\mapsto$  `negLits_nnf : (φ.nnf).negLits = φ.occs false` — surviving negative literals are *exactly* the odd-polarity atom occurrences — with the rejection face `nnf_no_negLit_iff` certifying the flavor text's “the neg-P rejection is a surface polarity check, stable under normalization.”

Width note: atoms and denotations are fully abstract  $(\alpha, \text{den} : \alpha \rightarrow \text{Set } \sigma)$  — the per-pair Streett context, the base-typed-request pass-through, and the probability flavour of the neg-P check are all consumers of these statements, not content of them; unconditional, `propext` alone, set-level witnesses covering the catalogued claim in full.

## Lean verifier receipts

module BernsteinLogicAux.NNFDuality; bound build 97564f8e3950... (bound from receipt 21f04961748c0302, 2026-07-29T01:03:33Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 1417cdf27f69842a | 2026-07-28T23:48:35Z | build+decorrelated                 | current |
| 65fc1b51055d38bd | 2026-07-29T00:00:33Z | build+decorrelated                 | current |
| 2abbbc61b6065811 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| 21f04961748c0302 | 2026-07-29T01:03:33Z | build+replay+hardened+decorrelated | current |
| 8879c5b9721acf0c | 2026-07-31T22:00:03Z | build+decorrelated                 | current |
| e47a2c2d7f152711 | 2026-07-31T22:28:57Z | build+decorrelated                 | current |
| ce1bb97f7cbcaf4f | 2026-08-02T23:31:07Z | build+decorrelated                 | current |
| 8755b28f77990ba0 | 2026-08-03T16:27:11Z | build+decorrelated                 | current |
| 64948e60a56ae3bc | 2026-08-03T22:45:55Z | build+decorrelated                 | current |

8 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

gate-layer coverage of the bound build:



build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*: the normalization is a producer-side rewrite completed before emission — certificates carry only its output, and the output's well-formedness is enforced by the typing pass, not by a checker replay of the rewrite. No wire object exists for a HOL statement to be about. Checked against the HOL sources: the `bern_catalogue` theory carries no spec-surface formula grammar.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template; the batch's shared template is Recipe [R50.1](#) of the `QSTRETTDEGENERATE` section.

**Recipe R54.1** (the filled-in application — one pair, normalized). Fix a Streett pair  $(\rho, \varphi)$  inside a `QSTRETTBATTERY` node, with  $\rho$  complement-typed. Write `nnf` for the negation-normal-form pass on the propositional skeleton (atoms,  $\neg$ ,  $\wedge$ ,  $\vee$ ): negations pushed inward by the `QNNFREWITE` rewrites until only literals remain. The rewrite is

$$(\text{cmpl } \rho, \varphi) \rightsquigarrow (\text{cmpl } (\text{elimNeg}(\text{nnf}(\rho))), \varphi)$$

— the body normalized, the complement tag *retained*: per the source discipline the complement applies at the consumption boundary (the complement-request rule), never as a literal. After the rewrite every literal of the request body is positive (atom-swapped where it was negative), and the pair is in exactly the input shape the bundle and the complement-request rule expect. A base-typed pair is already in that shape: on it the rewrite is the identity, the transport trivial, and the exactness identity reflexivity — stated once here so the corollary's case split below is complete. One pair to one pair, inside one node: the alias shape.

| Recipe R54.2 (payload transport).       |                                                                                                |
|-----------------------------------------|------------------------------------------------------------------------------------------------|
| un-normalized pair                      | normalized pair                                                                                |
| request body $\rho$ (cmpl tag retained) | <code>elimNeg(nnf(<math>\rho</math>))</code> , negative literals eliminated through dual atoms |
| response formula, certificates, rows    | verbatim                                                                                       |
| polarity bookkeeping                    | the polarity theorem (Proposition <a href="#">R54.4</a> ); nothing stored                      |

Unpack-function slot: no verified payload function in this edition (batch template); the structural mechanization exists (the normalizer and its theorems, named below; the four theorems hash-bound in this edition).

**Proposition R54.3** (the exactness identity — denotation preservation).  $\llbracket \text{nnf}(\rho) \rrbracket = \llbracket \rho \rrbracket$ , and eliminating negative literals through dual atoms preserves the denotation.

*Proof.* Induction on  $\rho$ , with the `QNNFREWITE` equalities as the cases — double negation and De Morgan drive the recursion, the atom swaps close the leaves. The elimination step substitutes, for each negative literal, the dual atom denoting the complement; equality is preserved literal by literal. (Machine-checked: `denot_nnf` and `denot_elimNeg`, hash-bound in this edition; the eliminated form has no negative literals at all, `negLits_elimNeg` — an unpinned auxiliary of the same module.) Complements of equal sets are equal, so the retained `cmpl` tag's consumption-boundary denotation  $S \setminus \llbracket \rho \rrbracket$  is preserved as well — the rewrite itself never discharges the complement.  $\square$

**Proposition R54.4** (the polarity theorem). *The surviving negative literals of  $\text{nnf}(\rho)$  are exactly the atom occurrences of  $\rho$  at odd polarity (an odd number of enclosing negations). Consequently the normal form is free of negated occurrences of a given atom class if and only if the input has no odd-polarity occurrence of that class.*

*Proof.* Strengthen to the polarity-indexed pass (normalizing  $\rho$  or  $\neg\rho$  according to a polarity bit) and induct: at an atom the bit decides the literal's sign; at a negation the bit flips; at  $\wedge$  and  $\vee$  the bit distributes (De Morgan swapping the connective when negative). The negative-literal set on one side and the odd-polarity occurrence set on the other satisfy the same recursion. The rejection face is a rewrite of the same equality. (Machine-checked: `negLits_nnf`, via the unpinned auxiliary `negLits_nnfAux`, and `nnf_no_negLit_iff`; both theorems hash-bound in this edition. Terminology, once: a *negative literal* is a signed-literal constructor

of the normal-form grammar, not an occurrence of the formula negation node — the grammar carries literals of both signs and no negation constructor, which is the sense in which negation-freedom of the output holds by *type*; the elimination pass then removes the negative-signed literals through dual atoms.)  $\square$

**Proposition R54.5** (soundness corollary). *A bundle node whose pair is normalized concludes identically to the un-normalized node: through QSTRETTBATTERY directly for base-typed requests, and through the QSTRETTCMPLREQUEST weakening for the consumed complement — with Proposition R54.3 guaranteeing both nodes constrain the same regions.*

*Proof.* Immediate from Proposition R54.3 and the target rules' theorems, each at its own proven scope — the rewrite itself is propositional and profile-free; whatever conditional mark the concluding bundle or weakening node carries is inherited there, not here; the alias adds no semantic content of its own.  $\square$

*Remark R54.6* (what the polarity theorem is protecting). The typing discipline rejects probability-atom occurrences at odd polarity — an odd number of enclosing negations, whether worn as a bare negated literal in even context or as a positive literal under an odd stack — because a negated lower bound is an upper bound, and the certificate calculus deliberately has no rule concluding probability upper bounds through this pass. The theorem makes the rejection *stable under normalization*: no sequence of dualities can manufacture or hide a rejected literal, so the check may run on the surface formula once, before normalization, with the same verdict. A guard that had to chase literals through rewrites would be a moving target; a polarity count is not.

| proof step                           | formal anchor                                                                                                           |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| normalizer, negation-freedom by type | Form.nnf, NForm (BernsteinLogicAux.NNFDuality; definitions — no statement pins, their theorems below carry the warrant) |
| Proposition R54.3                    | denot_nnf, denot_elimNeg (same module, hash-bound); negLits_elimNeg (unpinned auxiliary)                                |
| Proposition R54.4                    | negLits_nnf, nnf_no_negLit_iff (same module, hash-bound)                                                                |
| Proposition R54.5 targets            | the QSTRETTBATTERY and QSTRETTCMPLREQUEST sections                                                                      |

## Rule 55. QAVOID (alias rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — `QAvoid`. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PAlways}(\text{SNot } \beta))}{\mathcal{M}, x_0 \in R \models \text{P}_{\geq p}[\text{avoid } \beta]}$$

$\text{avoid } \beta := G\neg\beta$  — the unfolding is the rule (the premise is the target judgment at the expansion), so the alias is sound by definition: the equation is definitional at the set level (`rfl`-grade), and the negation atom is ruled in by the `QNNFREWRITE` swaps. The surface constructor builds its expansion rather than a new node kind — witnessed by the producer-side concrete-syntax canonicity battery; no rule sort exists and none is needed.

`QAVOID` is the avoidance sugar of the spec surface:  $\text{avoid } \beta := G\neg\beta$  — avoidance is safety of the negation, exact, the negation being ruled in by the `QNNFREWRITE` atom swaps. An alias unpacking to *one* application of an existing rule (the safety battery at the avoid set), and the strongest-fidelity member of the sugar trio (its siblings: `QSTREETTGF`, `QSTREETTFG`): the identity the definition rides is machine-checked as a path-event identity in the target rule's concluded shape (module `BernsteinLogicAux.QtslAliases`), `sugarAvoid_denot`, proof `rfl`, axiom-free — definitional exactness, certified as such by the kernel. A producer, and a reader of specifications, may use the sugar constructor freely: it builds its expansion rather than a new node kind, so nothing downstream — typing, compilation, checking — ever sees a sugar node, and no checker recomputation exists for this rule at all. There is no soundness lemma here because the equation is a definition; the witness that the surface really does build the expansion remains the reference implementation's concrete-syntax canonicity test battery — an implementation property, not a semantic claim, so it says a test rather than a theorem. No profile enters; the entry carries no conditional flag.

**notes** The avoid sugar is also the compatibility layer to plain safety specification: avoidance properties state directly against the safety kernel with no automaton machinery, which is the degenerate (and most common) case of the classical-surface translation the trio provides.

### Lean statement

`BernsteinLogicAux.QtslAliases.sugarAvoid_denot`

statement bound

```
/-- **`SugarBatch` (T10), first sugar**: `avoid β := G ¬β` — avoidance is
safety of the negation, definitionally (the proof is `rfl`; the negation is
ruled in by the `NNFDuality.lean` atom swaps (rule `QnnfRewrite`)). Exact: no direction is lost. -/
theorem sugarAvoid_denot {S : Type*} (B : Set S) :
  {ω : ℕ → S | ∀ n, ω n ∉ B} = {ω : ℕ → S | ∀ n, ω n ∈ Bc}
```

### Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed rule transports the target judgment across the defining equation  $\text{avoid } \beta := G\neg\beta$  (premise: the expansion's judgment; conclusion: the sugared form). `sugarAvoid_denot` certifies that equation and is `rfl` — definitional at the set level, the strongest possible fidelity, with the negation-atom step ruled in by the `QNNFREWRITE` swaps as the docstring notes. Width note: a per-path equivalence over arbitrary state types — no measure enters, so the identity transports under every profile's path measure by congruence; the remaining accompanying claim (the entry's flavor text: the certificate surface *builds* the expansion rather than minting a node kind) is a property of the concrete-syntax canonicity battery — a test, not a theorem, deliberately outside this bound set, exactly as the roster witness says.

### Lean verifier receipts

module `BernsteinLogicAux.QtslAliases`; bound build 570599779c29... (bound from receipt 5a60f4d01fd81ac5, 2026-07-21T05:39:02Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when

the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0a3819a978e31127 | 2026-07-21T04:51:36Z | build+decorrelated                 | current |
| 0bb1074df705d863 | 2026-07-21T05:21:58Z | build+decorrelated                 | current |
| 5a60f4d01fd81ac5 | 2026-07-21T05:39:02Z | build+replay+hardened+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

**comparator:** none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*, in the strongest form an alias can have: the sugar is a definitional constructor that builds its expansion at the surface — no sugar node exists in any representation downstream of parsing, wire format included, so there is no artefact for any checker port to be about. Checked against the HOL sources: the `bern_catalogue` theory carries no spec-surface grammar.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template; the shared template is Recipe [R50.1](#) of the `QSTREETTDEGENERATE` section. This rule is the degenerate case of even that template: the `unpack` is a *definition*, so the five artefacts collapse to the definitional equation, its reading, and the target section.

**Recipe R55.1** (the filled-in application).

$$\text{avoid } \beta := G\neg\beta$$

The concluding judgment discharges through one existing rule: avoidance through the safety battery (`QSAFETYBATTERY`) at the `avoid` set  $\llbracket \beta \rrbracket$ , with  $\neg\beta$  read by the semantic negation clause  $\llbracket \neg\beta \rrbracket = S \setminus \llbracket \beta \rrbracket$  (the `QNNFREWITE` atom dualities supply the syntactic normal form — the atom swap alone when  $\beta$  is an atom, De Morgan propagation when it is compound).

**Recipe R55.2** (payload transport). The expansion  $G\neg\beta$  carries the safety kit at  $U := \llbracket \beta \rrbracket$ , verbatim. No `unpack` function is owed here, because none will ever exist: the constructor *is* its expansion, so there is no sugar payload to transport from — the strongest possible form of producers-elaborate-before-emission.

**Proposition R55.3** (the exactness identity). *A path always avoids  $\llbracket \beta \rrbracket$  iff it always satisfies  $\neg\beta$  — definitional, by the negation clause  $\llbracket \neg\beta \rrbracket = S \setminus \llbracket \beta \rrbracket$ , with `QNNFREWITE` normalizing the syntax where needed.*

*Proof.* Definitional (the bound Lean identity is `rfl`, axiom-free — the kernel certifies the two sides as one term). □

**Proposition R55.4** (soundness corollary). *The sugared judgment is concluded by the target rule's theorem, with no additional obligation: the sugar constructor builds the expansion, so the sugared judgment is the target judgment.*

*Proof.* Definitional unfolding plus the target section's theorem. The witness that the surface implementation honours the definition — that the constructor builds its expansion rather than introducing a new node kind — is the reference implementation's concrete-syntax canonicity test battery; no checker-facing artefact depends on it, since no downstream representation contains a sugar node. □

| proof step                   | formal anchor                                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------------------------------------|
| target                       | the QSAFETYBATTERY section; negation via the QNNFREWRITE atom swaps                                               |
| the identity, event-level    | sugarAvoid_denot (rfl, axiom-free; bound in the entry)                                                            |
| siblings (the sugar trio)    | QSTREETTGF, QSTREETTFG                                                                                            |
| constructor builds expansion | the reference implementation's concrete-syntax canonicity battery (documentary; no wire artefact exists to check) |

## Rule 56. QSTREETTGF (alias rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QStreettGF*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\text{STrue}, \varphi))) \quad \text{paths total over the state space}}{\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{GF } \varphi)}$$

$\text{GF}\varphi := \text{Streett}((\top, \varphi))$  — recurrence is the one-pair Streett condition at the trivially-true request, *QSTREETTDEGENERATE*'s concluded shape, so the sugar lands on that rule's bound statement without adjustment. The totality premise is the bound chart, exactly the certified statement's scope. Killed-process classes are the named unbound residue: the identity extends whenever the cemetery is adjoined inside the region vocabulary *and* killed paths are continued as total paths at  $\dagger$  ( $\top = V \cup \{\dagger\}$ ) — that  $\dagger$ -extended restatement is not yet part of the certified semantics, so it stays here rather than in a premise; with the cemetery outside the region vocabulary the rewrite direction over-claims, and that class is kept closed by the batteries' full-mass gates. The surface constructor builds its expansion rather than a new node kind — witnessed by the producer-side concrete-syntax canonicity battery; no rule sort exists and none is needed.

*QSTREETTGF* is the recurrence sugar of the spec surface:  $\text{GF}\varphi := \text{Streett}((\top, \varphi))$  — recurrence is the degenerate Streett pair, the *QSTREETTDEGENERATE* section's concluded shape, so the sugar lands on that rule's bound statement without adjustment. An alias unpacking to *one* application of an existing rule; with its persistence twin *QSTREETTFG* it is the compatibility layer to the classical temporal-logic surface — GF/FG specifications translate to the Streett-primitive kernel without automaton products, the practical payoff of Büchi acceptance being a one-pair Streett condition. The identity the definition rides is machine-checked (module *BernsteinLogicAux.QtslAliases*): *sugarGF\_denot* — the per-pair Streett event at the trivially-true request *is* the recurrence event.

Scope, stated in the open because this member alone has one: the mechanized identity lives on the total path type, so it consumes the no-cemetery chart — exact on total-path state spaces and on killed-process classes with the cemetery adjoined inside the region vocabulary ( $\top = V \cup \{\dagger\}$ ); with the cemetery outside the region vocabulary the rewrite direction over-claims, and that class is kept closed by the batteries' full-mass gates. The named repair (the entry's residue): restate the identity on the  $\dagger$ -extended state type, matching the promotion lemma. The surface constructor builds its expansion rather than a new node kind; the witness for that remains the reference implementation's concrete-syntax canonicity test battery — an implementation property, a test rather than a theorem.

**notes** Why the factoring put the honesty here: as a batch row, one *PROVEN* badge covered three warrants of different semantic depths — *rfl-grade*, *chart-consuming*, and *pure-logic* — and the *chart-consuming* member's scope restriction was a memo-level footnote. As its own row it is display-level text: the exactness chart is part of what this rule *says*.

### Lean statement

*BernsteinLogicAux.QtslAliases* . **sugarGF\_denot**

statement bound

```
/-- **`SugarBatch` (T10), second sugar**: `GF φ := Streett((⊤, φ))` — the
per-pair Streett event at the trivially-true request (every state requests,
and along `atTop` over `N` the request is frequent) *is* the recurrence
event. The expansion's target is `QStreettDegenerate` (T9a), whose bound
statement `sound_QStreettDegenerate` concludes exactly the right-hand
event. -/
theorem sugarGF_denot {S : Type*} (G : Set S) (ω : ℕ → S) :
  ((∃! n in atTop, ω n ∈ (Set.univ : Set S)) → ∃! n in atTop, ω n ∈ G)
  ↔ ∃! n in atTop, ω n ∈ G
```

### Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

$\text{sugarGF\_denot}(\text{GF}\varphi := \text{Streett}((\top, \varphi)))$  certifies the pair event at request  $\text{Set.univ} \leftrightarrow$  the bare recurrence event — exactly  $\text{sound\_QStreettDegenerate}$ 's concluded form, so the sugar lands on  $\text{QStreettDegenerate}$ 's bound statement without adjustment. Fidelity caveat, bound honestly: the Lean statement lives on the total path type, which IS the displayed rule's totality premise (“paths total over the state space”) — the premise and the bound scope now coincide exactly. The flavor text's  $\dagger$ -extended case (cemetery adjoined inside the vocabulary with killed paths continued as total paths at  $\dagger$ : exact; cemetery outside: the rewrite over-claims) is the named unbound residue the bound statement does not and cannot witness; it is carried by the roster note's residue line, and the  $\dagger$ -extended restatement named there would move it into this bound set. No measure enters; the identity transports under every profile's path measure by congruence. The builds-expansions claim rides the canonicity battery, deliberately outside this bound set, exactly as the roster witness says.

## Lean verifier receipts

module BernsteinLogicAux.QtslAliases; bound build 570599779c29... (bound from receipt 5a60f4d01fd81ac5, 2026-07-21T05:39:02Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0a3819a978e31127 | 2026-07-21T04:51:36Z | build+decorrelated                 | current |
| 0bb1074df705d863 | 2026-07-21T05:21:58Z | build+decorrelated                 | current |
| 5a60f4d01fd81ac5 | 2026-07-21T05:39:02Z | build+replay+hardened+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current  
 replay (leanchecker (official kernel replay)) current  
 hardened (safe\_verify (hardened screen)) current  
 decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*: the sugar is a definitional constructor that builds its expansion at the surface — no sugar node exists in any representation downstream of parsing, wire format included, so there is no artefact for any checker port to be about. Checked against the HOL sources: the `bern_catalogue` theory carries no spec-surface grammar. (The chart restriction is a property of the Lean statement's ambient path type, not of any wire artefact — there is nothing HOL-side for it to attach to.)

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template; the shared template is Recipe [R50.1](#) of the `QStreettDegenerate` section. The unpack is a *definition*, so the five artefacts collapse to the definitional equation, its reading, its exactness chart, and the target section.

**Recipe R56.1** (the filled-in application).

$$\text{GF}\varphi := \text{Streett}((\top, \varphi))$$

The concluding judgment discharges through one existing rule: recurrence through the degenerate bundle (`QStreettDegenerate`).

**Recipe R56.2** (payload transport). The expansion  $\text{Streett}((\top, \varphi))$  carries the degenerate kit, verbatim. No unpack function is owed here, because none will ever exist: the constructor *is* its expansion, so there is no sugar payload to transport from — the strongest possible form of producers-elaborate-before-emission.



**Proposition R56.3** (the exactness identity, and its chart). *The pair  $(\top, \varphi)$  holds iff  $\varphi$ -states occur infinitely often (the degeneration identity, proved as the exactness identity, Proposition R50.4, of the QSTREETTDEGENERATE section). As mechanized, the identity is stated on the total path type; it therefore holds on total-path state spaces, and on killed-process classes whenever the cemetery is adjoined inside the region vocabulary  $(\top = V \cup \{\dagger\})$ . With the cemetery outside the region vocabulary the rewrite direction over-claims — a killed path visits no state cofinally, yet the vocabulary cannot say so — and that class is kept closed by the batteries' full-mass gates. The named repair is to restate the identity on the  $\dagger$ -extended state type, matching the promotion lemma; until then the chart restriction is carried in the rule's own display, in the open.*

*Proof.* The identity: by the degeneration argument of the target section, mechanized as the bound Lean statement. The chart restriction: read off the bound statement's ambient path type, as displayed.  $\square$

**Proposition R56.4** (soundness corollary). *On the stated chart, the sugared judgment is concluded by the target rule's theorem, with no additional obligation: the sugar constructor builds the expansion, so the sugared judgment is the target judgment.*

*Proof.* Definitional unfolding plus the target section's theorem, on the chart of Proposition R56.3. The witness that the surface implementation honours the definition — that the constructor builds its expansion rather than introducing a new node kind — is the reference implementation's concrete-syntax canonicity test battery; no checker-facing artefact depends on it, since no downstream representation contains a sugar node.  $\square$

| proof step                   | formal anchor                                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------------------------------------|
| target                       | the QSTREETTDEGENERATE section (degeneration identity proved there)                                               |
| the identity, event-level    | sugarGF_denot (bound in the entry; total path type = the chart)                                                   |
| the chart's named repair     | the $\dagger$ -extended restatement, $\top = V \cup \{\dagger\}$ (the promotion lemma's form)                     |
| twin                         | QSTREETTFG (the safety-polarized half; consumes no chart)                                                         |
| constructor builds expansion | the reference implementation's concrete-syntax canonicity battery (documentary; no wire artefact exists to check) |

## Rule 57. QSTREETFG (alias rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QStreetFG*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PDwellStreett}(\text{PairBase}(\beta, \text{SFalse})))}{\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{FG } \neg\beta)}$$

$\text{FG}\neg\beta := \text{Streett}((\beta, \perp))$  — eventual permanent avoidance is the one-pair Streett condition that forbids infinitely many  $\beta$ -visits: with an unsatisfiable response, the pair condition says exactly that  $\beta$ -states occur only finitely often. Pure per-path logic — the safety-polarized half of the recurrence/persistence pair, consuming no conservativity. The rule's honest scope is this polarized form (FG applied to  $\neg\beta$ ): general persistence  $\text{FG}\varphi$  is not this rule. The surface constructor builds its expansion rather than a new node kind — witnessed by the producer-side concrete-syntax canonicity battery; no rule sort exists and none is needed.

QSTREETFG is the persistence-of-avoidance sugar of the spec surface:  $\text{FG}\neg\beta := \text{Streett}((\beta, \perp))$  — eventual permanent avoidance is the one-pair Streett condition that forbids infinitely many  $\beta$ -visits: with an unsatisfiable response, the pair condition says exactly that  $\beta$ -states occur only finitely often. An alias unpacking to *one* application of an existing rule (one QSTREETBATTERY node at the pair  $(\beta, \perp)$ ); the safety-polarized half of the recurrence/persistence pair — its twin is QSTREETGF, and the two land at different points of the same Streett rule. The honest scope is the polarized form (FG applied to  $\neg\beta$ ): general persistence  $\text{FG}\varphi$  is not this rule. The identity the definition rides is machine-checked (module BernsteinLogicAux.QtslAliases): *sugarFG\_denot* — the per-pair Streett event at an unsatisfiable response is eventual permanent avoidance, the finitely-often reading; pure per-path logic, no conservativity and no chart hypothesis consumed. The surface constructor builds its expansion rather than a new node kind; the witness for that remains the reference implementation's concrete-syntax canonicity test battery — an implementation property, a test rather than a theorem. No profile enters; the entry carries no conditional flag.

**notes** The two dual sugars land at different points of the same rule:  $\text{GF}\varphi$  requests always and so erases the request machinery, while  $\text{FG}\neg\beta$  makes  $\beta$  itself the request and forbids response — the ranking certificate must then pay out at every  $\beta$ -visit with nothing ever refunding it, which is exactly a proof that  $\beta$ -visits are finitely many. That both directions are single nodes of one rule — where automaton-primitive designs pay a product construction for one of them — is the practical content of the Streett-primitive choice.

### Lean statement

BernsteinLogicAux.QtslAliases . **sugarFG\_denot**

statement bound

```
/-- **`SugarBatch` (T10), third sugar**: `FG ¬β := Streett((β, ⊥))` — the
per-pair Streett event at an unsatisfiable response says the request region
is visited at most finitely often, which over `N` *is* eventual permanent
avoidance. The pair condition forbids infinitely many `β`-visits: with
response `⊥`, "infinitely often `β` implies infinitely often `False`" is
exactly "eventually always `¬β`". -/
theorem sugarFG_denot {S : Type*} (B : Set S) (ω : ℕ → S) :
  ((∃f n in atTop, ω n ∈ B) → ∃f n in atTop, ω n ∈ (∅ : Set S))
  ↔ ∀f n in atTop, ω n ∉ B
```

### Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

*sugarFG\_denot* ( $\text{FG}\neg\beta := \text{Streett}((\beta, \perp))$ ) certifies the pair event at response  $\emptyset \leftrightarrow$  eventual permanent avoidance ( $\forall^*$ ), the finitely-often reading — stated, like its siblings, in the target rule's concluded shape, which is what makes it consumable as a rewrite rather than a mere fact. Width note: a per-path equivalence over arbitrary state types — no measure enters, so the identity transports under every profile's path measure by congruence; the remaining accompanying claim (the entry's flavor text: the certificate surface *builds* the expansion rather than minting a node kind) is a property of the concrete-syntax canonicity battery — a test, not a theorem, deliberately outside this bound set, exactly as the roster witness says.

## Lean verifier receipts

module BernsteinLogicAux.QtslAliases; bound build 570599779c29... (bound from receipt 5a60f4d01fd81ac5, 2026-07-21T05:39:02Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0a3819a978e31127 | 2026-07-21T04:51:36Z | build+decorrelated                 | current |
| 0bb1074df705d863 | 2026-07-21T05:21:58Z | build+decorrelated                 | current |
| 5a60f4d01fd81ac5 | 2026-07-21T05:39:02Z | build+replay+hardened+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*: the sugar is a definitional constructor that builds its expansion at the surface — no sugar node exists in any representation downstream of parsing, wire format included, so there is no artefact for any checker port to be about. Checked against the HOL sources: the `bern_catalogue` theory carries no spec-surface grammar.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template; the shared template is Recipe **R50.1** of the `QSTREETTDEGENERATE` section. The unpack is a *definition*, so the five artefacts collapse to the definitional equation, its reading, and the target section.

**Recipe R57.1** (the filled-in application).

$$\text{FG}\neg\beta := \text{Streett}((\beta, \perp))$$

The concluding judgment discharges through one existing rule: eventual permanent avoidance through one `QSTREETTBATTERY` node at the pair  $(\beta, \perp)$ . The honest scope is this polarized form (FG applied to  $\neg\beta$ ): general persistence  $\text{FG}\varphi$  is not this rule.

**Recipe R57.2** (payload transport). The expansion  $\text{Streett}((\beta, \perp))$  carries one bundle pair: requests  $\llbracket\beta\rrbracket$ , response empty. No unpack function is owed here, because none will ever exist: the constructor *is* its expansion, so there is no sugar payload to transport from — the strongest possible form of producers-elaborate-before-emission.

**Proposition R57.3** (the exactness identity). *The pair  $(\beta, \perp)$  holds iff  $\beta$ -states occur only finitely often, i.e. iff eventually the path permanently satisfies  $\neg\beta$ :*

$$((\exists^\infty n, \omega_n \in \llbracket\beta\rrbracket) \Rightarrow (\exists^\infty n, \omega_n \in \emptyset)) \iff \neg(\exists^\infty n, \omega_n \in \llbracket\beta\rrbracket),$$

since the consequent is false outright — two lines: an implication into falsehood is the negation of its hypothesis, and finitely many  $\beta$ -visits along a discrete path is precisely a tail of  $\neg\beta$ -states. Pure per-path logic; no conservativity and no chart hypothesis is consumed.

*Proof.* As displayed; the two lines are shown. □

**Proposition R57.4** (soundness corollary). *The sugared judgment is concluded by the target rule's theorem, with no additional obligation: the sugar constructor builds the expansion, so the sugared judgment is the target judgment.*

*Proof.* Definitional unfolding plus the target section's theorem. The witness that the surface implementation honours the definition — that the constructor builds its expansion rather than introducing a new node kind — is the reference implementation's concrete-syntax canonicity test battery; no checker-facing artefact depends on it, since no downstream representation contains a sugar node.  $\square$

*Remark R57.5* (why FG costs a bundle and GF costs only its degenerate case). The two dual sugars land at different points of the same rule.  $\text{GF}\varphi$  (the twin,  $\text{QSTREETTGF}$ ) requests always and so erases the request machinery;  $\text{FG}\neg\beta$  makes  $\beta$  itself the request and forbids response — the ranking certificate must then pay out at every  $\beta$ -visit with nothing ever refunding it, which is exactly a proof that  $\beta$ -visits are finitely many. That both directions are single nodes of one rule — where automaton-primitive designs pay a product construction for one of them — is the practical content of the Streett-primitive choice (see the  $\text{QSTREETTBATTERY}$  section's remark).

| proof step                   | formal anchor                                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------------------------------------|
| target                       | the $\text{QSTREETTBATTERY}$ section, one pair $(\beta, \perp)$                                                   |
| the identity, event-level    | $\text{sugarFG\_denot}$ (bound in the entry)                                                                      |
| twin                         | $\text{QSTREETTGF}$ (recurrence; the member that consumes a chart)                                                |
| constructor builds expansion | the reference implementation's concrete-syntax canonicity battery (documentary; no wire artefact exists to check) |

## Rule 58. QNegFALIAS (alias for QSAFETYBATTERY)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QNegFALias*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{child} : \text{QRegionSat}(\text{SProbGE}(p, \text{PAlways}(\text{SNot } \varphi)), R)}{\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PNotPath}(\text{PDeadline}(T, \varphi, \text{false})))}$$

QNegFALIAS discharges a negated deadline: to certify, with probability at least  $p$ , that  $\varphi$  does *not* occur within horizon  $T$  ( $\neg F_{\leq T} \varphi$ ), it suffices to certify the *unbounded* safety claim —  $\varphi$  never occurs at all — over the same region, and the alias rewrites the one judgment into the other: the child is a satisfaction judgment for always-not- $\varphi$ , the conclusion the negated-deadline judgment, region verbatim. It is an alias to the safety battery (QSAFETYBATTERY): one node to one node, and the entire mathematical content is a two-line denotation inclusion — a path on which  $\varphi$  never holds certainly does not reach  $\varphi$  within  $T$ , and not-reaching-within- $T$  is the bounded-always of the negation. The rewrite is choice-free (no producer parameter beyond the horizon already present in the goal), and the checker recomputes nothing new: the child is an ordinary safety-battery conclusion.

The rule is deliberately lossy in one direction — the unbounded claim is strictly stronger than the bounded one — which is the honest price of reusing the safety kernel with no horizon bookkeeping. A producer who needs the sharp bounded claim (because  $\varphi$  *does* occur eventually, only late) uses the deadline machinery instead; this alias serves the common case where the avoidance is permanent.

Mechanization state, plainly: the bound statements below are the two lines, as priced. `negF_denot_subset` (module `BernsteinLogicAux.QtslAliases`) is the denotation inclusion — a path on which  $\varphi$  never holds does not reach  $\varphi$  within  $T$  — pure path-set algebra, axiom-free (the kernel certifies it depends on no axioms at all); and `sound_QNegFALias` is the alias's transport: a lower bound on the unbounded safety event (the target's conclusion shape, with `sat :=  $\neg\varphi$` ) is a lower bound on the negated-deadline event, for *any* measure on path space — profile-generic, so the discrete-kernel and GSHS records read it alike, and the entry carries no conditional flag.

notes Two flags travel with the row, both from the source catalogue. Tier (flagged, not forced): alias as sorted, but if the folded denotation inclusion must itself be a rule application, the tier moves to admissible — machinery unchanged either way. And the mirror is a deliberate hole: there is no dual rule discharging a negated *always* by a bounded operator, because the conclusion it would need ( $F\varphi$  unbounded) is structurally unrepresentable in the spec grammar — the unbounded-eventually constructor was excluded by testable-negative design, unbounded eventuality surviving only as a producer-choice provenance flag on deadlines. The pair must never be ported as one duality entry: one side is a statable choice-free alias, the other hides a horizon choice and a metatheory line inside the deadline battery.

### Lean statement

BernsteinLogicAux.QtslAliases . **negF\_denot\_subset**

statement bound

```
/-- **The denotation inclusion** (QTSL-rules.toml row `QNegFALias` -
"the denotation inclusion  $\llbracket G \neg \varphi \rrbracket \subseteq \llbracket G_{\leq T} \neg \varphi \rrbracket = \llbracket \neg F_{\leq T} \varphi \rrbracket$ " folds into the
rewrite"; the catalogue's "two lines"): a path on which  $\varphi$  never holds
certainly does not reach  $\varphi$  within  $T$  — and not-reaching-within- $T$  *is*
the negated deadline event. Pure path-set algebra; no measure, no
measurability. -/
theorem negF_denot_subset {S : Type*} (sat : S → Prop) (T : ℕ) :
  {ω : ℕ → S | ∀ n, ¬ sat (ω n)} ⊆ {ω : ℕ → S | ¬ ∃ k ≤ T, sat (ω k)}
```

BernsteinLogicAux.QtslAliases . **sound\_QNegFALias**

statement bound

```
/-- **`sound_QNegFALias`** (rule row `QNegFALias`): the alias's
transport — a lower bound on the probability of the unbounded safety event
(`QSafetyBattery`'s conclusion shape, e.g. `sound_QSafetyBattery` /
`sound_QSafetyBattery_staged` with `sat :=  $\neg\varphi$ `) is a lower bound on the
probability of the negated-deadline event. Profile-generic (any measure on
path space — discrete-kernel and GSHS records alike), by monotonicity
through `negF_denot_subset`. The rewrite is choice-free: no producer
parameter beyond the horizon  $T$  already present in the goal. -/
theorem sound_QNegFALias {S : Type*} [MeasurableSpace S]
  (μ : Measure (ℕ → S)) {sat : S → Prop} {b : ENNReal} {T : ℕ}
  (h : b ≤ μ {ω | ∀ n, ¬ sat (ω n)}) :
  b ≤ μ {ω | ¬ ∃ k ≤ T, sat (ω k)}
```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed rewrite — child  $\text{QRegionSat}(\text{SProbGE}(p, \text{PAlways}(\text{SNot } \varphi)), R)$  concluding  $\text{SProbGE}(p, \text{PNotPath}(\text{PDeadline}(T, \varphi, \text{false})))$  —  $\mapsto$  the two-statement chain:  $\text{negF\_denot\_subset}$  (the denotation inclusion  $\{\omega \mid \forall n, \neg \text{sat } (\omega n)\} \subseteq \{\omega \mid \neg \exists k \leq T, \text{sat } (\omega k)\}$  — pure path-set algebra, axiom-free, no measure, no measurability) and  $\text{sound\_QNegFALias}$ , its transport: a lower bound  $b \leq \mu$  on the unbounded-safety event is a lower bound on the negated-deadline event, for *any* measure on path space — profile-generic by binder shape ( $\mu : \text{Measure } (\mathbb{N} \rightarrow S)$  arbitrary), the  $\text{QUNIONBOUND}$  pattern; discrete-kernel and GSHS records alike are instances. The display's choice-freeness is a binder fact: no producer parameter appears beyond the horizon  $T$  already in the goal. Width note: the child's  $\text{SProbGE}$  shape enters only as the abstract bound  $b \leq \mu \{...\}$  — the certified transport is wider than the displayed probabilistic judgment (any  $\text{ENNReal}$  bound transports), safe direction; paths are step-indexed ( $\mathbb{N} \rightarrow S$ ), the deadline read discretely, as at  $\text{TAYLORBOUND}$ .

## Lean verifier receipts

module  $\text{BernsteinLogicAux.QtslAliases}$ ; bound build 570599779c29... (bound from receipt 5a60f4d01fd81ac5, 2026-07-21T05:39:02Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0a3819a978e31127 | 2026-07-21T04:51:36Z | build+decorrelated                 | current |
| 0bb1074df705d863 | 2026-07-21T05:21:58Z | build+decorrelated                 | current |
| 5a60f4d01fd81ac5 | 2026-07-21T05:39:02Z | build+replay+hardened+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*: the alias unpacks to a single safety-battery node before emission — the wire never carries a negated-deadline node, so the HOL statement that would verify the discharged certificate is the safety battery's own (absent today: the satisfaction stratum has no HOL artefact, per that section). Checked against the HOL sources: the  $\text{bern\_catalogue}$  theory is the base-calculus checker cone and carries no spec-surface path operators.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template; the batch's shared template is Recipe [R50.1](#) of the  $\text{QSTREETTDEGENERATE}$  section. The soundness content is two lines, recorded as such in the source catalogue, and both are now mechanized and bound in the entry: the inclusion ( $\text{negF\_denot\_subset}$ , axiom-free) and its measure transport ( $\text{sound\_QNegFALias}$ , profile-generic).

**Recipe R58.1** (the filled-in application).

$$\frac{\text{child} : \text{QRegionSat}(\text{SProbGE}(p, \text{PAlways}(\text{SNot } \varphi)), R) \text{ (one QSAFETYBATTERY conclusion)}}{\mathcal{M}, x_0 \in R \models \text{SProbGE}(p, \text{PNotPath}(\text{PDeadline}(T, \varphi, \text{false})))} \text{QNEGFAlias}$$

One node to one node: the child is the safety battery's conclusion at avoid set  $\llbracket \varphi \rrbracket$ , the concluded judgment re-reads the same certified region against the negated deadline, horizon  $T$  taken from the goal. In the staged spec vocabulary the conclusion's formula is the negated-path reading of a deadline node whose met-flag is false; the child's is the always of the negation — the rewrite touches the formula only.

|                                          | QNEGFAlias node                                                                                 | QSAFETYBATTERY node                                                                                  |
|------------------------------------------|-------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <b>Recipe R58.2</b> (payload transport). | certified region $R$ , probability $p$<br>goal formula $\neg F_{\leq T} \varphi$<br>horizon $T$ | verbatim<br>child formula $G \neg \varphi$ (horizon dropped)<br>not consumed ( <b>Remark R58.5</b> ) |

Unpack-function slot: no verified payload function in this edition (batch template); the semantic witness is bound in the entry (negF\_denot\_subset + sound\_QNegFAlias).

**Proposition R58.3** (the exactness inclusion — two lines).  $\llbracket G \neg \varphi \rrbracket \subseteq \llbracket G_{\leq T} \neg \varphi \rrbracket = \llbracket \neg F_{\leq T} \varphi \rrbracket$ .

*Proof.* A path that never satisfies  $\varphi$  in particular does not satisfy  $\varphi$  at any time  $\leq T$  — the inclusion, by restriction of a universal quantifier. And not satisfying  $\varphi$  at any time  $\leq T$  is the definition of not reaching  $\varphi$  within  $T$  — the equality, definitional. Hence every path in the child's path set lies in the conclusion's, and the probability bound transports monotonically.  $\square$

**Proposition R58.4** (soundness corollary). *If the child judgment holds — from every state of  $R$ , with probability  $\geq p$  the path always avoids  $\llbracket \varphi \rrbracket$  — then from every state of  $R$ , with probability  $\geq p$  the path does not reach  $\llbracket \varphi \rrbracket$  within  $T$ .*

*Proof.* Proposition **R58.3** plus monotonicity of measures, applied under the safety battery's theorem. There is no other content — and this two-step corollary is precisely what the bound statements are: the inclusion and the transport, one decl each.  $\square$

**Remark R58.5** (the deliberate lossiness). The alias proves the bounded claim from the unbounded one, discarding  $T$ . When  $\varphi$  genuinely occurs eventually — only after the horizon — the child judgment is unprovable and this alias is the wrong tool; the deadline machinery (the deadline battery, with its geometric decay certificates) proves the sharp bounded claim instead. The alias exists because permanent avoidance is the common case, and for it the safety kernel is both cheaper and already ported.

**Remark R58.6** (the mirror that is not a rule). There is deliberately no dual entry discharging a negated *always*: its conclusion would need unbounded eventuality ( $F\varphi$ ), and the spec grammar has no unbounded-eventually constructor — an exclusion by testable-negative design, unbounded eventuality surviving only as a provenance flag recording that a deadline's horizon was producer-chosen. What replaces the missing rule is three separate pieces: a negation-normal-form duality (QNNFREWITE), a horizon instantiation (a producer choice, flagged), and a weakening line *inside* the deadline battery's lemma where the flag is discharged. The asymmetry is worth this remark because a port that treated the pair as one duality entry would smuggle a choice and a metatheory line into what looks like a definitional rewrite.

**Remark R58.7** (tier flag). Alias as sorted; if the folded inclusion of Proposition **R58.3** must itself be a rule application rather than a lemma of the unpack, the tier moves to admissible — flagged in the source catalogue, machinery unchanged either way.

| proof step                           | formal anchor                                                                   |
|--------------------------------------|---------------------------------------------------------------------------------|
| Recipe <b>R58.1</b> target           | the QSAFETYBATTERY section                                                      |
| Proposition <b>R58.3</b> + corollary | negF_denot_subset + sound_QNegFAlias (two lines, as priced; bound in the entry) |
| Remark <b>R58.6</b>                  | the spec grammar's constructor census (no unbounded-eventually node)            |



## 15.7 Discharge Kits, Closure, and Arithmetic Leaves

### Rule 59. HYPMULTIPLIER (admissible rule)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *HypMultiplier*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\vdash \lambda \geq 0 \text{ on cell} \quad \vdash N + \lambda g_q \leq 0 \text{ on cell}}{\vdash N \leq 0 \text{ on cell} \cap \{g_q \geq 0\}}$$

the child atom  $\{g_q \geq 0\}$  is a concluded child certificate's region.

the child atom enters through the presentation bridge; both premises are BERNCONe-family rows, replayed by the base kernel — realized as the verified pointwise warrant `hypMultiplier_sound`.

HYPMULTIPLIER is the discharge kit's multiplier move, and the only member of the kit that manufactures a new row rather than reshaping the scope of an existing one. The situation it serves: a premise row of a satisfaction-judgment rule must hold not on a whole cell but on the part of the cell where a child certificate's region atom holds —  $N \leq 0$  on  $\text{cell} \cap \{g_q \geq 0\}$ , with  $\{g_q \geq 0\}$  the region concluded by a nested certificate. The base calculus's rows are scoped to cells, not to cell-with-polynomial-atom intersections, so the obligation cannot be posted as a base row directly. The rule buys the restriction with one multiplier: exhibit  $\lambda$  and post two ordinary cone-family rows —  $\lambda \geq 0$  on the cell, and  $N + \lambda g_q \leq 0$  on the cell.

What the checker recomputes: nothing beyond the two premise rows, which are cone-membership sequents the base kernel already replays; the combination step is the degree-one corner of the positivstellensatz family, the same multiplier discipline as the calculus's Farkas machinery. The producer typically finds  $\lambda$  by linear programming; only the two rows ship.

Both hypotheses are load-bearing, in opposite ways. Drop the sign row and a negative multiplier flips the direction of the trade: with  $N \equiv 1$ ,  $\lambda \equiv -1$ ,  $g_q \equiv 1$ , the combined row  $N + \lambda g_q \equiv 0 \leq 0$  holds while the conclusion fails everywhere on the child atom. Widen the conclusion to the whole cell and the rule is unsound wherever the child atom fails: where  $g_q(x) < 0$  the term  $\lambda(x) g_q(x)$  is a subsidy rather than a tax, and the combined row bounds nothing. The proof section walks both, counterexamples attached.

**notes** Before this rule's mechanization, the move lived in the producer toolkit as unchecked metatheory — the discharge step performed the combination and asserted the scoped row. The verified lemma closes that trust boundary. The certificate row that would let the wire format carry the move natively is out of scope of the current baked theory, so today the lemma warrants the toolkit's elaboration rather than a kernel-replayed node.

#### Lean statement

The auxiliary library states the extension theory's structural lemmas over the compiled semantic class: a bare state type, scopes and cells as sets, observables as real-valued functions. The presentation-level child atom enters through the named bridge hypothesis (`hbridge`), which the presentation layer discharges definitionally at compile time; nothing else about the system enters at all.

`BernsteinLogicAux.Structural.hypMultiplier_sound`

statement bound

```
/-- **T2 HypMultiplier**, soundness. `childAtom` is the presentation-level
child atom; its bridge names the compiled superlevel reading `{x | 0 ≤ g x}`.
The premises are the two compiled BernCone-family rows. -/
theorem hypMultiplier_sound (cell childAtom : Set X) (N lam g : X → ℝ)
  (hbridge : childAtom = {x | 0 ≤ g x})
  (hlam : ∀ x ∈ cell, 0 ≤ lam x)
  (hcomb : ∀ x ∈ cell, N x + lam x * g x ≤ 0) :
  ∀ x ∈ cell ∩ childAtom, N x ≤ 0
```

#### Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as

a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

One bound statement, transcription-close. The displayed premises  $\vdash \lambda \geq 0$  on cell and  $\vdash N + \lambda g_q \leq 0$  on cell  $\mapsto$  hlam/hcomb, pointwise on cell; the conclusion  $\vdash N \leq 0$  on cell  $\cap \{g_q \geq 0\} \mapsto$  the goal  $\forall x \in \text{cell} \cap \text{childAtom}, N x \leq 0$ , with the child atom entering through the System-bridge hypothesis hbridge : childAtom = {x | 0 ≤ g x} — the presentation-level region named by an equality the presentation layer will discharge definitionally; until then it is an honest binder, not an assertion. Width note: N, lam, g are arbitrary real-valued functions and cell an arbitrary set — the BernCone row structure and polynomiality are consumed one layer down where the rows are replayed; the certified warrant is the pointwise Farkas step alone, degree-1 positivstellensatz as displayed, strictly wider than the polynomial instance, safe direction.

## Lean verifier receipts

module BernsteinLogicAux.Structural; bound build 1222cdcc326f... (bound from receipt cb0c3820c6877f22, 2026-07-29T01:03:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0b7e68907f45b27f | 2026-07-28T23:48:36Z | build+decorrelated                 | current |
| e97d5614d4f08c7e | 2026-07-29T00:00:35Z | build+decorrelated                 | current |
| 27de21f047e6b194 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| cb0c3820c6877f22 | 2026-07-29T01:03:36Z | build+replay+hardened+decorrelated | current |
| 3530841119a9a607 | 2026-07-31T22:00:22Z | build+decorrelated                 | current |
| b6cc23eb3e5b7f97 | 2026-07-31T22:29:16Z | build+decorrelated                 | current |
| bf3dacc3ab7f20ea | 2026-08-02T23:31:31Z | build+decorrelated                 | current |
| 384a8569d2005df7 | 2026-08-03T16:27:35Z | build+decorrelated                 | current |
| 717e07acf9acac8a | 2026-08-03T22:46:19Z | build+decorrelated                 | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development's object is the base calculus's wire-format checker (theory bern\_catalogue ports the catalogue check-function cone). This rule belongs to the extension theory's discharge kit: it postdates that port, and no row for the move exists in the baked theory — there is nothing for a HOL statement to be about. The box becomes fillable when a revision carries the move's certificate rows and the corresponding check is ported.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe; the general admissible template, both species included, is stated in the NORM section and instantiated here in its boundary species (discharge against a named verified theorem). The rule is sequent-side machinery: premises and conclusion are all  $\vdash$  rows, and what the extension theory contributes is the *scope* vocabulary — the child atom  $\{g_q \geq 0\}$  names a region concluded by a nested certificate, a shape no base cell can express. The rule closes a named trust boundary: before its mechanization, the move was performed producer-side and asserted.

**Recipe R59.1** (the `HYPMULTIPLIER` instance).

$$\frac{\mathcal{T}_\lambda \triangleright \vdash \lambda \geq 0 \text{ on cell} \quad \mathcal{T}_c \triangleright \vdash N + \lambda g_q \leq 0 \text{ on cell}}{\vdash N \leq 0 \text{ on cell} \cap \{g_q \geq 0\}} \quad \text{HYPMULTIPLIER} \rightsquigarrow \begin{array}{l} \text{the premise subtrees stand unchanged;} \\ \text{hypMultiplier\_sound discharges the scoped row.} \end{array}$$

The rewrite is degenerate in the same honest sense as the base catalogue's ground-fact rule: the conclusion is not derivable from catalogue nodes, because the base grammar has no row form scoped to a cell-with-polynomial-atom intersection — that scope exists one layer up, where the child certificate that concluded  $\{g_q \geq 0\}$  lives. The two premise subtrees are ordinary cone-family derivations, checked by the base kernel unchanged; the node itself discharges against the verified pointwise warrant below, which consumes the two premises plus the one bridge hypothesis naming the child atom's compiled reading (the closing remark) — and nothing else.

**Proposition R59.2** (semantic warrant). *If  $\lambda \geq 0$  everywhere on the cell and  $N + \lambda g_q \leq 0$  everywhere on the cell, then  $N \leq 0$  everywhere on  $\text{cell} \cap \{g_q \geq 0\}$ .*

*Proof.* Fix  $x$  in the intersection. Then  $g_q(x) \geq 0$  (membership in the child atom) and  $\lambda(x) \geq 0$  (the sign row), so  $\lambda(x) g_q(x) \geq 0$ , and

$$N(x) \leq N(x) + \lambda(x) g_q(x) \leq 0,$$

the first inequality by nonnegativity of the added term, the second by the combined row at  $x$ . That is the whole proof — `hypMultiplier_sound` is the two displayed inequalities.  $\square$

*Remark R59.3* (the sign row is half the rule). Drop  $\vdash \lambda \geq 0$  and the rule is unsound. Counterexample:  $N \equiv 1$ ,  $\lambda \equiv -1$ ,  $g_q \equiv 1$  on any nonempty cell (a singleton suffices). The combined row holds identically ( $1 - 1 = 0 \leq 0$ ), yet  $N \equiv 1$  fails everywhere on  $\text{cell} \cap \{g_q \geq 0\} = \text{cell}$ . A free-signed multiplier converts the tax the child atom levies into a subsidy: the combination can cancel exactly the positivity the conclusion denies.

*Remark R59.4* (the scope restriction is the other half). The conclusion cannot claim the whole cell. With  $N \equiv 1$ ,  $\lambda \equiv 1$ ,  $g_q \equiv -1$  on the same nonempty cell, both premises hold ( $\lambda \geq 0$ ;  $1 - 1 = 0 \leq 0$ ) while  $N > 0$  everywhere — and the rule is nonetheless sound, because  $\text{cell} \cap \{g_q \geq 0\} = \emptyset$  and the scoped claim is vacuous. Where  $g_q(x) < 0$  the term  $\lambda(x) g_q(x)$  subsidizes  $N$  instead of taxing it, and the combined row bounds nothing: the scoped conclusion is exactly as strong as the premises pay for, no stronger.

*Remark R59.5* (degree-one positivstellensatz, and where the search lives). The move is the degree-one corner of certifying positivity over a constrained set by nonnegative combinations: Farkas' multiplier discipline [73] (see [216] for the sign conventions) in the affine case, sitting at the bottom of the polynomial cone hierarchy of Krivine [144] and Handelman [110] that the calculus's cone rule draws on. Nothing beyond degree one is consumed here: one multiplier, one product, no certificate machinery beyond the two posted rows. The *search* for  $\lambda$  (typically a linear program over the cone-membership witnesses) is producer-side; only its result ships, and the checker replays the two rows without ever seeing the search.

*Remark R59.6* (the bridge hypothesis). In the verified statement the child atom is an abstract set with one bridge hypothesis naming its compiled superlevel reading ( $\{x \mid 0 \leq g_q(x)\}$ ). When the presentation layer of the extension theory lands, the hypothesis discharges definitionally from the compile — the scoped conclusion itself is unchanged; only the bridge binder becomes definitional. Until then it is the honest record of exactly what the warrant assumes about the child certificate's conclusion — its region is the superlevel set it claims to be, and nothing more.

| proof step            | formal anchor                                                                                                   |
|-----------------------|-----------------------------------------------------------------------------------------------------------------|
| Recipe R59.1 warrant  | <code>hypMultiplier_sound</code> (BernsteinLogicAux/Structural.lean)                                            |
| premise rows          | cone-family sequent machinery of the base calculus (the <code>BERNCONE</code> and <code>FARKAS</code> sections) |
| child-atom reading    | the <code>hbridge</code> hypothesis of <code>hypMultiplier_sound</code>                                         |
| trust boundary closed | the producer-side discharge step this lemma now warrants                                                        |

## Rule 60. QFullDomainWeaken (alias rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — *QFullDomainWeaken*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\vdash P \text{ on } D_m \quad \text{scope} \subseteq D_m \text{ (compile fact, not a posted premise)}}{\vdash P \text{ on scope}}$$

emitted shape:  $\text{scope} = D_m \setminus \text{Off}$  (the off-scoped row).

payload verbatim, scope replaced — one application at the region layer, realized as `fullDomainWeaken_sound` (axiom-free) + `_offScope`.

QFullDomainWeaken is the cheapest move in the discharge kit: an off-scoped row — an obligation  $P$  on  $D_m \setminus \text{Off}$ , the part of a mode's domain outside an off-region — is discharged by proving  $P$  on all of  $D_m$  and forgetting the carve-out. It is an alias: one scope-weakening application at the region layer, transporting the row payload verbatim and replacing the scope by the full domain.

A producer reaches for it first. When the row happens to hold globally, the off-region was doing no work for this row, and the discharge costs nothing beyond the containment of the scope in the domain — which is a compile guarantee, not a posted premise. When the global row fails (the off-region was earning its keep), the move is simply unavailable, and the kit's working move, QVerifiedCover, localizes the effort instead.

What the checker recomputes: nothing. Soundness is the monotonicity of universal quantification — a row asserted on a superset restricts to any subset — and the verified lemma is a one-line restriction along an inclusion, provable without any classical reasoning at all.

notes Two deliberate non-identifications. First, the alias's target is not a catalogue node: no region-weakening rule exists to apply, and the unpacking discharges directly against the verified monotonicity lemma — the alias-tier twin of the admissible tier's recorded reduction-to-verified-machinery boundary case. Second, the rule is deliberately NOT the base calculus's sequent weakening: that rule manages a sequent's premise list; this one manages the region a row is quantified over, one extension layer up. Unifying them would conflate two different quantifiers that merely rhyme.

### Lean statement

Stated over the compiled semantic class (sets over a bare state type). The off-scoped shape the discharge plan actually emits is the second declaration, an instance of the first; the scope-in-domain containment enters as the bridge hypothesis the compile guarantees.

BernsteinLogicAux.Structural .fullDomainWeaken\_sound

statement bound

```
/-- **T3a FullDomainWeaken**, soundness. `scope` is the presentation-level
off-scoped row's scope; the bridge names the one fact the compile
guarantees about it: it lies inside the full domain. -/
theorem fullDomainWeaken_sound (Dm scope : Set X) (P : X → Prop)
  (hbridge : scope ⊆ Dm)
  (hrow : ∀ x ∈ Dm, P x) :
  ∀ x ∈ scope, P x
```

BernsteinLogicAux.Structural .fullDomainWeaken\_offScope

statement bound

```
/-- The off-scope instance: the scope compiled as `D_m` minus an off-region.
(The shape the discharge plan actually emits.) -/
theorem fullDomainWeaken_offScope (Dm Off : Set X) (P : X → Prop)
  (hrow : ∀ x ∈ Dm, P x) :
  ∀ x ∈ Dm \ Off, P x
```

### Correspondence (Steele display ↔ Lean)

reviewer-certified against the displayed texts The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display

conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed premise  $\vdash P$  on  $D_m \mapsto \text{hrow} : \forall x \in D_m, P\ x$ ; the compile fact scope  $\subseteq D_m$  — displayed as “not a posted premise” —  $\mapsto$  the System-bridge hypothesis  $\text{hbridge} : \text{scope} \subseteq D_m$ , an explicit binder here because lean-aux lacks the compile layer that guarantees it; the conclusion  $\mapsto \forall x \in \text{scope}, P\ x$ . The emitted shape  $\text{scope} = D_m \setminus \text{Off}$  is bound separately as  $\text{fullDomainWeaken\_offScope}$ , where the bridge discharges by  $\text{Set.diff\_subset}$  — zero hypotheses beyond the row. Width note:  $P : X \rightarrow \text{Prop}$  arbitrary (payload verbatim, as displayed — the certified statement cannot even see the payload, which is the strongest possible form of “payload verbatim, scope replaced”); the root statement is axiom-free pure  $\forall$ -monotonicity, exactly the alias tier's price.

## Lean verifier receipts

module BernsteinLogicAux.Structural; bound build 1222cdcc326f... (bound from receipt cb0c3820c6877f22, 2026-07-29T01:03:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0b7e68907f45b27f | 2026-07-28T23:48:36Z | build+decorrelated                 | current |
| e97d5614d4f08c7e | 2026-07-29T00:00:35Z | build+decorrelated                 | current |
| 27de21f047e6b194 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| cb0c3820c6877f22 | 2026-07-29T01:03:36Z | build+replay+hardened+decorrelated | current |
| 3530841119a9a607 | 2026-07-31T22:00:22Z | build+decorrelated                 | current |
| b6cc23eb3e5b7f97 | 2026-07-31T22:29:16Z | build+decorrelated                 | current |
| bf3dacc3ab7f20ea | 2026-08-02T23:31:31Z | build+decorrelated                 | current |
| 384a8569d2005df7 | 2026-08-03T16:27:35Z | build+decorrelated                 | current |
| 717e07acf9acac8a | 2026-08-03T22:46:19Z | build+decorrelated                 | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development's object is the base calculus's wire-format checker (theory bern\_catalogue ports the catalogue check-function cone). This rule is scope machinery of the extension theory's discharge kit: it postdates that port, and no row for the move exists in the baked theory — there is nothing for a HOL statement to be about. The box becomes fillable when a revision carries the move's certificate rows and the corresponding check is ported.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template. An alias owes the book five artefacts: the verified  $1 \rightarrow 1$  unpack, the filled-in application of the target, the payload-transport table, the exactness identity, and the soundness corollary through the target's lemma. This alias is the tier's degenerate case — its target is a verified lemma rather than a catalogue node, because no region-weakening rule exists to apply — and the five artefacts below say so honestly rather than inventing a node.

**Recipe R60.1** (the alias template, degenerate-target form). An alias rule  $A$  is warranted by a verified function carrying its payload to the payload of a *single* target application — one node  $\rightarrow$  one node, nothing derived. Where the target is not a catalogue node but a verified theorem, the transport carries the payload into the theorem's hypotheses instead, and the soundness corollary is the theorem itself: the alias-tier twin of the admissible tier's recorded reduction-to-verified-machinery boundary case.

**Recipe R60.2** (the filled-in application).

$$\frac{\mathcal{T} \triangleright \vdash P \text{ on } D_m}{\vdash P \text{ on } D_m \setminus \text{Off}} \text{QFULLDOMAINWEAKEN} \rightsquigarrow \text{fullDomainWeaken\_offScope}[\mathcal{T}] \triangleright \vdash P \text{ on } D_m \setminus \text{Off}$$

with the general form — any scope the compile certifies inside the domain — carried by `fullDomainWeaken_sound`: from  $P$  on  $D_m$  and  $\text{scope} \subseteq D_m$ , conclude  $P$  on  $\text{scope}$ .

|                                          | off-scoped row                                                               | full-domain row                                                                                       |
|------------------------------------------|------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| <b>Recipe R60.3</b> (payload transport). | row payload $P$<br>scope $D_m \setminus \text{Off}$<br>scope $\subseteq D_m$ | $P$ (verbatim)<br>$D_m$ (the off-region dropped)<br>compile guarantee (hbridge), not a posted premise |

**Proposition R60.4** (exactness identity).  $D_m \setminus \text{Off} \subseteq D_m$ ; more generally, every off-scoped row's scope is contained in its mode's domain.

*Proof.* A set difference is contained in its left operand — one line (`Set.diff_subset`). The general containment is the compile fact the bridge hypothesis names, supplied by the presentation layer, never proven per-instance.  $\square$

**Proposition R60.5** (soundness corollary). If  $P$  holds everywhere on  $D_m$ , it holds everywhere on any  $\text{scope} \subseteq D_m$  — in particular on  $D_m \setminus \text{Off}$ .

*Proof.* Restriction of a universally quantified statement along an inclusion: for  $x \in \text{scope}$ , the inclusion gives  $x \in D_m$ , and the full-domain row gives  $P(x)$ . `fullDomainWeaken_sound` is this one line, and it is axiom-free — no classical reasoning, no choice: pure monotonicity of  $\forall$ .  $\square$

**Remark R60.6** (what is spent). Nothing, soundness-wise — and possibly everything, feasibility-wise: if  $P$  fails anywhere in  $D_m \cap \text{Off}$  (points of the off-region outside  $D_m$  never enter the row), the full-domain row is simply unprovable and the move is unavailable. The off-region existing at all signals that some row somewhere needed it; this move is the discharge for the rows that did not, and its availability is itself a diagnosis: the carve-out was free for this row. When it fails, the kit's working move (`QVERIFIEDCOVER`) localizes the work instead of abandoning the carve-out.

**Remark R60.7** (deliberately not the sequent weakening). The base calculus has a weakening rule; this is deliberately not it. Sequent weakening grows a premise *list* — what a sequent assumes; this move shrinks a quantification *region* — where a row is asserted. The two live one extension layer apart and neither subsumes the other: a base sequent has no scope field to weaken, and this move never touches a premise list. The kinship is real, and the identification would be a type error.

| proof step                                                     | formal anchor                                                                                                  |
|----------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| Recipe <b>R60.2</b> unpack<br>general form                     | <code>fullDomainWeaken_offScope</code> (the emitted shape)<br><code>fullDomainWeaken_sound</code> (axiom-free) |
| Proposition <b>R60.4</b><br>region-layer monotonicity, generic | <code>Set.diff_subset</code> ; the hbridge inclusion<br><code>regionSat_mono</code> (same module)              |

## Rule 61. QVERIFIEDCOVER (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QVerifiedCover*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\vdash P \text{ on } C \text{ (the cover row)} \quad \{\vdash K_i \subseteq \text{Off}\}_i \text{ (per-component containments)} \quad D_m \setminus C \subseteq \bigcup_i K_i \text{ (residual decomposition)}}{\vdash P \text{ on } D_m \setminus \text{Off}}$$

the per-component containments discharge by exact Bernstein upper bounds, and the residual decomposition is recomputed by the checker. Dimension-free as realized (*verifiedCover\_components\_sound*, arbitrary component index); the cover bookkeeping  $C \subseteq D_m$  is checker-side well-formedness, not consumed by soundness.

QVERIFIEDCOVER is the discharge kit's working move: an off-scoped row —  $P \text{ on } D_m \setminus \text{Off}$  — is discharged by proving the row on a cover  $C$  and certifying that everything of  $D_m$  the cover misses lies inside the off-region. The un-covered residual is decomposed into components  $K_i$ , and each component ships one containment premise  $K_i \subseteq \text{Off}$ , discharged by exact Bernstein upper bounds — ordinary sequent rows the base kernel replays.

A producer reaches for it when QFULLDOMAINWEAKEN fails — the row genuinely needs its carve-out — but the on-region can be covered cleanly. Cover *selection* is producer-side search (the current toolkit selects one-dimensional covers); the verified statement is dimension-free, so richer covers need no new warrant.

What the checker recomputes: the residual decomposition  $D_m \setminus C \subseteq \bigcup_i K_i$  (set bookkeeping over the posted components) and the per-component containment rows. The case analysis is exhaustive by construction: a point of the scope is either covered, and the row-on- $C$  premise applies, or un-covered, in which case it lies in some  $K_i \subseteq \text{Off}$  — contradicting membership in the scope. Drop the per-component containments and an un-covered point outside the off-region escapes both cases; drop the decomposition and the component list could silently miss part of the residual.

**notes** One premise of the catalogued statement is deliberately absent from the verified one: the cover bookkeeping  $C \subseteq D_m$ . The proof never consumes it — a cover reaching outside the domain proves more rows than needed, harmlessly — so the mechanized statement omits it, which is strictly stronger in the sound direction; the checker keeps the containment as cover well-formedness. The neighbouring scope-preparation steps (hull closure, cell covers, off-scope splitting, reset containment, floor boxes) are deliberately not rules at all: they are side conditions the checker recomputes, and rows for them would misstate that.

### Lean statement

Stated over the compiled semantic class (sets over a bare state type), dimension-free. The residual form is the kernel statement; the per-component form is the shape the discharge plan emits, with an arbitrary component index — nothing one-dimensional survives into the warrant.

BernsteinLogicAux.Structural.**verifiedCover\_sound**

statement bound

```
/-- **T3b VerifiedCover**, soundness (residual form). `scope` is the
off-scoped row's compiled scope `D_m \ Off`; the row is proven on the cover
`C`, and the un-covered residual `D_m \ C` is certified inside the
off-region. -/
theorem verifiedCover_sound (Dm C Off scope : Set X) (P : X → Prop)
  (hbridge : scope = Dm \ Off)
  (hrowC : ∀ x ∈ C, P x)
  (hresidual : Dm \ C ⊆ Off) :
  ∀ x ∈ scope, P x
```

BernsteinLogicAux.Structural.**verifiedCover\_components\_sound**

statement bound

```
/-- **T3b VerifiedCover**, per-component form: the residual is decomposed
into components `K i`, each separately certified inside the off-region (the
"exact Bernstein upper bounds" premises, one per component). -/
theorem verifiedCover_components_sound {ι : Type*} (Dm C Off scope : Set X)
  (K : ι → Set X) (P : X → Prop)
  (hbridge : scope = Dm \ Off)
  (hrowC : ∀ x ∈ C, P x)
```



```

(hdecomp : Dm \ C ⊆ ⋃ i, K i)
(hcomp : ∀ i, K i ⊆ Off) :
∀ x ∈ scope, P x

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed premises map onto `verifiedCover_components_sound` one for one: the cover row  $\vdash P$  on  $C \mapsto \text{hrowC}$ ; the per-component containments  $K_i \subseteq \text{Off} \mapsto \text{hcomp} : \forall i, K i \subseteq \text{Off}$  (the exact-Bernstein-upper-bound premises, one per component, over an *arbitrary* index type — the dimension-freedom the flavor text claims is a binder fact); the recomputed residual decomposition  $D_m \setminus C \subseteq \bigcup_i K_i \mapsto \text{hdecomp}$ , an explicit hypothesis (recompute rendered as binder, the `QSTREETTARGETCELLCOLLAPSE` pattern); the conclusion scope  $D_m \setminus \text{Off} \mapsto$  the System-bridge equality `hbridge : scope = Dm \ Off`. The residual form `verifiedCover_sound` is the same statement with the decomposition pre-collapsed. Width note, divergence `D1` as certified: the catalogued well-formedness  $C \subseteq D_m$  appears in `NO` binder — soundness demonstrably never consumes it, which the module header banks as the divergence and the flavor text carries; payload `P` arbitrary, per the `QFullDomainWeaken` pattern.

## Lean verifier receipts

module `BernsteinLogicAux.Structural`; bound build 1222cdcc326f... (bound from receipt cb0c3820c6877f22, 2026-07-29T01:03:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0b7e68907f45b27f | 2026-07-28T23:48:36Z | build+decorrelated                 | current |
| e97d5614d4f08c7e | 2026-07-29T00:00:35Z | build+decorrelated                 | current |
| 27de21f047e6b194 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| cb0c3820c6877f22 | 2026-07-29T01:03:36Z | build+replay+hardened+decorrelated | current |
| 3530841119a9a607 | 2026-07-31T22:00:22Z | build+decorrelated                 | current |
| b6cc23eb3e5b7f97 | 2026-07-31T22:29:16Z | build+decorrelated                 | current |
| bf3dacc3ab7f20ea | 2026-08-02T23:31:31Z | build+decorrelated                 | current |
| 384a8569d2005df7 | 2026-08-03T16:27:35Z | build+decorrelated                 | current |
| 717e07acf9acac8a | 2026-08-03T22:46:19Z | build+decorrelated                 | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**  
 replay (leanchecker (official kernel replay)) **current**  
 hardened (safe\_verify (hardened screen)) **current**  
 decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator:** none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development's object is the base calculus's wire-format checker (theory `bern_catalogue` ports the catalogue check-function cone). This rule is scope machinery of the extension theory's discharge kit: it postdates that port, and no row for the move exists in the baked theory — there is nothing for a HOL statement to be about. The box becomes fillable when the extension's certificate rows land in a wire format and the corresponding check is ported.

## HOL verifier receipts

**NONE**

no HOL verification receipt exists for this rule.

**Proof**

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. The rule discharges an off-scoped row by an exhaustive case analysis: prove the row on a cover, certify everything un-covered into the off-region. Its premises are ordinary sequent rows; the case analysis is the verified content.

The admissible template, general form, is stated once for the whole catalogue as Recipe **R16.1** of the **NORM** section; this section instantiates it.

**Recipe R61.1** (the  $\text{QVERIFIEDCOVER}$  instance).

$$\frac{\mathcal{T}_C \triangleright \vdash P \text{ on } C \quad \{ \mathcal{T}_i \triangleright \vdash K_i \subseteq \text{Off} \}_i \quad D_m \setminus C \subseteq \bigcup_i K_i}{\vdash P \text{ on } D_m \setminus \text{Off}} \quad \text{QVERIFIEDCOVER} \quad \begin{array}{l} \text{premise subtrees stand;} \\ \text{verifiedCover\_components\_sound} \\ \text{discharges the scoped row.} \end{array}$$

The per-component containments  $K_i \subseteq \text{Off}$  are exact Bernstein upper-bound rows — ordinary sequents the base kernel replays; the residual decomposition is checker-recomputed set bookkeeping over the posted components, not a proof obligation. The node discharges against the verified case analysis below.

**Proposition R61.2** (semantic warrant, residual form). *If  $P$  holds everywhere on  $C$  and  $D_m \setminus C \subseteq \text{Off}$ , then  $P$  holds everywhere on  $D_m \setminus \text{Off}$ .*

*Proof.* Fix  $x \in D_m \setminus \text{Off}$  and split on membership in the cover. If  $x \in C$ , the cover row gives  $P(x)$ . If  $x \notin C$ , then  $x \in D_m \setminus C \subseteq \text{Off}$  — contradicting  $x \notin \text{Off}$ . The two cases are exhaustive, so the second is vacuous and the first decides ( $\text{verifiedCover\_sound}$ ).  $\square$

**Proposition R61.3** (semantic warrant, per-component form). *If  $P$  holds everywhere on  $C$ , the residual decomposes as  $D_m \setminus C \subseteq \bigcup_i K_i$ , and each component satisfies  $K_i \subseteq \text{Off}$ , then  $P$  holds everywhere on  $D_m \setminus \text{Off}$ .*

*Proof.* The union of the components is contained in  $\text{Off}$  (each is), so  $D_m \setminus C \subseteq \text{Off}$  and Proposition **R61.2** applies ( $\text{verifiedCover\_components\_sound}$ ). The component index is an arbitrary type: no finiteness, no dimension, no ordering of components is consumed.  $\square$

**Remark R61.4** (each premise family is load-bearing). Drop the cover row and any  $x \in C \cap (D_m \setminus \text{Off})$  with  $P(x)$  false refutes the conclusion while both set-bookkeeping premises hold. Drop the per-component containments and an un-covered point outside the off-region escapes both cases:  $x \in D_m \setminus C$  with  $x \notin \text{Off}$  and  $P(x)$  false refutes the conclusion while the remaining premises hold. Drop the residual decomposition and the component list can silently miss part of the residual — the same point, now lying in no  $K_i$ , escapes again. The exhaustiveness of the case analysis is exactly the conjunction of the last two; the cover row is what it decides to.

**Remark R61.5** (the omitted premise — deliberate, and in the sound direction). The catalogued statement carries a cover-bookkeeping premise  $C \subseteq D_m$ ; the verified statement omits it, because the proof never consumes it: a cover reaching outside the domain proves more rows than needed, harmlessly. Omitting an un-consumed hypothesis makes the mechanized statement strictly stronger, in the sound direction — the checker retains the containment as cover *well-formedness*, which is where it belongs.

**Remark R61.6** (the kit as a spectrum). The corner  $C = D_m$  (empty residual, no components) is  $\text{QFULLDOMAINWEAKEN}$ ; the opposite corner  $C = \emptyset$  demands the entire domain certified into the off-region, at which point the scope is empty and the row vacuous. Between the corners, the cover is the dial trading global proof effort against per-component containment certificates — and cover *selection* is producer-side search (one-dimensional in the current toolkit), spent before anything ships. The verified statement is dimension-free: richer covers change the search, never the warrant.

**proof step**

Recipe **R61.1** warrant  
Proposition **R61.2**  
containment premises  
scope reading  
degenerate corner

**formal anchor**

$\text{verifiedCover\_components\_sound}$   
 $\text{verifiedCover\_sound}$  (the case split)  
exact Bernstein upper-bound rows (the **BERNCONE** section)  
the hbridge hypothesis (scope =  $D_m \setminus \text{Off}$ )  
the **QFULLDOMAINWEAKEN** section

## Rule 62. QPOSITIVECOMBINATIONROOT (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QPositiveCombinationRoot*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{child}_1 : \text{QRegionSat}(\varphi_1, R_1) \quad \text{child}_2 : \text{QRegionSat}(\varphi_2, R_2)}{\mathcal{M}, x_0 \in R_1 \cap R_2 \models \varphi_1 \wedge \varphi_2}$$

$$\frac{\text{child}_1 : \text{QRegionSat}(\varphi_1, R_1) \quad \text{child}_2 : \text{QRegionSat}(\varphi_2, R_2)}{\mathcal{M}, x_0 \in R_1 \cup R_2 \models \varphi_1 \vee \varphi_2}$$

$n$ -ary forms  $\bigcap_i R_i / \bigcup_i R_i$  over a block family (`positiveCombination{And,Or,Inter,Union}_sound`); positive connectives only — negation rides the normal-form and closure machinery, never this rule.

QPOSITIVECOMBINATIONROOT assembles a certificate for a root boolean combination out of certificates for its blocks, at zero new row cost: if the children conclude  $\text{QRegionSat}(\varphi_1, R_1)$  and  $\text{QRegionSat}(\varphi_2, R_2)$ , the conjunction is certified on  $R_1 \cap R_2$  and the disjunction on  $R_1 \cup R_2$  — and the  $n$ -ary forms take  $\bigcap_i R_i$  and  $\bigcup_i R_i$  over a whole block family. A producer uses it at the top of every specification whose root is a positive boolean shape: each block is certified by its own sub-derivation (a battery, an invariance certificate, a nested combination), and the root costs two set operations.

What the checker recomputes: the region intersection or union itself. The semantic clauses — that the conjunction denotes the intersection of the denotations, the disjunction the union — enter as definitional bridges supplied by the specification layer, never as posted premises.

The word *positive* in the name is load-bearing. Under-approximation is preserved by intersection and union: a region inside each denotation is inside their intersection; a union of regions inside denotations is inside the union of the denotations. It is REVERSED by complement: the complement of an under-approximation over-approximates — the empty region under-approximates every denotation, and its complement, the whole space, over-approximates every complement. A negation passed through this rule would silently flip the judgment kind's one orientation guarantee. Negation therefore never passes through it; it rides the normal-form and closure machinery instead, where the direction swap is handled deliberately.

notes Admissible, not alias: several children and two lemma directions — more than a single application. The node shape (a pair of binary connective nodes versus one  $n$ -ary combination node) is an enactment decision still open on the schema side; the verified lemmas cover both forms, so neither choice waits on new mathematics.

### Lean statement

RegionSat is the compiled reading of the workhorse judgment kind — the region under-approximates the denotation — quoted in full because the four lemmas' meaning turns on it. The connective denotations enter through bridge hypotheses (`denAnd` = intersection of the children's denotations, `denOr` = union), supplied by the specification layer at compile time.

BernsteinLogicAux.Structural . **positiveCombinationAnd\_sound**

statement bound

```

/-- **T11**, conjunction: intersection of under-approximations
under-approximates the conjunction. -/
theorem positiveCombinationAnd_sound (R1 R2 den1 den2 denAnd : Set X)
  (hbridge : denAnd = den1 ∩ den2)
  (h1 : RegionSat R1 den1) (h2 : RegionSat R2 den2) :
  RegionSat (R1 ∩ R2) denAnd

```

BernsteinLogicAux.Structural . **positiveCombinationOr\_sound**

statement bound

```

/-- **T11**, disjunction: union of under-approximations under-approximates
the disjunction. -/
theorem positiveCombinationOr_sound (R1 R2 den1 den2 denOr : Set X)
  (hbridge : denOr = den1 ∪ den2)
  (h1 : RegionSat R1 den1) (h2 : RegionSat R2 den2) :
  RegionSat (R1 ∪ R2) denOr

```

BernsteinLogicAux.Structural . **positiveCombinationInter\_sound**

statement bound

```

/-- **T11**, n-ary conjunction (the staged n-ary node shape). -/
theorem positiveCombinationInter_sound {ι : Type*} (R den : ι → Set X)
  (denAll : Set X)
  (hbridge : denAll = ⋂ i, den i)
  (h : ∀ i, RegionSat (R i) (den i)) :
  RegionSat (⋂ i, R i) denAll

```

BernsteinLogicAux.Structural . **positiveCombinationUnion\_sound**

statement bound

```

/-- **T11**, n-ary disjunction. -/
theorem positiveCombinationUnion_sound {ι : Type*} (R den : ι → Set X)
  (denAll : Set X)
  (hbridge : denAll = ⋃ i, den i)
  (h : ∀ i, RegionSat (R i) (den i)) :
  RegionSat (⋃ i, R i) denAll

```

BernsteinLogicAux.Structural . **RegionSat**

definition bound

```

/-- The satisfaction judgment's workhorse kind, compiled: region `R`
under-approximates denotation `den`. -/
def RegionSat (R den : Set X) : Prop := R ⊆ den

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The judgment kind itself is bound by display:  $\text{RegionSat } R \text{ den} = R \subseteq \text{den}$  —  $\text{QRegionSat}$  compiled, the formula appearing only through its denotation (the System bridge, module §0). The two displayed binary rules  $\mapsto$  `positiveCombinationAnd_sound` / `Or_sound`: children  $\text{QRegionSat}(\varphi_i, R_i) \mapsto h_1/h_2 : \text{RegionSat } R_i \text{ den}_i$ ; conclusions at  $R_1 \cap R_2$  /  $R_1 \cup R_2$  verbatim; and the semantic clauses  $\llbracket \varphi_1 \wedge \varphi_2 \rrbracket = \llbracket \varphi_1 \rrbracket \cap \llbracket \varphi_2 \rrbracket$  (resp.  $\cup$ )  $\mapsto$  the bridge hypotheses  $\text{hbridge} : \text{denAnd} = \text{den}_1 \cap \text{den}_2$  — supplied by the future spec layer, never asserted in lean-aux, exactly as the roster witness says. The  $n$ -ary forms (`Inter_sound`/`Union_sound`, arbitrary index) cover the baked node shapes (`QAndRegion`/`QOrRegion`, each  $n$ -ary over a children list). Width note: pure lattice monotonicity over arbitrary sets — no formula grammar, no positivity syntax; the “positive connectives only” restriction is enforced by which bridges exist to supply (negation has no intersection/union clause), not by any binder here — refusal lives in the spec layer, the certified statements are the two monotone directions the display prices.

## Lean verifier receipts

module BernsteinLogicAux.Structural; bound build 1222cdcc326f... (bound from receipt cb0c3820c6877f22, 2026-07-29T01:03:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0b7e68907f45b27f | 2026-07-28T23:48:36Z | build+decorrelated                 | current |
| e97d5614d4f08c7e | 2026-07-29T00:00:35Z | build+decorrelated                 | current |
| 27de21f047e6b194 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| cb0c3820c6877f22 | 2026-07-29T01:03:36Z | build+replay+hardened+decorrelated | current |
| 3530841119a9a607 | 2026-07-31T22:00:22Z | build+decorrelated                 | current |
| b6cc23eb3e5b7f97 | 2026-07-31T22:29:16Z | build+decorrelated                 | current |
| bf3dacc3ab7f20ea | 2026-08-02T23:31:31Z | build+decorrelated                 | current |
| 384a8569d2005df7 | 2026-08-03T16:27:35Z | build+decorrelated                 | current |
| 717e07acf9acac8a | 2026-08-03T22:46:19Z | build+decorrelated                 | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development's object is the base calculus's wire-format checker (theory `bern_catalogue` ports the catalogue check-function cone). This rule concludes a satisfaction judgment of the extension theory: it postdates that port, and although the combination nodes are baked in the extension theory (`QAndRegion/QOrRegion`), no HOL development consumes the extension theory's rows — there is no ported check for a HOL statement to be about. The box becomes fillable when a checker consuming the extension's certificate rows is ported.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. Admissible and not alias by count: several children, and two lemma directions (one per connective). The rule adds zero sequent rows; its entire content is two one-line lattice-monotonicity facts about under-approximation.

The admissible template, general form, is stated once for the whole catalogue as Recipe [R16.1](#) of the `NORM` section; this section instantiates it.

**Recipe R62.1** (the `QPOSITIVECOMBINATIONROOT` instance).

$$\frac{\mathcal{T}_1 \triangleright \text{QRegionSat}(\varphi_1, R_1) \quad \mathcal{T}_2 \triangleright \text{QRegionSat}(\varphi_2, R_2)}{\mathcal{M}, x_0 \in R_1 \cap R_2 \models \varphi_1 \wedge \varphi_2} \wedge\text{-FORM} \quad \frac{\mathcal{T}_1 \triangleright \text{QRegionSat}(\varphi_1, R_1) \quad \mathcal{T}_2 \triangleright \text{QRegionSat}(\varphi_2, R_2)}{\mathcal{M}, x_0 \in R_1 \cup R_2 \models \varphi_1 \vee \varphi_2} \vee\text{-FORM}$$

Each node rewrites to its children — which stand unchanged — plus one application of the corresponding lattice-monotonicity warrant (Propositions [R62.2](#) and [R62.3](#)); the  $n$ -ary forms replace the pair by a family and the binary operation by  $\cap_i$  or  $\cup_i$ . No sequent row is added anywhere: the whole rewrite happens at the judgment layer.

**Proposition R62.2** (conjunction: intersection preserves under-approximation). *If  $R_1$  under-approximates the denotation of  $\varphi_1$  and  $R_2$  that of  $\varphi_2$ , then  $R_1 \cap R_2$  under-approximates the denotation of  $\varphi_1 \wedge \varphi_2$  (which is the intersection of the two denotations, by the specification layer's definitional clause).*

*Proof.*  $x \in R_1 \cap R_2$  gives  $x \in R_1 \subseteq \llbracket \varphi_1 \rrbracket$  and  $x \in R_2 \subseteq \llbracket \varphi_2 \rrbracket$ , so  $x \in \llbracket \varphi_1 \rrbracket \cap \llbracket \varphi_2 \rrbracket = \llbracket \varphi_1 \wedge \varphi_2 \rrbracket$ . One line (`positiveCombinationAnd_sound`); the final equality is the `denAnd` bridge, supplied at compile time, never posted.  $\square$

**Proposition R62.3** (disjunction: union preserves under-approximation). *Under the same hypotheses,  $R_1 \cup R_2$  under-approximates the denotation of  $\varphi_1 \vee \varphi_2$  (the union of the denotations).*

*Proof.*  $x \in R_1 \cup R_2$  lies in some  $R_j \subseteq \llbracket \varphi_j \rrbracket \subseteq \llbracket \varphi_1 \rrbracket \cup \llbracket \varphi_2 \rrbracket = \llbracket \varphi_1 \vee \varphi_2 \rrbracket$ . One line (`positiveCombinationOr_sound`), with the `denOr` bridge closing it.  $\square$

**Remark R62.4** (the  $n$ -ary forms, and the empty corners). The  $n$ -ary lemmas (`positiveCombinationInter_sound`, `positiveCombinationUnion_sound`) quantify over an arbitrary index type — no finiteness is consumed — so a single  $n$ -ary combination node brings no new mathematical idea beyond the same monotonicity principle; the  $n$ -ary lemmas themselves are its warrant (an arbitrary-index family is not a fold of binary steps, which is why they are separate statements). The empty corners are sound instances rather than exclusions: an empty conjunction denotes everything and  $\bigcap_{i \in \emptyset} R_i$  is the whole space, which under-approximates it; an empty disjunction denotes nothing and the empty union is empty, which under-approximates that. Neither degenerate certificate claims anything false, and each is worthless in its own honest way: the disjunction corner certifies the empty region, the conjunction corner certifies a tautology.

**Remark R62.5** (the polarity fence, with the two-line counterexample). Complement REVERSES the direction this rule preserves: from  $R \subseteq \llbracket \varphi \rrbracket$  nothing at all follows about `cmpl`  $R$  under-approximating `cmpl`  $\llbracket \varphi \rrbracket$  — the containment flips. Sharpest instance:  $R = \emptyset$  under-approximates every denotation, and its complement is the whole space, which under-approximates only the trivial one. A negation routed through this rule would silently convert the judgment kind's under-approximation guarantee into its opposite. The name *positive* is that fence: negation rides the normal-form rewrites and the closure discipline (the `QCLOSUREEXACT` section and its siblings), where the direction swap is handled deliberately, one atom at a time.

*Remark R62.6* (the bridge hypotheses). The semantic clauses  $\llbracket \varphi_1 \wedge \varphi_2 \rrbracket = \llbracket \varphi_1 \rrbracket \cap \llbracket \varphi_2 \rrbracket$  and its  $\vee$ /union twin enter the verified statements as named bridge hypotheses (`denAnd`, `denOr`), to be discharged definitionally by the specification layer's compile. The warrant deliberately does not assert them: what the connectives denote is the specification layer's to define, and this rule's content is only that under-approximation respects whatever lattice structure that definition induces.

| proof step                                    | formal anchor                                                                             |
|-----------------------------------------------|-------------------------------------------------------------------------------------------|
| Recipe <a href="#">R62.1</a> , $\wedge$ -form | <code>positiveCombinationAnd_sound</code>                                                 |
| Recipe <a href="#">R62.1</a> , $\vee$ -form   | <code>positiveCombinationOr_sound</code>                                                  |
| $n$ -ary forms                                | <code>positiveCombinationInter_sound</code> , <code>positiveCombinationUnion_sound</code> |
| the judgment reading                          | <code>RegionSat</code> (quoted in full above)                                             |
| the polarity fence's other side               | <code>compl_over_approx</code> (the <code>QCLOSUREOUTWARD</code> section)                 |

## Rule 63. QCLOSUREEXACT (alias rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QClosureExact*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\overline{\text{cml}\{p < c\}} = \{c \leq p\}$$

$$\overline{\text{cml}\{c < p\}} = \{p \leq c\}$$

two axioms — deduction rules with nothing above the line, each an exact set identity, both orientations stated (`closureExactLt_sound/Gt`) — no margin spent, no side condition; the closed form is a closed set for continuous observables (`closureExact_isClosed_ge/le`).

The closure discipline is the calculus's answer to strict inequalities: rows quantify over closed cells and certificates consume closed atoms, so strict literals —  $\{p < c\}$ ,  $\{c < p\}$  — never reach a row. Three moves resolve them, keyed to where the literal sits. *QCLOSUREEXACT*, this rule, is the free one: the *complement* of a strict literal is already a closed atom, exactly —  $\text{cml}\{p < c\} = \{c \leq p\}$  — so a complemented strict literal is replaced by its closed form with no loss, no margin, and no side condition. The sibling moves pay: *QCLOSUREOUTWARD* closes the complement of a *closed* atom by an orientation swap that over-approximates (the sound direction at negative positions), and *QCLOSUREINWARDEPS* closes a strict literal at a positive position by rounding inward a producer-chosen  $\varepsilon > 0$  (the sound direction there). This rule is the alias of the trio; the other two are admissible.

What the checker recomputes: nothing — the rewrite is a definitional set identity, stated in both orientations because both strict forms occur in the atom grammar. The closedness that names the move is a fact about continuous observables: the closed form is a preimage of a closed ray, and the calculus's observables are polynomials.

notes The identity spends no margin — contrast the sibling moves, which each pay in one currency (a boundary cut, or an  $\varepsilon$ -band). Like *QFULLDOMAINWEAKEN*, the alias's target is a verified identity rather than a catalogue node. The  $\varepsilon$ -accounting for the inward move travels as producer annotation data on the certificate, never as a rule.

### Lean statement

The strict atom enters through the named bridge hypothesis (`hbridge`) fixing its compiled sublevel reading; the closedness lemmas ask only continuity of the observable.

BernsteinLogicAux.Structural . `closureExactLt_sound`

statement bound

```
/-- **T13 EXACT**, '<'-orientation: the complement of '{p < c}' IS the
closed atom '{c ≤ p}' — no rounding, no margin spent. -/
theorem closureExactLt_sound (p : X → ℝ) (c : ℝ) (strictAtom : Set X)
  (hbridge : strictAtom = {x | p x < c}) :
  strictAtomᶜ = {x | c ≤ p x}
```

BernsteinLogicAux.Structural . `closureExactGt_sound`

statement bound

```
/-- **T13 EXACT**, '>'-orientation. -/
theorem closureExactGt_sound (p : X → ℝ) (c : ℝ) (strictAtom : Set X)
  (hbridge : strictAtom = {x | c < p x}) :
  strictAtomᶜ = {x | p x ≤ c}
```

BernsteinLogicAux.Structural . `closureExact_isClosed_ge`

statement bound

```
/-- The closedness that names the move: '{c ≤ p}' is closed for continuous
'p'. -/
theorem closureExact_isClosed_ge [TopologicalSpace X] {p : X → ℝ}
  (hp : Continuous p) (c : ℝ) : IsClosed {x | c ≤ p x}
```

BernsteinLogicAux.Structural . `closureExact_isClosed_le`

statement bound

```
/-- Closedness, the other orientation: '{p ≤ c}' is closed for continuous
'p'. -/
theorem closureExact_isClosed_le [TopologicalSpace X] {p : X → ℝ}
  (hp : Continuous p) (c : ℝ) : IsClosed {x | p x ≤ c}
```



## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Transcription-close, both orientations bound separately (the recorded one-sidedness lesson as bind structure):  $\text{cmpl}\{p < c\} = \{c \leq p\} \mapsto \text{closureExactLt\_sound}$ , whose goal is the set identity  $\text{strictAtom}^c = \{x \mid c \leq p\ x\}$  under the System-bridge equality  $\text{hbridge} : \text{strictAtom} = \{x \mid p\ x < c\}$ ; the  $>$ -orientation  $\mapsto \text{closureExactGt\_sound}$ . The closedness that names the move  $\mapsto \text{closureExact\_isClosed\_ge/\_le}$ , and their binder row carries the one consumed regularity fact: Continuous  $p$  — the observable's continuity, which for polynomial atoms is free downstream but is named here, plus the  $[\text{TopologicalSpace } X]$  instance those two lemmas alone need. Width note:  $p : X \rightarrow \mathbb{R}$  arbitrary — the polynomial atom grammar is erased, safe direction; the identity pair is margin-free and side-condition-free exactly as displayed (no hypothesis beyond the bridge), and no completeness claim exists to state — an identity loses nothing by construction.

## Lean verifier receipts

module BernsteinLogicAux.Structural; bound build 1222cdcc326f... (bound from receipt cb0c3820c6877f22, 2026-07-29T01:03:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0b7e68907f45b27f | 2026-07-28T23:48:36Z | build+decorrelated                 | current |
| e97d5614d4f08c7e | 2026-07-29T00:00:35Z | build+decorrelated                 | current |
| 27de21f047e6b194 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| cb0c3820c6877f22 | 2026-07-29T01:03:36Z | build+replay+hardened+decorrelated | current |
| 3530841119a9a607 | 2026-07-31T22:00:22Z | build+decorrelated                 | current |
| b6cc23eb3e5b7f97 | 2026-07-31T22:29:16Z | build+decorrelated                 | current |
| bf3dacc3ab7f20ea | 2026-08-02T23:31:31Z | build+decorrelated                 | current |
| 384a8569d2005df7 | 2026-08-03T16:27:35Z | build+decorrelated                 | current |
| 717e07acf9acac8a | 2026-08-03T22:46:19Z | build+decorrelated                 | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development's object is the base calculus's wire-format checker (theory `bern_catalogue` ports the catalogue check-function cone). The closure discipline is a specification-surface rewrite layer of the extension theory: it postdates that port, and no row for any of the three moves exists in the baked theory — there is nothing for a HOL statement to be about. The box becomes fillable when a revision carries the closure moves' certificate rows and the corresponding check is ported.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template, here in the degenerate–target form: the target is a verified set identity, not a catalogue node. This section also carries the closure discipline's shared preamble; the `QCLOSUREOUTWARD` and `QCLOSUREINWARDEPS` sections build on it by reference.

**Recipe R63.1** (the closure discipline — shared preamble for the trio). The kernel's rows quantify over closed cells; its certificates consume closed atoms; its exact arithmetic decides non-strict comparisons. Strict literals therefore never reach a row: every strict atom is resolved, before rows are emitted, by one of three moves keyed to the literal's position.

| literal                        | move                                                               | cost                                                            |
|--------------------------------|--------------------------------------------------------------------|-----------------------------------------------------------------|
| complement of a strict atom    | <code>QCLOSUREEXACT</code> : replace by the closed form            | none (set identity)                                             |
| complement of a closed atom    | <code>QCLOSUREOUTWARD</code> : swap orientation, over-approximate  | the cut $\{p = c\}$                                             |
| strict atom, positive position | <code>QCLOSUREINWARDEPS</code> : round inward by $\varepsilon > 0$ | the band $(c - \varepsilon, c)$ ; mirror $(c, c + \varepsilon)$ |

The polarity logic: at a *positive* position — the atom contributes to what the certified region must lie inside — under-approximation is sound, so shrinking the atom (the inward move) is the safe direction. At a *negative* position — the atom arrives complemented — the consuming side wants the complement over-approximated, so growing it by the boundary cut (the outward move) is the safe direction. The exact move is direction-free: an identity is sound everywhere. Soundness-only is the discipline's contract: no move claims to preserve provability, and a specification tight at a critical threshold fails honestly — the certificate is refused — rather than being silently widened.

**Recipe R63.2** (the `QCLOSUREEXACT` instance).

$$\text{cmpl}\{p < c\} \rightsquigarrow \{c \leq p\} \qquad \text{cmpl}\{c < p\} \rightsquigarrow \{p \leq c\}$$

Payload transport: observable  $p$  and threshold  $c$  verbatim; orientation swapped; strictness dropped; the complement consumed. One literal  $\rightarrow$  one closed atom, and the respelling is *exact*: Proposition R63.3 is an equality, not a containment, so the rewrite is sound at every position, positive or negative — the only member of the trio with that property.

**Proposition R63.3** (the exactness identity).  $\text{cmpl}\{x \mid p(x) < c\} = \{x \mid c \leq p(x)\}$ , and symmetrically  $\text{cmpl}\{x \mid c < p(x)\} = \{x \mid p(x) \leq c\}$ .

*Proof.* Pointwise,  $\neg(p(x) < c) \iff c \leq p(x)$  — totality of the order on  $\mathbb{R}$ , nothing more. `closureExactLt_sound` and `closureExactGt_sound` are this equivalence under one set-extensionality step; there is no analytic content anywhere in the move.  $\square$

**Proposition R63.4** (the closedness that names the move). For continuous  $p$ , the atoms  $\{c \leq p\}$  and  $\{p \leq c\}$  are closed sets.

*Proof.* Each is the preimage under  $p$  of a closed ray (`closureExact_isClosed_ge`, `_le`). The calculus's observables are polynomials, hence continuous; the lemmas ask continuity only, so any continuous extension of the observable grammar inherits the move unchanged.  $\square$

**Remark R63.5** (both orientations, deliberately). The identity is stated twice because both strict forms occur in the atom grammar, and a one-sided statement of a complement lemma is a standing trap for author and porter alike. Stating both orientations costs two lines and removes the trap.

**Remark R63.6** (soundness corollary, and what the siblings inherit). Substituting equals for equals needs no direction analysis, so the soundness corollary is immediate: any judgment consuming  $\text{cmpl}\{p < c\}$  may consume  $\{c \leq p\}$  with identical denotation, and no margin is spent. The sibling moves are containments, not identities, and their soundness is position-dependent — which is exactly why the discipline routes each literal to the move whose direction its position wants (Recipe R63.1).

| proof step          | formal anchor                                                                 |
|---------------------|-------------------------------------------------------------------------------|
| Proposition R63.3   | <code>closureExactLt_sound</code> , <code>closureExactGt_sound</code>         |
| Proposition R63.4   | <code>closureExact_isClosed_ge</code> , <code>closureExact_isClosed_le</code> |
| strict-atom reading | the <code>hbridge</code> hypothesis of the identity lemmas                    |
| the sibling moves   | the <code>QCLOSUREOUTWARD</code> and <code>QCLOSUREINWARDEPS</code> sections  |

## Rule 64. QCLOSUREOUTWARD (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QClosureOutward*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{}{\text{cml}\{p \leq c\} \subseteq \{c \leq p\}} \qquad \frac{}{\text{cml}\{c \leq p\} \subseteq \{p \leq c\}} \qquad \frac{R \subseteq \llbracket \varphi \rrbracket}{\text{cml} \llbracket \varphi \rrbracket \subseteq \text{cml} R}$$

the closed swap over-approximates the true complement, losing only the cut  $\{p = c\}$  — the direction negative positions consume; atom face both orientations (*closureOutward\_sound/Ge*) + the antitone region face (*compl\_over\_approx*).

QCLOSUREOUTWARD is the closure discipline's move for complements of *closed* atoms (the discipline itself is set out in the QCLOSUREEXACT section). The true complement of  $\{p \leq c\}$  is the strict set  $\{c < p\}$  — not consumable by a row — and the move consumes the orientation swap  $\{c \leq p\}$  instead, which contains it: an over-approximation of the true complement, losing only the cut  $\{p = c\}$ . Over-approximation is exactly what the consuming side wants at negative positions: there the atom describes a set to be excluded or avoided, and a certificate against a superset is a fortiori a certificate against the set.

The warrant has two faces, both verified. The *atom* face is the pointwise containment above, in both orientations. The *region* face is the antitone kernel: a certified region under-approximates its denotation, so its complement over-approximates the denotation's complement — the direction transport that lets a negative position consume a child certificate's complement soundly. What the checker recomputes: the orientation swap itself, which is syntactic; the containments are the verified lemmas' content, spent once, globally.

The move is one-directional by design: at a *positive* position the same swap would overclaim — take  $p$  constant at  $c$ : the swap  $\{c \leq p\}$  is the whole space while the true complement of  $\{p \leq c\}$  is empty, the maximal over-claim — so the discipline's routing table, not a side condition, keeps this move on the negative side.

**notes** The conditional flag this entry once carried recorded the composite normal-form induction that pushes complements through connectives — deliberately outside this rule's own lemmas, since that induction is stated once, for all consumers, with the normal-form rewriting rules. It has since been mechanized where it lives, so the row carries no conditional flag: *denot\_nnf* (BernsteinLogicAux.NNFDuality, bound at the QSTREETTREQUESTNNF entry) is the denotation-preserving normal-form pass whose induction cases are exactly the QNNFREWRITE axioms, with *NForm.elimNeg* / *denot\_elimNeg* the consumption-boundary elimination. A reader tracing a complement through a full specification follows this rule for the atom step and the normal-form sections for the traversal; the composition-as-applied stays producer-side, which is why the traversal is cited here, not bound.

### Lean statement

RegionSat (quoted in full) is the compiled reading of the workhorse judgment kind, consumed by the region face; the atom faces fix the consumed closed swap through the bridge hypothesis (*hbridge*).

BernsteinLogicAux.Structural . **closureOutward\_sound**

statement bound

```
/-- **T13 OUTWARD**, `≤`-atom: the consumed closed swap `{c ≤ p}` contains
the true complement of `{p ≤ c}`. No margin is spent (the containment loses
only the measure-zero cut `{p = c}`). -/
theorem closureOutward_sound (p : X → ℝ) (c : ℝ) (cmlAtom : Set X)
  (hbridge : cmlAtom = {x | c ≤ p x}) :
  {x | p x ≤ c}ᶜ ⊆ cmlAtom
```

BernsteinLogicAux.Structural . **closureOutwardGe\_sound**

statement bound

```
/-- **T13 OUTWARD**, `≥`-atom orientation. -/
theorem closureOutwardGe_sound (p : X → ℝ) (c : ℝ) (cmlAtom : Set X)
  (hbridge : cmlAtom = {x | p x ≤ c}) :
  {x | c ≤ p x}ᶜ ⊆ cmlAtom
```

BernsteinLogicAux.Structural . **compl\_over\_approx**

statement bound

```
/-- The antitone region-complement kernel (region face): an
under-approximating certified region's complement OVER-approximates the
semantic complement — the direction negative positions consume. -/
theorem compl_over_approx {R den : Set X} (h : RegionSat R den) :
  denᶜ ⊆ Rᶜ
```

BernsteinLogicAux.Structural.RegionSat

definition bound

```

/-- The satisfaction judgment's workhorse kind, compiled: region `R`
under-approximates denotation `den`. -/
def RegionSat (R den : Set X) : Prop := R ≤ den

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Three displayed clauses, three bound statements. The atom face, both orientations:  $\text{cmpl}\{p \leq c\} \subseteq \{c \leq p\} \mapsto \text{closureOutward\_sound}(\text{goal}\{x \mid p \ x \leq c\}^c \subseteq \text{cmplAtom under the bridge } \text{cmplAtom} = \{x \mid c \leq p \ x\})$ , mirrored by  $\text{closureOutwardGe\_sound}$  — containments, not identities: the flavor text's “losing only the cut  $\{p = c\}$ ” is commentary on the gap, and the certified statement claims exactly the over-approximation direction, nothing finer. The region face  $R \subseteq \llbracket \varphi \rrbracket \Rightarrow \text{cmpl}\llbracket \varphi \rrbracket \subseteq \text{cmpl } R \mapsto \text{cmpl\_over\_approx} : \text{den}^c \subseteq R^c$  from  $\text{RegionSat } R \text{ den}$  — the antitone kernel, the region face of the closure discipline, two tokens of proof. Width note: this row once carried a conditional mark for a composite residue — the De Morgan *traversal* of a full specification — which is discharged: the traversal lives with the  $\text{QNnfRewrite}/\text{QStreettRequestNnf}$  normal-form machinery, deliberately not here, and is machine-checked there ( $\text{denot\_nnf} + \text{denot\_elimNeg}$ , cross-cited in the note), so the row carries no conditional flag and what remains is only that division of binding responsibility. The certified atom + region faces are unconditional set-level facts over arbitrary observables and regions; no topology, no continuity, no polynomiality is consumed anywhere in this bound set.

## Lean verifier receipts

module BernsteinLogicAux.Structural; bound build 1222cdcc326f... (bound from receipt cb0c3820c6877f22, 2026-07-29T01:03:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0b7e68907f45b27f | 2026-07-28T23:48:36Z | build+decorrelated                 | current |
| e97d5614d4f08c7e | 2026-07-29T00:00:35Z | build+decorrelated                 | current |
| 27de21f047e6b194 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| cb0c3820c6877f22 | 2026-07-29T01:03:36Z | build+replay+hardened+decorrelated | current |
| 3530841119a9a607 | 2026-07-31T22:00:22Z | build+decorrelated                 | current |
| b6cc23eb3e5b7f97 | 2026-07-31T22:29:16Z | build+decorrelated                 | current |
| bf3dacc3ab7f20ea | 2026-08-02T23:31:31Z | build+decorrelated                 | current |
| 384a8569d2005df7 | 2026-08-03T16:27:35Z | build+decorrelated                 | current |
| 717e07acf9acac8a | 2026-08-03T22:46:19Z | build+decorrelated                 | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development's object is the base calculus's wire-format checker (theory `bern_catalogue` ports the catalogue check-function cone). The closure discipline is a specification-surface rewrite layer of the extension theory: it postdates that port, and no row for any of the three moves exists in the baked theory — there is nothing for a HOL statement to be about. The box becomes fillable when a revision carries the closure moves' certificate rows and the corresponding check is ported.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. The closure discipline this move belongs to — why strict literals never reach a row, and how the three moves divide the positions — is set out in full in the QCLOSUREEXACT section; this section states only what is native to the outward move: the two verified faces of its warrant, and the direction analysis that confines it to negative positions.

The admissible template, general form, is stated once for the whole catalogue as Recipe R16.1 of the NORM section; this section instantiates it.

**Recipe R64.1** (the QCLOSUREOUTWARD instance — two faces). *Atom face.* At a negative position, the complement of a closed atom is consumed as its orientation swap:

$$\text{cmpl}\{p \leq c\} \rightsquigarrow \{c \leq p\} \qquad \text{cmpl}\{c \leq p\} \rightsquigarrow \{p \leq c\}$$

warranted by the containment  $\text{cmpl}\{p \leq c\} \subseteq \{c \leq p\}$  (Proposition R64.2): the consumed atom *over-approximates* the true complement, losing only the cut  $\{p = c\}$ .

*Region face.* A negative position consuming the complement of a *certified region*  $R$  (a child certificate's conclusion, not a syntactic atom) is warranted by the antitone kernel (Proposition R64.3): certified under-approximation of the denotation becomes over-approximation of the denotation's complement. Both faces discharge against their verified containment; the orientation swap itself is syntactic and recomputed by the checker.

**Proposition R64.2** (atom face).  $\text{cmpl}\{x \mid p(x) \leq c\} \subseteq \{x \mid c \leq p(x)\}$ , and symmetrically  $\text{cmpl}\{x \mid c \leq p(x)\} \subseteq \{x \mid p(x) \leq c\}$ .

*Proof.* If  $\neg(p(x) \leq c)$  then  $c < p(x)$ , hence  $c \leq p(x)$  — the strict fact weakened to its non-strict form (`closureOutward_sound`; the mirror is `closureOutwardGe_sound`). The two sides differ exactly by the cut  $\{p = c\}$ , which the swap adjoins.  $\square$

**Proposition R64.3** (region face — the antitone kernel). *If a certified region under-approximates a denotation,  $R \subseteq \llbracket \varphi \rrbracket$ , then  $\text{cmpl} \llbracket \varphi \rrbracket \subseteq \text{cmpl} R$ : the complement of the certificate over-approximates the complement of the denotation.*

*Proof.* Contrapositive, one line:  $x \notin \text{cmpl} R$  means  $x \in R \subseteq \llbracket \varphi \rrbracket$ , so  $x \notin \text{cmpl} \llbracket \varphi \rrbracket$  (`cmpl_over_approx`). This is the direction transport that lets negative positions consume child certificates at all: the judgment kind fixes one orientation, and complementation flips it in precisely the way the consuming side wants.  $\square$

**Remark R64.4** (what is lost, and why the direction is right). Only the cut  $\{p = c\}$  — and in the sound direction. At a negative position the atom names a set to be excluded or avoided; certifying against a superset of it is a fortiori certifying against it. The move never claims the cut is negligible — for a nondegenerate *polynomial* observable ( $p - c$  not identically zero on a full-dimensional cell) it is a measure-zero level set, but soundness nowhere consumes that: the containment alone carries the warrant, so even a degenerate observable whose cut is fat (at the extreme  $p \equiv c$ , the whole cell) only makes the certificate more conservative, never wrong.

**Remark R64.5** (one-directional by design). At a *positive* position the same swap would overclaim. Take  $p$  constant at  $c$ : the swap  $\{c \leq p\}$  is the whole space, while the true complement of  $\{p \leq c\}$  is empty — the maximal over-claim. Nothing in the move's own premises excludes this; the closure discipline's routing table (the preamble, QCLOSUREEXACT section) is the guard — the recipe's “at a negative position” names that routing assignment, not a premise the move checks for itself — sending positive-position strict literals to the inward move instead. A porter implementing the discipline ports the routing with it, or re-learns this remark the hard way.

**Remark R64.6** (what is deliberately not warranted here). The verified faces are the atom step and the region step. The composite induction — pushing complements through conjunctions, disjunctions, and nested operators until they reach atoms — is stated once, for all consumers, with the normal-form rewriting rules — this rule's warrant is deliberately partial in exactly that sense: a complement traversing a full specification rides the normal-form sections for the traversal and this rule for the atoms it arrives at.

### proof step

Proposition R64.2  
Proposition R64.3  
the judgment reading  
the routing guard  
the composite traversal

### formal anchor

`closureOutward_sound`, `closureOutwardGe_sound`  
`cmpl_over_approx`  
`RegionSat` (quoted in full above)  
the closure-discipline preamble (QCLOSUREEXACT section)  
the normal-form rewriting sections (deliberately not this rule)

## Rule 65. QCLOSUREINWARD EPS (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QClosureInwardEps*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\varepsilon \in \mathbb{Q}_{>0}}{\{p \leq c - \varepsilon\} \subseteq \{p < c\}} \qquad \frac{\varepsilon \in \mathbb{Q}_{>0}}{\{c + \varepsilon \leq p\} \subseteq \{c < p\}}$$

the producer-chosen  $\varepsilon$  elaborates a strict atom at a positive position to its rounded closed form ( $\{p < c\} \rightsquigarrow \{p \leq c - \varepsilon\}$ ,  $\{c < p\} \rightsquigarrow \{c + \varepsilon \leq p\}$ ), and the displayed containments are why that is sound: the rounded closed atom strengthens the claim (*closureInwardEpsLt\_sound/Gt*, any real  $\varepsilon > 0$  — rationality subsumed by cast); soundness-only, no completeness preservation claimed — the honest failure at a critical threshold is refusal. Closedness rider: *closureInwardEps\_isClosed*.

QCLOSUREINWARD EPS is the closure discipline's move for strict literals at positive positions (the discipline is set out in the QCLOSUREEXACT section): the literal  $\{p < c\}$  is replaced by the closed atom  $\{p \leq c - \varepsilon\}$  for a producer-chosen rational  $\varepsilon > 0$ . The rounded atom is contained in the literal, so at a position where the certified region must lie *inside* the denotation, the substitution strengthens the obligation — sound by construction — and hands the kernel a closed atom it can consume.

The  $\varepsilon$  is real spend: the band between  $c - \varepsilon$  and  $c$  is surrendered (between  $c$  and  $c + \varepsilon$  in the mirror orientation). The choice of  $\varepsilon$  lives in the producer toolkit — searchable, like a multiplier or a cover — and its accounting travels as annotation data on the certificate, not as a rule. What the checker recomputes: nothing beyond the rewritten atom; the containment is the verified lemma, spent once, globally.

The contract is soundness-only, and the honest failure mode is refusal: a specification exactly tight at its threshold — the intended witness satisfies  $p < c$  everywhere, but  $p$  approaches  $c$  on the region — admits no  $\varepsilon$ , and the certificate is refused rather than silently widened. No completeness preservation is claimed, deliberately.

notes The verified statement takes any real  $\varepsilon > 0$ ; the producer's  $\varepsilon$  being rational is a representation fact about certificates, subsumed by the cast — soundness for every positive real is a fortiori soundness for every positive rational. The closedness of the rounded atom is an instance of the exact move's closedness lemma at the shifted threshold.

### Lean statement

Both orientations of the rounding, plus the closedness rider; the rounded atom enters through the bridge hypothesis (*hbridge*) fixing its compiled sublevel reading, and  $\varepsilon$  enters as a real with  $0 < \varepsilon$  (rationality of the producer's choice is subsumed by the cast).

BernsteinLogicAux.Structural . *closureInwardEpsLt\_sound*

statement bound

```
/-- **T13 INWARD-EPS**, `<-orientation: `{p ≤ c - ε} ⊆ {p < c}` for any
`ε > 0`. Substituting the rounded closed atom at a positive position
strengthens the certified claim. -/
theorem closureInwardEpsLt_sound (p : X → ℝ) (c ε : ℝ) (hε : 0 < ε)
  (roundedAtom : Set X)
  (hbridge : roundedAtom = {x | p x ≤ c - ε}) :
  roundedAtom ⊆ {x | p x < c}
```

BernsteinLogicAux.Structural . *closureInwardEpsGt\_sound*

statement bound

```
/-- **T13 INWARD-EPS**, `>-orientation: `{c + ε ≤ p} ⊆ {c < p}`. -/
theorem closureInwardEpsGt_sound (p : X → ℝ) (c ε : ℝ) (hε : 0 < ε)
  (roundedAtom : Set X)
  (hbridge : roundedAtom = {x | c + ε ≤ p x}) :
  roundedAtom ⊆ {x | c < p x}
```

BernsteinLogicAux.Structural . *closureInwardEps\_isClosed*

statement bound

```
/-- The inward-rounding closedness rider: the rounded atom is closed
(instance of the EXACT move's closedness at the shifted threshold). -/
theorem closureInwardEps_isClosed [TopologicalSpace X] {p : X → ℝ}
  (hp : Continuous p) (c ε : ℝ) : IsClosed {x | p x ≤ c - ε}
```



## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Both displayed containments bound, orientation for orientation:  $\{p \leq c - \varepsilon\} \subseteq \{p < c\} \mapsto \text{closureInwardEpsLt\_sound}$  (the rounded atom entering through the bridge  $\text{roundedAtom} = \{x \mid p \ x \leq c - \varepsilon\}$ , the strict margin as  $\text{hE} : 0 < \varepsilon$ ), and the  $>$ -orientation  $\mapsto \text{closureInwardEpsGt\_sound}$ ; the inward-rounding closedness rider (the rounded atom is closed)  $\mapsto \text{closureInwardEps\_isClosed}$  (continuity consumed there alone, the  $\text{QClosureExact}$  pattern). Two width facts, one per direction of the display: the certified statements take *any real*  $\varepsilon > 0$  where the display asks  $\varepsilon \in \mathbb{Q}_{>0}$  — producer rationality subsumed by cast, divergence D3, statement wider in the safe direction; and no completeness–preservation statement exists in the module — the flavor text's “soundness-only” is certified by absence, with total loss at a critical cut remaining the honest refusal mode, exactly as the ledger inherits it.

## Lean verifier receipts

module BernsteinLogicAux.Structural; bound build 1222cdcc326f... (bound from receipt cb0c3820c6877f22, 2026-07-29T01:03:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0b7e68907f45b27f | 2026-07-28T23:48:36Z | build+decorrelated                 | current |
| e97d5614d4f08c7e | 2026-07-29T00:00:35Z | build+decorrelated                 | current |
| 27de21f047e6b194 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| cb0c3820c6877f22 | 2026-07-29T01:03:36Z | build+replay+hardened+decorrelated | current |
| 3530841119a9a607 | 2026-07-31T22:00:22Z | build+decorrelated                 | current |
| b6cc23eb3e5b7f97 | 2026-07-31T22:29:16Z | build+decorrelated                 | current |
| bf3dacc3ab7f20ea | 2026-08-02T23:31:31Z | build+decorrelated                 | current |
| 384a8569d2005df7 | 2026-08-03T16:27:35Z | build+decorrelated                 | current |
| 717e07acf9acac8a | 2026-08-03T22:46:19Z | build+decorrelated                 | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development's object is the base calculus's wire-format checker (theory `bern_catalogue` ports the catalogue check-function cone). The closure discipline is a specification-surface rewrite layer of the extension theory: it postdates that port, and no row for any of the three moves exists in the baked theory — there is nothing for a HOL statement to be about. The box becomes fillable when a revision carries the closure moves' certificate rows and the corresponding check is ported.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof



Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe. The closure discipline this move belongs to is set out in full in the `QCLOSUREEXACT` section; this section states what is native to the inward move: the rounding containment in both orientations, the closedness rider, and the two honesty clauses — strictness of  $\varepsilon > 0$ , and the deliberate absence of any completeness claim.

The admissible template, general form, is stated once for the whole catalogue as Recipe **R16.1** of the `NORM` section; this section instantiates it.

**Recipe R65.1** (the `QCLOSUREINWARDEPS` instance). At a positive position, a strict literal rounds inward by a producer-chosen  $\varepsilon > 0$ :

$$\{p < c\} \rightsquigarrow \{p \leq c - \varepsilon\} \qquad \{c < p\} \rightsquigarrow \{c + \varepsilon \leq p\}$$

warranted by the containment  $\{p \leq c - \varepsilon\} \subseteq \{p < c\}$  (Proposition **R65.2**): the rounded closed atom *strengthens* the certified claim, which is the sound direction where the certified region must lie inside the denotation. Payload transport: observable and threshold verbatim, threshold shifted by  $\varepsilon$ , strictness dropped; the  $\varepsilon$  itself travels as certificate annotation data, never as a rule premise.

**Proposition R65.2** (the rounding containment). *For every  $\varepsilon > 0$ :  $\{x \mid p(x) \leq c - \varepsilon\} \subseteq \{x \mid p(x) < c\}$ , and symmetrically  $\{x \mid c + \varepsilon \leq p(x)\} \subseteq \{x \mid c < p(x)\}$ .*

*Proof.*  $p(x) \leq c - \varepsilon < c$ , the second inequality because  $\varepsilon > 0$  — one line of linear arithmetic (`closureInwardEpsLt_sound`; mirror `closureInwardEpsGt_sound`). The verified statements take any *real*  $\varepsilon > 0$ ; the producer's  $\varepsilon$  being rational is a representation fact about certificates, subsumed by the cast.  $\square$

**Proposition R65.3** (closedness rider). *For continuous  $p$ , both rounded atoms are closed sets:  $\{p \leq c - \varepsilon\}$  and  $\{c + \varepsilon \leq p\}$ .*

*Proof.* An instance of the exact move's closedness at the shifted threshold  $c - \varepsilon$  (`closureInwardEps_isClosed`, one application of `closureExact_isClosed_le`); the mirrored orientation is  $\{c + \varepsilon \leq p\} = p^{-1}([c + \varepsilon, \infty))$ , closed by the same continuity — the exact move's  $\geq$ -orientation closedness at threshold  $c + \varepsilon$  (the cited binding covers the  $\leq$  orientation; the mirror discharges against the  $\geq$  twin). The move thus delivers exactly what the discipline asked of it: a closed atom the kernel can consume.  $\square$

**Remark R65.4** ( $\varepsilon > 0$  strictly — the boundary counterexample). At  $\varepsilon = 0$  the containment fails:  $\{p \leq c\}$  is not contained in  $\{p < c\}$ , and the failure can be total — take  $p$  constant at  $c$ : the un-rounded atom is the whole space while the literal is empty. The strict positivity of  $\varepsilon$  is therefore the whole rule, in the same sense that the sign row is half of `HYPMULTIPLIER`: remove it and the move manufactures the maximal over-claim from nothing.

**Remark R65.5** (soundness-only, and the refusal posture). No completeness preservation is claimed, deliberately. The  $\varepsilon$ -band is surrendered —  $(c - \varepsilon, c)$  in the first orientation,  $(c, c + \varepsilon)$  in the mirror — and a specification exactly tight at its threshold — the intended witness satisfies  $p < c$  everywhere while  $p$  approaches  $c$  on the region — admits no  $\varepsilon$  at all: every rounding excludes points the witness needs. The honest outcome is refusal (no certificate exists through this move), never silent widening; a producer meeting refusal here either finds slack elsewhere in the specification or reports the tightness, and the discipline's contract is that the checker is never the party doing the forgiving.

**Remark R65.6** (why not under a complement). Rounding an atom that will be complemented is doubly wrong as a discharge of *this* obligation: the complement of the rounded atom,  $\{c - \varepsilon < p\}$ , is strict again — the closedness goal is defeated, and the  $\varepsilon$  was spent for nothing. Complemented literals belong to the exact move (free, when the inner literal is strict) or the outward move (when the inner atom is closed); the routing table in the closure-discipline preamble (`QCLOSUREEXACT` section) assigns every case exactly one move, and the assignment is what makes the trio a discipline rather than three tricks.

| proof step               | formal anchor                                                                       |
|--------------------------|-------------------------------------------------------------------------------------|
| Proposition <b>R65.2</b> | <code>closureInwardEpsLt_sound</code> , <code>closureInwardEpsGt_sound</code>       |
| Proposition <b>R65.3</b> | <code>closureInwardEps_isClosed</code> (via <code>closureExact_isClosed_le</code> ) |
| rounded-atom reading     | the <code>hbridge</code> hypothesis of the rounding lemmas                          |
| $\varepsilon$ accounting | certificate annotation data, producer-side (not a rule)                             |
| the routing guard        | the closure-discipline preamble ( <code>QCLOSUREEXACT</code> section)               |

## Rule 66. TAYLORBOUND (alias for PCONSTEXACT)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTS�-rules.toml — *TaylorBound*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{q \in \mathbb{Q}; \quad \theta \in \mathbb{Q}_{>0}; \quad T \in \mathbb{N}; \quad n \in \mathbb{N} \quad q \leq \sum_{k \leq n} (\theta T)^k / k! \text{ holds under exact evaluation in } \mathbb{Q}}{\Phi \vdash q \leq \sum_{k \leq n} (\theta T)^k / k!} \text{ TAYLORBOUND}$$

one PCONSTEXACT application, displayed at the deadline battery's Taylor comparison — every term is wire-free (closed), so the comparison is decided by the verified closed evaluator; no QTS� sort exists and none is needed. The transcendental fact  $\sum_{k \leq n} (\theta T)^k / k! \leq e^{\theta T}$  belongs to the consuming lemma, never the checker: the checker never needs to know  $e$  exists.

TAYLORBOUND supplies the deadline battery's rational surrogate for a transcendental bound. The deadline rule's soundness argument wants  $q \leq e^{\theta T}$  for a producer-chosen rational  $q$ , with decay rate  $\theta \in \mathbb{Q}_{>0}$  and horizon  $T \in \mathbb{Q}_{\geq 0}$  from its payload: its concluded sublevel region  $\{V \leq (1-p)q\}$  grows with  $q$ , so the producer wants the sharpest  $q$  the arithmetic affords. No checker doing exact rational arithmetic can evaluate  $e^{\theta T}$ , and none needs to: the certificate posts the single wire-free comparison  $q \leq \sum_{k \leq n} (\theta T)^k / k!$  — a finite sum of rationals, checked in exact  $\mathbb{Q}$  — and the consuming lemma carries the enclosure  $\sum_{k \leq n} (\theta T)^k / k! \leq e^{\theta T}$  (nonnegative Taylor remainder at nonnegative argument), so  $q \leq e^{\theta T}$  follows without the checker ever knowing  $e$  exists. A producer picks the truncation order  $n$  and any convenient  $q$  at or below the partial sum.

What the checker recomputes is the target rule's entire check: exact closed evaluation of one comparison. No certificate ever carries a TAYLORBOUND node — no sort exists and none is needed; the producer emits the ground-fact row directly, and this row is the catalogue-level name for that elaboration.

What would break without each hypothesis: with rounded rather than exact evaluation,  $q$  could creep above the true partial sum and the surrogate above  $e^{\theta T}$ ; without wire-freedom the “constant” could depend on state and the enclosure is meaningless; and the sign condition  $\theta T \geq 0$  is load-bearing in the consuming lemma — at negative arguments the Taylor remainder alternates in sign and the enclosure direction fails at alternate truncation orders (the proof section carries the counterexample).

**notes** Deliberately *not* aliased onto the interval-enclosure rule: that would import witness-bracket machinery for a metatheory one-liner, and the enclosure rule today lives only in a separate reference-checker implementation besides. The seam this rule instantiates is a design invariant of the whole extension page: transcendental facts live in soundness lemmas (the Lean-side deadline lemma consumes the standard library's partial-sum-below-exponential fact), arithmetic lives in the checker. Mechanization state, plainly: the bound statements below are the enclosure and its exact- $\mathbb{Q}$  face. `taylorLeaf_le_exp` (module `BernsteinLogicAux.QtslAliases`) is the enclosure step —  $q$  at or below the truncated sum is below  $e^x$  at nonnegative argument, the nonnegative Taylor remainder — and `taylorLeaf_rat_le_exp` is the row end to end: the wire-free comparison exactly as the checker replays it, in  $\mathbb{Q}$ , with the  $(\theta, T)$  payload shape, concluding  $q \leq e^{\theta T}$ . The consuming site is `sound_QDeadlineBattery's hq` premise (the target's entry), whose internal enclosure step these statements name as a rule-level fact. The sign condition  $0 \leq \theta T$  is a stated hypothesis, exactly as load-bearing as the commentary's counterexample says it is. No profile enters; the entry carries no conditional flag. The target's comparator exclusion ( $\neq$  inadmissible) travels with the alias to any port.

### Lean statement

`BernsteinLogicAux.QtslAliases . taylorLeaf_le_exp`

statement bound

```
-- **The enclosure step** (QTS�-rules.toml row `TaylorBound (T7)`): a
-- producer-chosen `q` at or below the order-`n` Taylor partial sum of
-- `exp` at a nonnegative argument is below `exp` itself — the nonnegative
-- Taylor remainder (`Real.sum_le_exp_of_nonneg`). The sign condition
-- `0 ≤ x` is load-bearing: at negative arguments the remainder alternates
-- in sign, so the enclosure fails at alternate truncation orders (at
-- `x = -1` it fails at `n = 0`, since `1 > e⁻¹`, and holds at `n = 1`) —
-- there is no all-`n` enclosure to be had (counter-witnesses in the
-- book's proof section). -/
theorem taylorLeaf_le_exp {q x : ℝ} {n : ℕ} (hx : 0 ≤ x)
  (hq : q ≤ ∑ k ∈ range (n + 1), x ^ k / (Nat.factorial k : ℝ)) :
  q ≤ Real.exp x
```

`BernsteinLogicAux.QtslAliases . taylorLeaf_rat_le_exp`

statement bound

```

/-- **TaylorBound`, the exact- $\mathbb{Q}$  face, end to end** (rule row
`TaylorBound (T7)` – the single wire-free closed comparison, checked in
exact  $\mathbb{Q}$ , that stands in for ` $q \leq e^{\{0T\}}$ ): decay rate ` $\theta \in \mathbb{Q}_{\geq 0}$ ` and
horizon ` $T \in \mathbb{N}$ ` from the deadline payload, truncation order ` $n$ `
producer-chosen. The checker replays the hypothesis; the conclusion –
which mentions ` $e$ ` – lives only here (the catalogue's "the checker never
needs to know ` $e$ ` exists"). Consuming site: `sound_QDeadlineBattery`'s
`hq` premise. -/
theorem taylorLeaf_rat_le_exp {qq  $\theta$  :  $\mathbb{Q}$ } {T n :  $\mathbb{N}$ } (h $\theta$  :  $\theta \leq 0$ )
  (hleaf : qq  $\leq \sum k \in \text{range } (n + 1), (\theta * T) ^ k / (\text{Nat.factorial } k : \mathbb{Q})$ ) :
  (qq :  $\mathbb{R}$ )  $\leq \text{Real.exp } (\theta * T)$ 

```

The payload-level elaborator, displayed whole.  $(q, \theta, T, n)$  to the one PConstExact payload, the right side expanded producer-side in exact  $\mathbb{Q}$ ; the checker replays one closed comparison, and the transcendental conclusion appears only in `taylorElab_le_exp`'s goal.

BernsteinLogic.QtslElaborators . **taylorFormula\_true\_iff**

statement bound

```

/-- The wire-level replay identity: the checker's `true` verdict is
exactly the  $\mathbb{Q}$  comparison against the partial sum. -/
theorem taylorFormula_true_iff (q  $\theta$  : Rat) (T n : Nat) :
  (taylorFormula q  $\theta$  T n).evalClosed = some true  $\leftrightarrow q \leq \text{taylorPartialSum } \theta T n$ 

```

BernsteinLogic.QtslElaborators . **taylorElab\_sound**

statement bound

```

/-- The factoring, BL face: `sound_pConstExact` at the elaborated
payload (`prop:tb-sound`'s first half – validity of the posted
comparison at every valuation). -/
theorem taylorElab_sound {H : Hypernet} {seq : Sequent} {q  $\theta$  : Rat} {T n : Nat}
  (hEval : (taylorFormula q  $\theta$  T n).evalClosed = some true)
  (hpost : seq.post = taylorFormula q  $\theta$  T n) :
  seq.semanticallyValid H

```

BernsteinLogic.QtslElaborators . **taylorElab\_le\_exp**

statement bound

```

/-- The semantic face, end to end (`prop:tb-sound` composed with
`prop:tb-enclosure`): the checked wire comparison bounds ` $q$ ` by
`e^{\{0T\}}` through the bound enclosure – the elaborator factoring through
the kernel replay into `taylorLeaf_rat_le_exp`. The sign condition
` $\theta \leq 0$ ` is load-bearing (the proof section's negative-argument
counterexample); the deadline rule's typing supplies it. -/
theorem taylorElab_le_exp {q  $\theta$  : Rat} {T n : Nat} (h $\theta$  :  $\theta \leq 0$ )
  (hEval : (taylorFormula q  $\theta$  T n).evalClosed = some true) :
  (q :  $\mathbb{R}$ )  $\leq \text{Real.exp } (\theta * T)$ 

```

BernsteinLogic.QtslElaborators . **taylorPartialSum**

definition bound

```

/-- The order- $n$  Taylor partial sum of ` $\text{exp } (\theta * T)$ `, computed in exact
 $\mathbb{Q}$  – the producer-side expansion of `rec:tb-transport`'s right side. -/
def taylorPartialSum ( $\theta$  : Rat) (T n : Nat) : Rat :=
   $\sum k \in \text{Finset.range } (n + 1), (\theta * (T : \text{Rat})) ^ k / (\text{Nat.factorial } k : \text{Rat})$ 

```

BernsteinLogic.QtslElaborators . **taylorElab**

definition bound

```

/-- **The TaylorBound payload elaborator** (`rec:tb-transport`): (q,  $\theta$ ,
T, n) plus the checked comparison, to the one `PConstExactData`
inhabitant. No `TaylorBound` node sort exists and none is needed
(`rec:tb-template`) – this function is the catalogue-level name for the
producer's elaboration, mechanized. -/
def taylorElab (H : Hypernet) (seq : Sequent) (q  $\theta$  : Rat) (T n : Nat)
  (hEval : (taylorFormula q  $\theta$  T n).evalClosed = some true)
  (hpost : seq.post = taylorFormula q  $\theta$  T n) :
  PConstExactData H seq :=
  .mk H seq (.const q) (.const (taylorPartialSum  $\theta$  T n)) .le (by decide)
  hEval hpost

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed wire-free comparison  $q \leq \sum_{k \leq n} (\theta T)^k / k!$  standing in for  $q \leq e^{\theta T} \mapsto$  the bound pair: `taylorLeaf_le_exp` (the enclosure step — partial sum below `Real.exp` at nonnegative argument, the `nonneg-remainder` citation) and `taylorLeaf_rat_le_exp`, the end-to-end exact- $\mathbb{Q}$  face: `qq`  $\theta : \mathbb{Q}, T : \mathbb{N}$ , hypothesis the exact- $\mathbb{Q}$  partial-sum comparison the checker replays, conclusion  $(qq : \mathbb{R}) \leq \text{Real.exp } (\theta * T)$  — the flavor text's “the checker never needs to know  $e$  exists” as a binder fact:  $e$  appears in the goal alone, never in a hypothesis. The sign condition  $\mapsto$  `h0`  $\theta \leq 0$  (load-bearing — the alternating-remainder counterexample in the proof section is why the binder exists); the truncation order  $n$  is producer-chosen, unconstrained. Width note: the consuming site (`sound_QDeadlineBattery`'s `hq` premise) is cited, not bound here — correct package-boundary behaviour, the deadline battery's own entry carries it; the displayed horizon  $T \in \mathbb{N}$  matches the bound end-to-end lemma's  $T : \mathbb{N}$  exactly (the rational-horizon generalization is mathematically available from `taylorLeaf_le_exp` but is neither bound nor displayed); and the displayed  $\theta \in \mathbb{Q}_{>0}$  is a deliberately stronger surface restriction than the bound `h0`  $\theta \leq 0$  — safe direction, the deadline battery's own decay-rate positivity.

### Lean verifier receipts

module `BernsteinLogicAux.QtslAliases`; bound build 570599779c29... (bound from receipt 5a60f4d01fd81ac5, 2026-07-21T05:39:02Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0a3819a978e31127 | 2026-07-21T04:51:36Z | build+decorrelated                 | current |
| 0bb1074df705d863 | 2026-07-21T05:21:58Z | build+decorrelated                 | current |
| 5a60f4d01fd81ac5 | 2026-07-21T05:39:02Z | build+replay+hardened+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.QtslElaborators`; bound build 0205ec213b91... (hashed from the built olean at bind time). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a237f4cff27cb15a | 2026-07-22T10:06:47Z | build+decorrelated | current |

#### gate-layer coverage of the bound build:

|                                                           |         |
|-----------------------------------------------------------|---------|
| build (lake build (elaboration))                          | current |
| replay (leanchecker (official kernel replay))             | current |
| hardened (safe_verify (hardened screen))                  | current |
| decorrelated (lean4export + nanoda (decorrelated kernel)) | current |

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

### HOL4 statement

#### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*, not by backlog: the wire format never carries a `TAYLORBOUND` node (no sort exists and none is needed — the producer emits the ground-fact comparison row directly), so no wire certificate a HOL checker reads could ever contain one; HOL coverage for this rule *is* the target rule's. Today even that statement is not writable: the HOL4 development (theory `bern_catalogue`, the base catalogue's check-function cone) predates `PCONSTEXACT`, whose wire tag is unallocated — see that rule's section for the honest state of the target's own box.

### HOL verifier receipts

**NONE**

no HOL verification receipt exists for this rule.

**Proof**

Genre: alias rule (tier 3) — a respelling template. The five artefacts an alias owes the book: the 1→1 unpack, the filled-in target application, the payload-transport table, the exactness identity, and the soundness corollary through the target's lemma. All five follow, with one honest asymmetry marked where it occurs: this rule's exactness identity is not a ring identity but an *enclosure* — the Taylor-remainder inequality — and it lives in the consuming lemma, never in the checker. Mechanization state: the target's lemma is kernel-verified; the alias-level semantic content — the enclosure and its exact- $\mathbb{Q}$  face — is mechanized and bound in the entry, and so — now that the extension certificate schema exists — is the payload-level elaborator (the remark below names the bound functions).

**Recipe R66.1** (the template instance — no node sort, by design). The deadline rule (QDEADLINEBATTERY) wants the fact  $q \leq e^{\theta T}$  for a producer-chosen rational surrogate  $q$ , with decay rate  $\theta \in \mathbb{Q}_{>0}$  and horizon  $T \in \mathbb{Q}_{\geq 0}$  from its payload: its concluded sublevel region  $\{V \leq (1-p)q\}$  grows with  $q$ , so the producer wants the sharpest  $q$  the arithmetic affords. No checker doing exact rational arithmetic can evaluate  $e^{\theta T}$ , and none needs to: the certificate posts the single wire-free comparison

$$q \leq \sum_{k \leq n} (\theta T)^k / k!$$

for a producer-chosen truncation order  $n$  — a finite sum of rationals, one closed comparison — and the consuming lemma carries the enclosure to  $e^{\theta T}$  (Proposition R66.4). The unpack image is a single PCONSTEXACT node; indeed no TAYLORBOUND node sort exists and none is needed — the producer emits the ground-fact row directly, and this rule is the catalogue-level name for that elaboration. One node → one node, nothing derived.

**Recipe R66.2** (the filled-in application).

$$\frac{q \leq \sum_{k \leq n} (\theta T)^k / k! \text{ holds under exact evaluation in } \mathbb{Q}}{\Phi \vdash q \leq \sum_{k \leq n} (\theta T)^k / k!} \text{ PCONSTEXACT}$$

with arbitrary premise  $\Phi$ , as the target licenses (the rule is premise-free). Both sides are closed rational terms — the sum is finite, every summand exact — so the target's verified evaluator decides the comparison in exact  $\mathbb{Q}$ , and the comparator ( $\leq$ ) lies inside the target's admissible comparator set.

**Recipe R66.3** (payload transport — field for field).

| TAYLORBOUND data                      | PCONSTEXACT payload                                                         |
|---------------------------------------|-----------------------------------------------------------------------------|
| $q$ (the rational surrogate)          | left side of the comparison                                                 |
| $\theta, T, n$ (rate, horizon, order) | the right side $\sum_{k \leq n} (\theta T)^k / k!$ , expanded producer-side |
| comparator                            | $\leq$ (inside the admissible set)                                          |
| wire-freedom                          | implied, not declared: the evaluator returns none on any wire               |

The checker sees one closed comparison and nothing else;  $\theta$ ,  $T$ , and  $n$  survive only through the value of the right-hand term. The transcendental target  $e^{\theta T}$  appears in *no* field — that is the point.

**Proposition R66.4** (the enclosure — the exactness identity, honest form). *For  $\theta T \geq 0$  and any  $n$ ,*

$$\sum_{k \leq n} (\theta T)^k / k! \leq e^{\theta T}, \quad \text{hence} \quad q \leq \sum_{k \leq n} (\theta T)^k / k! \Rightarrow q \leq e^{\theta T}.$$

*Proof.* Two lines, though of a different species than a ring identity: at  $x := \theta T \geq 0$  every term of the exponential series is nonnegative, so each partial sum is dominated by the limit — in the classical Taylor form,  $e^x = \sum_{k \leq n} x^k / k! + R_n$  with the Lagrange remainder  $R_n = e^\xi x^{n+1} / (n+1)! \geq 0$  for some  $\xi \in [0, x]$  [214]. The sign condition is load-bearing: at negative arguments the remainder alternates in sign and the enclosure direction fails at alternate truncation orders ( $n=0$ ,  $x=-1$ : the partial sum is  $1 > e^{-1}$ ; at  $n=1$  it happens to hold again — there is no all- $n$  enclosure to be had) — the deadline rule's typing ( $\theta > 0$ ,  $T \geq 0$ ) is what entitles the lemma to the inequality, via  $\theta T \geq 0$ . Unlike the base-calculus exemplar alias, whose two-line identity is checked by the kernel as exact polynomial arithmetic, this identity is consumed *inside the deadline rule's soundness lemma* (the Lean side consumes the standard library's partial-sum-below-exponential fact, `Real.sum_le_exp_of_nonneg`, there) and never reaches the checker: the checker's whole obligation is the exact rational comparison of Recipe R66.2. The checker never needs to know  $e$  exists.  $\square$

**Proposition R66.5** (soundness corollary). *If the posted comparison checks and  $\theta T \geq 0$  (the sign hypothesis of Proposition R66.4), then  $q \leq e^{\theta T}$ , which is the fact the deadline rule's lemma consumes.*

*Proof.* `sound_pConstExact` (the target's kernel-verified lemma, quoted in the PCONSTEXACT section) turns the true evaluation into the validity of  $q \leq \sum_{k \leq n} (\theta T)^k / k!$  at every valuation; Proposition R66.4 extends to  $e^{\theta T}$  inside the consuming lemma. There is no other content — the alias genre working as intended.  $\square$

**Remark R66.6** (the payload-level elaborator, mechanized). The *semantic* content of the respelling is bound in the entry: `taylorLeaf_le_exp` is Proposition R66.4's enclosure step, and `taylorLeaf_rat_le_exp` is Proposition R66.5's exact- $\mathbb{Q}$  face end to end. The payload-level elaborator — previously this page's one absent artefact, held back until the extension certificate

schema existed — is bound with it: `taylorElab` (module `BernsteinLogic.QtsLElaborators`) takes  $(q, \theta, T, n)$  to exactly the one `PConstExact` payload of Recipe [R66.3](#), the right side expanded producer-side in exact  $\mathbb{Q}$  (`taylorPartialSum`), as an inhabitant of the target's own payload type — the comparator confinement discharged at elaboration, once — with `taylorElab_sound` the factoring through `sound_pConstExact`, `taylorFormula_true_iff` the wire-level replay identity, and `taylorElab_le_exp` the semantic face end to end (the checked comparison bounding  $q$  by  $e^{\theta T}$  through the bound enclosure — the transcendental conclusion still appears in no field, only in the goal). Stated at the discrete horizon,  $T : \mathbb{N}$ , matching the bound exact- $\mathbb{Q}$  face — the entry's width note carries the divergence from the recipe's rational-time payload; the enclosure of Proposition [R66.4](#) itself needs only  $\theta T \geq 0$ .

*Remark R66.7* (deliberately not an enclosure-rule alias). The calculus also has interval-enclosure machinery with witness brackets. Respelling this leaf onto it was considered and declined: that would import witness-bracket payload for a metatheory one-liner, and the enclosure rule today lives only in a separate reference-checker implementation besides. The seam this rule instantiates is a design invariant of the whole extension page: transcendental facts live in soundness lemmas, arithmetic lives in the checker.

*Remark R66.8* (the inherited comparator confinement). `PConstExact` excludes one comparator ( $\neq$ ) from its admissible set, for reasons of a second soundness invariant unpacked in that rule's section. This leaf uses  $\leq$  only — comfortably inside — but any port or mechanization of this rule must *preserve* the exclusion rather than re-derive the comparator set from the semantic lemma alone; the flag travels with the alias.

| proof step                                                | formal anchor                                                                                                                                                                                                                                    |
|-----------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| target lemma (Prop. <a href="#">R66.5</a> )               | <code>sound_pConstExact</code> ; <code>Formula.holds_of_evalClosed_true</code> (quoted in the <code>PConstExact</code> section)                                                                                                                  |
| wire-freedom implied at the leaf                          | <code>Formula.evalClosed</code> (none on any wire; same section)                                                                                                                                                                                 |
| the enclosure, mechanized where consumed                  | <code>Real.sum_le_exp_of_nonneg</code> inside the deadline lemma<br><code>sound_QDeadlineBattery</code> (see that rule's entry)                                                                                                                  |
| the enclosure + exact- $\mathbb{Q}$ face, rule-level      | <code>taylorLeaf_le_exp</code> , <code>taylorLeaf_rat_le_exp</code> (bound in the entry)                                                                                                                                                         |
| payload-level elaborator ( <a href="#">Remark R66.6</a> ) | <code>taylorElab</code> + <code>taylorElab_sound</code> + <code>taylorElab_le_exp</code> ,<br>over <code>taylorPartialSum</code> + <code>taylorFormula_true_iff</code> (module <code>BernsteinLogic.QtsLElaborators</code> , bound in the entry) |



## Rule 67. ROOTEval (alias for PConstExact)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *RootEval*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{R = \{x : V(x) \leq y\} \text{ (sublevel root atom)} \quad \vdash V(x_0) \leq y \text{ (one PConstExact leaf: the substituted comparison is closed)}}{\vdash x_0 \in R}$$

$$\frac{R = \prod_i [a_i, b_i] \text{ (box root atom)} \quad \forall i. a_i \leq (x_0)_i \leq b_i \text{ (recomputed in exact } \mathbb{Q})}{\vdash x_0 \in R}$$

catalogued as the eval leaf of the region-to-satisfaction bridge (a BL Proof leaf; the bridge's node sort is out of scope of the current baked theory — its baked warrant seam is `qsat_warrant`); sublevel atoms discharge through PConstExact (the first form is that rule's application at the substituted point), box atoms through the registry interval check (the second form: conformance recomputation, not a rule application).

ROOTEval is the membership leaf itself, catalogued as a rule of its own because every initialized certificate ends with one: membership of the specification's initial point in the *root atom* — the region atom concluded at the certificate's root (“root” names the certificate's root claim, not roots of polynomials) — by exact rational evaluation. In the certificate schema it is not a node sort at all: it is the eval leaf of the region-to-satisfaction bridge (QSatFromRegion — whose own node sort is out of scope of the current baked theory; the baked warrant seam is `qsat_warrant`), a base-proof leaf. For a sublevel atom  $\{x : V(x) \leq y\}$  the leaf is one PConstExact application: substitute the rational point into the polynomial — exact, no rounding anywhere — and decide one closed comparison. For a box atom, the checker recomputes per-coordinate containment directly (the side-condition registry: conformance recomputation, not a rule application).

What the checker recomputes: everything — the leaf *is* a recompute; there is no payload beyond the point and the atom. What would break without exactness: under naive (non-directed) rounding a point lying  $\varepsilon$  outside the region could certify membership, and the bridge rule would conclude satisfaction from a false premise — directed outward rounding keeps that direction sound but surrenders the equivalence at the boundary — the exactness identity in the proof section is a literal identity of values, and it is the whole rule. The comparator exclusion inherited from the target ( $\neq$  inadmissible, a second-soundness-invariant exclusion unpacked in the target's section) must survive any port.

**notes** The judgment-level composition — how the checked leaf feeds the instantiation that concludes root satisfaction — lives in the QSatFromRegion section; this row owns the leaf. Mechanization state, plainly: the bound statements below are the exactness identity itself. `rootEval_eval_cast` (module `BernsteinLogicAux.QtslAliases`) is the literal identity of values — evaluating the  $\mathbb{Q}$ -coefficient polynomial at the rational point and casting equals evaluating the cast polynomial at the cast point — and `rootEval_sublevel_iff` / `rootEval_box_iff` are its two atom shapes, each an *equivalence*: the exact- $\mathbb{Q}$  comparison decides the real-semantics membership in both directions, which is precisely the content a rounded evaluator would lose. The statements are over the generic polynomial-evaluation vocabulary (`MvPolynomial` over  $\mathbb{Q}$ , cast to  $\mathbb{R}$ ); the wire-term evaluator that produces the  $\mathbb{Q}$ -comparison is the target's own kernel-verified theorem, quoted in the PConstExact section, and the box-atom branch's registry recompute is checker-side by design. No profile enters; the entry carries no conditional flag.

### Lean statement

`BernsteinLogicAux.QtslAliases . rootEval_eval_cast`

statement bound

```
/-- **The exactness identity** (QTSL-rules.toml row `RootEval (T19)` – “a
literal identity of values, and it is the whole rule”): evaluating a
 $\mathbb{Q}$ -coefficient polynomial at a rational point and casting the result equals
evaluating the cast polynomial at the cast point. This is why the checker's
exact rational arithmetic (`PConstExact`'s replay) is the real-semantics
evaluation, with no rounding gap anywhere. -/
theorem rootEval_eval_cast {σ : Type*} (P : MvPolynomial σ  $\mathbb{Q}$ ) (xq : σ →  $\mathbb{Q}$ ) :
  ((MvPolynomial.eval xq P :  $\mathbb{Q}$ ) :  $\mathbb{R}$ )
    = MvPolynomial.eval (fun i => (xq i :  $\mathbb{R}$ )) (MvPolynomial.map (Rat.castHom  $\mathbb{R}$ ) P)
```

`BernsteinLogicAux.QtslAliases . rootEval_sublevel_iff`

statement bound

```
/-- **`RootEval`, sublevel atom, as an equivalence**: the exact  $\mathbb{Q}$  comparison
at the initial point decides membership of the cast point in the cast
```



```

sublevel atom `{x : V(x) ≤ thr}` – both directions, which is the content a
rounded evaluator would lose (one direction would survive rounding; the
identity does not). The forward direction is the leaf as the checker
discharges it (rule row `RootEval (T19)`, sublevel branch →
`PConstExact`). -/
theorem rootEval_sublevel_iff {σ : Type*} (P : MvPolynomial σ ℚ) (xq : σ → ℚ)
  (thr : ℚ) :
  MvPolynomial.eval xq P ≤ thr
  ↔ MvPolynomial.eval (fun i => (xq i : ℝ))
    (MvPolynomial.map (Rat.castHom ℝ) P) ≤ (thr : ℝ)

```

BernsteinLogic.QtslAliases . **rootEval\_box\_iff**

statement bound

```

/-- **`RootEval`, box atom, as an equivalence**: per-coordinate exact ℚ
containment decides containment of the cast point in the cast box (rule row
`RootEval (T19)`, box branch – the side-condition registry's per-coordinate
recompute). -/
theorem rootEval_box_iff {ι : Type*} (aq bq xq : ι → ℚ) :
  (∀ i, aq i ≤ xq i ∧ xq i ≤ bq i)
  ↔ ∀ i, ((aq i : ℝ) ≤ (xq i : ℝ) ∧ (xq i : ℝ) ≤ (bq i : ℝ))

```

**The payload-level elaborator, displayed whole.** The closed left-hand term is built by verified substitution (polyClosedTerm; its evaluation lemma says the checker's exact replay computes the value  $V(x_0)$  itself), and the elaborated payload inhabits the target's own payload type — comparator confinement discharged at elaboration.

BernsteinLogic.QtslElaborators . **polyClosedTerm\_evalQ**

statement bound

```

@[simp] theorem polyClosedTerm_evalQ (x0 : Nat → Rat) :
  ∀ p : Poly, (polyClosedTerm x0 p).evalQ? = some (Poly.eval x0 p)
| [] => by simp [polyClosedTerm, Poly.eval, Term.evalQ?]
| mc :: rest => by
  have ih

```

BernsteinLogic.QtslElaborators . **rootEvalFormula\_true\_iff**

statement bound

```

/-- The wire-level exactness identity, ℚ face (`prop:re-exact` at the
payload: the checker's `true` verdict IS the exact membership
inequality – both directions; the ℝ face stays
`Aliases.rootEval_sublevel_iff`). -/
theorem rootEvalFormula_true_iff (x0 : Nat → Rat) (V : Poly) (thr : Rat) :
  (rootEvalFormula x0 V thr).evalClosed = some true ↔ Poly.eval x0 V ≤ thr

```

BernsteinLogic.QtslElaborators . **rootEvalElab\_sound**

statement bound

```

/-- The factoring, BL face: the elaborated payload is sound through the
target's kernel-verified lemma – one application of
`sound_pConstExact`, which is itself one step to
`Formula.holds_of_evalClosed_true` (`prop:re-sound`). -/
theorem rootEvalElab_sound {H : Hypernet} {seq : Sequent} {x0 : Nat → Rat}
  {V : Poly} {thr : Rat}
  (hEval : (rootEvalFormula x0 V thr).evalClosed = some true)
  (hpost : seq.post = rootEvalFormula x0 V thr) :
  seq.semanticallyValid H

```

BernsteinLogic.QtslElaborators . **polyClosedTerm**

definition bound

```

/-- The closed polynomial term at a rational point. Wire-free by
construction, and its exact evaluation is `Poly.eval` at that point
(`polyClosedTerm_evalQ`) – the checker computes the value `V(x_0)`
itself, not an approximation to it. -/
def polyClosedTerm (x0 : Nat → Rat) (p : Poly) : Term :=
  p.foldr (fun mc acc => .add (.mul (.const mc.2) (monoClosedTerm x0 mc.1)) acc)
    (.const 0)

```

BernsteinLogic.QtslElaborators . **rootEvalFormula**

definition bound

```

/-- The elaborated leaf formula:  $\forall(x_0) \leq \text{thr}$  as one closed exact
comparison. -/
def rootEvalFormula (x0 : Nat → Rat) (V : Poly) (thr : Rat) : Formula :=
  .atom .le (polyClosedTerm x0 V) (.const thr)

```

BernsteinLogic.QtslElaborators . **rootEvalElab**

definition bound

```

/-- **The RootEval payload elaborator** (`rec:re-transport`, sublevel
branch): the alias's data – point, atom – plus its own checked leaf, to
the one `PConstExactData` inhabitant. The comparator confinement
travels by construction (`le ≠ .ne` discharged here, once). -/
def rootEvalElab (H : Hypernet) (seq : Sequent) (x0 : Nat → Rat) (V : Poly)
  (thr : Rat)
  (hEval : (rootEvalFormula x0 V thr).evalClosed = some true)
  (hpost : seq.post = rootEvalFormula x0 V thr) :
  PConstExactData H seq :=
  .mk H seq (polyClosedTerm x0 V) (.const thr) .le (by decide) hEval hpost

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed leaf rules — exact rational evaluation concluding root-atom membership, one form per atom shape —  $\mapsto$  three statements, and the strength gradation is the dictionary's content. `rootEval_eval_cast` is the literal identity of values:  $\mathbb{Q}$ -evaluation, cast, equals  $\mathbb{R}$ -evaluation of the cast polynomial at the cast point (MvPolynomial over any variable type — dimension-free). The two atom shapes are then *equivalences*, not implications: `rootEval_sublevel_iff` (exact- $\mathbb{Q}$  comparison  $\leftrightarrow$  membership in the cast sublevel atom) and `rootEval_box_iff` (per-coordinate  $\mathbb{Q}$  containment  $\leftrightarrow$  cast-box membership) — two-directional exactness, which is precisely what a rounded evaluator would lose (under directed outward rounding the sound forward direction survives and the iff does not; undirected rounding loses even soundness); the entry's “no rounding gap anywhere” as bind structure. Width note: the checker consumes only the forward directions (leaf discharge via `PConstExact` / the registry interval recompute); the certified equivalences are strictly stronger, safe direction, and no profile, measure, or topology enters any binder.

## Lean verifier receipts

module BernsteinLogicAux.QtslAliases; bound build 570599779c29... (bound from receipt 5a60f4d01fd81ac5, 2026-07-21T05:39:02Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0a3819a978e31127 | 2026-07-21T04:51:36Z | build+decorrelated                 | current |
| 0bb1074df705d863 | 2026-07-21T05:21:58Z | build+decorrelated                 | current |
| 5a60f4d01fd81ac5 | 2026-07-21T05:39:02Z | build+replay+hardened+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.QtslElaborators; bound build 0205ec213b91... (hashed from the built olean at bind time). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a237f4cff27cb15a | 2026-07-22T10:06:47Z | build+decorrelated | current |

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope for the HOL side *by design*, not by backlog: this rule is a *field* of the bridge rule (QSATFROMREGION's eval slot), never a node, so no wire certificate a HOL checker reads could ever contain a ROOTEval node; HOL coverage for this rule *is* the target rule's, and the box-atom branch is loader-side registry recomputation, likewise not a wire-format rule. Today even the target's statement is not writable: the HOL4 development (theory bern\_catalogue, the base catalogue's check-function cone) predates PCONSTEXACT, whose wire tag is unallocated — see that rule's section for the honest state of the target's own box.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template: the  $1 \rightarrow 1$  unpack, the filled-in target application, the payload-transport table, the exactness identity, and the soundness corollary through the target's lemma. This row is the membership leaf itself, catalogued as a rule of its own because every initialized certificate ends with one; in the extension certificate schema it is a *field* of the bridge rule (QSATFROMREGION's eval slot), not a node sort. The judgment-level story lives in the bridge rule's section — this section owns the leaf. Mechanization state: the target's lemma is kernel-verified; the exactness identities are mechanized and bound in the entry, and so — now that the extension certificate schema exists — is the payload-level elaborator (the remark below names the bound functions).

**Recipe R67.1** (the template instance). One leaf  $\rightarrow$  one node: membership of the initial point  $x_0$  in the certificate's *root atom* — the region atom concluded at the root of the certificate (naming note: “root” refers to the certificate's root claim, not to roots of polynomials) — by exact rational evaluation. For a *sublevel* atom  $\{x : V(x) \leq y\}$  the unpack image is a single PCONSTEXACT application (Recipe R67.2). For a *box* atom the leaf is not a rule application at all: per-coordinate interval containment is recomputed directly by the checker's side-condition registry (conformance recomputation) — the alias reading covers the sublevel branch, and the box branch is stated so the respelling never overclaims. Nothing else is derived: the rule is a compact spelling of one ground-fact row.

**Recipe R67.2** (the filled-in application). Sublevel atom  $R = \{x : V(x) \leq y\}$  with  $V$  a polynomial over  $\mathbb{Q}$ , threshold  $y \in \mathbb{Q}$ , and initial point  $x_0$  with rational coordinates:

$$\frac{V(x_0) \leq y \text{ holds under exact evaluation in } \mathbb{Q}}{\Phi \vdash V(x_0) \leq y} \text{ PCONSTEXACT}$$

with arbitrary premise  $\Phi$ , as the target licenses (the rule is premise-free).  $V(x_0)$  denotes the closed term obtained by substituting every coordinate of  $x_0$  into  $V$  — closed by construction, so the target's verified evaluator decides the comparison in exact  $\mathbb{Q}$ , and a true verdict certifies closedness as a byproduct (the target section unpacks why).

### Recipe R67.3 (payload transport — field for field).

| ROOTEval data                | PCONSTEXACT payload                                        |
|------------------------------|------------------------------------------------------------|
| $x_0$ (rational coordinates) | substituted into the closed left-hand term                 |
| sublevel atom $(V, y)$       | the comparison's two sides                                 |
| comparator                   | $\leq$ (inside the admissible set)                         |
| box atom (per-axis bounds)   | <i>not</i> transported — registry recompute (Recipe R67.1) |

**Proposition R67.4** (the exactness identity).

$$x_0 \in \{x : V(x) \leq y\} \iff \text{evalClosed}(V(x_0) \leq y) = \text{true}.$$

*Proof.* Two lines, which is the point. Substituting a rational point into a rational polynomial is exact arithmetic — no rounding enters at any step — so the evaluator computes the value  $V(x_0)$  itself, not an approximation to it; and the decidable comparison of two exact rationals is the membership predicate verbatim. This is the degenerate — and best — case of the exactness identity an alias owes: not an algebraic rearrangement but a literal identity of values. It is also the whole rule: under naively rounded (non-directed) evaluation, a point lying  $\varepsilon$  outside the region could certify membership, and the bridge rule's instantiation (its section, the extension page) would conclude satisfaction from a false premise; directed outward rounding would keep that direction sound but surrender the equivalence — boundary points, where  $V(x_0) = y$  exactly, typically become undecidable (an outward enclosure that straddles the threshold cannot decide; an evaluation whose enclosure does not cross it — an exact zero-width value, or a one-sided enclosure touching the threshold — still decides).  $\square$

**Proposition R67.5** (soundness corollary). *If the leaf checks,  $x_0$  belongs to the root atom.*

*Proof.* `Formula.holds_of_evalClosed_true` (the target's kernel-verified evaluator theorem, quoted in the `PConstExact` section): a closed comparison whose exact evaluation returns true holds at every valuation. By Proposition R67.4 that is membership: substitution closed the formula, so  $V(x_0) \leq y$  mentions no wires, its truth is valuation-independent, and that truth is literally the sublevel atom's membership predicate at  $x_0$ . There is no other content — the alias genre working as intended. (For the box branch the recomputed per-coordinate containment plays the same role, warranted by the `recompute`'s definition rather than a rule lemma.)  $\square$

*Remark R67.6* (the payload-level elaborator, mechanized). The *semantic* content of the respelling is bound in the entry: `rootEval_eval_cast` is Proposition R67.4's literal identity of values, and `rootEval_sublevel_iff` / `rootEval_box_iff` are the two atom shapes as equivalences. The payload-level elaborator — previously this page's one absent artefact, held back until the extension certificate schema existed — is bound with it: `rootEvalElab` (module `BernsteinLogic.QtslElaborators`) takes  $(x_0, \text{atom})$  to exactly the one `PConstExact` payload of Recipe R67.3 (sublevel branch), as an inhabitant of the target's own payload type — the comparator confinement discharged at elaboration, once. The closed left-hand term is built by verified substitution (`polyClosedTerm`, whose evaluation lemma `polyClosedTerm_evalQ` says the checker's exact replay computes *the value  $V(x_0)$  itself*), `rootEvalFormula_true_iff` is Proposition R67.4's identity at the wire (the  $\mathbb{Q}$  face — both directions, as the proposition demands), and `rootEvalElab_sound` is the factoring through `sound_pConstExact`, itself one step to `Formula.holds_of_evalClosed_true`. The box branch is not elaborated, by design (Recipe R67.1: registry `recompute`, not a rule application).

*Remark R67.7* (the inherited comparator confinement). `PConstExact` excludes one comparator ( $\neq$ ) from its admissible set, for reasons of a second soundness invariant unpacked in that rule's section. The membership leaf uses  $\leq$  only — comfortably inside — but any port or mechanization of this rule must *preserve* the exclusion rather than re-derive the comparator set from the semantic lemma alone; the flag travels with the alias.

| proof step                                             | formal anchor                                                                                                                                                                                                                                        |
|--------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| target lemma (Prop. R67.5)                             | <code>Formula.holds_of_evalClosed_true</code> ; <code>sound_pConstExact</code> (quoted in the <code>PConstExact</code> section)                                                                                                                      |
| closedness implied at the leaf<br>exactness identities | <code>Formula.evalClosed</code> (none on any wire; same section)<br><code>rootEval_eval_cast</code> , <code>rootEval_sublevel_iff</code> , <code>rootEval_box_iff</code> (bound in the entry)                                                        |
| payload-level elaborator (Remark R67.6)                | <code>rootEvalElab</code> + <code>rootEvalElab_sound</code> , over <code>polyClosedTerm</code> + <code>polyClosedTerm_evalQ</code> + <code>rootEvalFormula_true_iff</code> (module <code>BernsteinLogic.QtslElaborators</code> , bound in the entry) |
| box-atom branch<br>consumption site                    | checker-side registry <code>recompute</code> , by design — no rule anchor<br>the bridge rule's eval field ( <code>QSATFROMREGION</code> section)                                                                                                     |

## Rule 68. BoxBERN (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *BoxBern*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} N \in \mathbb{Q}[z_1, \dots, z_n]; \quad D = \prod_v [a_v, b_v] \subseteq \mathbb{Q}^n \\ \varphi = (A, A^{-1}, b) : \quad A, A^{-1} : \mathbb{Q}^{[n] \times [n]}, \quad b : \mathbb{Q}^{[n]}; \quad \varphi(w) = A w + b \text{ (affine-rung witness; } \varphi \text{ omitted reads as id)} \\ \vdash A^{-1} A = 1_{[n]} \text{ (the one certified product, exact)} \\ \hat{N} = \varphi \circ N \text{ (exact substitution)} \quad \{b_\alpha \leq 0\}_\alpha \text{ (every tensor-product Bernstein coefficient of } \hat{N} \text{ over } D) \end{array}}{D \vdash \hat{N} \leq 0} \text{BoxBERN}$$

the witness is data with index structure — two square rational tensors and an offset vector, the declared inverse carried rather than computed, its sole obligation the exact product (over a field, one side suffices) — and the ladder above the affine rung is graded by each rung's own grader: an unimplemented rung is a named refusing variant, never a partial check. The pullback  $\hat{N} = \varphi \circ N$  is recomputed by the checker and never rides the certificate, so a laundered pullback is unrepresentable rather than detected. The enacted rule is stated from its landed realization (`check_box_row/check_wire_family` in the workshop checker, over the wire-family grammar; refusal witness  $(\alpha, b_\alpha)$ ) — its certificate content is out of scope of the current baked theory: a box-cell row family is the named successor, and this cell moves onto the theory row when a revision carries it.

BoxBERN is the tensor-product (box) sibling of BERNcone: where BERNcone certifies a polynomial predicate over a simplex-shaped region by exact Bernstein enclosure, BoxBERN certifies  $\hat{N} \leq 0$  over an axis-aligned rational box, with the polynomial optionally pulled back through a certified reparameterization witness first. A producer reaches for it wherever certificate cells are boxes rather than simplices — grid covers, dyadic bisection cells, per-cell contraction rows for discrete maps — because the obligation count is the product of per-axis degrees rather than the simplex count times a multinomial: at the worked fixture's shape (four variables, per-axis degrees (2, 4, 2, 4), total degree 8), one box costs 225 exact coefficients where the Kuhn-simplex decomposition of the same box costs  $24 \cdot 495 = 11,880$  — a structural gap that compounds with per-coefficient cost, measured at three orders of magnitude wall-clock on the acceptance fixture.

What the checker recomputes, given the row  $(N, \varphi, D)$ : that the reparameterization witness certifies (for an affine witness, shapes and the exact left-inverse identity  $A^{-1} A = 1$ ; later ladder rungs by their own graders; an unimplemented rung is a *named refusing variant*, never a partial check); the pullback  $\hat{N} = \varphi \circ N$  by exact substitution — no pullback ever rides the certificate, so a laundered  $\hat{N}$  is unrepresentable rather than detected; and then the enclosure itself: substitute  $x_v = a_v + (b_v - a_v) t_v$  per axis, change basis per axis at the polynomial's actual degrees, and test every tensor-product Bernstein coefficient against  $\leq 0$ , refusing with the first offending multi-index and its exact value. Everything is exact rational arithmetic; nothing consults a float.

Each hypothesis is load-bearing. Drop the slot certification and an uninvertible “witness” could smuggle a coordinate change under which the claim means something else entirely. Drop the recompute-the-pullback discipline and the checked polynomial decouples from the claimed one. Drop the per-coefficient sweep and the conclusion is unrelated to the box. The conclusion is deliberately narrow — one sign fact on one box — because the row *species* that consumes it (coverage of the operative region, S-procedure multipliers  $\lambda \geq 0$ , init/containment side rows) owns every regional and compositional obligation; this rule is the arithmetic leaf they all stand on.

**notes** The enclosure bound is conservative — coefficients overshoot the true range except at corners, where they interpolate — and completeness is recovered by subdivision exactly as for the simplex rule: SUBDIVIDE remains the completeness partner. Degrees enter the certified statement as *data*: any degree family admitting a valid tensor-Bernstein expansion encloses, so “use the actual degrees” is a checker cost policy, not a soundness premise. Degenerate axes ( $a_v = b_v$ ) are inside the statement — the substitution sends the axis to a constant and the enclosure is unaffected.

### Lean statement

BernsteinLogicAux.BoxBern .boxBern\_enclosure

statement bound

```
-- **The tensor-basis enclosure lemma** (soundness part (ii) of the
box-Bernstein rule): if the box-shifted claim polynomial
`N` on `(lo + (hi - lo)·u)` has tensor-Bernstein coefficients `b`, then any
bounds on `b` bound `N` on the box. `hexp` is the checker's exact-`Q`
replay obligation (the shift and basis-conversion passes); the conclusion
quantifies over the landed arena's box set. -/
theorem boxBern_enclosure (lo hi : Fin n → Q) (N : MvPolynomial (Fin n) Q)
  (d : Fin n → ℕ) (b : (∀ v, Fin (d v + 1)) → Q) (m M : Q)
```

```
(hexp : boxShift lo hi N =  $\sum \alpha$ , C (b  $\alpha$ ) * tensorBern d  $\alpha$ )
(hm :  $\forall \alpha$ , m  $\leq$  b  $\alpha$ ) (hM :  $\forall \alpha$ , b  $\alpha \leq$  M) :
 $\forall x \in (\text{ratBox lo hi}).\text{set}, (m : \mathbb{R}) \leq \text{aeval } x \text{ N} \wedge \text{aeval } x \text{ N} \leq (M : \mathbb{R})$ 
```

BernsteinLogicAux.BoxBern . **boxBern\_enclosure\_minmax**

statement bound

```
/-- The doc-verbatim min/max form (part (ii) as displayed):
`min_α b_α ≤ N(x) ≤ max_α b_α` on the box. -/
theorem boxBern_enclosure_minmax (lo hi : Fin n → ℚ)
  (N : MvPolynomial (Fin n) ℚ)
  (d : Fin n → ℕ) (b : (∀ v, Fin (d v + 1)) → ℚ)
  (hexp : boxShift lo hi N =  $\sum \alpha$ , C (b  $\alpha$ ) * tensorBern d  $\alpha$ ) :
 $\forall x \in (\text{ratBox lo hi}).\text{set},$ 
  ((Finset.univ.inf' Finset.univ_nonempty b : ℚ) : ℝ) ≤ aeval x N ∧
  aeval x N ≤ ((Finset.univ.sup' Finset.univ_nonempty b : ℚ) : ℝ)
```

BernsteinLogicAux.BoxBern . **boxBern\_nonpos**

statement bound

```
/-- **The checker-facing sign corollary** (the rule's accept condition:
"accept iff every coefficient `b_α ≤ 0`"): nonpositive coefficients certify
`N ≤ 0` on the box — exactly one `hcells` premise of `Repar.t0_soundness` /
`t05_soundness_gridcx`, over the same `GridCx.Box` object. -/
theorem boxBern_nonpos (lo hi : Fin n → ℚ) (N : MvPolynomial (Fin n) ℚ)
  (d : Fin n → ℕ) (b : (∀ v, Fin (d v + 1)) → ℚ)
  (hexp : boxShift lo hi N =  $\sum \alpha$ , C (b  $\alpha$ ) * tensorBern d  $\alpha$ )
  (hcoeff :  $\forall \alpha$ , b  $\alpha \leq 0$ ) :
 $\forall x \in (\text{ratBox lo hi}).\text{set}, \text{aeval } x \text{ N} \leq 0$ 
```

BernsteinLogicAux.BoxBern . **aeval\_boxShift**

statement bound

```
/-- **Per-axis affine substitution exactness** (doc (ii) proof-shape item
"per-axis affine substitution exactness — a diagonal T0 map"): evaluating
the shifted polynomial at `u` is evaluating the original at the shifted
point. -/
theorem aeval_boxShift (lo hi : Fin n → ℚ) (u : Fin n → ℝ)
  (N : MvPolynomial (Fin n) ℚ) :
  aeval u (boxShift lo hi N)
    = aeval (fun v => (lo v : ℝ) + ((hi v : ℝ) - (lo v : ℝ)) * u v) N
```

BernsteinLogicAux.BoxBern . **sum\_aeval\_tensorBern**

statement bound

```
/-- Sub-lemma 2c (partition of unity): the tensor basis sums to `1` on the
unit cube — "products of partitions of unity are partitions of unity"
(doc (ii)); univariate `bernsteinPolynomial.sum` distributed over the
product by `Fintype.prod_sum`. -/
theorem sum_aeval_tensorBern (d : Fin n → ℕ) (u : Fin n → ℝ) :
 $\sum \alpha : \forall v, \text{Fin } (d v + 1), \text{aeval } u (\text{tensorBern } d \alpha) = 1$ 
```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed rule — certified slot, exact pullback, coefficient sweep, conclusion  $\hat{N} \leq 0$  on  $D$  — maps onto the bound statements as follows.  $D \mapsto \text{lo hi} : \text{Fin } n \rightarrow \mathbb{Q}$  through `ratBox` (the same `GridCx.Box` object the transport theorems quantify over);  $\hat{N} \mapsto N : \text{MvPolynomial } (\text{Fin } n) \ \mathbb{Q}$  — the *pulled-back* polynomial, because the checker recomputes the pullback and hands the enclosure lemma its result: the slot legs (certification, replay) are consumed upstream by the transport theorems (`t0_soundness`, `t05_soundness_gridcx`, and the `T1/TC` rung lemmas), quoted and bound in their own sections, never re-stated here. One orientation divergence in the slot leg is named rather than left to the reader: the enacted checker verifies the *displayed* product — its single exact multiplication computes  $A^{-1}A$  and compares it with  $1_{[n]}$  (`affine_left_certified`) — whereas the Lean witness structure stores the *opposite* orientation (`AffineWitness.certified : A · A-1 = 1`) and derives the displayed one as the theorem `left_certified : A-1 · A = 1` (via `mul_eq_one_comm`; the matrices are square over the commutative ring  $\mathbb{Q}$ , so either product



determines the other and the two orientations are equivalent). The displayed coefficient table  $\mapsto$  the expansion hypothesis of `boxBern_enclosure`: an explicit representation of the `boxShift`-substituted polynomial as  $\sum_{\alpha} b_{\alpha} \prod_v B_{\alpha_v}^{d_v}$  — the u-space/x-space transport is a theorem (`aeval_boxShift`), not notation, and the degree family  $d$  is a binder, not a fixed choice. The displayed accept condition (every  $b_{\alpha} \leq 0$ )  $\mapsto$  the hypothesis of `boxBern_nonpos`, whose conclusion is verbatim one `hcells` premise of the transport theorems — the two compose without glue, which is the certified content of “exactly one premise of the landed transport theorems.” Width note, two-sided: `boxBern_enclosure_minmax` is strictly stronger than the accept condition needs (both range ends, where the rule consumes only the max side), and nothing certified is wider than the display — the slot legs live upstream by construction, and no coverage or multiplier obligation enters any binder here (the row species owns them, as the display says).

## Lean verifier receipts

module `BernsteinLogicAux.BoxBern`; bound build 110ba4bda664... (bound from receipt 6935ac926dca961f, 2026-08-03T20:13:44Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 63a3e12578528f22 | 2026-08-02T02:06:52Z | build+decorrelated | current |
| c2fe5c957feb2507 | 2026-08-02T23:30:40Z | build+decorrelated | current |
| 665f63f0c0026da8 | 2026-08-03T16:26:44Z | build+decorrelated | current |
| 6935ac926dca961f | 2026-08-03T20:13:44Z | build+decorrelated | current |
| a3e7c09b848c5808 | 2026-08-03T22:45:28Z | build+decorrelated | current |

8 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) stale only

hardened (safe\_verify (hardened screen)) stale only

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL family port (`checkFam_boxbern` + a `famEq` arm, the A1/A3 mechanism) is a named successor of the kernel leg that landed this rule — absence recorded at landing, with the mechanism that will discharge it, so this reason cannot silently go stale the way pre-wave absence reasons did.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: admissible rule (tier 2) — admissible by *reduction to verified checking machinery*, the taxonomy's recorded boundary case. The reduction shape is stated first; the enclosure proposition it stands on is then proved in full, because it is the section's one piece of genuinely new mathematics — the tensor twin of the simplex grounding that `BERNCONE` already carries.

**Recipe R68.1** (the reduction — what the checker replays). A `BoxBERN` row is  $(N, \varphi?, D)$ : a claim polynomial  $N \in \mathbb{Q}[z_1, \dots, z_n]$ , an optional reparameterization witness  $\varphi$  from the ladder ( $\emptyset$  = identity), and an axis-aligned box  $D = \prod_v [a_v, b_v] \subseteq \mathbb{Q}^n$  in the witness's working coordinates. The discharge is three replays, all exact  $\mathbb{Q}$ , each standing on a named verified theorem rather than on catalogue structure:

1. *slot certification* — for an affine witness, shapes and the exact left-inverse identity  $A^{-1}A = 1$  (a left inverse suffices over a field); later ladder rungs certify by their own graders, and a rung without a grader is a *named refusing variant* — refused loudly, never half-checked.
2. *substitution replay* —  $\hat{N} = \varphi \circ N$  by exact substitution. No pullback field exists in the row, so a forged  $\hat{N}$  is unrepresentable rather than detected.



3. *enclosure* — the tensor-product Bernstein coefficients of  $\hat{N}$  over  $D$  (Proposition R68.2); accept iff every coefficient is  $\leq 0$ , refusing with the witness  $(\alpha, b_\alpha)$  otherwise.

The conclusion,  $\hat{N} \leq 0$  on  $D$ , is deliberately exactly one `hcells` premise of the landed transport theorems: coverage of the operative region, S-procedure multipliers, and `init/containment` side rows are owned by the consuming row species, never by this leaf.

**Proposition R68.2** (tensor-product Bernstein enclosure). *Let  $p \in \mathbb{Q}[x_1, \dots, x_n]$ , let  $D = \prod_v [a_v, b_v]$  with rational endpoints ( $a_v \leq b_v$ , degenerate axes allowed), and suppose that after the per-axis affine substitution  $x_v = a_v + (b_v - a_v)u_v$  the resulting polynomial admits the tensor-product Bernstein representation*

$$p(a + (b - a) \odot u) = \sum_{\alpha} b_{\alpha} \prod_{v=1}^n B_{\alpha_v}^{d_v}(u_v), \quad B_i^d(t) = \binom{d}{i} t^i (1-t)^{d-i},$$

for some degree family  $(d_v)$  and coefficients  $b_{\alpha} \in \mathbb{Q}$ ,  $\alpha \in \prod_v \{0, \dots, d_v\}$ . Then for every  $x \in D$ ,

$$\min_{\alpha} b_{\alpha} \leq p(x) \leq \max_{\alpha} b_{\alpha}.$$

In particular, if every  $b_{\alpha} \leq 0$  then  $p \leq 0$  on  $D$ .

*Proof.* Fix  $x \in D$ . Each coordinate lies in its interval, so there is a  $u \in [0, 1]^n$  with  $x_v = a_v + (b_v - a_v)u_v$  for every axis: on a non-degenerate axis take  $u_v = (x_v - a_v)/(b_v - a_v)$ ; on a degenerate axis ( $a_v = b_v$ ) the substitution is the constant  $a_v$  regardless of  $u_v$ , and  $u_v = 0$  serves. Substitution exactness — that evaluating  $p$  at this  $x$  equals evaluating the substituted polynomial at  $u$  — is a per-axis affine replay, not a notational identification (the bound statement keeps the two polynomials distinct and transports along the substitution lemma).

On  $[0, 1]$ , each Bernstein basis polynomial is nonnegative ( $t^i \geq 0$ ,  $(1-t)^{d-i} \geq 0$ , binomial coefficients positive) and each degree- $d$  family sums to 1:  $\sum_i B_i^d(t) = (t + (1-t))^d = 1$  by the binomial theorem. Products of partitions of unity are partitions of unity, so the tensor family  $\{\prod_v B_{\alpha_v}^{d_v}(u_v)\}_{\alpha}$  is a family of nonnegative weights summing to 1 at every  $u \in [0, 1]^n$ . Evaluating the representation at  $u$  therefore exhibits  $p(x)$  as a convex combination of the  $b_{\alpha}$ , and a convex combination lies between the minimum and the maximum of its generators. The accept condition is the corollary: if every generator is  $\leq 0$ , so is every convex combination.  $\square$

*Remark R68.3* (degrees are data; degenerate axes are inside the statement). Nothing in Proposition R68.2 privileges the polynomial's actual per-axis degrees: *any* degree family admitting a valid representation encloses, so the checker's "use the actual degrees after the shift" is a cost policy — smallest coefficient table — and never a soundness premise. Likewise a degenerate axis needs no exclusion: the substitution collapses it to a constant, degrees drop, and the convex-combination argument is untouched. Both points are formalization deltas made *explicit* in the bound statements, in the conservative direction — the expansion hypothesis speaks about the substituted polynomial (the u-space/x-space transport is a theorem), and the degree family is a binder.

*Remark R68.4* (conservatism, and where completeness comes from). The bound is not tight: coefficients overshoot the true range except at the corners of  $D$ , where the representation interpolates ( $B_i^d(0) = [i = 0]$ ,  $B_i^d(1) = [i = d]$ , so corner coefficients *equal* corner values). A true-but-refused row is repaired by subdivision — Bernstein enclosures converge under subdivision and degree elevation — so SUBDIVIDE is the completeness partner here exactly as for the simplex rule, and refusal is always the failure direction.

*Remark R68.5* (why a box rule at all — the obligation-count structure). At per-axis degrees  $(d_v)$ , one box costs  $\prod_v (d_v + 1)$  coefficients; the kernel-checkable simplex shape for the same box costs  $n!$  Kuhn simplices at  $\binom{d+n}{n}$  coefficients each (total degree  $d$ ). At the acceptance fixture's shape —  $n = 4$ , degrees  $(2, 4, 2, 4)$ ,  $d = 8$  — that is  $3 \cdot 5 \cdot 3 \cdot 5 = 225$  against  $24 \cdot \binom{12}{4} = 11,880$  per cell, and the per-coefficient work also drops (one-dimensional passes against multinomial walks). The measured same-harness gap on the fixture's 26 cells is three orders of magnitude wall-clock. None of this changes what is certified — the rule is an optimization of checker cost, not a widening of any claim.

| proof step              | formal anchor                                                                                                             |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------|
| Recipe R68.1 slot leg   | <code>t0_soundness / t05_soundness_gridcx</code> (their sections); later rungs per their graders, refusing variants named |
| Recipe R68.1 replay leg | <code>pullback</code> recomputed, never carried — laundering unrepresentable by row shape                                 |
| Proposition R68.2       | <code>boxBern_enclosure</code> (bound form), <code>boxBern_enclosure_minmax</code> (verbatim min/max form)                |
| substitution exactness  | <code>aeval_boxShift</code> (per-axis affine replay; degenerate axes via the unit-scalar witness)                         |
| partition of unity      | <code>sum_aeval_tensorBern</code> (tensor family sums to 1)                                                               |
| the accept corollary    | <code>boxBern_nonpos</code> — conclusion verbatim one <code>hcells</code> premise of the transport theorems               |

## Rule 69. PEXPTYPEENCLOSURE (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *PExpTypeEnclosure*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{f \in F \quad n \in \mathbb{N} \quad r, R \in \mathbb{Q} \quad a_k = a_k^f \text{ for } k = 0..n \text{ (exact coefficient recurrence)} \quad 0 \leq r < n + 2 \quad R \geq \frac{r^{n+1}}{(n+1)!} \cdot \frac{n+2}{n+2-r}}{\forall x, |x| \leq r \Rightarrow \left| f(x) - \sum_{k \leq n} a_k x^k \right| \leq R} \text{ PEXPTYPEENCLOSURE}$$

the coefficients are recomputed by the checker (never carried); the check is pure  $\mathbb{Q}$ -arithmetic and never evaluates  $f$ ; the real functions enter only in the kernel-checked soundness theorem. Scope of  $F$ : entire functions with exact- $\mathbb{Q}$  Maclaurin coefficients dominated by  $|a_k| \leq 1/k!$  (entire of exponential type, normalized) — exp, sin, cos, and (qtsl tower) the reduced error-function kernel; a new in-class function costs one coefficient recurrence + one domination case, the tail argument untouched. Finite-radius functions (log, rational powers) are certified by sibling routes, not by this rule; every downstream consumption decomposes into existing polynomial sequents (recentering, Taylor-model combination, endpoint substitution).

PEXPTYPEENCLOSURE is the calculus's one transcendental leaf: a remainder-bound rule that brackets a non-polynomial function between two polynomials with all data rational, so that every downstream obligation — recentering, Taylor-model combination, endpoint substitution into a generator inequality — decomposes into polynomial sequents the existing rules already check. A producer reaches for it wherever certified dynamics carry transcendental drift terms (pendulum-family sin / cos, rate and decay exp, Gaussian-kernel reductions): the rule certifies a Maclaurin truncation once, at a radius, with an explicit remainder, and from there the trusted arithmetic is exact rational computation exactly as for every other leaf.

The witness is  $(f, n, r, R)$ : a function tag  $f$  from the registered library, a truncation degree  $n$ , a radius  $r \in \mathbb{Q}_{\geq 0}$ , and a remainder budget  $R \in \mathbb{Q}$ . The check is pure rational arithmetic — the coefficients  $a_k^f$  are recomputed from  $(f, n)$  by exact per-tag recurrence rather than carried (a laundered coefficient list is unrepresentable on the wire), the radius must satisfy  $r < n + 2$ , and  $R$  must dominate the closed-form tail bound  $\frac{r^{n+1}}{(n+1)!} \cdot \frac{n+2}{n+2-r}$ . The checker never evaluates  $f$ ; the real functions enter only in the kernel-checked soundness theorem, which concludes the two-sided bracket  $|f(x) - \sum_{k \leq n} a_k x^k| \leq R$  on  $[-r, r]$ .

The scope is deliberately exact: *entire functions with exact-rational Maclaurin coefficients dominated by  $|a_k| \leq 1/k!$*  — entire of exponential type, normalized so  $e^{|x|}$  dominates coefficientwise. All three conditions are load-bearing: exact- $\mathbb{Q}$  recurrences make clause (i) checkable term-by-term; the  $1/k!$  domination is what lets ONE tail formula serve every tag; entirety means any rational radius works, subject only to the formula's own guard  $r < n + 2$ . This is strictly narrower than “anything with a closed-form Taylor series”: log and rational powers are analytic but not entire (a finite-radius variant would need a radius-ratio geometric tail and a domain side condition; they are certified today by sibling routes built on their own bounds), and tan is meromorphic (handled as sin / cos plus division at the consumer). Extending the library to a new in-class function costs one coefficient recurrence and one domination case — the tail argument is untouched. The error-function kernel exercised exactly this clause: under the  $y = z^2$  reduction its coefficients are  $(-1)^k / (k! (2k + 1))$ , dominated by  $1/k!$  verbatim, and the soundness statement needed no edits beyond the per-tag cases.

**notes** Admissible by reduction to verified checking machinery (the same boundary clause as BoxBERN): the checker replays encCheck, and the trust lands on the kernel-checked meaning theorems, never on catalogue structure. Conservativity: no new sequent theorems — the rule concludes into a sibling judgment kind, and its consumers re-enter the calculus as ordinary polynomial rows. Stated outright rather than smoothed: this is the one rule where transcendental meaning enters the trust story — a deliberate, priced, single-rule growth of the trusted base, carried by the soundness theorems cited below. Slack is the producer's business (pick  $n$ , recenter, subdivide): subdivision shrinks bracket width first-order while degree shrinks it factorially, and the measured budgets on the acceptance systems ran at degree 9 with remainders near  $10^{-7}$  — orders below the certificate margins they feed.

### Lean statement

BernsteinLogicAux.Enclosure . encCheck\_sound

statement bound

```
-- **The remainder-bound rule is sound** (note §1's boxed rule): a witness passing
`encCheck` certifies `|f(x) - P(x)| ≤ R` for every `|x| ≤ r`, where `f` is the tag's
real function and `P` the witness polynomial. -/
theorem encCheck_sound (w : EncWitness) (h : encCheck w = true) :
  ∀ x : ℝ, |x| ≤ (w.r : ℝ) →
    |w.tag.eval x - evalPoly w.coeffs x| ≤ (w.R : ℝ)
```

BernsteinLogicAux.Enclosure . **encCheck\_sound\_bracket**

statement bound

```

/-- Two-sided bracket form:  $P(x) - R \leq f(x) \leq P(x) + R$  on  $[-r, r]$  — the pair of
polynomial brackets the certificate's sequents consume (note §3). -/
theorem encCheck_sound_bracket (w : EncWitness) (h : encCheck w = true) :
   $\forall x : \mathbb{R}, |x| \leq (w.r : \mathbb{R}) \rightarrow$ 
    evalPoly w.coeffs x - (w.R :  $\mathbb{R}$ )  $\leq$  w.tag.eval x  $\wedge$ 
    w.tag.eval x  $\leq$  evalPoly w.coeffs x + (w.R :  $\mathbb{R}$ )

```

BernsteinLogicAux.Enclosure . **encCheck\_point**

statement bound

```

/-- Point-bracket corollary (producer recentering, note §3(1)): evaluating the witness at
a rational point `c` with  $|c| \leq r$  brackets the transcendental constant  $f(c)$ . -/
theorem encCheck_point (w : EncWitness) (h : encCheck w = true) (c :  $\mathbb{Q}$ )
  (hc :  $|c| \leq w.r$ ) :
    evalPoly w.coeffs (c :  $\mathbb{R}$ ) - (w.R :  $\mathbb{R}$ )  $\leq$  w.tag.eval (c :  $\mathbb{R}$ )  $\wedge$ 
    w.tag.eval (c :  $\mathbb{R}$ )  $\leq$  evalPoly w.coeffs (c :  $\mathbb{R}$ ) + (w.R :  $\mathbb{R}$ )

```

BernsteinLogicAux.Enclosure . **linear\_interp\_le**

statement bound

```

/-- Generator VCs are linear in each transcendental leaf occurrence: if  $A + C \cdot w \leq 0$ 
holds at both endpoints of  $[l, h]$ , it holds for every  $w \in [l, h]$ . With this, per-cell
interval hypotheses for the enclosed leaf discharge as TWO ordinary polynomial sequents —
no new wire dimension. -/
theorem linear_interp_le {A C l h w :  $\mathbb{R}$ } (hlw :  $l \leq w$ ) (hwh :  $w \leq h$ )
  (hl :  $A + C \cdot l \leq 0$ ) (hh :  $A + C \cdot h \leq 0$ ) :  $A + C \cdot w \leq 0$ 

```

QtslAux.Enclosure . **encCheck\_sound**

statement bound

```

theorem encCheck_sound (w : EncWitness) (h : encCheck w = true) :
   $\forall x : \mathbb{R}, |x| \leq (w.r : \mathbb{R}) \rightarrow$ 
     $|w.tag.eval x - evalPoly w.coeffs x| \leq (w.R : \mathbb{R})$ 

```

QtslAux.Enclosure . **encCheck\_sound\_bracket**

statement bound

```

theorem encCheck_sound_bracket (w : EncWitness) (h : encCheck w = true) :
   $\forall x : \mathbb{R}, |x| \leq (w.r : \mathbb{R}) \rightarrow$ 
    evalPoly w.coeffs x - (w.R :  $\mathbb{R}$ )  $\leq$  w.tag.eval x  $\wedge$ 
    w.tag.eval x  $\leq$  evalPoly w.coeffs x + (w.R :  $\mathbb{R}$ )

```

QtslAux.Enclosure . **integral\_exp\_neg\_sq**

statement bound

```

/-- The Gaussian integral reduces to the error-function kernel by the
substitution  $t = z * s$ :  $\int t \text{ in } (0:\mathbb{R})..z, \exp(-(t^2)) = z * T(z^2)$ .
Both sides are odd in  $z$ , and the oriented interval integral makes the
identity hold for every real  $z$  without a sign case split. Together with a
bracket for  $2 / \sqrt{\pi}$  (see two_div_sqrt_pi_mem) this carries polynomial
enclosures of  $T$  to enclosures of the error function. -/
theorem integral_exp_neg_sq (z :  $\mathbb{R}$ ) :
   $\int t \text{ in } (0:\mathbb{R})..z, \text{Real.exp } -(t^2) = z * \text{FnTag.erfT.eval } (z^2)$ 

```

Correspondence (Steele display  $\leftrightarrow$  Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed rule maps onto the bound statements as follows. The witness  $(f, n, r, R) \mapsto \text{EncWitness}$  (fields `func`, `deg`, `radius`, `rem`, plus the coefficient list governed by the displayed recurrence clause, recompute semantics per the flavor text); the three displayed check clauses  $\mapsto$  the definition of `encCheck`, whose acceptance is the single hypothesis of every soundness statement. The displayed conclusion  $\mapsto$  `encCheck_sound` (the absolute-value form) and `encCheck_sound_bracket` (the same content as a two-sided polynomial bracket — equal width, different packaging); `encCheck_point` instantiates the bracket at a rational point (the recentering step's constant legs), and `linear_interp_le` carries the endpoint-substitution step the consumers use. One deliberate width difference,

named outright: on the wire the coefficient vector is never serialized — the checker recomputes it — while the Lean witness carries the list explicitly and clause (i) checks it against the recurrence; the wire grain is therefore strictly narrower than the mechanized statement, and the coefficient-mutation receipts live kernel-side precisely because those mutations are unrepresentable on the wire. The two Lean packages prove the same statement shape over different tag libraries (the qtsl package adds the error-function kernel, with its meaning stated against the Gaussian integral and bridged by `integral_exp_neg_sq`); the HOL statements are the side-by-side port against transcTheory's natively per-index series — reindexing-free exactly where the Lean proof needed injection reindexing.

## Lean verifier receipts

module BernsteinLogicAux.Enclosure; bound build aeef11c4ffdf... (bound from receipt c61f02086268c593, 2026-07-21T00:02:40Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| ec9147f09c619209 | 2026-07-17T21:43:40Z | build+replay+hardened+decorrelated | current |
| ef188baecdf2d5cc | 2026-07-19T18:23:51Z | build+decorrelated                 | current |
| 74824f999e9c6c1a | 2026-07-20T01:21:03Z | build+decorrelated                 | current |
| c61f02086268c593 | 2026-07-21T00:02:40Z | build+decorrelated                 | current |
| 5f18c0fcd7885f0e | 2026-07-22T18:26:52Z | build+decorrelated                 | current |
| 0b776138b6a8c8d7 | 2026-07-31T21:59:44Z | build+decorrelated                 | current |
| 2ee240defc1ac528 | 2026-07-31T22:28:37Z | build+decorrelated                 | current |
| 1fb846e38f09ba02 | 2026-08-02T23:30:45Z | build+decorrelated                 | current |
| 2f17504ab4610d57 | 2026-08-03T16:26:49Z | build+decorrelated                 | current |
| c53cac58e2ed77d9 | 2026-08-03T22:45:33Z | build+decorrelated                 | current |

module QtslAux.Enclosure; bound build 7de2b931c19f... (bound from receipt 840791fb258b3984, 2026-07-26T15:38:06Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 969915618955be8f | 2026-07-25T11:27:08Z | build+decorrelated                 | current |
| 840791fb258b3984 | 2026-07-26T15:38:06Z | build+replay+hardened+decorrelated | current |

### gate-layer coverage of the bound build:

- build (lake build (elaboration)) current
- replay (leanchecker (official kernel replay)) current
- hardened (safe\_verify (hardened screen)) current
- decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_enclosure, HOL4 development script bern\_enclosureScript.sml script bound

### Definition encCheck\_def

```

Definition encCheck_def:
  encCheck w ⇔
    LENGTH w.coeffs = w.n + 1 ∧
    w.coeffs = GENLIST (maclaurin w.tag) (w.n + 1) ∧
    0 ≤ w.r ∧ w.r < &w.n + 2 ∧
    tailBound w.n w.r ≤ w.R
End

```

### Theorem encCheck\_sound

```
Theorem encCheck_sound:
  ∀w x.
    encCheck w ∧ abs x ≤ real_of_rat w.r ⇒
    abs (fnEval w.tag x - evalPolyR w.coeffs x) ≤ real_of_rat w.R
```

#### Theorem encCheck\_sound\_bracket

```
Theorem encCheck_sound_bracket:
  ∀w x.
    encCheck w ∧ abs x ≤ real_of_rat w.r ⇒
    evalPolyR w.coeffs x - real_of_rat w.R ≤ fnEval w.tag x ∧
    fnEval w.tag x ≤ evalPolyR w.coeffs x + real_of_rat w.R
```

## HOL verifier receipts

**documentary** Honest partial, stated as such: the port deliberately carries the MEANING theorem (`encCheck_sound` against `transcTheory's exp / sin / cos`), unlike the neighbouring slices whose meaning stays Lean-side. Of the theory's fifteen theorems, the eleven foundation theorems (datatypes through the factorial bound, including the full rational-to-real cast bridge) were machine-verified in a batch build; the four analysis theorems (`TERM_LE`, `MAJORANT_SUMS`, `TAIL_MAIN`, `encCheck_sound` with corollaries) are assembled from interactively-verified tactic pieces, and a full in-file batch re-verification currently exceeds the memory of every build host used to date (~117 GB resident observed at failure; a ≥256 GB host is the named path). Until that build lands, the HOL half is a stated port with interactive verification, not a batch-verified theory — the badge here says exactly that. The `encReceipts` table (seven rows, expected TTTFFFF) replays the same seven witnesses that appear byte-identically in the Lean kernel receipts and the reference checker's battery: three accepts (including one with  $R$  exactly at the tail bound) and four single-mutation rejects.

HOL receipts are documentary: quoted results of `hol-strict` runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: admissible rule (tier 2) — admissible by *reduction to verified checking machinery*, the taxonomy's recorded boundary case. The reduction is stated first; the tail bound it stands on is then proved in full, because it is the section's one piece of genuinely new mathematics — and the one place in the whole calculus where the meaning of a transcendental function enters the trusted base.

**Recipe R69.1** (the reduction — what the checker replays). A `PExprTypeEnclosure` row is  $(f, n, r, R)$ : a function tag  $f$  from the registered library  $F$ , a truncation degree  $n \in \mathbb{N}$ , a radius  $r \in \mathbb{Q}_{\geq 0}$ , and a remainder budget  $R \in \mathbb{Q}$ . The discharge is three replays, all exact  $\mathbb{Q}$ :

1. *coefficient recompute* — the Maclaurin prefix  $a_0, \dots, a_n$  is regenerated from  $(f, n)$  by the tag's exact rational recurrence; no coefficient ever rides the certificate, so a laundered coefficient list is unrepresentable rather than detected.
2. *radius guard* —  $0 \leq r < n + 2$ , the side condition under which the closed-form tail bound below is finite and valid.
3. *remainder comparison* —  $R \geq \text{tail}(n, r)$  where  $\text{tail}(n, r) = \frac{r^{n+1}}{(n+1)!} \cdot \frac{n+2}{n+2-r}$ , a comparison of two explicit rationals.

Nothing here evaluates  $f$ . The conclusion — for all real  $x$  with  $|x| \leq r$ ,  $|f(x) - \sum_{k \leq n} a_k x^k| \leq R$ , i.e. the two-sided polynomial bracket  $P - R \leq f \leq P + R$  on  $[-r, r]$  — is the statement of the kernel-checked soundness theorem, and that theorem is where the real `exp`, `sin`, `cos` (as the proof assistant's own library defines them) enter, once, for the whole calculus.

**Proposition R69.2** (uniform Maclaurin tail bound). *Let  $f$  be entire with Maclaurin coefficients satisfying  $|a_k| \leq 1/k!$  for all  $k$  (rationality of the  $a_k$  is what makes the check exact; the bound itself needs only the domination). For  $n \in \mathbb{N}$  and  $0 \leq r < n + 2$ ,*

$$\sup_{|x| \leq r} \left| f(x) - \sum_{k \leq n} a_k x^k \right| \leq \frac{r^{n+1}}{(n+1)!} \cdot \frac{n+2}{n+2-r} = \text{tail}(n, r).$$

*Proof.* For  $|x| \leq r$  the omitted tail is dominated termwise:

$$\left| \sum_{k > n} a_k x^k \right| \leq \sum_{k > n} \frac{r^k}{k!}.$$

Write  $N = n + 1$  and index the tail by  $j \geq 0$  as  $k = N + j$ . The factorial lower bound  $(N + j)! \geq N! (N + 1)^j$  — induction on  $j$ : the base is trivial, and the step multiplies by  $N + j + 1 \geq N + 1$  — gives

$$\frac{r^{N+j}}{(N+j)!} \leq \frac{r^N}{N!} \left( \frac{r}{N+1} \right)^j.$$

The guard  $r < n + 2 = N + 1$  makes the ratio  $q = r/(N + 1)$  satisfy  $0 \leq q < 1$ , so the geometric series sums to  $1/(1 - q) = (N + 1)/(N + 1 - r)$ , and

$$\sum_{k > n} \frac{r^k}{k!} \leq \frac{r^N}{N!} \cdot \frac{N + 1}{N + 1 - r} = \frac{r^{n+1}}{(n+1)!} \cdot \frac{n+2}{n+2-r}.$$

□

Soundness of the rule is now the two-line composition the mechanized statements carry: clause (1) certifies that the recomputed prefix agrees with  $f$ 's true coefficients (a per-tag recurrence identity, proven once per library member against the library definition of  $f$ ); every registered tag satisfies the domination  $|a_k| \leq 1/k!$  (a per-tag lemma — for  $\exp$  an identity, for  $\sin / \cos$  the odd/even zero pattern, for the reduced error-function kernel  $|a_k| = 1/(k! (2k + 1))$ ); and Proposition R69.2 turns clauses (2)–(3) into the bracket. Accepting a witness with  $R$  *exactly* at  $\text{tail}(n, r)$  is sound — the receipt battery pins one such row on purpose — and every strengthening of the check (a smaller  $R$ , a larger  $r$ , a shortened prefix, a flipped coefficient sign) is refused by construction, which is what the four mutation receipts witness.

*Remark R69.3* (scope — what the class is, and is not). The hypotheses of Proposition R69.2 are the exact scope of the rule: entire, exact- $\mathbb{Q}$  coefficients, dominated by  $1/k!$  (entire of exponential type, normalized so  $e^{|x|}$  dominates coefficientwise). Each condition excludes something familiar. Finite-radius functions —  $\log$ , rational powers — fail entirety: a variant for them would carry a radius-ratio geometric tail *and* a domain side condition, and they are certified today by sibling routes built on their own bounds rather than by stretching this rule.  $\tan$  is meromorphic and out entirely; consumers build it from  $\sin / \cos$  and division. And  $e^{x^2}$  is entire but not of exponential type — its coefficients escape the domination, so no choice of  $n$  rescues the one uniform tail formula. Conversely, admitting a new in-class function costs exactly one coefficient recurrence and one domination case; the tail argument, the check, and the soundness statement shape are untouched — the error-function kernel entered this way, under a  $y = z^2$  reduction chosen precisely to land its coefficients inside the domination.

*Remark R69.4* (slack is the producer's business). The rule certifies one truncation at one radius; tightness is policy, not soundness. Recentering (the point brackets of the mechanized `encCheck_point`, combined by Taylor-model arithmetic over  $\mathbb{Q}$ ) and subdivision shrink bracket width first-order, while raising the degree shrinks it factorially — so producers buy accuracy with  $n$  and locality with cells, and every combination step re-enters the calculus as ordinary polynomial rows. On the measured acceptance systems, degree 9 at unit radius gave remainders near  $10^{-7}$ , orders below the certificate margins consuming them.



## 15.8 Morse Topology

### Rule 70. QMORSEDISCHARGE (core rule)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QMorseDischarge*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\begin{array}{l}
 L : \text{QBlockLattice (blocks over one rational grid)} \\
 \{F_f \vdash \sigma_f \mathcal{A}\ell_f > 0\}_{f \in \text{facets}(L)} \text{ (facet-sign rows: } \ell_f \text{ the facet's normal coordinate, } \sigma_f \in \{\pm 1\} \text{ the declared crossing sign, } F_f \text{ the facet)} \\
 h : \text{QCellMap (cell} \rightarrow \text{vertex} \sqcup \{\text{transit}\}) \quad \{\vdash T(b) \subseteq \rho(h(b))\}_{h(b) \neq \text{transit}} \text{ (containments, per vertex cell)} \\
 \{T(b) \vdash \mathcal{A}V_b + \varepsilon_b \leq 0\}_{h(b) = \text{transit}}, \quad \varepsilon_b \in \mathbb{Q}_{>0} \text{ (transits, per transit cell)} \\
 \{\vdash \rho(u) \cap \rho(v) = \emptyset\}_{u \neq v \in G}, \quad \{\vdash \rho(v) \subseteq D\}_{v \in G} \text{ (}\rho\text{-wf: rows where not box-aligned, recomputed where box-aligned)} \\
 \text{CR}(\mathcal{M}|_D) \subseteq \bigcup_{h(b) \neq \text{transit}} T(b) \text{ (cell-level chain-recurrence cover)} \\
 \text{MorseAssembly}(L, h, G, \rho) \text{ (the assembly ledger, defined below)} \\
 \hline
 \mathcal{M} \models \text{Morse}(D, G, \rho)
 \end{array}$$

the lattice payload carries name, grid, arena, and blocks, and enters as certificate data (a shared reference; the rows impose the claims); its per-facet sign rows are displayed inline as the second premise — each certifies a uniform strict sign of the flow across one grid facet ( $\mathcal{A}\ell_f$  is the vector field's facet-normal component on the facet, frozen at the gridline), and the transit certificates  $V_b$  ride the same payload. The  $\rho$ -wf obligations split by alignment: box-aligned disjointness/containment is bounded interval arithmetic, recomputed; non-box-aligned pairs are per-pair BL rows. The displayed *MorseAssembly* package comprises the assembly ledger — arena compactness, per-cell-pair crossing certificates on shared facets, orbit-interval order rows — plus profile identification and the ambient metric structure chain recurrence needs (model-class); the chain-recurrence cover is displayed as its own premise. Checker-recomputed: lattice/cell-map presentation conformance (closure recompute, join-irreducible readout, the acyclicity check on the spec graph, the arena inside the top member's realization — the nonemptiness row).

QMORSEDISCHARGE concludes the global Morse judgment: all recurrence of the model inside the arena lives in the declared regions  $\rho(v)$ , and transitions between them run only down the declared order. It is reduction-to-base in its cleanest form: the certificate is a lattice of blocks over one rational grid whose every internal facet carries a strictly-inward sign row of the base calculus, a map  $h$  sending the lattice's cells to spec vertices or to “transit”, containment rows placing each vertex cell inside its region, and per-transit-cell decrease rows  $\mathcal{A}V + \varepsilon \leq 0$  — every leaf a sequent the base kernel already checks, and zero new kernel rules.

A producer reaches for it from the refinement loop: synthesize the block family (splitting cells where a facet sign fails to certify, merging where refinement is not needed), pick transit observables from low-degree templates, and match cells to the spec graph. Failure mode is refusal with a pointer: a spec whose resolution exceeds what the certificate can contain reports exactly which cells obstruct, and that report is itself the useful artefact.

What the checker recomputes rather than trusts: the lattice closure under union and grid-level intersection; the readout — the join-irreducible members, their unique lower covers, the cells  $T(b) = \text{cl}(b \searrow b^-)$ , and the induced cell order; convexity of  $h$ 's fibres and monotonicity of  $h$  on vertex cells; acyclicity of the spec graph; and the box-aligned well-formedness checks on  $\rho$ . The witness the rule assembles is  $M(v) := \text{Inv}(\bigcup_{h(b)=v} T(b))$  — the invariant part of the fibre's cell-union, not the union of per-cell invariant parts; the proof section shows the distinction is load-bearing.

Each hypothesis earns its place. Drop the containment rows and the location clause fails. Drop the transit rows and a transit cell could harbour recurrence the  $\rho$ -cover misses. Drop fibre convexity and the judgment could assert a strict order the certificate does not enforce. Drop facet strictness and three arguments die at once: no-facet-riding, fibre disjointness, and the chain-margin bound (“strictness pays a second time”).

The mechanization is conditional, honestly: the bound theorems prove the witness assembly with a ledger of named hypothesis families — G3 (compactness of the invariant parts), discharged outright from model-class facts; G4 (disjointness of distinct fibres), reduced to transit certificates on fibre overlaps; G5 (the chain-recurrence cover), reduced to the cell-level cover; and G6 (the order clause), which remains a hypothesis of the statements bound here, its content being the block lattice's orbit combinatorics. The proof section renders each as mathematics, together with G2, the pointwise-to-integrated bridge that every certificate-shaped hypothesis consumes.

**notes** The judgment is deliberately an “at most” claim: empty Morse sets are permitted, so the rule certifies confinement of recurrence and of transition order, never existence of structure — the existence half (Conley indices, connection witnesses) is a separately priced annotation tier, absent by design. End-to-end certificate receipts exist in the development's exact-arithmetic certificate-checker lane, which accepts both direct and product-assembled Morse-family certificates against a mutant battery; those receipts ride the checker's own test suite, not this book's receipt store.



## Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The rule's official certificate content in the geolog theory the checker loads. **QMorse** is the concluded judgment kind — the graph-over-regions claim, with its arena — and **QMorseDischarge** is this rule's payload entity, field for field the display's premise kit (lattice, cell map, containment/transit/well-formedness proof-leaf lists, and the per-transit-cell certificate polynomials in `transit_certs`). The arena tie is the quoted axiom: a discharged claim's arena is the lattice's, as a stated equation between hash-consed values. The R1 side condition is the demand relation `qtsl_morse_readout` — lattice closure and readout recompute, discharged by bounded evaluation at the engine's ground seam. And the transit rows'  $\Delta V$  is not an ambient operator: `qtsl_gen_form` is the derived generator-form seam tying the subject system and the certificate component  $V$  to the hypernet that says how  $V$  can change with duration — recomputed, never trusted as stored data.

theory Qtsl . **QMorse**

quoted from the theory source at render

```
QMorse : Set; // entity: graph-over-regions claim
```

theory Qtsl . **QMorse**/\*

quoted from the theory source at render

```
QMorse/judgment : QMorse -> QJudgment;
QMorse/graph : QMorse -> QDag;
QMorse/rho : QMorse -> QVertexRegionList;
QMorse/arena : QMorse -> QArena;
```

theory Qtsl . **QMorseDischarge**

quoted from the theory source at render

```
QMorseDischarge : Set; // entity: the lattice-reading discharge
```

theory Qtsl . **QMorseDischarge**/\*

quoted from the theory source at render

```
QMorseDischarge/proof : QMorseDischarge -> QProof;
QMorseDischarge/lattice : QMorseDischarge -> QBlockLattice;
QMorseDischarge/h : QMorseDischarge -> QCellMap;
QMorseDischarge/containments : QMorseDischarge -> QProofList;
QMorseDischarge/transits : QMorseDischarge -> QProofList;
QMorseDischarge/transit_certs : QMorseDischarge -> QCellPolyList;
QMorseDischarge/rho_wf : QMorseDischarge -> QProofList;
```

theory Qtsl . **ax/qmorsedischarge/arena**

quoted from the theory source at render

```
chase ax/qmorsedischarge/arena : forall r : QMorseDischarge,
  p : QProof, j : QMorse, lat : QBlockLattice, k : QJudgment,
  ac : Shape, gc : QGridCx.
  r QMorseDischarge/proof = p,
  p QProof/concludes = k,
  j QMorse/judgment = k,
  r QMorseDischarge/lattice = lat,
  lat QBlockLattice/arena = ac,
  lat QBlockLattice/cx = gc
|- j QMorse/arena = {cells: ac, cx: gc} qarMk;
```

theory Qtsl . **qtsl\_morse\_readout**

quoted from the theory source at render

```
qtsl_morse_readout : {dag: QDag, h: QCellMap, lat: QBlockLattice,
  rho: QVertexRegionList} -> Prop;
```

theory Qtsl . **qtsl\_gen\_form**

quoted from the theory source at render

```
qtsl_gen_form : {sys: System, cert: HypernetRef, gen: HypernetRef}
  -> Prop; // side-condition (oracle discharge / checker
```

## Lean statement

BernsteinLogic.QtslSoundness . **qMorseDischarge\_sound**

statement bound

```

/-- QMorseDischarge soundness – Lemma M(v), transcribed as the witness
assembly `M(v) := MaxInv(U_{h(b)=v} T(b))` (the note's own definition,
"the invariant part of the fiber's cell-union, NOT the union of per-cell
invariant parts" – its self-caught first-draft defect). PROVED outright:
the location clause (from the containment rows) and the invariance
clause (`maxInv_isInvariant`). INTERFACE (gaps G3–G6, each the named
prose paragraph of the note's M(v)/M(ii)/M(iii) proofs): compactness,
fiber disjointness, the chain-recurrence cover – into which the transit
rows enter via `transit_maxInv_empty` (M(iv), mechanized above) – and
the order clause. A successor leg mechanizing a gap discharges the
corresponding hypothesis; the statement is already the roster's. -/
theorem qMorseDischarge_sound (φ : Semiflow X) (D : Set X)
  (ρ : Vtx → Set X) (cells : ι → Set X) (hmap : ι → Option Vtx)
  (hContain : ∀ i v, hmap i = some v → cells i ⊆ ρ v)
  (hCompact : ∀ v, IsCompact (φ.MaxInv (fiberUnion cells hmap v)))
  (hDisj : ∀ v w, v ≠ w →
    Disjoint (φ.MaxInv (fiberUnion cells hmap v))
      (φ.MaxInv (fiberUnion cells hmap w)))
  (hCR : ∀ x ∈ D, ChainRecurrentIn φ D x →
    ∃ v, x ∈ φ.MaxInv (fiberUnion cells hmap v))
  (hOrder : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    (∀ v, ¬ Set.range γ ⊆ φ.MaxInv (fiberUnion cells hmap v)) →
    ∃ v w, w < v ∧ alphaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap v) ∧
      omegaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap w)) :
  MorseSat φ D ρ

```

BernsteinLogic.QtslSoundness.MorseWitness

statement bound

```

/-- A witness for `M ⊆_D Morse(G, ρ)` – the graded Morse-decomposition
semantics, clause for clause. `Vtx` carries the QDag's reachability
order (the edge set is presentation; v1 semantics consumes only the
reachability order); empty `M v` are permitted (the deliberate v1
"at most this structure" polarity). Order convention: source-to-sink,
`w < v` = connecting orbits may run from `M v` down to `M w`. -/
structure MorseWitness (φ : Semiflow X) (D : Set X) [PartialOrder Vtx]
  (ρ : Vtx → Set X) where
  /-- the family `{M(v)}` of Morse sets. -/
  M : Vtx → Set X
  /-- each compact... -/
  isCompact : ∀ v, IsCompact (M v)
  /-- ...invariant... -/
  invariant : ∀ v, φ.IsInvariantIn (M v)
  /-- clause 1 (location): `M(v) ⊆ ρ(v)`. -/
  located : ∀ v, M v ⊆ ρ v
  /-- ...pairwise disjoint. -/
  disjoint : ∀ v w, v ≠ w → Disjoint (M v) (M w)
  /-- clause 2 (completeness of the cover): every chain-recurrent point
of `φ|_D` lies in some `M(v)`. -/
  crCover : ∀ x ∈ D, ChainRecurrentIn φ D x → ∃ v, x ∈ M v
  /-- clause 3 (order): a complete orbit in `D` contained in no `M(v)`
connects a strictly higher Morse set to a strictly lower one. -/
  order : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    (∀ v, ¬ Set.range γ ⊆ M v) →
    ∃ v w, w < v ∧ alphaSet γ ⊆ M v ∧ omegaSet γ ⊆ M w

```

BernsteinLogic.QtslSoundness.MorseSat

statement bound

```

/-- The `QMorse` judgment's semantics: some family witnesses it. -/
def MorseSat (φ : Semiflow X) (D : Set X) [PartialOrder Vtx]
  (ρ : Vtx → Set X) : Prop

```

BernsteinLogic.QtslDischarge.qMorseDischarge\_sound\_modelClass

statement bound

```

/-- `qMorseDischarge_sound` with the gap ledger at its post-discharge
state. Same conclusion, same rows; the interface hypotheses are now:

```

```

* model-class facts (the ODE-doctrine profile's warrants, per the
  System notion): spacewise-continuous flow, compact cells –
  G3 is GONE, these replace it;
* `hOverlap` – G4's residue: shared-facet overlaps carry transit
  certificates ("strictness pays a second time");
* `hCell` – G5's residue: the CELL-level chain-recurrence cover
  (M(iii), KMV-III, literature strength);
* `hTransitCert` – M(iv)'s input, now in certificate form;
* `hOrder` – G6, unchanged (wall named in the module header).

```

The witness assembly and the proved clauses (location, invariance) are inherited from `qMorseDischarge\_sound` untouched. -/

```

theorem qMorseDischarge_sound_modelClass (φ : Semiflow X) (D : Set X)
  (ρ : Vtx → Set X) (cells : ι → Set X) (hmap : ι → Option Vtx)
  (hflow : ∀ t, Continuous (φ.flow t))
  (hcells : ∀ i, IsCompact (cells i))
  (hContain : ∀ i v, hmap i = some v → cells i ⊆ ρ v)
  (hOverlap : ∀ v w, v ≠ w →
    φ.HasTransitCert
      (fiberUnion cells hmap v n fiberUnion cells hmap w))
  (hCell : ∀ x ∈ D, ChainRecurrentIn φ D x → ∃ i, x ∈ φ.MaxInv (cells i))
  (hTransitCert : ∀ i, hmap i = none → φ.HasTransitCert (cells i))
  (hOrder : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    (∀ v, ¬ Set.range γ ⊆ φ.MaxInv (fiberUnion cells hmap v)) →
    ∃ v w, w < v ∧ alphaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap v) ∧
    omegaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap w)) :
  MorseSat φ D ρ

```

BernsteinLogic.QtslDischarge.**maxInv\_isCompact**

statement bound

```

/-- G3 discharged: `MaxInv` of a compact region is compact, given a
Hausdorff state space and spacewise-continuous flow maps – the "flow
continuity" the gap ledger names, and nothing else. Route: the set of
complete orbits inside `C` is a closed subset of the compact product
`V_ : ℝ, C` (Tychonoff; orbit constraints are equalizers of
continuous maps, closed in T2), and `MaxInv C` is its continuous image
under evaluation at 0. -/
theorem maxInv_isCompact [T2Space X] {φ : Semiflow X} {C : Set X}
  (hflow : ∀ t, Continuous (φ.flow t)) (hC : IsCompact C) :
  IsCompact (φ.MaxInv C)

```

BernsteinLogic.QtslDischarge.**fiberUnion\_isCompact**

statement bound

```

/-- Finitely many compact cells have compact fiber unions. -/
theorem fiberUnion_isCompact {Vtx ι : Type*} [Finite ι]
  {cells : ι → Set X} {hmap : ι → Option Vtx}
  (hcells : ∀ i, IsCompact (cells i)) (v : Vtx) :
  IsCompact (fiberUnion cells hmap v)

```

BernsteinLogic.QtslDischarge.**HasTransitCert.disjoint\_maxInv**

statement bound

```

/-- The G4 reduction ("strictness pays a second time"): if the overlap of two regions carries a transit
certificate, their maximal invariant sets are disjoint – a common
point would be forward-confined to the overlap by
`maxInv_forward_confined` applied on both sides. No backward
uniqueness, no injectivity of the flow maps. -/
theorem HasTransitCert.disjoint_maxInv {φ : Semiflow X} {C₁ C₂ : Set X}
  (h : φ.HasTransitCert (C₁ n C₂)) :
  Disjoint (φ.MaxInv C₁) (φ.MaxInv C₂)

```

BernsteinLogic.QtslDischarge.**HasTransitCert.maxInv\_empty**

statement bound

```

/-- Lemma M(iv) through the certificate interface: a transit-certified
region has empty maximal invariant set. (Shorter than the original
route: forward confinement suffices.) -/
theorem HasTransitCert.maxInv_empty {φ : Semiflow X} {C : Set X}
  (h : φ.HasTransitCert C) : φ.MaxInv C = ∅

```

## BernsteinLogic.QtslDischarge.crCover\_assemble

statement bound

```

/-- The cover-completeness paragraph of Lemma M(v), mechanized: the
fiber-level chain-recurrence cover follows from the CELL-level cover
(M(iii), the KMV-III margin argument – the residue, deliberately at
literature strength) plus transit certificates on transit cells
(M(iv), through `HasTransitCert.maxInv_empty`). -/
theorem crCover_assemble {φ : Semiflow X} {D : Set X}
  {cells : ι → Set X} {hmap : ι → Option Vtx}
  (hCell : ∀ x ∈ D, ChainRecurrentIn φ D x → ∃ i, x ∈ φ.MaxInv (cells i))
  (hTransit : ∀ i, hmap i = none → φ.HasTransitCert (cells i)) :
  ∀ x ∈ D, ChainRecurrentIn φ D x →
    ∃ v, x ∈ φ.MaxInv (fiberUnion cells hmap v)

```

## BernsteinLogic.QtslOrderAssembly.qMorseDischarge\_sound\_orderAssembled

statement bound

```

/-- `qMorseDischarge_sound_modelClass` with G6 at its residue: `hOrder`
is DERIVED (`orderClause_assemble`), and the interface carries instead
arena compactness (`hD`, a model-class fact: the arena is a compact
box-union), the two R1 shadows (`hConv`, `hMono`), and G6a
(`hInterval` – M(ii)'s orbit-interval conclusion, prose tier A; see
`orderClause_assemble`). No statement in `QtslSoundness.lean` or the
leg-5 module changes shape. -/
theorem qMorseDischarge_sound_orderAssembled (φ : Semiflow X) (D : Set X)
  (ρ : Vtx → Set X) (cells : ι → Set X) (hmap : ι → Option Vtx)
  (hD : IsCompact D)
  (hflow : ∀ t, Continuous (φ.flow t))
  (hcells : ∀ i, IsCompact (cells i))
  (hContain : ∀ i v, hmap i = some v → cells i ⊆ ρ v)
  (hOverlap : ∀ v w, v ≠ w →
    φ.HasTransitCert
      (fiberUnion cells hmap v n fiberUnion cells hmap w))
  (hCell : ∀ x ∈ D, ChainRecurrentIn φ D x → ∃ i, x ∈ φ.MaxInv (cells i))
  (hTransitCert : ∀ i, hmap i = none → φ.HasTransitCert (cells i))
  (hConv : ∀ i j k v, i ≤ j → j ≤ k →
    hmap i = some v → hmap k = some v → hmap j = some v)
  (hMono : ∀ i j v w, i ≤ j →
    hmap i = some v → hmap j = some w → v ≤ w)
  (hInterval : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    ∃ βα bw : ι, βα ≤ bw ∧
      alphaSet γ ⊆ cells βα ∧ omegaSet γ ⊆ cells bw ∧
      ∀ t : ℝ, ∃ i, βα ≤ i ∧ i ≤ bw ∧ γ t ∈ cells i) :
  MorseSat φ D ρ

```

## BernsteinLogic.QtslEventualMember.qMorseDischarge\_sound\_eventualMember

statement bound

```

/-- `qMorseDischarge_sound_orderAssembled` with G6a DERIVED: the
`hInterval` hypothesis is replaced by the block-family interface – the
five R1-recompute combinatorial rows, per-member forward invariance
(G1's class), closedness of join-irreducible realizations, and the
strict-inwardness shadows (F1's meet-confinement clause and the two
F2 absorption facts). No statement in `QtslSoundness.lean` or the
leg-5/leg-6 modules changes shape. What remains of G6 after this leg:
nothing under the name G6a – the residue classes are those already on
the ledger (G1b facet-row bridges per member, G2-class
pointwise-to-integrated for the shadows, R1 recompute conformance). -/
theorem qMorseDischarge_sound_eventualMember (φ : Semiflow X) (D : Set X)
  (ρ : Vtx → Set X) (cells : ι → Set X) (hmap : ι → Option Vtx)
  (blocks : β → Set X) (emb : ι → β) (cover : ι → β) (bD : β)
  (hD : IsCompact D)
  (hflow : ∀ t, Continuous (φ.flow t))
  (hcells : ∀ i, IsCompact (cells i))
  (hContain : ∀ i v, hmap i = some v → cells i ⊆ ρ v)
  (hOverlap : ∀ v w, v ≠ w →
    φ.HasTransitCert
      (fiberUnion cells hmap v n fiberUnion cells hmap w))

```

```

(hCell : ∀ x ∈ D, ChainRecurrentIn φ D x → ∃ i, x ∈ φ.MaxInv (cells i))
(hTransitCert : ∀ i, hmap i = none → φ.HasTransitCert (cells i))
(hConv : ∀ i j k v, i ≤ j → j ≤ k →
  hmap i = some v → hmap k = some v → hmap j = some v)
(hMono : ∀ i j v w, i ≤ j →
  hmap i = some v → hmap j = some w → v ≤ w)
(hbD : D ⊆ blocks bD)
(hJoin : ∀ b : β, ∀ x ∈ blocks b,
  ∃ i : ι, x ∈ blocks (emb i) ∧ blocks (emb i) ⊆ blocks b)
(hcover : ∀ i : ι, blocks (cover i) ⊆ blocks (emb i))
(hcellsSup : ∀ i : ι, blocks (emb i) \ blocks (cover i) ⊆ cells i)
(horder : ∀ i j : ι, blocks (emb i) ⊆ blocks (emb j) → i ≤ j)
(hclosed : ∀ i : ι, IsClosed (blocks (emb i)))
(hInv : ∀ b : β, φ.Invariant (blocks b))
(hMeet : ∀ a b : β, ∃ c : β, blocks c ⊆ blocks a ∧ blocks b ∧
  ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
  ∀ s t : ℝ, (∀ u, s ≤ u → u ≤ t → γ u ∈ blocks a ∧ blocks b) →
  ∀ u, s ≤ u → u ≤ t → γ u ∈ blocks c)
(hAbsorbOmega : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
  ∀ b : β, (omegaSet γ n blocks b).Nonempty →
  ∃ T : ℝ, ∀ t, T ≤ t → γ t ∈ blocks b)
(hAbsorbAlpha : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
  ∀ b : β, (alphaSet γ n blocks b).Nonempty →
  ∃ T : ℝ, ∀ t, t ≤ T → γ t ∈ blocks b) :
MorseSat φ D ρ

```

BernsteinLogic.QtslCrossing.qMorseDischarge\_sound\_cellCover

statement bound

```

/-- `qMorseDischarge_sound_modelClass` with G4 at its sharpened residue:
the fiber-overlap hypothesis `hOverlap` (one transit certificate on each
whole fiber-intersection – not the row shape) is replaced by per-cell-
pair crossing certificates `hCross` (the row shape: one certified facet
functional per shared facet). New model-class fact consumed: trajectory
continuity `htraj` – the same hypothesis G1's
`invariant_of_boundaryDwell` already takes. Everything else is
inherited unchanged; G5's and G6's residues are exactly as in
`qMorseDischarge_sound_modelClass`. -/
theorem qMorseDischarge_sound_cellCover (φ : Semiflow X) (D : Set X)
  (p : Vtx → Set X) (cells : ι → Set X) (hmap : ι → Option Vtx)
  (hflow : ∀ t, Continuous (φ.flow t))
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hcells : ∀ i, IsCompact (cells i))
  (hContain : ∀ i v, hmap i = some v → cells i ⊆ p v)
  (hCross : ∀ v w, v ≠ w → ∃ i j, hmap i = some v → hmap j = some w →
    φ.HasCrossingCert (cells i n cells j))
  (hCell : ∀ x ∈ D, ChainRecurrentIn φ D x → ∃ i, x ∈ φ.MaxInv (cells i))
  (hTransitCert : ∀ i, hmap i = none → φ.HasTransitCert (cells i))
  (hOrder : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    (∀ v, ¬ Set.range γ ⊆ φ.MaxInv (fiberUnion cells hmap v)) →
    ∃ v w, w < v ∧ alphaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap v) ∧
    omegaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap w)) :
MorseSat φ D ρ

```

## Correspondence (Steele display ↔ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed judgment  $\mathcal{M} \models \text{Morse}(D, G, \rho) \mapsto$  the goal  $\text{MorseSat } \phi D \rho = \text{Nonempty } (\text{MorseWitness } \phi D \rho)$  — the Morse note §2.2 clauses (compactness, invariance, location, disjointness, chain-recurrence cover, order) packaged as a structure, so the certified conclusion is existential exactly as the judgment reads; the witness assembly  $M(v) := \text{MaxInv}(\bigcup_{h(b)=v} T(b))$  lives inside the proof

term of `qMorseDischarge_sound`, never in its statement. The display's spec graph  $G$  enters as the binder pair  $\text{Vtx} + [\text{PartialOrder } \text{Vtx}]$ : the semantics consumes only the reachability order (the `MorseWitness` docstring says so in words), and the dag check on the presentation's edge set is R1 recompute with no Lean counterpart — correctly. Order polarity carried exactly: `hOrder`'s conclusion places  $\alpha$ -limits in the strictly *higher* set and  $\omega$ -limits in the lower ( $w < v \wedge \alpha\text{Set } \gamma \subseteq \dots v \wedge \omega\text{Set } \gamma \subseteq \dots w$ ), the display's “transitions run only down the declared order.”

- the E2 lattice  $L$  and cell map  $h \mapsto$  the abstract binders `cells :  $\iota \rightarrow \text{Set } X$`  and `hmap :  $\iota \rightarrow \text{Option } \text{Vtx}$`  (`none = transit`, the display's  $\sqcup \{\text{transit}\}$ ): the whole lattice stratum — closure recompute, join-irreducible readout,  $T(b)$ , fibre convexity and monotonicity of  $h$ , box-aligned  $\rho$ -wf — is presentation the checker recomputes (side condition R1), and the bound statements never see it; they quantify over arbitrary cell families, strictly wider than the display, in the safe direction.
- containment rows  $\mapsto$  `hContain (hmap i = some v  $\rightarrow$  cells i  $\subseteq$   $\rho$  v)`, one-to-one at the semantic stratum.
- transit rows  $\mathcal{AV} + \varepsilon \leq 0 \mapsto$  `hTransitCert i` (per `hmap i = none`) in the semantic-shadow form `HasTransitCert`: some observable  $V$ , bounded below on the cell, with explicit margin  $\varepsilon > 0$  and the *integrated* decrease clause  $V(\Phi_t x) \leq V(x) - \varepsilon t$  along confined prefixes. The pointwise-to-integrated bridge (gap G2; the `FlowDerivOn` profile identification) is outside this bound set — the roster witness cites its mechanization (`QtslMeanValue.lean`), not bound here.
- $\rho$ -wf rows  $\mapsto$  nothing: no bound hypothesis consumes them (fibre disjointness arrives through `hOverlap` instead) — R1 territory, like the lattice.
- the conditional ledger, exactly as bound: `qMorseDischarge_sound` carries G3–G6 as hypotheses (`hCompact/hDisj/hCR/hOrder`); `_modelClass` discharges G3 outright (`maxInv_isCompact`: Tychonoff, consuming `[T2Space X]`, spacewise flow continuity `hflow`, `[Finite  $\iota$ ]` + compact cells via `fiberUnion_isCompact` — model-class facts all), reduces G4 to overlap transit certificates (`HasTransitCert.disjoint_maxInv`: forward confinement on both sides, no backward uniqueness), reduces G5 to the cell-level cover `hCell (M(iii), KMV-III, literature strength)` through `crCover_assemble + HasTransitCert.maxInv_empty (M(iv))`, and carries G6 (`hOrder`) untouched.

Width note, both directions stated. Three assembly statements extend the bound set (the three assembly blocks below): the order assembly (`qMorseDischarge_sound_orderAssembled`, module `QtslOrderAssembly`) derives `hOrder` — G6 proper — carrying instead arena compactness, the two R1 shadows, and G6a, the orbit-interval row; the eventual-member assembly (`qMorseDischarge_sound_eventualMember`, module `QtslEventualMember`) derives G6a in turn from the block-family interface (the five R1-recompute combinatorial rows, per-member forward invariance, closedness, strict-inwardness shadows), leaving nothing of G6 under its own name; and the crossing assembly (`qMorseDischarge_sound_cellCover`, module `QtslCrossing`) sharpens G4 to the row shape — per-cell-pair crossing certificates on shared facets in place of whole-fibre-overlap transit certificates. All three are hash-bound, so the certified reach extends past `_modelClass`'s ledger and the conditional residue shrinks to the ledger's G1b/G2-class profile identifications (the `FlowDerivOn V (AV)` chain-rule bridges — the mean-value and Dini profile strata remain cited theorems, not bound), the cell-level chain-recurrence cover — now a displayed premise ( $\text{CR}(\mathcal{M}|_D) \subseteq \bigcup T(b)$  over vertex cells) —  $\mapsto$  `hCell` (the fibre-to-cell margin argument is the proof's stated residue), and the displayed `MorseAssembly` package  $\mapsto$  the remaining crossing/order/interface hypotheses plus recompute conformance. Binder cargo: `[PseudoMetricSpace X]` is consumed by chain recurrence itself —  $(\varepsilon, T)$ -chains need distances — ambient structure the assembly package carries.

## Lean verifier receipts

module `BernsteinLogic.QtslSoundness`; bound build 77005991d61d... (bound from receipt 8130a7a7a679cef4, 2026-08-04T01:16:06Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 8130a7a7a679cef4 | 2026-08-04T01:16:06Z | build+decorrelated | current |
| 9eb3706354082144 | 2026-08-04T02:23:12Z | build+decorrelated | current |

5 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.QtslDischarge`; bound build 4aa46910a600... (bound from receipt dd2946805f790f99, 2026-08-04T01:16:09Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| dd2946805f790f99 | 2026-08-04T01:16:09Z | build+decorrelated | current |
| e1bc68890e2aebf1 | 2026-08-04T02:23:11Z | build+decorrelated | current |

14 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.QtslOrderAssembly`; bound build `1fef28f78168...` (bound from receipt `f6e3628b18a7ed89`, 2026-07-20T08:27:03Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers                             | status  |
|-------------------------------|----------------------|------------------------------------|---------|
| <code>b066c7d1be7303f1</code> | 2026-07-19T04:29:30Z | build+decorrelated                 | current |
| <code>ef9ccb70d0449d3b</code> | 2026-07-19T09:07:23Z | build+decorrelated                 | current |
| <code>f6e3628b18a7ed89</code> | 2026-07-20T08:27:03Z | build+decorrelated                 | current |
| <code>08fb682d6ccdc7</code>   | 2026-07-20T20:44:50Z | build+decorrelated                 | current |
| <code>62d115665ebe5578</code> | 2026-07-20T21:01:19Z | build+replay+hardened+decorrelated | current |
| <code>b2fd112afe983043</code> | 2026-07-21T05:28:46Z | build+replay+hardened+decorrelated | current |

module `BernsteinLogic.QtslEventualMember`; bound build `9282e7682307...` (bound from receipt `839bcd47d53458bf`, 2026-07-20T08:27:05Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers                             | status  |
|-------------------------------|----------------------|------------------------------------|---------|
| <code>d9e8d15537b8c931</code> | 2026-07-19T11:09:44Z | build+decorrelated                 | current |
| <code>dcd494c897af133b</code> | 2026-07-19T11:29:34Z | build+decorrelated                 | current |
| <code>839bcd47d53458bf</code> | 2026-07-20T08:27:05Z | build+decorrelated                 | current |
| <code>0b348bff26bc835c</code> | 2026-07-20T20:44:43Z | build+decorrelated                 | current |
| <code>fbf234c5b555fbd6</code> | 2026-07-20T21:01:12Z | build+replay+hardened+decorrelated | current |
| <code>a67d63ba90d91fb5</code> | 2026-07-21T05:28:37Z | build+replay+hardened+decorrelated | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.QtslCrossing`; bound build `29a1be816112...` (bound from receipt `3dec8934e00e8465`, 2026-07-20T08:27:06Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt                       | minted               | layers                             | status  |
|-------------------------------|----------------------|------------------------------------|---------|
| <code>58226ade60f5d15c</code> | 2026-07-19T04:33:17Z | build+decorrelated                 | current |
| <code>8fa273e016e7de3b</code> | 2026-07-19T09:07:20Z | build+decorrelated                 | current |
| <code>3dec8934e00e8465</code> | 2026-07-20T08:27:06Z | build+decorrelated                 | current |
| <code>204c60d7cf1cf318</code> | 2026-07-20T20:44:39Z | build+decorrelated                 | current |
| <code>574f67e1c43258ae</code> | 2026-07-20T21:01:08Z | build+replay+hardened+decorrelated | current |
| <code>5e4d769e0abef0a1</code> | 2026-07-21T05:28:33Z | build+replay+hardened+decorrelated | current |

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

**comparator:** none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

Absent, and honestly so at two levels. The HOL4 development's object is the base-kernel checker cone — the first-order computational surface of the sequent-stratum certificate contract (dtype factorization, exact rational arithmetic, the catalogue check functions) — and the satisfaction-judgment stratum this rule concludes in has no HOL twin: no Morse-family or judgment-stratum definition exists in the ported scripts (verified against the port's definition inventory). The base leaves this rule consumes — facet-sign, containment, and decrease rows — ARE covered by the ported checker; what has no HOL counterpart is the judgment-stratum soundness development itself, which today is Lean-only. A port would be new assurance surface over the semiflow stratum, not a translation of existing checker code.



## HOL verifier receipts

**NONE**

no HOL verification receipt exists for this rule.

**Proof**

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail; the largest mechanized lemma set in this book. The rule’s mechanization is *conditional*: the soundness theorem is proved with a ledger of four named hypothesis families, of which one is discharged outright below, two are reduced to transit-certificate form, and the fourth — the order clause — is carried as an explicit hypothesis by the discharged-floor theorem and discharged by the separately bound assembly legs (ledger remark). The ledger remark near the end states each as mathematics. Every hypothesis is used and remarked on; the formal-anchor table maps each step to the Lean declaration that carries it.

**Setting.** Fix a forward semiflow  $\varphi$  on a metric state space  $X$  ( $\Phi_0 = \text{id}$ ,  $\Phi_{s+t} = \Phi_t \circ \Phi_s$ ), and a compact forward-invariant arena  $D \subseteq X$ . Four standard notions, exactly as the mechanized statements carry them:

(complete orbits)  $\gamma : \mathbb{R} \rightarrow X$  is a complete orbit when  $\gamma(s+t) = \Phi_t(\gamma(s))$  for all  $s \in \mathbb{R}$ ,  $t \geq 0$  — a two-sided trajectory compatible with the forward flow.

(maximal invariant part) for  $C \subseteq X$ ,

$$\text{Inv}(C) := \{x : \exists \gamma \text{ complete orbit, } \gamma(0) = x, \text{ range } \gamma \subseteq C\}.$$

An *invariant set*  $S$  (the sense fit for a semiflow) is one that is forward-invariant and satisfies  $S \subseteq \text{Inv}(S)$  — every point rides a complete orbit inside.

(limit sets)  $\omega(\gamma) := \bigcap_T \text{cl } \gamma([T, \infty))$  and  $\alpha(\gamma) := \bigcap_T \text{cl } \gamma((-\infty, T])$ .

(chains) an  $(\varepsilon, T)$ -chain from  $x$  to  $y$  inside  $D$  is a finite concatenation of orbit segments of duration  $\geq T$  with jumps of size  $< \varepsilon$  at points of  $D$  (generated by: a single step  $x, y \in D$ ,  $t \geq T$ ,  $\text{dist}(\Phi_t x, y) < \varepsilon$ ; and transitivity). A point is *chain-recurrent* in  $D$  when for every  $\varepsilon > 0$  and  $T$  there is an  $(\varepsilon, T)$ -chain from  $x$  to itself.

**The judgment.**  $\mathcal{M} \models \text{Morse}(D, G, \rho)$  holds when some family  $\{M(v)\}_{v \in V(G)}$  — indexed by the poset the graph  $G$  presents by reachability — satisfies: each  $M(v)$  compact and invariant; the family pairwise disjoint; (location)  $M(v) \subseteq \rho(v)$ ; (completeness of the cover) every chain-recurrent point of  $\varphi|_D$  lies in some  $M(v)$ ; and (order) every complete orbit in  $D$  contained in no single  $M(v)$  has  $\alpha(\gamma) \subseteq M(v)$  and  $\omega(\gamma) \subseteq M(w)$  for some  $w < v$ . The order runs source-to-sink:  $w < v$  means connecting orbits may run from  $M(v)$  down to  $M(w)$ , so the  $\alpha$ -side vertex is the higher one. Empty  $M(v)$  are permitted — the judgment is an “at most this structure” claim (recurrence confined to the  $\rho(v)$ , transitions confined to the order), never an existence claim; the existence half is a separately priced annotation tier (Conley indices, connection matrices), deliberately absent here.

**The rule’s data and the checker’s recompute.** The certificate supplies: the family object — a finite lattice of box-union blocks over one declared rational grid, each internal facet carrying a strictly-inward sign row (the `QBLOCKINV` section’s row species; the same rows serve both rules); a cell map  $h$  from the lattice’s cells to  $V(G) \sqcup \{\text{transit}\}$ ; per vertex cell a containment row  $T(b) \subseteq \rho(h(b))$ ; per transit cell a polynomial  $V$  with decrease rows  $\mathcal{A}V + \varepsilon \leq 0$ ; and well-formedness rows for  $\rho$ . The checker *recomputes*, rather than trusts, the lattice readout: closure under union and grid-level intersection, the join-irreducible members  $b$  with unique lower covers  $b^-$ , the cells  $T(b) := \text{cl}(b \setminus b^-)$ , the induced cell order, convexity of  $h$ ’s fibres and monotonicity of  $h$  on vertex cells, and acyclicity of  $G$ . Writing  $\text{fib}(v) := \bigcup_{h(b)=v} T(b)$  for the fibre’s cell-union, the witness the rule assembles is

$$M(v) := \text{Inv}(\text{fib}(v))$$

— the invariant part of the fibre’s cell-union, not the union of per-cell invariant parts. The distinction is load-bearing (Remark R70.11).

**Lemma R70.1** (Inv is located and invariant). (i)  $\text{Inv}(C) \subseteq C$ . (ii)  $\text{Inv}(C)$  is an invariant set.

*Proof.* (i) The witnessing orbit passes through  $x$  at time 0 with range in  $C$ . (ii) Two facts, both by time-shifting: shifting a complete orbit by any  $s$  ( $u \mapsto \gamma(u+s)$ ) is again a complete orbit, and every point  $\gamma(s)$  of an orbit with range in  $C$  therefore lies in  $\text{Inv}(C)$  with the shifted orbit as witness. Forward invariance: for  $x \in \text{Inv}(C)$  with orbit  $\gamma$ ,  $\Phi_t x = \gamma(0+t)$  is such a point. The complete-orbit-through clause: the same statement at every shift shows  $\text{range } \gamma \subseteq \text{Inv}(C)$ , so the witnessing orbit itself runs inside  $\text{Inv}(C)$ .  $\square$

**Lemma R70.2** (location). Under the containment rows,  $\text{fib}(v) \subseteq \rho(v)$ , hence  $M(v) \subseteq \rho(v)$  by Lemma R70.1(i).

*Proof.* A union of cells each inside  $\rho(v)$  is inside  $\rho(v)$ .  $\square$

**Lemma R70.3** (forward confinement). If  $x \in \text{Inv}(C)$  then  $\Phi_t x \in C$  for every  $t \geq 0$ .

*Proof.* The witnessing orbit’s forward half is the forward orbit:  $\Phi_t x = \Phi_t(\gamma(0)) = \gamma(t) \in C$ . This one-line lemma is the section’s quiet workhorse: a semiflow is deterministic *forward*, so membership in an Inv set constrains the actual forward trajectory — and nothing here assumes backward uniqueness (a semiflow may branch backward; complete orbits through a point need not be unique). The disjointness reduction below survives precisely because it only ever uses forward halves.  $\square$

**Lemma R70.4** (transit certificates kill invariance). Call  $C$  transit-certified when there are  $V : X \rightarrow \mathbb{R}$  and  $\varepsilon > 0$  with  $V$  bounded below on  $C$  and, along every forward trajectory confined to  $C$ ,  $V(\Phi_t x) \leq V(x) - \varepsilon t$ . Then no point is forward-confined to  $C$ ; in particular  $\text{Inv}(C) = \emptyset$ .

*Proof.* Suppose  $\Phi_t x \in C$  for all  $t$ , and let  $m$  bound  $V$  below on  $C$ . By the Archimedean property choose  $n$  with  $n\varepsilon > V(x) - m$ ; the decrease clause at horizon  $n$  gives  $V(\Phi_n x) \leq V(x) - \varepsilon n < m$ , contradicting  $\Phi_n x \in C$ . The Inv statement follows by Lemma R70.3. (The certified rows are pointwise  $-\mathcal{A}V + \varepsilon \leq 0$  on the cell, for the deterministic generator  $\mathcal{A} = f \cdot \nabla$  — and reach the integrated clause used here by integrating the derivative along the trajectory; that bridge is ledger item G2 in Remark R70.10, stated once and consumed by every certificate-shaped hypothesis in this section.)  $\square$

**Lemma R70.5** (disjointness from overlap certificates). *If  $C_1 \cap C_2$  is transit-certified, then  $\text{Inv}(C_1)$  and  $\text{Inv}(C_2)$  are disjoint.*

*Proof.* A common point is forward-confined to  $C_1$  and to  $C_2$  (Lemma R70.3 twice), hence to  $C_1 \cap C_2$  — which Lemma R70.4 forbids. No backward uniqueness, no injectivity of the flow maps.  $\square$

*Remark R70.6* (why the certificate sits on the overlap — “strictness pays a second time”). Distinct fibres’ cell-unions intersect only in shared boundary faces (cell interiors are disjoint by the readout’s construction — generically facets, but possibly lower-dimensional: two cells may meet in an edge or a corner alone). On a certified facet incident to a shared face, the facet polynomial supplies the observable — oriented to decrease: the rows certify inward growth of  $\ell$ , so the transit observable Lemma R70.4 consumes is  $V = -\ell$ , bounded below on the compact overlap, with the certified strict rate as its  $\varepsilon$ ; the shared face lies in  $\ell$ ’s zero set, and confinement to the face would force  $\ell \equiv 0$ . Where the overlap is a union of several faces, one signed observable per face does not by itself assemble into a single global certificate — the row-shaped assembly is per-cell-pair crossing certificates on shared facets, the development’s G4 sharpening, bound in this edition (anchor table) — and the hypothesis Theorem R70.9 takes is the assembled overlap certificate. The strict margin of the QBLOCKINV rows is thus consumed a second time, as the transit raw material for facet regions. One might hope to weaken the hypothesis to per-piece certificates — each  $C_i$  separately transit-certified — and conclude something about the union; that is false. On the circle rotation, two overlapping closed arcs each carry a monotone angular observable (a transit certificate for the arc), yet the union is the whole circle, every point recurrent: certificates on pieces do not compose across an overlapping union, because no global observable exists. The overlap form is the correct, and certifiable, hypothesis.

**Lemma R70.7** (compactness of the invariant parts). *Suppose  $X$  is Hausdorff, each flow map  $\Phi_t : X \rightarrow X$  is continuous, and  $C$  is compact. Then  $\text{Inv}(C)$  is compact. Consequently each  $\text{fib}(v)$ , a finite union of compact cells, has compact invariant part.*

*Proof.* Let  $K \subseteq X^{\mathbb{R}}$  (product topology) be the set of complete orbits with range in  $C$ :

$$K = \left( \bigcap_{s \in \mathbb{R}, t \geq 0} \{ \gamma : \gamma(s+t) = \Phi_t(\gamma(s)) \} \right) \cap \prod_{r \in \mathbb{R}} C.$$

Each orbit constraint is the equalizer of two continuous maps — evaluation at  $s+t$ , and  $\Phi_t \circ$  (evaluation at  $s$ ) — hence closed,  $X$  being Hausdorff; the product  $\prod_{\mathbb{R}} C$  is compact by Tychonoff (the one genuinely infinitary step of this section); a closed subset of a compact set is compact. And  $\text{Inv}(C)$  is exactly the image of  $K$  under evaluation at 0, a continuous map, hence compact. The fibre statement: a finite union of compact cells is compact, and the above applies to it.  $\square$

**Lemma R70.8** (the cover assembles from the cell level). *Suppose (cell-level cover) every chain-recurrent point of  $\varphi|_D$  lies in  $\text{Inv}(T(b))$  for some cell  $b$ , and (transit rows) every transit cell is transit-certified. Then every chain-recurrent point of  $\varphi|_D$  lies in some  $M(v) = \text{Inv}(\text{fib}(v))$ .*

*Proof.* Let  $x$  be chain-recurrent,  $x \in \text{Inv}(T(b))$ . If  $h(b) = \text{transit}$ , Lemma R70.4 makes  $\text{Inv}(T(b))$  empty — contradiction; so  $h(b)$  is a vertex  $v$ . Then  $T(b) \subseteq \text{fib}(v)$ , and  $\text{Inv}$  is monotone in its region (a complete orbit inside the cell is inside the fibre’s union), so  $x \in \text{Inv}(\text{fib}(v))$ .  $\square$

**Theorem R70.9** (soundness of QMORSEDISCHARGE, at the discharged floor). *Suppose: (model class)  $X$  is a Hausdorff metric space, each flow map  $\Phi_t$  is continuous, and every cell is compact; (containment rows)  $T(b) \subseteq \rho(h(b))$  for every vertex cell; (overlap certificates) for  $v \neq w$ , the overlap  $\text{fib}(v) \cap \text{fib}(w)$  is transit-certified; (cell-level cover) every chain-recurrent point of  $\varphi|_D$  lies in some cell’s invariant part; (transit rows) every transit cell is transit-certified; and (order clause) every complete orbit in  $D$  contained in no single  $M(v)$  connects a strictly higher fibre’s invariant part to a strictly lower one, in the sense of the judgment’s order clause with  $M(v) = \text{Inv}(\text{fib}(v))$ . Then  $\mathcal{M} \models \text{Morse}(D, G, \rho)$ .*

*Proof.* Take the family  $M(v) := \text{Inv}(\text{fib}(v))$  and check the witness clauses. *Location:* Lemma R70.2. *Invariance:* Lemma R70.1(ii). *Compactness:* Lemma R70.7 under the model-class facts. *Disjointness:* Lemma R70.5 applied to the overlap certificates. *Cover:* Lemma R70.8 from the cell-level cover and the transit rows. *Order:* the hypothesis, verbatim. All six clauses hold, so the family witnesses the judgment.  $\square$

*Remark R70.10* (the gap ledger, as mathematics). The rule was first mechanized as the bare witness assembly: location and invariance proved outright, and four clauses — labelled G3–G6 in the development, the labels this book adopts as the rule’s hypothesis names — taken as opaque interface hypotheses. The discharge stratum bound by this section moves the ledger to the state Theorem R70.9 exhibits:

G3 (compactness of the invariant parts): *proven*. Lemma R70.7 derives it from model-class facts alone — Hausdorff state space, spacewise-continuous flow, compact cells. No compactness hypothesis survives at the fibre level.

G4 (disjointness of distinct fibres’ invariant parts): *reduced, and sharpened to rows*. Lemma R70.5 trades it for transit certificates on pairwise overlaps; the residue is the overlap observable of Remark R70.6 — shared-facet functionals whose certified strict signs are the raw material — together with the G2 bridge below; and the row-shaped assembly (per-cell-pair crossing certificates on shared facets in place of whole-overlap certificates) is mechanized and bound (qMorseDischarge\_sound\_cellCover, anchor table).

G5 (completeness of the chain-recurrence cover): *reduced*. Lemma R70.8 trades the fibre-level cover for the cell-level one. The residue is the margin argument: a chain-recurrent point’s chains stay within its own cell, because the strict facet margin is a uniform

bound on admissible chain jumps near facets — an  $\varepsilon$ -chain with  $\varepsilon$  below the margin can no more exit a block than an orbit can. The classical home of this containment is Conley’s theory [47]; its algorithmic form is the chain-recurrence algorithm of Kalies–Mischaikow–van der Vorst [130].

*G6 (the order clause): a hypothesis of Theorem R70.9, discharged by the bound assembly legs.* Its mathematical content is the orbit combinatorics of the block lattice: the *eventual member* of a forward orbit — the minimal lattice member containing a forward tail, well-defined because the tail-containing family is finite, closed under the lattice meet, and nonempty: the arena sits inside the top member’s realization (one of the combinatorial interface rows the assembly legs consume), so the top member contains every forward tail — descends the cell order and stabilizes, giving every complete orbit in  $D$  extreme cells  $b_\omega \leq b_\alpha$  that bracket every visited cell; the fibre-convexity case analysis then converts bracketing into the witness’s order clause (the extremes are not transit, by Lemma R70.4 against their nonempty invariant limit sets; a same-fibre pair traps the whole orbit in its fibre by convexity, contradicting the “no single  $M(v)$ ” hypothesis; distinct fibres yield the strict order by monotonicity of  $h$ ). The development’s assembly legs mechanize exactly this — the cell order entering as theorem-interface data, the eventual-member construction proved against combinatorial row hypotheses — and are bound in this edition (qMorseDischarge\_sound\_orderAssembled, \_eventualMember; anchor table), so nothing of G6 survives under its own name: the residue is the R1-recompute combinatorial rows and strict-inwardness shadows those legs consume, plus the G2-class profile identifications below. Theorem R70.9 is stated at the discharged floor and takes the clause as an explicit hypothesis there; the rule’s conditional badge is judged against the full bound set.

*G2 (the pointwise-to-integrated bridge), named for completeness:* every “transit-certified” hypothesis above is the integrated semantic shadow of pointwise rows  $AV + \varepsilon \leq 0$ ; the bridge is one-variable calculus along trajectories, and its surviving analytic content is the identification of the certified polynomial rate with the trajectory-wise derivative on the region — the same model-class profile warrant as ledger item (a) of the QBLOCKINV section, whose G1 (block invariance) completes the shared ledger.

*Remark R70.11 (the witness is Inv of the union, not the union of Invs).* Define instead  $M'(v) := \bigcup_{h(b)=v} \text{Inv}(T(b))$  and the order clause fails: a complete orbit crossing between two cells of the *same* fibre lies in no per-cell invariant part (its range is in neither cell alone), so it belongs to no  $M'(v)$  while also connecting nothing strictly ordered — the “orbit contained in no Morse set” case triggers with no higher/lower pair to name. The correct definition absorbs such orbits into their fibre’s  $M(v)$ , which is exactly the role of fibre convexity: every cell between the orbit’s extremes maps to the same vertex, so the whole orbit lies in the fibre’s union and hence in its invariant part. Without convexity the judgment could assert a strict order the certificate does not enforce — convexity of  $h$ ’s fibres is well-formedness, not a nicety, and the checker recomputes it.

*Remark R70.12 (relation to the literature).* The judgment is Conley’s Morse decomposition [47] in certified form: a decomposition indexed by a two-element order is an attractor–repeller pair, and the general form is the lattice-refined iteration of that idea; the completeness clause is the containment of chain recurrence in the Morse sets, and the certified staircase of lattice members is a finite shard of a complete Lyapunov function (Conley’s fundamental theorem). The block-lattice route — attracting blocks closed under union and intersection, join-irreducibles, the tessellation into cells  $T(b)$ , and Birkhoff duality [22] between the attractor lattice and the Morse-graph poset — is the program of Kalies–Mischaikow–van der Vorst’s lattice series [131, 132]; the chain-recurrence cover’s algorithmic form is their earlier chain-recurrence algorithm [130]; and the computational tradition of certifying global dynamics by index machinery is surveyed in Mischaikow–Mrozek [183]. What the rule adds to that tradition is the artefact discipline: those pipelines end in trusted computation, while here every analytic claim rides certified sign rows — base kernel sequents in exact  $\mathbb{Q}$  — plus combinatorics the checker recomputes, with the semantic warrant being the lemma chain above.

| proof step                                     | formal anchor                                                                                                                                |
|------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| witness assembly, location + invariance proved | qMorseDischarge_sound (QtslSoundness.lean); maxInv_subset, maxInv_isInvariant                                                                |
| Lemma R70.1                                    | maxInv_subset, maxInv_invariant, maxInv_isInvariant                                                                                          |
| Lemma R70.2                                    | fiberUnion_subset; rows hContain                                                                                                             |
| Lemma R70.3                                    | maxInv_forward_confined (QtslDischarge.lean)                                                                                                 |
| Lemma R70.4                                    | HasTransitCert, HasTransitCert.not_forward_confined, HasTransitCert.maxInv_empty; the bare form transit_maxInv_empty (QtslSoundness.lean)    |
| Lemma R70.5                                    | HasTransitCert.disjoint_maxInv; hypothesis h0overlap                                                                                         |
| Lemma R70.7                                    | maxInv_isCompact (Tychonoff route), fiberUnion_isCompact                                                                                     |
| Lemma R70.8                                    | crCover_assemble; maxInv_mono, subset_fiberUnion; hypotheses hCell, hTransitCert                                                             |
| Theorem R70.9                                  | qMorseDischarge_sound_modelClass; order hypothesis h0order (ledger G6)                                                                       |
| judgment semantics                             | MorseWitness, MorseSat, EpsChainIn, ChainRecurrentIn, omegaSet, alphaSet                                                                     |
| G6’s assembly legs (bound in this edition)     | orderClause_assemble, eventualMember_interval, qMorseDischarge_sound_orderAssembled, _eventualMember (QtslOrderAssembly, QtslEventualMember) |
| G4’s row sharpening (bound in this edition)    | qMorseDischarge_sound_cellCover (QtslCrossing)                                                                                               |

## Rule 71. QMORSEPRODUCT (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QMorseProduct*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\mathcal{M}_1 \models \text{Morse}(D_1, G_1, \rho_1) \quad \mathcal{M}_2 \models \text{Morse}(D_2, G_2, \rho_2) \quad L_1, L_2 : \text{QBlockLattice (the factor lattices)}}{\mathcal{M}_1 \times \mathcal{M}_2 \models \text{Morse}(D_1 \times D_2, G_1 \times G_2, \rho_1 \times \rho_2)}$$

the two factor judgments enter as derivation premises (judgment-kind edges); the factor lattices enter as certificate data (shared references — the premise judgments' own certificates impose their claims), consulted by the checker's product tie.

QMORSEPRODUCT composes two global judgments along the state-space axis: disjoint variable sets, product arena, product spec graph, product regions. A producer reaches for it exactly where global structure is engineered compositionally — certify each factor subsystem's Morse graph, then take the product — because the alternative (re-discharging the composite through QMORSEDISCHARGE on the composite lattice) recomputes an exponentially larger readout and re-derives the cell map.

The rule is fenced to the deterministic-flow class, and the fence is the rule's own mathematical content: the composite certificate's readout — join-irreducibles, cells, cell order — must be the product of the factor readouts, and for flows that is a theorem (the attractor lattice of a product flow is the coproduct of the factor attractor lattices), while for maps it is false: products of maps exhibit emergent, phase-locked attractors outside the coproduct, so a composite readout computed from factor certificates would misrepresent the product's global structure. For maps the rule must not be derived at all.

What the checker recomputes: wire-disjointness of the factors, the product surgery on graphs and regions, and — on the certificate route — the composite lattice's readout. Facet and transit rows ride along by citation: every composite facet is normal to one factor's axis and its inwardness polynomial is that factor's, the other factor's coordinates entering as free box variables; a factor transit observable still decreases on cell  $\times$  anything, the other factor's drift differentiating it to zero.

The witness-level lemma chain is mechanized in full: the product construction on the semiflow stratum (Semiflow.prod), the factoring lemmas for orbits, limit sets, and chains, the six-clause witness transport (MorseWitness.prod), and the soundness theorem (qMorseProduct\_sound) — unconditional, with no interface hypotheses: from the two factor judgments, the product judgment, on the product arena, product order, and product regions. The bound statements consume neither compactness nor invariance of the arenas (the proof never needed them), and Hausdorffness exactly where the transport says “compact, hence closed.”

**notes** The proof section separates two strata deliberately: the witness-transport argument (compactness, invariance, location, disjointness, cover, order — all coordinate-wise) is elementary and does not consume continuity of time, while the readout-transport theorem that warrants the certificate route is flow-specific. The fence binds the rule as catalogued; the decomposition is recorded so the mechanization and any map-side successor judgment know where the flow hypothesis actually bites. The mechanization respects the split exactly: the bound statements carry the witness-level entailment in full and import nothing of Kalies–Wehbe — the fence's warrant remains at literature strength, where it belongs, since it certifies the canonicity of the certificate-route composite readout rather than the satisfaction-level soundness bound here.

### Lean statement

BernsteinLogic.QtslProduct . qMorseProduct\_sound

statement bound

```
-- QMorseProduct soundness, witness level (thm:qmp-sound; the bind
target). From the two factor judgments — the rule's E1 premises `left,
right : QMorse` — conclude the product judgment on the product arena,
product reachability order, and product regions. UNCONDITIONAL: no
interface hypotheses (contrast the `dcond` family); the analytic
content is the witness transport above. The E2 lattice premises and the
`flows_only` fence's readout-transport warrant (Kalies–Wehbe) live at
the certificate stratum and are deliberately not consumed — see the
module header and the book page's fence remark. -/
theorem qMorseProduct_sound
  (h1 : MorseSat ϕ1 D1 ρ1) (h2 : MorseSat ϕ2 D2 ρ2) :
  MorseSat (ϕ1.prod ϕ2) (D1 ×s D2)
  (fun p : Vtx1 × Vtx2 => ρ1 p.1 ×s ρ2 p.2)
```

BernsteinLogic.QtslProduct . MorseWitness.prod

statement bound

```

/-- Witness transport (lem:qmp-transport): the product family
`M(v, w) := M1(v) × M2(w)`, indexed by the product order, witnesses the
product judgment. Six clauses, each the book paragraph:

* compactness – a product of compacts is compact;
* invariance – coordinate-wise forward invariance, and a product point
rides the pairing of the factor witnesses' complete orbits;
* location – the product of the factor locations;
* disjointness – distinct product vertices differ in some coordinate,
and that factor's disjointness kills the intersection;
* cover – a chain-recurrent product point has chain-recurrent
coordinates (§4), each covered by its factor family;
* order – a product orbit contained in no `M(v, w)` has an uncontained
coordinate; three cases (both uncontained / first only / second
only), the contained coordinate handled by compact-hence-closed and
the closure bound (§3), the  $\alpha$ -side vertex strictly higher in the
product order exactly as the source-to-sink convention demands. -/
def MorseWitness.prod (W1 : MorseWitness  $\phi_1$  D1  $\rho_1$ )
  (W2 : MorseWitness  $\phi_2$  D2  $\rho_2$ ) :
  MorseWitness ( $\phi_1$ .prod  $\phi_2$ ) (D1 ×s D2)
  (fun p : Vtx1 × Vtx2 =>  $\rho_1$  p.1 ×s  $\rho_2$  p.2)

```

BernsteinLogic.QtslProduct.**completeOrbit\_prod\_iff**

statement bound

```

/-- Orbits factor (lem:qmp-factor(i)): `Y` is a complete orbit of the
product iff each coordinate is a complete orbit of its factor. -/
theorem completeOrbit_prod_iff {Y :  $\mathbb{R} \rightarrow X_1 \times X_2$ } :
  ( $\phi_1$ .prod  $\phi_2$ ).CompleteOrbit Y  $\leftrightarrow$ 
   $\phi_1$ .CompleteOrbit (fun s => (Y s).1)  $\wedge$ 
   $\phi_2$ .CompleteOrbit (fun s => (Y s).2)

```

BernsteinLogic.QtslProduct.**omegaSet\_prod\_subset**

statement bound

```

/--  $\omega$ -limit sets factor (lem:qmp-factor(ii)):
` $\omega(Y) \subseteq \omega(Y_1) \times \omega(Y_2)$ `. Book proof verbatim: the image of `[T,  $\infty$ )`
under `Y` sits inside the product of the coordinate images; closure of
a product is the product of closures. -/
theorem omegaSet_prod_subset (Y :  $\mathbb{R} \rightarrow X_1 \times X_2$ ) :
  omegaSet Y  $\subseteq$ 
  (omegaSet fun s => (Y s).1) ×s (omegaSet fun s => (Y s).2)

```

BernsteinLogic.QtslProduct.**alphaSet\_prod\_subset**

statement bound

```

/--  $\alpha$ -limit sets factor (lem:qmp-factor(ii), the `likewise for  $\alpha$ `). -/
theorem alphaSet_prod_subset (Y :  $\mathbb{R} \rightarrow X_1 \times X_2$ ) :
  alphaSet Y  $\subseteq$ 
  (alphaSet fun s => (Y s).1) ×s (alphaSet fun s => (Y s).2)

```

BernsteinLogic.QtslProduct.**EpsChainIn.fst**

statement bound

```

/-- Chains project, first coordinate (lem:qmp-factor(iii)): under the
max metric (mathlib's product metric, `Prod.dist_eq`) factor distance
is  $\leq$  product distance, so a chain step projects to a chain step – jump
points project into `D1`, durations unchanged – and concatenations
project to concatenations. -/
theorem EpsChainIn.fst {D1 : Set X1} {D2 : Set X2} { $\epsilon$  :  $\mathbb{R}$ } {T :  $\mathbb{R}_{\geq 0}$ }
  {x y : X1 × X2}
  (h : EpsChainIn ( $\phi_1$ .prod  $\phi_2$ ) (D1 ×s D2)  $\epsilon$  T x y) :
  EpsChainIn  $\phi_1$  D1  $\epsilon$  T x.1 y.1

```

BernsteinLogic.QtslProduct.**EpsChainIn.snd**

statement bound

```

/-- Chains project, second coordinate. -/
theorem EpsChainIn.snd {D1 : Set X1} {D2 : Set X2} { $\epsilon$  :  $\mathbb{R}$ } {T :  $\mathbb{R}_{\geq 0}$ }
  {x y : X1 × X2}
  (h : EpsChainIn ( $\phi_1$ .prod  $\phi_2$ ) (D1 ×s D2)  $\epsilon$  T x y) :
  EpsChainIn  $\phi_2$  D2  $\epsilon$  T x.2 y.2

```



## BernsteinLogic.QtslProduct . ChainRecurrentIn.fst

statement bound

```

/-- Chain recurrence projects, first coordinate (lem:qmp-factor(iii),
"hence a point chain-recurrent in `D1 × D2` has both coordinates
chain-recurrent in their arenas"). -/
theorem ChainRecurrentIn.fst {D1 : Set X1} {D2 : Set X2} {x : X1 × X2}
  (h : ChainRecurrentIn (φ1.prod φ2) (D1 ×s D2) x) :
  ChainRecurrentIn φ1 D1 x.1

```

## BernsteinLogic.QtslProduct . ChainRecurrentIn.snd

statement bound

```

/-- Chain recurrence projects, second coordinate. -/
theorem ChainRecurrentIn.snd {D1 : Set X1} {D2 : Set X2} {x : X1 × X2}
  (h : ChainRecurrentIn (φ1.prod φ2) (D1 ×s D2) x) :
  ChainRecurrentIn φ2 D2 x.2

```

## BernsteinLogic.QtslProduct . CompleteOrbit.prodMk

statement bound

```

/-- Pairing complete orbits (lem:qmp-factor(i), ⇐): factor orbits ride
together as a product orbit. -/
theorem CompleteOrbit.prodMk {γ1 : ℝ → X1} {γ2 : ℝ → X2}
  (h1 : φ1.CompleteOrbit γ1) (h2 : φ2.CompleteOrbit γ2) :
  (φ1.prod φ2).CompleteOrbit fun s => (γ1 s, γ2 s)

```

## BernsteinLogic.QtslProduct . omegaSet\_subset\_closure\_range

statement bound

```

/-- `ω(γ) ⊆ cl(range γ)` — one intersect and one image bound; feeds the
order clause's contained cases (book:
`α(γ2), ω(γ2) ⊆ cl(range γ2) ⊆ M2(w0)`). -/
theorem omegaSet_subset_closure_range (γ : ℝ → X) :
  omegaSet γ ⊆ closure (Set.range γ)

```

## BernsteinLogic.QtslProduct . alphaSet\_subset\_closure\_range

statement bound

```

/-- `α(γ) ⊆ cl(range γ)` — the α twin. -/
theorem alphaSet_subset_closure_range (γ : ℝ → X) :
  alphaSet γ ⊆ closure (Set.range γ)

```

## Correspondence (Steele display ↔ Lean)

## NO CORRESPONDENCE DICTIONARY YET

no reviewer-certified dictionary maps the displayed rule onto the mechanized statements above; until one is written and certified, that mapping rests on this section's editorial care alone.

## Lean verifier receipts

module BernsteinLogic.QtslProduct; bound build 1b85f883dfdc... (bound from receipt d5b482c9d6ed9b6c, 2026-07-20T03:30:32Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| d5b482c9d6ed9b6c | 2026-07-20T03:30:32Z | build+decorrelated                 | current |
| 15fa0566d44e84fd | 2026-07-20T20:44:52Z | build+decorrelated                 | current |
| e8214fce26d91980 | 2026-07-20T21:01:21Z | build+replay+hardened+decorrelated | current |
| b87a989a1475042c | 2026-07-21T05:28:48Z | build+replay+hardened+decorrelated | current |

## gate-layer coverage of the bound build:

build (lake build (elaboration)) current  
 replay (leanchecker (official kernel replay)) current  
 hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

Absent at the judgment stratum: the HOL4 development's object is the base-kernel checker cone, and no Morse-family definition exists in the ported scripts (verified against the port's definition inventory) — so the satisfaction-judgment semantics this rule's soundness is stated in has no HOL twin to mirror the now-complete Lean mechanization into. A future port has its interface fixed twice over: the witness clauses quoted in the QMORSEDISCHARGE section and the product lemma chain bound here.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail, mechanized *in full*: the accompanying development proves the product construction, the factoring lemmas, the witness transport, and the soundness theorem with no interface hypotheses (contrast the QMORSEDISCHARGE ledger; within the Morse family only QBASIN shares this fully-proved status). The lemma chain below fixed the mechanization's interface in advance, and the formal-anchor table maps each step to the declaration that carries it. The readout-transport theorem of the fence remark is deliberately *not* mechanized: it warrants the canonicity of the certificate-route composite readout, not the witness-level entailment proved here, and remains at literature strength.

**Setting and claim.** Let  $\varphi_1, \varphi_2$  be forward semiflows on Hausdorff metric state spaces  $X_1, X_2$ , with compact forward-invariant arenas  $D_1, D_2$ . The *product semiflow*  $\varphi_1 \times \varphi_2$  on  $X_1 \times X_2$  (with the max metric) flows coordinate-wise:  $\Phi_t(x_1, x_2) = (\Phi_t^1 x_1, \Phi_t^2 x_2)$ ; the semiflow laws hold coordinate-wise. The rule: from  $\mathcal{M}_1 \models \text{Morse}(D_1, G_1, \rho_1)$  and  $\mathcal{M}_2 \models \text{Morse}(D_2, G_2, \rho_2)$ , with disjoint variable sets, conclude

$$\mathcal{M}_1 \times \mathcal{M}_2 \models \text{Morse}(D_1 \times D_2, G_1 \times G_2, \rho_1 \times \rho_2),$$

where  $G_1 \times G_2$  presents the product order  $((v_1, v_2) \geq (w_1, w_2) \text{ iff } v_i \geq w_i \text{ in both coordinates})$  and  $(\rho_1 \times \rho_2)(v_1, v_2) := \rho_1(v_1) \times \rho_2(v_2)$ .

**Lemma R71.1** (orbits, limit sets, and chains factor). (i)  $\gamma = (\gamma_1, \gamma_2)$  is a complete orbit of  $\varphi_1 \times \varphi_2$  iff each  $\gamma_i$  is a complete orbit of  $\varphi_i$ . (ii)  $\omega(\gamma) \subseteq \omega(\gamma_1) \times \omega(\gamma_2)$ , and likewise for  $\alpha$ . (iii) the projection of an  $(\varepsilon, T)$ -chain inside  $D_1 \times D_2$  is an  $(\varepsilon, T)$ -chain inside  $D_i$ ; hence a point chain-recurrent in  $D_1 \times D_2$  has both coordinates chain-recurrent in their arenas.

*Proof.* (i) is the coordinate-wise flow law read in both directions. (ii):  $\gamma([T, \infty)) \subseteq \gamma_1([T, \infty)) \times \gamma_2([T, \infty))$ ; closure of a product is the product of closures; and an intersection of product sets is the product of the intersections — pointwise,  $(x_1, x_2)$  lies in every  $A_T \times B_T$  iff each  $x_i$  lies in every factor set. (iii): under the max metric, factor distance is  $\leq$  product distance, so a single chain step projects to a chain step (jump points project into  $D_i$ , durations are unchanged), and concatenations project to concatenations.  $\square$

**Lemma R71.2** (witness transport). Let  $\{M_1(v)\}_{v \in V(G_1)}$  and  $\{M_2(w)\}_{w \in V(G_2)}$  witness the factor judgments. Then the product family

$$M(v, w) := M_1(v) \times M_2(w),$$

indexed by the product order, witnesses the product judgment.

*Proof.* Six clauses.

**Compactness:** a product of two compact sets is compact.

**Invariance:** forward invariance holds coordinate-wise; and a point  $(x_1, x_2)$  of the product rides the pairing of the factor witnesses' complete orbits through  $x_1$  and  $x_2$  (Lemma R71.1(i)), which stays inside the product set.

**Location:**  $M_1(v) \times M_2(w) \subseteq \rho_1(v) \times \rho_2(w)$ , the product of the factor locations.

**Disjointness:** if  $(v, w) \neq (v', w')$  they differ in some coordinate, say  $v \neq v'$ ; then  $M_1(v) \cap M_1(v') = \emptyset$  makes the products disjoint regardless of the other coordinate.

**Cover:** let  $(x_1, x_2)$  be chain-recurrent in  $D_1 \times D_2$ . By Lemma R71.1(iii) each  $x_i$  is chain-recurrent in  $D_i$ , hence covered by its factor family:  $x_1 \in M_1(v)$ ,  $x_2 \in M_2(w)$  for some  $v, w$ ; so  $(x_1, x_2) \in M(v, w)$ .

**Order:** let  $\gamma = (\gamma_1, \gamma_2)$  be a complete orbit in  $D_1 \times D_2$  contained in no single  $M(v, w)$ . Containment in a product set is coordinate-wise, so — negating “some  $(v, w)$  contains  $\gamma$ ” — either  $\gamma_1$  is contained in no  $M_1(v)$ , or  $\gamma_2$  in no  $M_2(w)$ . Three cases. (both uncontained): the factor order clauses give  $v > v'$  with  $\alpha(\gamma_1) \subseteq M_1(v)$ ,  $\omega(\gamma_1) \subseteq M_1(v')$ , and  $w > w'$  likewise for  $\gamma_2$ ; by Lemma R71.1(ii),  $\alpha(\gamma) \subseteq M(v, w)$  and  $\omega(\gamma) \subseteq M(v', w')$ , with  $(v, w) > (v', w')$  strictly in the product order. ( $\gamma_1$  uncontained,  $\gamma_2 \subseteq M_2(w_0)$ ): the first factor gives  $v > v'$  as above; for the second, range  $\gamma_2 \subseteq M_2(w_0)$  with  $M_2(w_0)$  compact in a Hausdorff space, hence closed, gives  $\alpha(\gamma_2), \omega(\gamma_2) \subseteq \text{cl}(\text{range } \gamma_2) \subseteq M_2(w_0)$ ; so  $\alpha(\gamma) \subseteq M(v, w_0)$ ,  $\omega(\gamma) \subseteq M(v', w_0)$ , and  $(v, w_0) > (v', w_0)$  strictly. (the symmetric case): exchange the roles of the factors. In each case the  $\alpha$ -side vertex is the strictly higher one, as the source-to-sink convention demands.  $\square$



**Theorem R71.3** (soundness of QMORSEPRODUCT, witness level). *If both factor judgments hold, the product judgment holds.*

*Proof.* Choose factor witnesses and apply Lemma R71.2. □

*Remark R71.4* (where the flow hypothesis actually binds — the rule’s fence). Nothing in Lemmas R71.1–R71.2 consumes continuity of time: read over  $\mathbb{N}$ -time systems, the same six clauses transport verbatim, at the same “at most” polarity. The rule is nonetheless fenced to the deterministic-*flow* class, for reasons that live above the witness stratum. The certificate-level discharge route does not consume semantic witnesses: it composes the factor *lattices* (the join-closure of the products of factor blocks, every composite facet’s inwardness polynomial being a factor’s with the other factor’s coordinates riding along as free box variables) and must know that the composite lattice’s recomputed readout — its join-irreducibles, cells, and cell order — is the product of the factor readouts, so that the composite tessellation represents the product dynamics at the resolution the factors certified. For flows this is a theorem: the attractor lattice of a product flow is the coproduct of the factor attractor lattices (Kalies–Wehbe [133]), whose Birkhoff dual [22] is exactly “composite Morse graph = product of factor graphs.” For maps it is false: products of maps exhibit emergent, phase-locked attractors outside the coproduct (Kalies–Wehbe’s counterexample — already visible in the product of two period-two orbits, whose diagonal and anti-diagonal are attractors of the product generated by no product of factor attractors), so a composite readout computed from factor certificates would silently misrepresent the product’s global structure, and the judgment’s ecosystem — which reads a certified Morse graph as the canonical decomposition at the declared resolution — would be poisoned at exactly the canonicity it relies on. Hence the fence: for maps the rule must not be derived at all; for flows the readout-transport theorem is the rule’s semantic import, and it is the reason this core rule carries its own lemma rather than reducing to lattice combinatorics.

*Remark R71.5* (what rides along for free). Two certificate-level transports cost nothing beyond citation, and are worth recording with the rule even though the witness-level proof above does not touch them. *Facet rows*: every internal facet of a composite block is normal to one factor’s axis, and its inwardness polynomial *is* that factor’s facet polynomial — the other factor’s coordinates enter as free box variables, so the composite row’s enclosure data is the factor’s, replicated along the free coordinates. *Transit rows*: if  $\mathcal{A}_1 V \leq -\varepsilon$  on a factor cell, the same  $V$  read on the product satisfies  $\mathcal{A}_{1 \times 2} V = \mathcal{A}_1 V \leq -\varepsilon$  on cell  $\times$  anything — the other factor’s drift differentiates  $V$  to zero. Both ride the same disjointness of the variable sets — the facet rows by weakening alone, the transit rows by weakening plus the product-generator identity  $\mathcal{A}_{1 \times 2}(V \circ \pi_i) = (\mathcal{A}_i V) \circ \pi_i$ , which is the precise content of “differentiates  $V$  to zero.”

*Remark R71.6* (the mechanization consumes no arena hypotheses). The setting above posits compact forward-invariant arenas because that is what arenas are in this catalogue’s ecosystem; the transport itself never uses either property, and the mechanized statements quantify over bare sets  $D_1, D_2$  — strictly more general, and faithful to the witness clauses, which constrain the arena only through the judgments. Hausdorffness is consumed exactly once, where the proof of Lemma R71.2 says “compact in a Hausdorff space, hence closed”; the mechanization takes it as a topological hypothesis over a pseudometric base, which every metric state space satisfies.

*Remark R71.7* (relation to the literature). The rule is the certified face of compositionality results for global dynamics: Kalies–Wehbe [133] prove the attractor-lattice coproduct isomorphism for product flows and delimit it (maps fail; skew products degrade to one-way results), inside the lattice-theoretic program of Kalies–Mischaikow–van der Vorst [131, 132]. The witness-level transport (Lemma R71.2) is elementary Conley theory [47] over products; the observation that its “at most” clauses survive even where the canonical readout does not (Remark R71.4) is exactly why the judgment’s fence is stated at the rule, not at the witness.

| proof step                               | formal anchor                                                                                                                                        |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| product semiflow (Setting)               | Semiflow.prod, prod_flow (QtslProduct.lean); the max metric is the product metric, Prod.dist_eq                                                      |
| Lemma R71.1(i)                           | completeOrbit_prod_iff, CompleteOrbit.prodMk                                                                                                         |
| Lemma R71.1(ii)                          | omegaSet_prod_subset, alphaSet_prod_subset                                                                                                           |
| Lemma R71.1(iii)                         | EpsChainIn.fst/.snd, ChainRecurrentIn.fst/.snd                                                                                                       |
| Lemma R71.2                              | MorseWitness.prod (all six clauses; the contained cases of the order clause via alphaSet_subset_closure_range, omegaSet_subset_closure_range)        |
| Theorem R71.3                            | qMorseProduct_sound — unconditional, no interface hypotheses                                                                                         |
| witness clauses (the interface targeted) | MorseWitness, MorseSat (QtslSoundness.lean, bound in the QMORSEDISCHARGE section)                                                                    |
| Remark R71.4 (readout transport, flows)  | <i>deliberately not mechanized</i> : Kalies–Wehbe [133], attractor-lattice coproduct isomorphism; Birkhoff duality [22]                              |
| interim discharge path                   | QMORSEDISCHARGE on the composite lattice (that section’s theorem chain) — no longer needed as warrant; retained as the certificate-route alternative |

## Rule 72. QBLOCKINV (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QBlockInv*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{c} N : \text{QRegionAtom (the block)} \\ \{F_f \vdash \mathcal{A}\ell_f \geq \varepsilon_f\}_{f \in \text{facets}(N)}, \quad \varepsilon_f \in \mathbb{Q}_{>0} \text{ } (\ell_f \text{ the facet's inward affine coordinate; } F_f \text{ its facet region}) \\ \text{FacetBridge}(\mathcal{M}, D, N, (F_f, \varepsilon_f)_f) \text{ (the bridge package, defined below)} \quad \vdash X_0 \subseteq N \text{ (init\_containment, a single BL leaf)} \end{array}}{\mathcal{M} \models \text{QInvScope}(N, X_0)}$$

each facet-inward row bounds the flow derivative of the facet's inward coordinate ( $\ell_f$ : affine, vanishing on the facet hyperplane, nonnegative on the block's side) below by a strictly positive rational margin, on the facet region  $F_f$ . The displayed **FacetBridge** package comprises: the block realized closed in its arena; each facet region a neighbourhood of its entire facet box, corners included (tangential margin, not merely normal thickening); the per-facet wedge test (one-directional by design); arena invariance ( $N \subseteq D$ ,  $D$  invariant); trajectory continuity and the pointwise-to-integrated row bridge — checker-recomputed or interface-carried per profile (model-class).

**QBLOCKINV** concludes the invariance-scope judgment: the block  $N$  is forward-invariant and encloses the declared initial region. It is the Morse family's workhorse producer — the judgment it mints is consumed by **QBASIN**'s safety factory, by scope-restriction of reach obligations, and as the arena warrant of nested certificates — and its premise kit is minimal: per internal facet of the box-union block, one strictly-inward sign row with explicit positive rational margin (facets on the arena boundary owe nothing), plus one containment row  $X_0 \subseteq N$ .

What the checker recomputes: the block's grid geometry — that the bitmap realizes a closed box-union inside its arena, and the facet-list well-formedness — plus the replay of every row through the base kernel. What would break without each hypothesis: without the sign rows, exit through a facet; without strictness-by-margin, a strict field sign could not be certified through the non-strict enclosure predicates of exact- $\mathbb{Q}$  arithmetic (and the Morse rules that cite the same rows would lose the arguments where strictness is genuinely load-bearing); without the containment row, a start outside the block; without closedness of the realization, the first-exit argument's sup-time step fails.

The mechanization is conditional at a named interface: the bound theorems descend the invariance obligation through a first-exit skeleton (closed block, continuous trajectories, boundary dwell) and a facet-row bridge (integrated growth of the facet observables through a one-directional wedge test), leaving exactly three hypothesis families — the integrated row clause (reachable from the pointwise rows through a mean-value bridge whose surviving content is the profile-level identification of the certified rate with the trajectory-wise derivative), the grid facts of the box-union, and arena invariance (nested citation or model-class warrant, terminating at the whole space). The proof section states the ledger, and why the wedge test must be one-directional (reentrant corners of box-unions).

**notes** The same facet rows serve three consumers — this rule, the lattice membership of the family object, and the Morse discharge's facet arguments — which is the design reason strict margins are mandatory here even though this rule's own conclusion would survive non-strict signs (the proof section's strictness remark).

### Lean statement

BernsteinLogic.QtslSoundness.qBlockInv\_sound

statement bound

```
-- QBlockInv soundness, the M(i) corollary (roster: "2 lines"). The
-- facet rows deliver `hInv` through Lemma M(i) — strict facet-inwardness
-- of the compact box-union block implies forward invariance (gap G1, tier
-- A prose); `init_containment` delivers `hInit`. The rule-level content is
-- the assembly of the `QInvScope` claim. -/
theorem qBlockInv_sound {φ : Semiflow X} {N X0 : Set X}
  (hInv : φ.Invariant N) (hInit : X0 ⊆ N) :
  φ.InvScopeSat N X0
```

BernsteinLogic.QtslDischarge.qBlockInv\_sound\_of\_dwell

statement bound

```
-- QBlockInv soundness with G1 at its residue: the interface fact
-- drops from "the block is invariant" to "boundary points dwell"
-- (G1b — the facet-row bridge). -/
```

```

theorem qBlockInv_sound_of_dwell {φ : Semiflow X} {N X0 : Set X}
  (hN : IsClosed N)
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hdwell : ∀ x ∈ N, x ∈ frontier N →
    ∃ δ : ℝ≥0, 0 < δ ∧ ∀ s ≤ δ, φ.flow s x ∈ N)
  (hInit : X0 ⊆ N) : φ.InvScopeSat N X0

```

BernsteinLogic.QtslDischarge.invariant\_of\_boundaryDwell

statement bound

-- G1's tier-A "first-exit argument", mechanized. A closed region with continuous trajectories is forward-invariant as soon as every BOUNDARY point dwells (stays in the region for some positive time) – interior points dwell by continuity alone. The residue G1b (certified strict facet-inward sign rows  $\Rightarrow$  the boundary dwell) is the polynomial-field bridge, prose tier A, same class as G2.

Proof shape = the note's first-exit: the set of times whose whole prefix stays in `N` is nonempty, closed under the sup (closedness + continuity), and extends past its sup by the dwell at the sup point – so it is unbounded, i.e. all of  $\mathbb{R}_{\geq 0}$ . -/

```

theorem invariant_of_boundaryDwell {φ : Semiflow X} {N : Set X}
  (hN : IsClosed N)
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hdwell : ∀ x ∈ N, x ∈ frontier N →
    ∃ δ : ℝ≥0, 0 < δ ∧ ∀ s ≤ δ, φ.flow s x ∈ N) :
  φ.Invariant N

```

BernsteinLogic.QtslDischarge.HasFacetRows.boundaryDwell

statement bound

-- G1b mechanized at its honest stratum: facet rows + block-in-arena + a forward-invariant arena  $\Rightarrow$  the boundary dwell that `invariant\_of\_boundaryDwell` consumes. Proof shape = the entry half of the interior-entry argument ("interior entry is the same sign fact read forward"): at a boundary point, trajectory continuity confines a short prefix to the wedge neighborhood and every active certified region at once (finitely many); on that prefix each active observable grows from  $\emptyset$  at rate  $\varepsilon_i$  – nonnegative – arena invariance supplies the  $\mathcal{D}$  conjunct, and the wedge test re-admits every prefix point. Inactive facets never appear: the wedge test quotes only active ones. -/

```

theorem HasFacetRows.boundaryDwell {φ : Semiflow X} {D N : Set X}
  (h : φ.HasFacetRows D N) (hND : N ⊆ D) (hD : φ.Invariant D)
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x) :
  ∀ x ∈ N, x ∈ frontier N →
    ∃ δ : ℝ≥0, 0 < δ ∧ ∀ s ≤ δ, φ.flow s x ∈ N

```

BernsteinLogic.QtslDischarge.qBlockInv\_sound\_of\_facetRows

statement bound

-- QBlockInv soundness from the certificate's facet rows – the composed G1 corollary at the current floor. Standing interface facts, each at its named honest stratum (module header, G1 entry): `hN` and `hND` are grid facts of the block (compact box-union in its arena), `hD` is the model-class arena fact (`invariant\_univ` when the block has no  $\partial D$ -faces), `hrows` carries the G2-class integrated row residue and the R1-class wedge geometry, `htraj` is flow continuity (G3's model-class clause). -/

```

theorem qBlockInv_sound_of_facetRows {φ : Semiflow X} {D N X0 : Set X}
  (hN : IsClosed N)
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hND : N ⊆ D) (hD : φ.Invariant D) (hrows : φ.HasFacetRows D N)
  (hInit : X0 ⊆ N) : φ.InvScopeSat N X0

```

BernsteinLogic.QtslMeanValue.qBlockInv\_sound\_of\_rows

statement bound

-- QBlockInv soundness from pointwise rows – G1's chain at its new floor: per-facet row bounds + profile + wedge geometry + arena invariance + model-class regularity  $\Rightarrow$  `InvScopeSat`. Composes

```

`hasFacetRows_of_rows` into `qBlockInv_sound_of_facetRows`; every
hypothesis is now either R1-recomputable grid geometry, a model-class
fact, exact row arithmetic, or the profile identification. -/
theorem qBlockInv_sound_of_rows {φ : Semiflow X} {D N X0 : Set X}
  {n : ℕ} {ℓ : Fin n → X → ℝ} {ε : Fin n → ℝ} {F : Fin n → Set X}
  (W : Fin n → X → ℝ)
  (hN : IsClosed N)
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hND : N ⊆ D) (hD : φ.Invariant D)
  (hε : ∀ i, 0 < ε i)
  (hℓ : ∀ i, ContinuousOn (ℓ i) (F i))
  (hW : ∀ i, φ.FlowDerivOn (ℓ i) (W i) (F i))
  (hrow : ∀ i, ∀ y ∈ F i, ε i ≤ W i y)
  (hlocal : ∀ x ∈ frontier N, ∃ (A : Set (Fin n)) (U : Set X), U ∈ ℳ × Λ
    (∀ i ∈ A, ℓ i x = 0 ∧ F i ∈ ℳ x) ∧
    ∀ y ∈ U, y ∈ D → (∀ i ∈ A, 0 ≤ ℓ i y) → y ∈ N)
  (hInit : X0 ⊆ N) : φ.InvScopeSat N X0

```

BernsteinLogic.QtslGridGeometry.qBlockInv\_sound\_of\_grid

statement bound

```

/-- QBlockInv soundness on the grid model, end to end: grid geometry
discharged outright; the remaining hypotheses are exactly the named
residues – the arena invariance interface `hArena` (residue (c), see
the two reduction theorems below) and the integrated row clause
(residue (a), G2's class) – plus flow continuity (G3's model-class
clause).-/
theorem qBlockInv_sound_of_grid (φ : Semiflow (Fin d → ℝ)) (B : Finset G.CellIdx)
  {X0 : Set (Fin d → ℝ)}
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hArena : φ.Invariant G.arena)
  (ε : G.FacetData → ℝ) (F : G.FacetData → Set (Fin d → ℝ))
  (hε : ∀ p, G.IsFacet B p → 0 < ε p)
  (hF : ∀ p, G.IsFacet B p → ∀ x ∈ G.facetBox p, F p ∈ ℳ x)
  (hrow : ∀ p, G.IsFacet B p → ∀ x ∈ F p, ∀ t : ℝ≥0,
    (∀ s ≤ t, φ.flow s x ∈ F p) →
    G.obs p x + ε p * (t : ℝ) ≤ G.obs p (φ.flow t x))
  (hInit : X0 ⊆ G.blockRegion B) :
  φ.InvScopeSat (G.blockRegion B) X0

```

BernsteinLogic.QtslGridGeometry.qBlockInv\_sound\_of\_pointwiseRows

statement bound

```

/-- CAPSTONE of the same-day gap-fill pair (this file × the interval
mean-value bridge, `QtslMeanValue.lean`): grid block + POINTWISE row
bounds, consumed through the profile predicate `FlowDerivOn`, ⇒
`InvScopeSat`. Every remaining hypothesis is a model-class regularity
fact (`htraj`, `hV`), exact row arithmetic (`hε`, `hrow`), the arena
interface (`hArena`, residue (c)'s reductions), the certified-region
neighborhood fact (`hF`), or the profile identification (`hW` –
`W p = ℳ(obs p)` for the ODE doctrine, the sole analytic remainder of
the whole G1 chain). -/
theorem qBlockInv_sound_of_pointwiseRows (φ : Semiflow (Fin d → ℝ))
  (B : Finset G.CellIdx) {X0 : Set (Fin d → ℝ)}
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hArena : φ.Invariant G.arena)
  (ε : G.FacetData → ℝ) (F : G.FacetData → Set (Fin d → ℝ))
  (W : G.FacetData → (Fin d → ℝ) → ℝ)
  (hε : ∀ p, G.IsFacet B p → 0 < ε p)
  (hF : ∀ p, G.IsFacet B p → ∀ x ∈ G.facetBox p, F p ∈ ℳ x)
  (hV : ∀ p, G.IsFacet B p → ContinuousOn (G.obs p) (F p))
  (hW : ∀ p, G.IsFacet B p → φ.FlowDerivOn (G.obs p) (W p) (F p))
  (hrow : ∀ p, G.IsFacet B p → ∀ y ∈ F p, ε p ≤ W p y)
  (hInit : X0 ⊆ G.blockRegion B) :
  φ.InvScopeSat (G.blockRegion B) X0

```

## Correspondence (Steele display ↔ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed judgment  $\mathcal{M} \models \text{QInvScope}(N, X_0) \mapsto$  the goal  $\text{InvScopeSat } N \ X_0 = \emptyset. \text{Invariant } N \wedge X_0 \subseteq N$  — the claim record's denotation verbatim. The five bound statements form one discharge chain: `qBlockInv_sound` (the roster's “two lines”: invariance + containment  $\Rightarrow$  the claim,  $M(i)$  as hypothesis), `_of_dwell` riding `invariant_of_boundaryDwell` (G1 mechanized: closed block + trajectory continuity + boundary dwell  $\Rightarrow$  invariance, the first-exit sup-time argument), and `_of_facetRows` riding `HasFacetRows.boundaryDwell`, the deepest bound form, whose kit is the honest ledger:

- the block  $N : \text{QRegionAtom} \mapsto$  an abstract  $N : \text{Set } X$  with  $hN : \text{IsClosed } N$ : the grid stratum (bitmap realizes a closed box-union in its arena; facet-list well-formedness) is checker-side recompute, and closedness is the one grid fact the bound statements consume — as an explicit hypothesis, not a derived fact.
- facet-inward sign rows  $\mapsto$  `hrows :  $\emptyset. \text{HasFacetRows } D \ N$` : finitely many facet observables  $\ell_i$  with explicit positive margins ( $\forall i, 0 < \varepsilon_i$  — the display's strictness-by-margin, carried exactly), each growing at *integrated* rate  $\geq \varepsilon_i$  along confined prefixes — the pointwise row  $\mathcal{A} \ell \geq \varepsilon$  reaches that form through the G2-class mean-value bridge, outside this bound set — plus the local wedge test, one-directional by design (reentrant corners of box-unions defeat the  $\leftrightarrow$  form), with “ $\partial D$ -faces owe nothing” landing as the wedge test's  $y \in D$  conjunct.
- `init_containment`  $\mapsto$  `hInit :  $X_0 \subseteq N$` , one-to-one, unchanged at every link of the chain.
- the displayed `FacetBridge` package (its contents defined in the flavor text)  $\mapsto$  `htraj` (trajectory continuity — model-class), and the arena pair `hND :  $N \subseteq D + hD : \emptyset. \text{Invariant } D$`  — the certified statement makes the invariance regress explicit (the roster witness names its termination at `invariant_univ`, which is in the module but not this bound set).

Width note: the conditional flag's three named families are visible verbatim in the facet-row statement's binders — the integrated row clause (inside `HasFacetRows`), the grid facts (`hN`, plus the wedge-test geometry carried *inside* `hrows` rather than discharged), and arena invariance (`hD`). The bound set extends past the two-line core (the two blocks below): the row-to-shadow discharge `qBlockInv_sound_of_rows` (module `QtslMeanValue` — pointwise rows reach the integrated shadow form through the interval mean-value bridge, leaving the profile identification `hW` as the sole analytic remainder of the G1 chain), the grid-model discharge `qBlockInv_sound_of_grid` (module `QtslGridGeometry` — grid geometry discharged outright, residues exactly the arena interface and the integrated row clause), and the capstone `qBlockInv_sound_of_pointwiseRows` (grid block + pointwise rows through `FlowDerivOn` to the judgment) are hash-bound, so the conditional residue shrinks to the profile identification plus the arena-invariance interface (its reduction theorems terminate at `invariant_univ`), the facet-neighbourhood interface clause (each certified region  $F_i$  a neighbourhood of every point of its facet box, corners included — the capstone consumes it as an explicit hypothesis) and model-class regularity. One deliberate width inversion: the display's conclusion tie is an unconditional judgment about the model, while `qBlockInv_sound` alone is wider than the rule (any invariant set qualifies); the facet-row form narrows back to certificate-shaped input. No hypothesis in the bound set demands the block be a box-union — the geometry enters only through what the wedge test can actually be discharged for.

## Lean verifier receipts

module `BernsteinLogic.QtslSoundness`; bound build 77005991d61d... (bound from receipt 8130a7a679cef4, 2026-08-04T01:16:06Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 8130a7a679cef4   | 2026-08-04T01:16:06Z | build+decorrelated | current |
| 9eb3706354082144 | 2026-08-04T02:23:12Z | build+decorrelated | current |

5 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module `BernsteinLogic.QtslDischarge`; bound build 4aa46910a600... (bound from receipt dd2946805f790f99, 2026-08-04T01:16:09Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| dd2946805f790f99 | 2026-08-04T01:16:09Z | build+decorrelated | current |
| e1bc68890e2aebf1 | 2026-08-04T02:23:11Z | build+decorrelated | current |

14 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.QtslMeanValue; bound build ed04b606934b... (bound from receipt c2f57d21d0b1c270, 2026-07-20T08:27:08Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 160082fd7894faf1 | 2026-07-19T07:39:26Z | build+decorrelated                 | current |
| 260eab5039951515 | 2026-07-19T09:07:22Z | build+decorrelated                 | current |
| c2f57d21d0b1c270 | 2026-07-20T08:27:08Z | build+decorrelated                 | current |
| f75d120f9128f80c | 2026-07-20T20:44:49Z | build+decorrelated                 | current |
| 891ea7ab9c4f79ff | 2026-07-20T21:01:17Z | build+replay+hardened+decorrelated | current |
| af025f8810d8a6cc | 2026-07-21T05:28:44Z | build+replay+hardened+decorrelated | current |

module BernsteinLogic.QtslGridGeometry; bound build 9060a661920e... (bound from receipt 64d6a21041b4042f, 2026-07-20T08:27:10Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| bf51898b80b37884 | 2026-07-19T12:35:27Z | build+decorrelated                 | current |
| 64d6a21041b4042f | 2026-07-20T08:27:10Z | build+decorrelated                 | current |
| c7deda94ed05f8e5 | 2026-07-20T20:44:45Z | build+decorrelated                 | current |
| 6bab7a61aa41db61 | 2026-07-20T21:01:14Z | build+replay+hardened+decorrelated | current |
| 5e72e577b5f436e5 | 2026-07-21T05:28:39Z | build+replay+hardened+decorrelated | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

#### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

theory bern\_tr\_t5qdyn, HOL4 development script bern\_tr\_t5qdynScript.sml script bound

#### Definition sfInvariant\_def

```
Definition sfInvariant_def:
  sfInvariant (fl:real->'x->'x) (sv:'x->bool) ⇔
    ∀x. x IN sv ⇒ ∀t. 0 ≤ t ⇒ fl t x IN sv
End
```

#### Definition sfInvScopeSat\_def

```
Definition sfInvScopeSat_def:
  sfInvScopeSat (fl:real->'x->'x) (sv:'x->bool) (X0:'x->bool) ⇔
    sfInvariant fl sv ∧ X0 SUBSET sv
End
```

#### Theorem sfQBlockInv\_sound

```
Theorem sfQBlockInv_sound:
  ∀fl (N:'x->bool) X0.
    sfInvariant fl N ∧ X0 SUBSET N ⇒ sfInvScopeSat fl N X0
```



## HOL verifier receipts

**documentary** hol-strict CLEAN: theory bern\_tr\_t5qdyn, 22 theorems, 0 cheat-tagged, 0 nonstandard axioms (whitelist: DISK\_THM; the four bool axioms) — verification run of 2026-07-22 on an isolated build host, against a content-addressed checkout of the exact sources cited here (git blob d64e1954...); an independent acceptance host ran the same build green at the porting seating (2026-07-21). What is ported is the semiflow-stratum root: the flow law, the invariance and invariance-scope-satisfaction predicates, and the root soundness theorem concluding the scope judgment from invariance plus containment — the exact root the Lean development's satisfaction stratum supplies for this rule. The facet-row bridge (the first-exit skeleton and integrated-growth descent that discharge invariance from the certificate's sign rows) and the grid-geometry recomputation remain Lean-side; the conditional reading of this section's mechanization carries over to the HOL twin unchanged. The facet-sign and containment rows the rule consumes are covered by the ported base checker as before.

HOL receipts are documentary: quoted results of hol-strict runs and census rows from the HOL4 development's build evidence, not minted receipt artefacts. A HOL receipt store (the Lean store's twin) is named future work.

## Proof

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail. The rule's mechanization is *conditional*: the section descends the invariance obligation from an opaque interface fact to three named hypothesis families — integrated facet-row clauses, grid geometry of the block, arena invariance — and the ledger remark at the end states exactly what remains and why. Every hypothesis is used and remarked on; the formal-anchor table maps each step to the Lean declaration that carries it.

**Setting and claim.** Fix a forward semiflow  $\varphi$  on a state space  $X$  ( $\Phi_0 = \text{id}$ ,  $\Phi_{s+t} = \Phi_t \circ \Phi_s$ ; the QBASIN section's setting, which also fixes the QInvScope denotation:  $S$  forward-invariant and  $X_0 \subseteq S$ ). The rule's data: a *block*  $N$  — the realization of a box-union over the certificate's declared rational grid, a compact closed set — sitting in an *arena*  $D \supseteq N$ ; per internal facet of  $N$ , one strictly-inward sign row of the base calculus (a BERNCONe-family sequent asserting the field's normal component points into the block, with an explicit positive rational margin; facets on  $\partial D$  owe nothing); and one containment row  $X_0 \subseteq N$ . The claim: these data warrant  $\text{QInvScope}(N, X_0)$  — established below at the discharged floor, conditional on the named ledger of [Remark R72.9](#).

**Lemma R72.1** (rule-level assembly). *If  $N$  is forward-invariant and  $X_0 \subseteq N$ , then  $\text{QInvScope}(N, X_0)$  holds.*

*Proof.* The judgment is that conjunction. The lemma is stated to fix where the rule-level content ends: everything below descends the invariance conjunct toward the certificate's rows.  $\square$

**Lemma R72.2** (the first-exit skeleton: closed + continuous + boundary dwell  $\Rightarrow$  invariant). *Suppose  $N$  is closed, every trajectory  $t \mapsto \Phi_t x$  is continuous, and every boundary point dwells: for each  $x \in N \cap \partial N$  there is  $\delta > 0$  with  $\Phi_s x \in N$  for all  $0 \leq s \leq \delta$ . Then  $N$  is forward-invariant.*

*Proof.* First, every point of  $N$  dwells: an interior point  $x$  has  $\Phi_0 x = x \in \text{int } N$ , so continuity of the trajectory at time 0 confines a short prefix to  $\text{int } N$  — the dwell hypothesis is needed only at the frontier, which is where the certificate's rows live.

Now fix  $x \in N$  and  $t \geq 0$ , and suppose toward a contradiction that  $\Phi_t x \notin N$ . Consider the set of *prefix-confined times*

$$S := \{u \geq 0 : \Phi_s x \in N \text{ for all } s \leq u\}.$$

$S$  contains 0, and every element of  $S$  is  $\leq t$  (an element  $> t$  would confine time  $t$  itself); throughout, times  $s, u$  range over  $[0, \infty)$ . Let  $T := \sup S \leq t$ . Every time  $s < T$  is confined (it is below some element of  $S$ ). The sup itself is confined: for  $T > 0$  the preimage of  $N$  under the continuous trajectory is closed and contains  $[0, T)$ , hence contains  $T \in \text{cl } [0, T)$ ; for  $T = 0$  it is the hypothesis  $x \in N$ . So  $T \in S$ . But now the dwell at the point  $\Phi_T x \in N$  supplies  $\delta > 0$  with  $\Phi_s(\Phi_T x) \in N$  for  $0 \leq s \leq \delta$ , i.e. (semiflow law)  $\Phi_u x \in N$  for  $T \leq u \leq T + \delta$ ; combined with  $T \in S$  this puts  $T + \delta \in S$ , contradicting  $T = \sup S$  with  $\delta > 0$ .  $\square$

**Remark R72.3** (each skeleton hypothesis is load-bearing). *Closedness*: on  $X = \mathbb{R}$  with  $\Phi_t x = x + t$  and the half-open  $N = [0, 1)$ , every point of  $N$  dwells (interior points by continuity; the one frontier point of  $N$ , namely 0, dwells since  $\Phi_s 0 = s \in [0, 1)$  for small  $s$  — the other frontier point 1 is not in  $N$ ), yet the trajectory from 0 exits: the proof fails exactly at “the sup itself is confined”, which is where closedness is consumed. *Trajectory continuity*: with a discontinuous time-dependence the preimage argument dies and a trajectory may jump across the frontier without ever presenting a boundary point to the dwell hypothesis; in this stratum continuity is a model-class fact (polynomial-drift flows), not a theorem. *Dwell at the frontier*: dropping it re-opens exit through the boundary — it is precisely the obligation the facet rows discharge (next lemma).

**Lemma R72.4** (the facet-row bridge: certified rows  $\Rightarrow$  boundary dwell). *Suppose the certificate supplies, for block  $N$  in arena  $D$ : finitely many facet observables  $\ell_1, \dots, \ell_n : X \rightarrow \mathbb{R}$  (the affine facet polynomials), margins  $\varepsilon_i > 0$ , and certified regions  $F_i$  (boxes around the facets, thickened to full dimension with tangential margin: each  $F_i$  must be a neighbourhood of every point of its facet, corners included — a box thickened only in the normal direction is a neighbourhood of no corner point, and the wedge test below demands one), such that*

(row, integrated form) *along any trajectory segment confined to  $F_i$ ,  $\ell_i$  grows at rate at least  $\varepsilon_i$ : if  $\Phi_s x \in F_i$  for all  $s \leq t$ , then  $\ell_i(\Phi_t x) \geq \ell_i(x) + \varepsilon_i t$ ; and*

(local wedge test) *every frontier point  $x$  of  $N$  has an active set  $A \subseteq \{1, \dots, n\}$  — each active observable vanishing at  $x$ , each active  $F_i$  a neighbourhood of  $x$  — and a neighbourhood  $U$  of  $x$  such that every  $y \in U$  with  $y \in D$  and  $\ell_i(y) \geq 0$  for all  $i \in A$  lies in  $N$ .*

*If moreover  $N \subseteq D$ ,  $D$  is forward-invariant, and trajectories are continuous, then every boundary point of  $N$  dwells.*



*Proof.* Fix  $x \in N \cap \partial N$  with active set  $A$  and wedge neighbourhood  $U$ . The set  $U \cap \bigcap_{i \in A} F_i$  is a finite intersection of neighbourhoods of  $x$ , hence a neighbourhood; continuity of the trajectory at time 0 (where it sits at  $x$ ) confines a prefix  $[0, \delta]$  to it, some  $\delta > 0$ . Now take any  $0 \leq s \leq \delta$  and check the wedge test’s three conditions at  $y = \Phi_s x$ : membership in  $U$  is the confinement; membership in  $D$  is arena invariance applied to  $x \in N \subseteq D$ ; and for each active  $i$ , the whole prefix up to  $s$  is confined to  $F_i$ , so the integrated row gives  $\ell_i(\Phi_s x) \geq \ell_i(x) + \varepsilon_i s = 0 + \varepsilon_i s \geq 0$ , using exactly the activation fact  $\ell_i(x) = 0$ . The wedge test re-admits  $\Phi_s x \in N$ . Inactive facets never appear: the test quotes only active ones.  $\square$

**Theorem R72.5** (soundness of QBLOCKINV, at the discharged floor). *Suppose:  $N$  is closed with  $N \subseteq D$ ;  $D$  is forward-invariant; trajectories are continuous; the certificate’s facet rows satisfy Lemma R72.4’s two clauses; and  $X_0 \subseteq N$ . Then  $\text{QInvScope}(N, X_0)$  holds.*

*Proof.* Lemma R72.4 delivers the boundary dwell; Lemma R72.2 converts it to forward invariance of  $N$ ; Lemma R72.1 assembles the judgment with the containment row.  $\square$

**Remark R72.6** (the wedge test is one-directional by design). At a *reentrant* corner of a box-union block — an inner corner of an L-shape — the block is locally the *union* of two half-spaces  $\{\ell_1 \geq 0\} \cup \{\ell_2 \geq 0\}$ , while the wedge test’s hypothesis set is their *intersection*: the implication “both  $\ell_i \geq 0 \Rightarrow y \in N$ ” holds (the intersection sits inside the union), but its converse is false there — points of  $N$  with one active observable negative exist on the other branch. The biconditional form of the test would therefore be unsatisfiable at reentrant corners, while the one-directional form is exactly what the first-exit argument consumes (Lemma R72.4 never needs to *characterize* the frontier, only to re-admit prefix points). A certificate whose rows are jointly unsatisfiable at a boundary point — checkerboard blocks touching at a corner and demanding opposite strict signs there — simply makes the bridge’s hypotheses unfulfillable: soundness only consumes what is given.

**Remark R72.7** ( $\partial D$ -faces owe nothing; the arena regress terminates). Facets of  $N$  on the arena boundary carry no rows: in Lemma R72.4 they enter only through the conjunct  $y \in D$ , which arena invariance supplies. The arena’s own invariance is either a nested QInvScope citation (a chart sub-block certified by its own facet rows — this same rule, one level up) or the model-class fact that a chart’s denotation confines trajectories to its domain; the regress terminates at the whole space, which is invariant vacuously.

**Remark R72.8** (strictness of the margins: where it is and is not consumed). The invariance conclusion of this section survives  $\varepsilon_i = 0$ : the integrated clause then says  $\ell_i$  is non-decreasing along confined segments, and growth-from-zero still yields  $\ell_i \geq 0$  — consistent with the classical invariance theorem (Nagumo [187]), which needs only subtangentiality. The explicit positive rational margin is nonetheless mandatory in the certificate convention, for two reasons that live outside this section. First, exactness: a *strict* field sign cannot be certified directly by the non-strict enclosure predicates of the base calculus; “non-strict row on  $\mathcal{A}\ell - \varepsilon$  with explicit  $\varepsilon > 0$ ” is how strictness becomes checkable in exact  $\mathbb{Q}$ . Second, the same rows are consumed by the Morse-family rules, where strictness is genuinely load-bearing — no orbit rides a facet, distinct fibres’ invariant parts are disjoint, and the facet margin bounds admissible chain jumps (“strictness pays a second time”; see the QMORSEDISCHARGE section’s ledger). A margin quietly dropped to zero would leave this rule’s conclusion intact and break those.

**Remark R72.9** (the hypothesis ledger — what remains, named). The chain above discharges the rule’s invariance obligation down to three hypothesis families, and the rule’s Lean column renders with the hypothesis-flavoured check because of exactly these:

(a) *the integrated row clause: discharged, and bound.* The certificate’s rows are pointwise: per facet box, a base-calculus sequent bounding the field’s certified rate,  $\mathcal{A}\ell_i \geq \varepsilon_i$  (the generator  $\mathcal{A} = f \cdot \nabla$  of the deterministic model class). Lemma R72.4 consumes the *integrated* clause along trajectory segments. The bridge between them is one-variable calculus — a mean-value argument along the trajectory, mechanized and bound in this edition (anchor table) — and its surviving analytic content is the identification of the certified polynomial rate with the trajectory-wise derivative on the region: a model-class profile warrant (for smooth flows the identification is literal; switched and hybrid profiles replace it by one-sided derivative bounds, which is their own seam, deliberately not this section’s).

(b) *grid geometry of the block: closed for grid blocks, and bound — except one interface clause.* Closedness of  $N$ ,  $N \subseteq D$ , the vanishing of each facet observable on its facet box, and the wedge re-admission implication are finite geometry of the box-union over its declared rational grid — recomputable combinatorics, proved outright for grid blocks in the grid-geometry module and bound in this edition (anchor table), end-to-end through the capstone that takes grid block plus pointwise row bounds to the concluded judgment. The one geometric clause that stays interface-level even there: each certified region  $F_i$  must be a neighbourhood of every point of its facet box, corners included — the tangential-margin requirement of Lemma R72.4 — which the capstone consumes as a hypothesis on the certificate’s regions rather than proving from the boxes. The interface-level theorems above remain stated with (b) as hypotheses, which is what the general (non-grid) reading consumes.

(c) *arena invariance.* As in the regress remark: nested citation, or model-class warrant, terminating at the whole space.

None of the three is an opaque dynamical assumption; each names a specific fact with a specific owner — (a) the profile stratum, (b) the checker’s recompute (and the grid module), (c) the arena’s own certificate. That is the sense in which this core rule is *conditionally* mechanized: the cited theorems are sorry-free, and their hypotheses are the ledger above.

**Remark R72.10** (relation to the literature). The block-with-inward-facets certificate is the exact-arithmetic descendant of Conley theory’s isolating blocks [48]: compact sets whose boundary behaviour is transverse, so that dynamical claims reduce to boundary sign checks. The invariance argument itself is Nagumo’s theorem [187] in first-exit form, specialized to polyhedral sets, where the subtangentiality condition becomes finitely many polynomial sign rows — the form set-invariance control has long exploited [23]. What the certificate stratum adds is the discipline: strict signs via non-strict predicates with exact rational margins (Remark R72.8), and a one-directional wedge test chosen so that box-union (nonconvex) blocks are admissible without any frontier characterization (Remark R72.6).

| proof step                             | formal anchor                                                                                                                                          |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lemma R72.1                            | qBlockInv_sound (QtslSoundness.lean)                                                                                                                   |
| Lemma R72.2, first-exit skeleton       | invariant_of_boundaryDwell (QtslDischarge.lean); composed as qBlockInv_sound_of_dwell                                                                  |
| Lemma R72.4, row shadow + wedge test   | HasFacetRows (the definition); HasFacetRows.boundaryDwell                                                                                              |
| Theorem R72.5, end-to-end              | qBlockInv_sound_of_facetRows                                                                                                                           |
| arena regress terminus                 | invariant_univ; nested-citation path invariant_of_invScopeSat (development)                                                                            |
| ledger item (a), the mean-value bridge | integrated_ge_of_flowDeriv_ge, composed as qBlockInv_sound_of_rows (QtslMeanValue; <b>bound in this edition</b> )                                      |
| ledger item (b), grid facts            | Grid.wedge_containment, Grid.qBlockInv_sound_of_grid; capstone Grid.qBlockInv_sound_of_pointwiseRows (QtslGridGeometry; <b>bound in this edition</b> ) |
| facet rows                             | strictly-inward BERNCONe-family sequents with explicit rational margin (base kernel)                                                                   |

## Rule 73. QLATTICEINV (alias for QBLOCKINV)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QLatticeInv*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{L : \text{QBlockLattice} \quad C \text{ (downset, a cell-set; membership and down-closedness recomputed)} \quad \mathcal{M} \models \text{QInvScope}(\bigcup_{i \in C} \text{blocks}(\text{emb } i), X_0) \text{ (the cited QBLOCKINV conclusion)} \quad \vdash X_0 \subseteq \bigcup \downarrow C \text{ (init\_containment)}}{\mathcal{M} \models \text{QInvScope}(\bigcup \downarrow C, X_0)}$$

QLATTICEINV is the family object's invariance alias: any down-closed cell-set  $C$  of the recomputed readout names an invariant region  $\bigcup \downarrow C$ , and the rule concludes its invariance-scope judgment by citation — the unpacking is a single QBLOCKINV application whose block is the lattice member realizing the down-set (existence by join-closure) and whose facet rows are that member's own, already present in the family object. No row is re-proven; the producer merely selects which down-set encloses the initial region away from the avoid sets.

What the checker recomputes: membership of the realization in the lattice, and down-closedness of  $C$  in the recomputed cell order — readout facts, not dynamics. What would break without them: a cell-set that is not down-closed names a region orbits can leave (an orbit descends the cell order out of it), and the exactness identity between the down-set's cell union and the member's realization fails — already on a two-member chain: the top member alone is not down-closed, and its cell omits everything the bottom member contributes.

Mechanization state, plainly: the bound statements below are the exactness identity and the soundness corollaries, at both semantic strata (module BernsteinLogic.QtslLatticeInv). *qLatticeInv\_exact* proves the identity that makes the respelling exact — for down-closed  $C$ , the union of  $C$ 's cells equals the union of its join-irreducible realizations — by the discharge chain's own well-founded descent through lower covers (*exists\_cellMem\_subset*, over the same descent rows the eventual-member interface consumes — rows that assume no distributivity, plus one row this alias adds: cell inside member, the readout closure half's set-level shadow). One sharp point, because the page's classical route consumes distributivity: the mechanized route consumes none, and the row pack does not encode join-primality — absorption-positive families satisfy every row, and the bound (union-form) identity holds in them regardless — because the descent runs inside a single witnessed member of  $C$  and never forms the join. The join-form reading (the union as the member  $N_C$ 's realization) costs exactly join-realizes-as-union, the checker's recomputed membership of the down-set's realization, strictly weaker than distributivity; that recompute is also what refuses the diamond-lattice witness when its top is realized past the atoms' union. The corollaries *Semiflow.qLatticeInv\_sound* / *Behavior.qLatticeInv\_sound* (and their *\_of\_blockInv* end-to-end forms) are  $\text{sound}_A = \text{sound}_T \circ \text{unpack}$  through the cited QBLOCKINV conclusion, delivered as judgment-level cargo. Unconditional: the lattice rows are the checker's recomputed readout facts entering as theorem parameters (R1 shadows), and join-closure membership of the down-set's realization is deliberately *not* consumed — with down-closedness consumed and the cited member judgment supplied, the entailment holds for the bare cell union, so the rule is sound a fortiori with the checker's extra side condition. The payload-level unpack elaborator is bound beside the corollaries (as with the sibling extension rules; module BernsteinLogic.QtslElaborators): *LatticeInvPayload* reifies the wire payload — the leg-7 row pack, the down-set, the containment row — *LatticeInvPayload.unpack* is the verified transport onto the QBLOCKINV payload (block := the down-set's realization; the containment row rewritten along the exactness identity; facets a citation, never payload data), and the *qLatticeInv\_sound\_elab* pair is  $\text{sound}_A = \text{sound}_T \circ \text{unpack}$  with *qBlockInv\_sound* as the literal  $\text{sound}_T$ , at both semantic strata. The tier is flagged, not forced: if registry-backed citation of the member's rows is ruled heavier than the alias criterion allows, the rule reclassifies as admissible with all machinery unchanged.

**notes** The warrant for the reading — once an orbit enters a lattice member it never leaves, so visited cells descend the cell order and down-sets are forward-invariant — is the same orbit combinatorics the Morse discharge's order clause rests on, consumed here at its easy end.

### Lean statement

BernsteinLogic.QtslLatticeInv . *qLatticeInv\_exact*

statement bound

```
-- **prop:qli-exact, the exactness identity.** For a down-closed cell-set
`C`, the union of the cells of `C` equals the union of the realizations of
`C`'s join-irreducible members — the realization of the display's member
`N_C = v C` (join = union in the union-closed family object). The  $\supseteq$  half
is leg 7's well-founded descent through lower covers
(`Semiflow.exists_cellMem_subset`): a member point lands in the cell of a
join-irreducible below it, and down-closedness (through `horder`) pins
that cell inside `C`. The  $\subseteq$  half is `hcellSub`, cell inside member. This
is what makes the alias's respelling exact rather than merely ` $\supseteq$ '. -/
```

```

theorem qLatticeInv_exact
  (hdc : IsLowerSet C)
  (hJoin :  $\forall b : \beta, \forall x \in \text{blocks } b,$ 
     $\exists i : \iota, x \in \text{blocks } (\text{emb } i) \wedge \text{blocks } (\text{emb } i) \subseteq \text{blocks } b$ )
  (hcover :  $\forall i : \iota, \text{blocks } (\text{cover } i) \subset \text{blocks } (\text{emb } i)$ )
  (hcellsSup :  $\forall i : \iota, \text{blocks } (\text{emb } i) \setminus \text{blocks } (\text{cover } i) \subseteq \text{cells } i$ )
  (horder :  $\forall i j : \iota, \text{blocks } (\text{emb } i) \subseteq \text{blocks } (\text{emb } j) \rightarrow i \leq j$ )
  (hcellSub :  $\forall i : \iota, \text{cells } i \subseteq \text{blocks } (\text{emb } i)$ ) :
  ( $\bigcup i \in C, \text{cells } i = \bigcup i \in C, \text{blocks } (\text{emb } i)$ )

```

BernsteinLogic.QtslLatticeInv.member\_subset\_cells\_le

statement bound

```

/-- ***cells tile members*** (the page proof's first readout fact), at the
principal down-set: every point of a join-irreducible's realization lies
in the cell of some join-irreducible at-or-below it in the cell order.
The identity's  $\supseteq$  half, exposed as its own statement because the book page
names it. -/
theorem member_subset_cells_le
  (hJoin :  $\forall b : \beta, \forall x \in \text{blocks } b,$ 
     $\exists i : \iota, x \in \text{blocks } (\text{emb } i) \wedge \text{blocks } (\text{emb } i) \subseteq \text{blocks } b$ )
  (hcover :  $\forall i : \iota, \text{blocks } (\text{cover } i) \subset \text{blocks } (\text{emb } i)$ )
  (hcellsSup :  $\forall i : \iota, \text{blocks } (\text{emb } i) \setminus \text{blocks } (\text{cover } i) \subseteq \text{cells } i$ )
  (horder :  $\forall i j : \iota, \text{blocks } (\text{emb } i) \subseteq \text{blocks } (\text{emb } j) \rightarrow i \leq j$ )
  (i :  $\iota$ ) :
   $\text{blocks } (\text{emb } i) \subseteq \bigcup j \in \text{Set.Iic } i, \text{cells } j$ 

```

BernsteinLogic.QtslLatticeInv.Semiflow.qLatticeInv\_sound

statement bound

```

/-- **prop:qli-sound, through the target** – the alias soundness corollary
in its genre-native shape `sound_A = sound_T ◦ unpack`: the premise is the
cited `QBlockInv` node's conclusion on the member `N_C` (its realization:
the union of the down-set's join-irreducible realizations), delivered as
judgment-level cargo (the `qBasin_sound` precedent – the facet rows live
in the cited node, not here); the conclusion is the alias's displayed
judgment on  $\bigcup C$ . The bridge is `qLatticeInv_exact`, and there is no
other content – the alias genre working as intended. -/
theorem Semiflow.qLatticeInv_sound { $\phi$  : Semiflow X} {X0 : Set X}
  (hdc : IsLowerSet C)
  (hJoin :  $\forall b : \beta, \forall x \in \text{blocks } b,$ 
     $\exists i : \iota, x \in \text{blocks } (\text{emb } i) \wedge \text{blocks } (\text{emb } i) \subseteq \text{blocks } b$ )
  (hcover :  $\forall i : \iota, \text{blocks } (\text{cover } i) \subset \text{blocks } (\text{emb } i)$ )
  (hcellsSup :  $\forall i : \iota, \text{blocks } (\text{emb } i) \setminus \text{blocks } (\text{cover } i) \subseteq \text{cells } i$ )
  (horder :  $\forall i j : \iota, \text{blocks } (\text{emb } i) \subseteq \text{blocks } (\text{emb } j) \rightarrow i \leq j$ )
  (hcellSub :  $\forall i : \iota, \text{cells } i \subseteq \text{blocks } (\text{emb } i)$ )
  (hMember :  $\phi.\text{InvScopeSat } (\bigcup i \in C, \text{blocks } (\text{emb } i)) \text{ X0}$ ) :
   $\phi.\text{InvScopeSat } (\bigcup i \in C, \text{cells } i) \text{ X0}$ 

```

BernsteinLogic.QtslLatticeInv.Semiflow.qLatticeInv\_sound\_of\_blockInv

statement bound

```

/-- **rec:qli-filled, composed end-to-end**: the filled-in `QBlockInv`
application. Premises: forward invariance of the member's realization
(the cited facet-row chain's output – dischargeable per member via
`qBlockInv_sound_of_facetRows` and kin) and the rule's own containment
row, stated on the alias's displayed region and transported verbatim
across the identity. -/
theorem Semiflow.qLatticeInv_sound_of_blockInv { $\phi$  : Semiflow X} {X0 : Set X}
  (hdc : IsLowerSet C)
  (hJoin :  $\forall b : \beta, \forall x \in \text{blocks } b,$ 
     $\exists i : \iota, x \in \text{blocks } (\text{emb } i) \wedge \text{blocks } (\text{emb } i) \subseteq \text{blocks } b$ )
  (hcover :  $\forall i : \iota, \text{blocks } (\text{cover } i) \subset \text{blocks } (\text{emb } i)$ )
  (hcellsSup :  $\forall i : \iota, \text{blocks } (\text{emb } i) \setminus \text{blocks } (\text{cover } i) \subseteq \text{cells } i$ )
  (horder :  $\forall i j : \iota, \text{blocks } (\text{emb } i) \subseteq \text{blocks } (\text{emb } j) \rightarrow i \leq j$ )
  (hcellSub :  $\forall i : \iota, \text{cells } i \subseteq \text{blocks } (\text{emb } i)$ )
  (hInv :  $\phi.\text{Invariant } (\bigcup i \in C, \text{blocks } (\text{emb } i))$ )
  (hInit :  $X0 \subseteq \bigcup i \in C, \text{cells } i$ ) :
   $\phi.\text{InvScopeSat } (\bigcup i \in C, \text{cells } i) \text{ X0}$ 

```

BernsteinLogic.QtslLatticeInv . Behavior.qLatticeInv\_sound

statement bound

```

/-- `Semiflow.qLatticeInv_sound` at the behavioral floor – the identity is
dynamics-free, so the alias transports over solution-trajectory-set
semantics verbatim (the two-directions carve: `QInvScope` is
safety-directional, sound here without existence hypotheses). -/
theorem Behavior.qLatticeInv_sound {B : Behavior X} {X0 : Set X}
  (hdc : IsLowerSet C)
  (hJoin : ∀ b : β, ∀ x ∈ blocks b,
    ∃ i : ι, x ∈ blocks (emb i) ∧ blocks (emb i) ⊆ blocks b)
  (hcover : ∀ i : ι, blocks (cover i) ⊆ blocks (emb i))
  (hcellsSup : ∀ i : ι, blocks (emb i) \ blocks (cover i) ⊆ cells i)
  (horder : ∀ i j : ι, blocks (emb i) ⊆ blocks (emb j) → i ≤ j)
  (hcellSub : ∀ i : ι, cells i ⊆ blocks (emb i))
  (hMember : B.InvScopeSat (⋃ i ∈ C, blocks (emb i)) X0) :
  B.InvScopeSat (⋃ i ∈ C, cells i) X0

```

BernsteinLogic.QtslLatticeInv . Behavior.qLatticeInv\_sound\_of\_blockInv

statement bound

```

/-- `Semiflow.qLatticeInv_sound_of_blockInv` at the behavioral floor. -/
theorem Behavior.qLatticeInv_sound_of_blockInv {B : Behavior X} {X0 : Set X}
  (hdc : IsLowerSet C)
  (hJoin : ∀ b : β, ∀ x ∈ blocks b,
    ∃ i : ι, x ∈ blocks (emb i) ∧ blocks (emb i) ⊆ blocks b)
  (hcover : ∀ i : ι, blocks (cover i) ⊆ blocks (emb i))
  (hcellsSup : ∀ i : ι, blocks (emb i) \ blocks (cover i) ⊆ cells i)
  (horder : ∀ i j : ι, blocks (emb i) ⊆ blocks (emb j) → i ≤ j)
  (hcellSub : ∀ i : ι, cells i ⊆ blocks (emb i))
  (hInv : B.Invariant (⋃ i ∈ C, blocks (emb i)))
  (hInit : X0 ⊆ ⋃ i ∈ C, cells i) :
  B.InvScopeSat (⋃ i ∈ C, cells i) X0

```

**The payload-level unpack, displayed whole.** Recipe [R73.3](#) as a function: the wire payload reified field-for-field against the register, the containment row transported along the exactness identity, the target's soundness consumed literally in the factoring pair.

BernsteinLogic.QtslElaborators . LatticeInvPayload

statement bound

```

/-- **The QLatticeInv wire payload, Lean face** – fields mirror the
geolog register (section J's adjudication: "registers already staged;
the elaborator's domains are complete"): the family object with its
recomputed readout (`QLatticeInv/lattice`, as the leg-7 R1-shadow row
pack – no lattice structure minted, the leg-7 discipline), the
down-set (`QLatticeInv/downset`, with down-closedness the checker's
recomputed side condition), and the rule's own containment row
(`QLatticeInv/init_containment`). Membership of the down-set's
realization in the lattice is deliberately NOT a field – checker-side,
not consumed (soundness holds for the bare union, a fortiori with the
extra condition). -/
structure LatticeInvPayload (X ι β : Type*) [Finite ι] [PartialOrder ι] where
  /-- Member realizations (the family object's blocks). -/
  blocks : β → Set X
  /-- Join-irreducible embedding over the cell index. -/
  emb : ι → β
  /-- Lower covers. -/
  cover : ι → β
  /-- Cells `T(b) = cl(b \ b⁻)` , as realized sets. -/
  cells : ι → Set X
  /-- The down-set `C` (wire field `downset`). -/
  downset : Set ι
  /-- The initial region `X₀`. -/
  initRegion : Set X
  /-- Down-closedness in the recomputed cell order (R1 recompute). -/
  hdc : IsLowerSet downset
  /-- Birkhoff join-decomposition, realization form (R1 shadow). -/
  hJoin : ∀ b : β, ∀ x ∈ blocks b,
    ∃ i : ι, x ∈ blocks (emb i) ∧ blocks (emb i) ⊆ blocks b

```

```

/-- Strict lower covers (R1 shadow). -/
hcover :  $\forall i : \iota, \text{blocks } (\text{cover } i) \subset \text{blocks } (\text{emb } i)$ 
/-- Cell  $\supseteq$  member-minus-cover (R1 shadow). -/
hcellsSup :  $\forall i : \iota, \text{blocks } (\text{emb } i) \setminus \text{blocks } (\text{cover } i) \subseteq \text{cells } i$ 
/-- Order reflects realization inclusion (R1 shadow). -/
horder :  $\forall i j : \iota, \text{blocks } (\text{emb } i) \subseteq \text{blocks } (\text{emb } j) \rightarrow i \leq j$ 
/-- Cell  $\subseteq$  its member's realization (the row this alias adds). -/
hcellSub :  $\forall i : \iota, \text{cells } i \subseteq \text{blocks } (\text{emb } i)$ 
/-- The rule's own containment row (wire field `init_containment`),
stated on the alias's displayed region. -/
hInit :  $\text{initRegion} \subseteq \bigcup i \in \text{downset}, \text{cells } i$ 

```

BernsteinLogic.QtslElaborators . **BlockInvPayload**

statement bound

```

/-- **The QBlockInv payload, Lean face** (`rec:qli-transport`'s
codomain): the block region and the containment row. The facet rows are
a CITATION (judgement-level cargo, the `qBasin_sound` precedent) – they
enter the factoring theorems as the cited node's invariance, never as
payload data. -/
structure BlockInvPayload (X : Type*) where
  /- The block `N_C` – here its realization, the bare union (the
  join realizes as the union in the union-closed family object; that
  the union is a lattice MEMBER is the checker's recomputed side
  condition, deliberately not consumed). -/
  block : Set X
  /- The initial region `X_0`, transported verbatim. -/
  initRegion : Set X
  /- `init_containment`, rewritten along `qLatticeInv_exact`. -/
  hInit :  $\text{initRegion} \subseteq \text{block}$ 

```

BernsteinLogic.QtslElaborators . **Semiflow.qLatticeInv\_sound\_elab**

statement bound

```

/-- **sound_A = sound_T  $\circ$  unpack, semiflow stratum** – the target's
soundness lemma (`qBlockInv_sound`, literally sound_T) applied to the
UNPACKED payload, concluding the alias's displayed judgement. The cited
member's facet-row chain enters as `hInv` (judgement-level cargo). -/
theorem Semiflow.qLatticeInv_sound_elab { $\phi$  : Semiflow X}
  (p : LatticeInvPayload X  $\iota$   $\beta$ ) (hInv :  $\phi$ .Invariant p.unpack.block) :
   $\phi$ .InvScopeSat ( $\bigcup i \in p.\text{downset}, p.\text{cells } i$ ) p.initRegion

```

BernsteinLogic.QtslElaborators . **Behavior.qLatticeInv\_sound\_elab**

statement bound

```

/-- **sound_A = sound_T  $\circ$  unpack, behavioural floor** – the identity is
dynamics-free, so the unpack transports at every semantics for free. -/
theorem Behavior.qLatticeInv_sound_elab {B : Behavior X}
  (p : LatticeInvPayload X  $\iota$   $\beta$ ) (hInv : B.Invariant p.unpack.block) :
  B.InvScopeSat ( $\bigcup i \in p.\text{downset}, p.\text{cells } i$ ) p.initRegion

```

BernsteinLogic.QtslElaborators . **LatticeInvPayload.unpack**

definition bound

```

/-- **The verified unpack** – `rec:qli-filled` as a function, the first
of the five alias artefacts (`unpack : Step_A  $\rightarrow$  Step_T`): block :=
the down-set's realization `N_C`, facets cited (not payload),
`init_containment` transported along the exactness identity
`qLatticeInv_exact` ( $\bigcup i \in \text{N\_C}$  as regions – the alias's conclusion is
the target's on the nose). -/
def LatticeInvPayload.unpack (p : LatticeInvPayload X  $\iota$   $\beta$ ) : BlockInvPayload X where
  block :=  $\bigcup i \in p.\text{downset}, p.\text{blocks } (p.\text{emb } i)$ 
  initRegion := p.initRegion
  hInit := by
    rw [← qLatticeInv_exact p.hdc p.hJoin p.hcover p.hcellsSup p.horder p.hcellSub]
    exact p.hInit

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed judgment  $\mathcal{M} \models \text{QInvScope}(\bigcup \downarrow C, X_0) \mapsto \text{InvScopeSat}(\bigcup i \in C, \text{cells } i) X_0$ , at BOTH semantic strata (Semiflow./Behavior.qLatticeInv\_sound — the exactness identity is dynamics-free, so the behavioural transport is verbatim). The alias's entire content is qLatticeInv\_exact: for down-closed  $C$  ( $\text{hdc} : \text{IsLowerSet } C$  — the R1 down-closedness recompute as a binder), the union of  $C$ 's cells EQUALS the union of its join-irreducible realizations — the member  $N_C$ 's realization exactly where the checker's membership recompute holds (join realizes as union; the bound statement is on the bare union and consumes no distributivity) — so the respelling is exact, not merely one-sided.

- the E2 lattice  $L \mapsto$  the Birkhoff row pack, abstract:  $\text{hJoin}$  (every block point lands in a join-irreducible below it),  $\text{hcover}$  (strict lower covers),  $\text{hcellsSup}$  ( $\text{cell} \supseteq \text{member}$  minus cover),  $\text{horder}$  (order reflects realization inclusion) — byte-identical to the Morse discharge's eventual-member interface: no lattice structure is minted, the rows ARE the interface — plus the one row new to this alias,  $\text{hcellSub}$  ( $\text{cell} \subseteq \text{member}$ : the readout closure half's set-level shadow).
- the cited QBlockInv premise  $\mapsto \text{hMember} : \text{InvScopeSat}(\bigcup i \in C, \text{blocks } (\text{emb } i)) X_0$  — judgment-level cargo on the member's realization (the qBasin\_sound precedent: facet rows live in the cited node, never here);  $\text{\_of\_blockInv}$  composes the target in end-to-end (its  $\text{hInv/hInit}$  pair is qBlockInv\_sound's kit, the containment row transported across the identity).
- $\text{init\_containment} \mapsto \text{hInit} : X_0 \subseteq \bigcup i \in C, \text{cells } i$ , stated on the alias's displayed region.
- $\text{member\_subset\_cells\_le}$  exposes the page's “cells tile members” fact at the principal down-set — bound because the proof section names it.

Width note: the module is topology-free and dynamics-free except through the cited judgment —  $\text{cells/blocks}$  arbitrary set families over an abstract ordered index; the grid realization of the lattice is R1 territory, strictly wider in the safe direction. Module row bound to an attested build (the receipt provenance is recorded in the bindings).

## Lean verifier receipts

module BernsteinLogic.QtslLatticeInv; bound build 467dc57e1c30... (bound from receipt 98755f4d8dbefb10, 2026-07-21T04:48:40Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 98755f4d8dbefb10 | 2026-07-21T04:48:40Z | build+decorrelated                 | current |
| 210205ac3ea031f4 | 2026-07-21T05:17:21Z | build+decorrelated                 | current |
| 9a870584fbb58e05 | 2026-07-21T05:28:41Z | build+replay+hardened+decorrelated | current |

3 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogic.QtslElaborators; bound build 0205ec213b91... (hashed from the built olean at bind time). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a237f4cff27cb15a | 2026-07-22T10:06:47Z | build+decorrelated | current |

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement



**NO HOL STATEMENT YET**

Absent at the judgment stratum: no judgment-stratum definition exists in the ported base-kernel checker scripts (verified against the port's definition inventory), so the satisfaction-level witnesses — now mechanized on the Lean side (module `QtslLatticeInv`, the entry's bound statements) — have no HOL twin to mirror them. The target rule's rows (facet signs, containment) ARE covered by the ported checker.

**HOL verifier receipts****NONE**

no HOL verification receipt exists for this rule.

**Proof**

Genre: alias rule (tier 3) — a respelling template. An alias owes the book five artefacts: the verified  $1 \rightarrow 1$  unpack function, the filled-in application of the target rule, the payload-transport table, the exact algebraic identity that makes the respelling correct, and the soundness corollary through the target rule's lemma. All five are present — the filled-in application, the transport table, the exactness identity and the soundness corollary mechanized and bound in the entry (`qLatticeInv_exact`, Proposition [R73.4](#); the `qLatticeInv_sound` family, Proposition [R73.6](#), at the semiflow *and* behavioural strata), and — now that the extension certificate schema exists — the verified payload-level unpack function itself, `LatticeInvPayload.unpack` with its factoring pair (the remark below names the bound functions). (Tier flag, carried honestly: the unpack cites facet rows through the checker's recomputed lattice readout — if registry-backed citation is ruled heavier than the alias criterion allows, the rule reclassifies as admissible with all machinery unchanged.)

*Remark R73.1* (the payload-level elaborator, mechanized). The *semantic* content of this page is bound in the entry: `qLatticeInv_exact` is Proposition [R73.4](#) (over the discharge chain's own Birkhoff rows — no lattice structure is minted), and the `qLatticeInv_sound` family is Proposition [R73.6](#) in both its through-the-target and end-to-end forms, at both semantic strata. The payload-level elaborator — previously this page's one absent artefact, held back until the extension certificate schema existed — is bound with it, as with the sibling extension rules: `LatticeInvPayload` (module `BernsteinLogic.QtslElaborators`) reifies the rule's wire payload — family object as the leg-7 row pack, down-set, containment row — and `LatticeInvPayload.unpack` takes it to the single `QBlockInv` payload of Recipe [R73.3](#) (`BlockInvPayload`: `block` := the down-set's realization, the containment row transported along Proposition [R73.4](#); facets are a citation, entering the factoring theorems as the cited node's invariance, never as payload data). The factoring pair `Semiflow.qLatticeInv_sound_elab / Behavior.qLatticeInv_sound_elab` is  $\text{sound}_A = \text{sound}_T \circ \text{unpack}$  with `qBlockInv_sound` as the literal  $\text{sound}_T$ , applied to the unpacked payload. The checker-side ingredients (membership of the down-set's realization in the lattice, down-closedness of the cell-set) are recomputed side conditions, already specified with the rule; the mechanization deliberately consumes the down-closedness and leaves the membership fact checker-side (with down-closedness consumed and the cited member judgment supplied, soundness holds for the bare cell union, a fortiori with the extra condition).

**Recipe R73.2** (the alias template). An alias rule  $A$  is warranted by a verified function  $\text{unpack} : \text{Step}_A \rightarrow \text{Step}_T$  into the payload of a *single* target node  $T$ , such that the  $A$ -node and its image conclude the same judgment from the same premises: one node  $\rightarrow$  one node, payload transported, nothing derived. Soundness is then a corollary,  $\text{sound}_A = \text{sound}_T \circ \text{unpack}$ .

**The rule.** The certificate supplies the family object (the block lattice of the `QMORSEDISCHARGE` section, with its recomputed readout: join-irreducible members  $b$ , lower covers  $b^-$ , cells  $T(b) := \text{cl}(b \setminus b^-)$ , the induced cell order), a cell-set  $C$  of join-irreducibles that is *down-closed* in the cell order, and one containment row  $X_0 \subseteq \bigcup \downarrow C$ . It concludes  $\text{QInvScope}(\bigcup \downarrow C, X_0)$ , writing  $\bigcup \downarrow C$  for the union of the cells of  $C$ .

**Recipe R73.3** (the filled-in application — `QLATTICEINV` rewritten *into* the `QBLOCKINV` instance). Let  $N_C := \bigvee_{b \in C} b$  — the lattice join of the down-set's members, itself a member of the family object by join-closure. The `QLATTICEINV` node rewrites to the single `QBLOCKINV` application

$$\frac{\begin{array}{l} N_C \text{ (the member: block data from the family object)} \\ \{ \vdash \text{ facet-inward sign rows of } N_C \} \text{ (cited from the family object, byte-identical)} \\ \vdash X_0 \subseteq N_C \text{ (the rule's own containment row, rewritten along Proposition \a href\#R73.4)} \end{array}}{\mathcal{M} \models \text{QInvScope}(N_C, X_0)} \quad \text{QBLOCKINV}$$

and the conclusion is the alias's conclusion *on the nose* by Proposition [R73.4](#):  $N_C = \bigcup \downarrow C$  as regions. Every internal facet of  $N_C$  already carries its sign row in the family object — membership of  $N_C$  in the lattice is exactly the checker's recomputed side condition, and no row is re-proven.

**Proposition R73.4** (the exactness identity). *For a down-closed set  $C$  of join-irreducibles,*

$$\bigcup_{b \in C} T(b) = \bigvee_{b \in C} b \quad (\text{equality of realizations}).$$

*Proof.* Two readout facts, one lattice fact. (*cells tile members*) Every member  $N$  of the lattice is the union of the cells of the join-irreducibles below it,  $N = \bigcup \{T(b) : b \in J, b \subseteq N\}$ : downward induction along lower covers —  $T(b)$  is by definition what  $b$  adds over  $b^-$ , so peeling cells descends through the lattice and exhausts  $N$ . (*Birkhoff exactness*) In a finite *distributive* lattice (a lattice

of regions closed under union and intersection — a ring of sets — qualifies) the map  $N \mapsto \{b \in J : b \subseteq N\}$  is an isomorphism onto down-closed sets of join-irreducibles [22]; for down-closed  $C$  the join  $N_C$  then satisfies  $\{b \in J : b \subseteq N_C\} = C$ . Distributivity — join-primality of the irreducibles — is what forbids the join from absorbing new join-irreducibles, and it is not decoration: in the diamond lattice  $M_3$ , two of the three atoms form a down-closed set of join-irreducibles whose join absorbs the third. Substituting the second fact into the first:  $N_C = \bigcup_{b \in C} T(b)$ . (Where each half lives: the tiling is the readout’s own construction, recomputed by the checker per certificate; the isomorphism is the one classical fact, and down-closedness of  $C$  is the recomputed side condition that makes the identity exact rather than merely  $\supseteq$ .) *Mechanization note, the honest disanalogy, adjudicated against the descent:* `qLatticeInv_exact` binds the identity in union form —  $\bigcup_{b \in C} T(b) = \bigcup_{b \in C} b$  as realizations, the right side the bare union over the down-set — and proves it from the row pack alone: *no distributivity hypothesis is consumed*, and none is encoded, since the rows are satisfiable in absorption-positive families. The route never forms the join: each point of the union arrives with a witnessing member  $b \in C$ ; the well-founded descent through lower covers (the discharge chain’s `exists_cellMem_subset`) runs *inside that one member*, landing the point in the cell of a join-irreducible  $b' \subseteq b$ ; and the pin-step is two rows — the induced-order binding `horder` turns  $b' \subseteq b$  into  $b' \leq b$  in the cell order, and down-closedness closes  $b' \in C$ . Join-primality forbids absorption below the join; the descent never visits the join, so the prohibition is sidestepped, not re-proven. What distributivity carries is therefore only the display seam — reading the bare union as the realization of the member  $N_C = \bigvee C$  — and the seam consumes less than the full hypothesis: it needs exactly join-realizes-as-union, which is the checker’s recomputed membership of the down-set’s realization (a member that is a union of members is their least upper bound), a consequence of the ring-of-sets structure strictly weaker than distributivity (union-closed absorption-positive families exist: there the identity holds in *both* forms while the classifying map still fails). The  $M_3$  witness above, realized as sets — atom realizations  $\{1\}, \{2\}, \{3\}$ , top  $\{1, 2, 3\}$ ,  $C$  the first two atoms — separates the strata cleanly: every row holds, the union form holds ( $\{1, 2\} = \{1, 2\}$ ), the join form fails ( $\{1, 2\} \neq \{1, 2, 3\}$ ), and it is the membership recompute that refuses. Nothing mechanized rests on the classical citation; it remains as warrant genealogy for this proof’s route.  $\square$

**Recipe R73.5** (payload transport — field for field).

| QLatticeInv payload                           | QBlockInv payload                                                                   |
|-----------------------------------------------|-------------------------------------------------------------------------------------|
| family object + down-set $C$<br>(no own rows) | block $N_C = \bigvee C$ (member, by join-closure)                                   |
| init_containment                              | facets := $N_C$ ’s rows, cited from the family object                               |
| conclusion region $\bigcup \downarrow C$      | init_containment (rewritten along Proposition R73.4: $\bigcup \downarrow C = N_C$ ) |
|                                               | conclusion region $N_C$ (Proposition R73.4)                                         |

The checker’s recomputed side conditions at the alias:  $C$  down-closed in the recomputed cell order, and  $N_C$  a member whose facet rows are all present — both facts about the readout, not about the dynamics.

**Proposition R73.6** (soundness corollary, through the target). *If the family object’s rows are valid and  $X_0 \subseteq \bigcup \downarrow C$ , then — under the QBlockInv section’s standing hypotheses (its conditional ledger, applied to the member  $N_C$ ) —  $\text{QInvScope}(\bigcup \downarrow C, X_0)$  holds.*

*Proof.* The member  $N_C$  is a strictly facet-inward compact block of the family object, so the QBlockInv section’s theorem chain applies to it verbatim (its rows *are* rows of the family object); the conclusion transports along Proposition R73.4. Mechanized, this proposition is one composition,  $\text{sound}_{\text{QLatticeInv}} = \text{sound}_{\text{QBlockInv}} \circ \text{unpack}$  — there is no other content, which is the alias genre working as intended: the through-the-target form (`Semiflow.qLatticeInv_sound / Behavior.qLatticeInv_sound`) takes the cited node’s `QInvScope` conclusion as judgment-level cargo and rewrites along the identity; the end-to-end form (`_of_blockInv`) composes through `qBlockInv_sound` verbatim. Both strata are covered because the identity is dynamics-free — the alias transports at every semantics for free.  $\square$

**Remark R73.7** (why the rule earns its keep). Down-closed cell-sets are the invariant enclosures the Morse readout offers for free: once an orbit enters a member it never leaves, so the sequence of cells it visits is non-increasing in the cell order, and the down-set of any initial cell-set is forward-invariant and encloses the reachable set. The alias turns each such enclosure into a one-node citation — the member’s rows already exist in the family object — where a direct proof would re-derive facet rows the certificate already carries. Downstream, the minted `QInvScope` judgment feeds QBASIN’s safety factory and scope-restriction of reach obligations, which is where the row-count payoff compounds.

| proof step                                    | formal anchor                                                                                                                                                                                                                                                          |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| target soundness                              | <code>qBlockInv_sound</code> ( <code>QtslSoundness.lean</code> ) and the QBlockInv section’s discharge chain                                                                                                                                                           |
| Proposition R73.4                             | <code>qLatticeInv_exact</code> ( <code>QtslLatticeInv.lean</code> , bound in the entry; union form, no distributivity consumed) via the <code>exists_cellMem_subset</code> descent; Birkhoff representation [22] as warrant genealogy (nothing mechanized rests on it) |
| “cells tile members”<br>alias-level corollary | <code>member_subset_cells_le</code> (principal down-set instance)<br><code>Semiflow.qLatticeInv_sound</code> + <code>_of_blockInv</code> and the behavioural-floor pair (bound in the entry)                                                                           |
| payload-level elaborator (Remark R73.1)       | <code>LatticeInvPayload.unpack</code> + <code>Semiflow.qLatticeInv_sound_elab</code><br>+ <code>Behavior.qLatticeInv_sound_elab</code> (module <code>BernsteinLogic.QtslElaborators</code> , bound in the entry)                                                       |
| side conditions                               | down-closedness: consumed ( <code>IsLowerSet</code> + <code>horder</code> ); membership of $N_C$ in the lattice: checker-side, deliberately not consumed                                                                                                               |

## Rule 74. QBASIN (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — *QBasin*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\mathcal{M} \models \text{QInvScope}(S, X_0) \quad \{\vdash S \cap \llbracket \beta_j \rrbracket = \emptyset\}_j \text{ (avoid\_disjoint rows)}}{\mathcal{M}, x_0 \in X_0 \models \text{SProbGE}(1, \text{PAlways}(\text{SNot } \beta_j))}$$

the invariance premise is deliberately the `textsQInvScope` judgment and not the Morse-graph judgment: basin avoidance consumes invariance of  $S$  from  $X_0$  alone, and asking the stronger judgment would couple the rule to machinery its soundness never uses.

QBASIN is the campaign rule: safety atoms by citation. From one invariance-scope judgment and one disjointness fact per avoid region, it concludes the safety-shaped region judgment `SProbGE(1, PAlways(SNot  $\beta$ ))` (surface syntax  $P_{\geq 1}[G \neg \beta]$ ) on the scope's init region — the deterministic reading, with no probabilistic machinery: for the deterministic model class the certified claim is universal avoidance, and no supermartingale certificate is synthesized or needed. The economics are the point: the invariance judgment is proved once (typically by `QBLOCKINV` or `QLATTICEINV`, whose facet rows already live in the family object), and each additional avoid region costs exactly its own disjointness rows and nothing else — the invariance side is cited, never re-certified, even when the regions are taken jointly (the batch theorem).

What the checker recomputes: the conclusion tie (the cited judgment's region and init fields equal the rule's own), plus the per-box disjointness sign rows, or the grid bookkeeping when the avoid region is cell-aligned. What would break without each hypothesis: without invariance, a trajectory exits the scope and reaches the avoid set (the proof section's drift counterexample); without the containment, failure at time zero; without disjointness, failure at time zero inside the scope.

The premise is deliberately the invariance judgment and NOT the global Morse judgment: recurrence structure alone implies no enclosure of the init region — an orbit may transit the avoid set en route to its Morse set. Basin-minted and battery-minted safety atoms are type-identical conclusions, so downstream citations consume either producer indifferently; keying on provenance is forbidden by design.

This is the one Morse-family rule whose mechanization is unconditional: the cited theorems are proved outright, with no interface hypotheses.

**notes** The probability-one costume (`SProbGE(1, ·)`) matters for composition: the conclusion lands in the same judgment form the probabilistic batteries mint, so tower citations, boolean combination at the root, and the assume-guarantee spine consume deterministic and stochastic safety atoms through one interface.

### Lean statement

`BernsteinLogic.QtslSoundness.qBasin_sound`

statement bound

```
-- QBasin soundness (roster: "3 lines against the Morse note's lemma
corpus"; axioms note §3.5) — FULLY PROVED, no interface facts. Premises:
the cited `QInvScope` judgment (the E1 edge `QBasin/inv`) and the
avoid-disjointness fact (`QBasin/avoid_disjoint`'s rows). Conclusion:
the safety-shaped `QRegionSat` on the scope's init region. Note the
invariance premise arrives as judgment-level cargo — the certificate-
level fact `QInvScope` carries — which is why no dynamics beyond the
semiflow algebra is consumed here (the roster's §4.6 correction: the
premise is NOT the `QMorse` judgment). -/
theorem qBasin_sound {φ : Semiflow X} {S X0 B : Set X}
  (hscope : φ.InvScopeSat S X0) (hAvoid : S ∩ B = ∅) :
  φ.SafetySat X0 B
```

`BernsteinLogic.QtslSoundness.qBasin_batch`

statement bound

```
-- The batch-factory payoff, mechanized ("one family object, many
G-judgments, no new rows" — zero new rows per additional `β` after the
first containment): one invariant scope
discharges every avoid-region at the price of its own disjointness fact
alone — even jointly. -/
```

```

theorem qBasin_batch {φ : Semiflow X} {S X0 : Set X} {J : Type*}
  {B : J → Set X} (hscope : φ.InvScopeSat S X0)
  (hAvoid : ∀ j, S ∩ B j = ∅) :
  φ.SafetySat X0 (⋃ j, B j)

```

## Correspondence (Steele display ↔ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Both statements are read against the abstract model class: the display's model  $\mathcal{M}$  is any forward semiflow  $\varphi$  (Semiflow X) on a bare state type; the v1 model class realizes it, and nothing in either statement consumes the realization.

- $S, X_0$  (the scope's region and init region)  $\mapsto$  the binders  $S \ X_0 : \text{Set } X$ ; the avoid predicate's extent  $\llbracket \beta_j \rrbracket \mapsto B : \text{Set } X$  in  $\text{qBasin\_sound}$ , and the indexed family  $B : J \rightarrow \text{Set } X$  in  $\text{qBasin\_batch}$  (the display's  $j$ -indexed premise row, taken jointly).
- premise  $\text{inv} : \text{QInvScope}(S, X_0) \mapsto \text{hscope} : \varphi.\text{InvScopeSat } S \ X_0$  — definitionally the conjunction of forward invariance of  $S$  and  $X_0 \subseteq S$ , the judgment's two clauses.
- each  $\text{avoid\_disjoint}$  row  $\vdash S \cap \llbracket \beta_j \rrbracket = \emptyset \mapsto \text{hAvoid} : S \cap B = \emptyset$  (batch:  $\forall j, S \cap B j = \emptyset$ ).
- conclusion  $\mathcal{M}, x_0 \in X_0 \models \text{SProbGE}(1, \text{PAlways}(\text{SNot } \beta_j)) \mapsto$  the goal  $\varphi.\text{SafetySat } X_0 \ B$ : every trajectory from  $X_0$  avoids  $B$  at every forward time — the deterministic probability-one reading of the displayed judgment.  $\text{qBasin\_batch}$  concludes  $\varphi.\text{SafetySat } X_0 (\bigcup_j B j)$ , which yields the displayed per- $j$  family of conclusions by monotonicity.

Width note: the display's E1 annotation, glossed by the flavor text (the premise is the invariance judgment, *not* the Morse judgment), is enforced by type rather than by hypothesis —  $\text{qBasin\_sound}$  consumes an  $\text{InvScopeSat}$  fact and nothing of Morse shape fits that premise slot. No hypothesis of either statement is unused, and neither statement is wider than the displayed rule.

## Lean verifier receipts

module BernsteinLogic.QtslSoundness; bound build 77005991d61d... (bound from receipt 8130a7a7a679cef4, 2026-08-04T01:16:06Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 8130a7a7a679cef4 | 2026-08-04T01:16:06Z | build+decorrelated | current |
| 9eb3706354082144 | 2026-08-04T02:23:12Z | build+decorrelated | current |

5 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current  
 replay (leanchecker (official kernel replay)) stale only  
 hardened (safe\_verify (hardened screen)) stale only  
 decorrelated (lean4export + nanoda (decorrelated kernel)) current

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

Absent: the HOL4 development's object is the base-kernel checker cone, and the satisfaction-judgment stratum this rule concludes in has no HOL twin — no judgment-stratum definition exists in the ported scripts (verified against the port's definition inventory). The disjointness rows this rule consumes ARE covered by the ported checker; the three-line judgment-stratum soundness lemma is Lean-only today.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1) — an ordinary mathematical proof at Putnam-exam-solution detail. Every hypothesis of the rule is used below and remarked on; the formal-anchor table maps each step to the Lean declaration that carries it. This rule concludes a judgment one stratum above the sequent calculus: premises typed  $\vdash$  are ordinary sequents the base kernel replays, and the rule's own soundness lives at the judgment stratum, over a dynamical model. Uniquely among the Morse-family rules in this book, its mechanization is *unconditional*: the cited theorems are proved outright, with no interface hypotheses.

**Setting and claim.** Fix a *forward semiflow*  $\varphi$  on a state space  $X$ : a map  $\Phi : \mathbb{R}_{\geq 0} \times X \rightarrow X$ , written  $\Phi_t$ , with  $\Phi_0 = \text{id}$  and  $\Phi_{s+t} = \Phi_s \circ \Phi_t$  (the abstract shadow of the model class: a deterministic polynomial-drift flow on its chart domains; the two semiflow laws are model-frame data — the proofs below consume only the rule premises (a)–(c)). Regions denote subsets of  $X$ . Two judgment forms participate:

(*invariance scope*)  $\text{QInvScope}(S, X_0)$  asserts the conjunction:  $S$  is forward-invariant ( $x \in S \Rightarrow \Phi_t x \in S$  for all  $t \geq 0$ ) and  $X_0 \subseteq S$ .

(*safety*) the region judgment  $\mathcal{M}, x_0 \in X_0 \vdash \text{SProbGE}(1, \text{PAlways}(\text{SNot } \beta))$  asserts, in the deterministic reading, that every point of  $X_0$  avoids  $B := \llbracket \beta \rrbracket$  at every forward time:  $\forall x \in X_0, \forall t \geq 0, \Phi_t x \notin B$ . The probability-one former  $\text{SProbGE}(1, \cdot)$  is the degenerate probabilistic costume of this universal statement — no measure, no stochastic machinery, and no supermartingale certificate participates anywhere in this section (the model-class fence: deterministic flows only).

The rule's premises: the cited invariance judgment (an internal edge — the claim is consumed, not its derivation), and one disjointness fact  $S \cap \llbracket \beta_j \rrbracket = \emptyset$  per avoid region, discharged by per-box sign rows of the base calculus (or by cell bookkeeping the checker recomputes, when  $\beta_j$  is aligned to the certificate's grid).

**Theorem R74.1** (soundness of QBASIN). *Let  $S, X_0, B \subseteq X$  with (a)  $S$  forward-invariant, (b)  $X_0 \subseteq S$  (jointly: the cited  $\text{QInvScope}(S, X_0)$  judgment), and (c)  $S \cap B = \emptyset$ . Then every  $x \in X_0$  satisfies  $\Phi_t x \notin B$  for all  $t \geq 0$ .*

*Proof.* Three sentences. By (b),  $x \in S$ ; by (a),  $\Phi_t x \in S$  for every  $t$ . Were  $\Phi_t x \in B$ , it would lie in  $S \cap B$ , which (c) declares empty. Hence  $\Phi_t x \notin B$ .  $\square$

**Proposition R74.2** (the batch form). *Let  $\{B_j\}_{j \in J}$  be any family of avoid regions with, for each  $j$ , the single fact  $S \cap B_j = \emptyset$ . Under (a) and (b) of Theorem R74.1, every  $x \in X_0$  avoids  $\bigcup_j B_j$  at every forward time — equivalently, each  $B_j$  separately, and all of them jointly.*

*Proof.* A point of the union lies in some  $B_j$ ; apply Theorem R74.1 to that  $j$ . Nothing about  $J$  is used — the index family may be infinite, though a certificate of course carries finitely many rows.  $\square$

**Remark R74.3** (each hypothesis is load-bearing). Drop (a), keeping (b) and (c): on  $X = \mathbb{R}$  with the unit-speed drift  $\Phi_t x = x + t$ , take  $S = X_0 = [0, 1]$  and  $B = [2, 3]$ . Then  $X_0 \subseteq S$  and  $S \cap B = \emptyset$ , yet the trajectory from  $x = 0$  enters  $B$  at  $t = 2$  — invariance is the entire dynamical content of the rule, and it arrives wholesale through the cited judgment. Drop (b): a point  $x \in X_0 \setminus S$  may sit in  $B$  already, violating the conclusion at  $t = 0$  — concretely, the identity flow with  $S = \{0\}$  and  $X_0 = B = \{1\}$ : (a) and (c) hold, and the start state is the violation. Drop (c): a point of  $S \cap B$  reachable from  $X_0$  (at worst a point of  $X_0 \cap B$  itself, again at  $t = 0$ ) violates it — concretely, the identity flow with  $X_0 = S = B = \{0\}$ : (a) and (b) hold, and the start state sits in the avoid set. The two set-theoretic hypotheses fail at time zero when dropped; the dynamical one fails at positive time — which is why (a) is a judgment with its own producing rules while (b) and (c) are single containment/disjointness rows.

**Remark R74.4** (the premise is the invariance judgment, not the Morse judgment). A natural mis-design would take the global Morse judgment as the premise: it is the stronger-looking claim. But  $\text{Morse}(G, \rho)$  alone implies no particular invariant enclosure of  $X_0$  — an orbit from  $X_0$  may transit  $\llbracket \beta \rrbracket$  en route to its Morse set (concretely:  $\dot{x} = 3 - x$  on  $\mathbb{R}$ , Morse structure the attractor  $\{3\}$ ,  $X_0 = \{0\}$ ,  $B = (1, 2)$  — the orbit crosses  $B$  on its way to the attractor), so recurrence structure without an enclosure proves no safety at all. The sound premise is the certificate-level invariance fact, and the Morse machinery contributes through the *factory* instead: every lattice member of the family object (and every down-closed cell-set, via  $\text{QLATTICEINV}$ ) yields a  $\text{QInvScope}$  judgment whose facet rows already exist in the family object. This rule then mints one safety atom per avoid region at the price of that region's own disjointness rows and *nothing else* — the invariance side is cited, never re-certified, so no invariance or facet row is ever added however many avoid regions arrive — which is the row-count payoff the batch form (Proposition R74.2) states as mathematics.

**Remark R74.5** (what the checker recomputes). At this rule the checker verifies: that the cited judgment's region and init fields equal the rule's own scope data (the conclusion tie), and the disjointness rows — per-box, per-atom sign sequents of the base calculus, or grid bookkeeping when  $\beta_j$  is cell-aligned. Nothing else: soundness is pure logic over the judgment's semantic content, which is why the mechanized statement is three lines and hypothesis-free. Basin-minted and battery-minted safety atoms are *type-identical* conclusions, deliberately: downstream citations consume the judgment and may not key on its producer.

**Remark R74.6** (relation to the literature). Safety-by-invariance is the classical frame of set-invariance control — Nagumo's subtangentiality theorem [187] is the ancestral invariance certificate, and Blanchini's survey [23] the modern map of the field — and of barrier certificates [201], which in their usual packaging synthesize one function per specification, coupling the invariance

argument to the safety argument. The rule’s point is the *decoupling*: invariance is certified once, as a judgment (by strictly inward facet rows, at QBLOCKINV), and each safety claim is thereafter a citation plus disjointness rows. The batch corollary is the certificate-side face of the elementary observation that one invariant set separates its contents from every disjoint avoid set at once.

| proof step                                               | formal anchor                                                                             |
|----------------------------------------------------------|-------------------------------------------------------------------------------------------|
| Theorem R74.1                                            | qBasin_sound (QtslSoundness.lean); premise cargo hscope, hAvoid                           |
| Proposition R74.2                                        | qBasin_batch                                                                              |
| invariance-scope denotation                              | Semiflow.InvScopeSat                                                                      |
| safety denotation (SProbGE(1, ·), deterministic reading) | Semiflow.SafetySat                                                                        |
| disjointness rows                                        | per-box BERNONE-family sequents (base kernel); cell-aligned case = recomputed bookkeeping |



## 15.9 Existence Certificates

### Rule 75. QZEROBYMIRANDA (core rule)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QZeroByMiranda*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{b : \text{QMirandaBattery} \quad \text{qtsl\_miranda\_battery}(b, C, g, \Pi)}{\models \text{QZeroExists}(g, C)}$$

QZEROBYMIRANDA is Miranda's theorem as a certificate:  $2m$  face rows over an equidimensional polynomial view, each row a sign sequent claiming the signed component restricted to one face of the cell nonpositive on the lower face and nonnegative on the upper — and the theorem concludes a zero of the vector field inside the cell. Existence only: nothing in this family can conclude uniqueness, count, or non-existence. Face geometry is derived from each row's indices, never stored — a producer-supplied face would be a trusted row — and the face-restriction recompute (exact-rational substitution) is the walk's derived-sequent authority.

**notes** The battery walk checks completeness (each (axis, face) exactly once), per-axis component agreement, component bijectivity, and index ranges as decided-false violations; non-box atoms, non-canonical or over-wide polynomials, list ambiguity, and budget overruns refuse with no verdict. Entered at the roster census (baked, never catalogued); no Lean statement binds it yet.

#### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

theory Qtsl . QZeroByMiranda

quoted from the theory source at render

| QZeroByMiranda : Set;

theory Qtsl . QZeroByMiranda/\*

quoted from the theory source at render

| QZeroByMiranda/proof : QZeroByMiranda -> QProof;  
 | QZeroByMiranda/g : QZeroByMiranda -> PolyVec;  
 | QZeroByMiranda/cell : QZeroByMiranda -> QRegionAtom;  
 | QZeroByMiranda/battery : QZeroByMiranda -> QMirandaBattery;

theory Qtsl . qtsl\_miranda\_battery

quoted from the theory source at render

| qtsl\_miranda\_battery : {battery: QMirandaBattery,  
                           cell: QRegionAtom, g: PolyVec,  
                           params: QRegionAtom} -> Prop;

#### Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

#### HOL4 statement

##### NO HOL STATEMENT YET

No HOL development of the extension theory exists.

#### HOL verifier receipts



**NONE**

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), existence group. This section states the rule’s soundness obligation exactly, per the catalogue’s completeness-over-polish charter (a rule that can only be named-and-gapped is still a row); the discharge is open debt on the mechanization column, and the schema quoted in the entry is the baked ground truth.

**Soundness obligation.** Miranda’s theorem, certified: if each signed component of the equidimensional polynomial view is nonpositive on its lower face and nonnegative on its upper face of the cell (the  $2m$  face rows, with face geometry derived from row indices and each restriction recomputed by exact-rational substitution), then the view has a zero in the cell — for every parameter point of the parameter cell. Existence only; uniqueness, count, and non-existence are outside the family by design.

*Remark* R75.1 (status). No Lean statement binds this node. Miranda’s theorem (Poincaré–Miranda) is classical; the mechanization route would state it over the boxed cell with the walk’s derived face restrictions as hypotheses.

## Rule 76. QEQUILIBRIUMFROMINWARD (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QEquilibriumFromInward*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{\text{inv} : \text{QBlockInv} \text{ (strictly-inward facets, box block)}}{\models \text{QEquilibriumExists}(f, C)}$$

QEQUILIBRIUMFROMINWARD is a derived rule: a strictly-inward facet battery on a box block — every facet's sign sequent points the field inward — is already the existence battery for the negated field with identity pairing, so an equilibrium inside the block follows by citation, with zero new oracle demand. The premises are the block-invariant evidence rows themselves; the unpacking instantiates the zero-existence obligation.

**notes** For fixed points the battery's subject is the map minus the identity; the elaboration recompute tying the two is a named deferral at the ground seam. Entered at the roster census (baked, never catalogued); no Lean statement binds it yet.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

theory Qtsl . QEquilibriumFromInward

quoted from the theory source at render

| QEquilibriumFromInward : Set;

theory Qtsl . QEquilibriumFromInward/\*

quoted from the theory source at render

| QEquilibriumFromInward/proof : QEquilibriumFromInward -> QProof;  
| QEquilibriumFromInward/inv : QEquilibriumFromInward -> QBlockInv;

### Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

### HOL4 statement

**NO HOL STATEMENT YET**

No HOL development of the extension theory exists.

### HOL verifier receipts

**NONE**

no HOL verification receipt exists for this rule.

### Proof

Genre: admissible rule (tier 2), existence group. This section states the rule's soundness obligation exactly, per the catalogue's completeness-over-polish charter (a rule that can only be named-and-gapped is still a row); the discharge is open debt on the mechanization column, and the schema quoted in the entry is the baked ground truth.

**Soundness obligation.** A strictly-inward facet battery on a box block is the existence battery for the negated field with identity pairing: inwardness on every facet gives exactly the sign pattern Miranda's theorem demands for  $-f$ , so the unpacking cites the block-invariant evidence rows and instantiates the zero-existence obligation — premises by citation, zero new oracle demand. Soundness is therefore inherited from the zero-existence rule's obligation.

*Remark R76.1 (status).* No Lean statement binds this node; its obligation factors entirely through the zero-existence rule's. For fixed points the battery's subject is the map minus the identity, and the elaboration recompute tying the two is a named deferral at the ground seam.

## Rule 77. QCOVERBYMIRANDA (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QCoverByMiranda*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{b : \text{QMirandaBattery} \quad \text{qtsl\_miranda\_battery}(b, W, \sigma - \text{id}_Z, Z)}{\models Z \subseteq \sigma(W)}$$

QCOVERBYMIRANDA is the chart-image discharge at per-(patch cell, target cell) granularity: the battery's subject is chart minus target in joint variables, with the target cell as parameters — a certified zero of  $\sigma(w) - z$  in the patch cell for every  $z$  in the target cell is exactly the containment of the target cell in the chart image. Union to a region cover rides the existing cover machinery; a cover is never inferred from an image enclosure — enclosures bound images from outside, and only the zero-existence argument bounds them from inside.

notes Entered at the roster census (baked, never catalogued); no Lean statement binds it yet. The battery walk is shared with the zero-existence rule.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

theory Qtsl . QCoverByMiranda

quoted from the theory source at render

| QCoverByMiranda : Set;

theory Qtsl . QCoverByMiranda/\*

quoted from the theory source at render

| QCoverByMiranda/proof : QCoverByMiranda -> QProof;  
| QCoverByMiranda/chart : QCoverByMiranda -> PolyVec;  
| QCoverByMiranda/wcell : QCoverByMiranda -> QRegionAtom;  
| QCoverByMiranda/zcell : QCoverByMiranda -> QRegionAtom;  
| QCoverByMiranda/battery : QCoverByMiranda -> QMirandaBattery;

### Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

### HOL4 statement

#### NO HOL STATEMENT YET

No HOL development of the extension theory exists.

### HOL verifier receipts

#### NONE

no HOL verification receipt exists for this rule.

### Proof

Genre: core rule (tier 1), existence group. This section states the rule's soundness obligation exactly, per the catalogue's completeness-over-polish charter (a rule that can only be named-and-gapped is still a row); the discharge is open debt on the mechanization column, and the schema quoted in the entry is the baked ground truth.

**Soundness obligation.** For each target point  $z$  of the target cell, the battery (subject: chart minus target in joint variables, parameters the target cell) certifies a zero of  $\sigma(w) - z$  in the patch cell — i.e. a preimage of  $z$  under the chart — so the target cell is contained in the chart image. Union to a region cover rides the existing cover machinery; a cover is never inferred from an image enclosure.

*Remark R77.1* (status). No Lean statement binds this node; the obligation factors through the zero-existence rule's, applied parametrically in the target point.

## 15.10 Composition and Aggregation

### Rule 78. QWIRECOMPOSE (core rule)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QWireCompose*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$\{\Sigma_i \models (I_i, \alpha_i, \gamma_i[\cdot, \kappa_i])\}_{i=1}^n$      $\text{QWireCert}(w, (\text{spec}_i)_i \Rightarrow \text{spec}_{\text{out}})$

$\text{rank} : \{1, \dots, n\} \rightarrow \mathbb{N}; \quad \{\text{rank}_i < \text{rank}_j\}_{(i \rightarrow j) \in w}$     (the causality certificate: one rank per component)

W0: the composite martingale problem admits a càdlàg strong-Markov solution, unique in law, from every initial state (stochastic mechanism)

W1: transition data factor through per-place colour multisets (SDCPN mechanisms; typing-)

W2: the within-instant immediate-jump relation is well-founded, of uniformity

W3: simultaneous enablings carry a declared serialization or a certified reset commutation (a.s.)

$\kappa_{\text{out}} := \bigoplus_{\text{legs}} \kappa_{\text{leg}}, \quad \kappa_a \oplus \kappa_b := \kappa_a + \kappa_b$  (slack addition along lens composition; check)

$\text{SysWire}(w, [\Sigma_1, \dots, \Sigma_n]) \models \text{spec}_{\text{out}}$

satisfaction of an interface-bearing spec is plain *QSat* — behaviours whose inputs conform to the assumption deliver the guarantee; closed  $\mathcal{M} \models s$  is the empty-interface instance (defined once at the judgment-kinds home; the former subscripted display  $\models_1$  and the *QSatI* sort are both dissolved). The first two premises are derivation edges (judgment-kind premises); the well-posedness families enter as proof leaves on the node's own well-posedness list, each owed only by the mechanism profiles that name it — W0 by stochastic profiles, W1 by SDCPN profiles, W2/W3 the syntactic class (their four-way recompute relation is not yet enacted; the leaves enter as posted proof rows) — and the causality ranks ride the node as one natural per component. The causality certificate and the slack composite are recomputed by the checker. Tier split derived from mechanism field-family population, never a stored tag: **admissible** where every mechanism derives ODE/discrete — the composite certificate  $\sum_i \lambda_i V_i$  is syntactically constructible and the node unpacks through the sequent-layer transport (PROMISE/SYZGY) — **core** where any mechanism is stochastic (SDE/GSHS): the warrant is the per-profile semantic theorems. This display is the acyclic  $n$ -ary form (the causality certificate makes a directed cycle unrepresentable); feedback rides the binary small-gain cells, sum-form and gauge-form.

QWIRECOMPOSE is the satisfaction-judgment stratum's composition rule — assume-guarantee transport over a structured system presentation. From interface-indexed satisfaction of each component (*QSatI*( $\Sigma_i, \text{spec}_i$ ): every behaviour of  $\Sigma_i$  whose inputs satisfy the assumption  $\alpha_i$  delivers the guarantee  $\gamma_i$ , with quantitative slack where declared) and a wiring certificate (the lens square transporting delivered guarantees into downstream assumptions, with slack  $\kappa \in \mathcal{K}_{\infty}^0$ ), it concludes satisfaction of the wired composite at the output spec. A producer uses it to certify a system of interacting components — each certified once, against its own datasheet — without ever building the monolithic certificate.

The soundness warrant factors, and the factorization organizes everything on this page:  $\text{sound} = (\text{transport arithmetic}) \circ (\text{semantic reading})$ . The transport half is profile-generic exact polynomial arithmetic — the  $(1, \lambda, 1, 1)$  cancellation of cross-supplies, the sequent-layer machinery of PROMISE and SYZGY — and consumes only linearity of the obligation operator over the wiring. The semantic half is per-profile, and it is where this entry's conditional check lives. What the cited theorems prove, per profile and shape: for general stochastic hybrid systems (GSHS), *series* composition (the conditioning tower, qualitative and product-bound forms, plus the certified pipeline through the maximal-inequality kernel), *parallel* composition (the weighted-sum supermartingale — the semantic face of the syzygy weights), and *feedback in sum-form* (the cancellation run under conditional expectation along the joint jump skeleton, with non-expansive boundary rows) are proven; the  $n$ -ary *acyclic* assembly is proven at any  $n$  by strong induction on the causality certificate, with union-bound slack composition; and the per-profile tier split is itself a proven, derived function of the presentation. *Feedback in gauge-form* (jumps that expand the certificate, repaid by flow decay) is partial: its reduction theorem and the max-form piecewise-deterministic instance are machine-checked end-to-end, while the finite-index price of the diffusion term and the admissible-window pinch remain open — with the diffusion cell's parking reason retired: within-segment decay for gauged certificates is now available at the martingale-problem profile, so the open part is the accounting, not the stochastic calculus. The deterministic-ODE row is at statement level; the discrete-kernel row's assembly is proven — B1  $n$ -ary acyclic, B2 parallel, and B3 sum-form small-gain, end-to-end over the trajectory kernel — while the sampled-data row remains stated, not yet assembled. The discrete row carries a substrate asymmetry worth stating with the grid: it consumes *no* interface hypothesis fields (contrast the GSHS cells' D0–D4 data — Dynkin, début sampling, skeleton adaptedness), because its execution semantics is mathlib's own `Kernel.traj`. The commentary states this grid plainly because the dashboard's conditional mark warrants exactly the proven cells, never the whole catalogued rule.

Two fences are part of the statement, not omissions. Undirected stochastic wire sharing (common noise) is out of scope: stochastic components sharing a wire need exactly one determining side, and the proven parallel cell covers directed coupling and disjoint tensor only. And the  $n$ -ary theorems are acyclic by construction — the presentation's causality certificate makes a directed cycle

unrepresentable — so feedback rides the binary small-gain cells, where the coupling of jump times between the two components is handled by the joint-skeleton argument.

What a checker recomputes, the rule's row sorts now baked: the causality certificate (a rank function strictly increasing along wires), the slack composite (lens composition plus slack addition, with the class membership rows discharged by `QWIRECERTDISCHARGE`), and the well-posedness row families for the stochastic profiles. The component satisfactions and the wiring squares are cited judgments and inherited sequent leaves — the rule adds no kernel arithmetic of its own.

**notes** The tier split is derived from mechanism field-family population, never stored as a tag: admissible where every mechanism in the presentation derives the ODE or discrete profile (the composite certificate  $\sum_i \lambda_i V_i$  is syntactically constructible and the node unpacks through the sequent-layer transport), core where any mechanism is stochastic (no unpacking exists; the semantic theorems are the warrant). One rule node either way. The judgment sorts the rule concludes into are baked in the extension theory; the checker chain that would consume them is not yet ported — the rule is proven ahead of being wireable, and the dashboard row shows exactly that state.

## Lean statement

BernsteinLogicAux.AGCompose.GshsSeries .certified

statement bound

```

/-- **C1, certified form**: the full conditional-on-interface pipeline. Component
certificates enter as I-bridge outputs on the composite law – nonnegative
supermartingales along the stopped joint skeleton [D1][D2], with barrier values on
their bad sets (floor-row transport [D2]) and violation-witness inclusions [D3]
(the stopped skeleton samples the début of a violation; for component 2 the
certificate process is understood  $\alpha_2$ -conformance-stopped, which is why its witness
hypothesis is relative to `conform2`). The landed [K1] (`reach_avoid_div`,
`Ville.lean`) converts each certificate to a failure bound; `fail_bound` composes
them. Conclusion: the composite QSatI failure bound with certificate-valued  $\varepsilon$ 's,

`μ (conform1 n satOutc) ≤ ofReal (∫ Y1 0) / ofReal a1 + ofReal (∫ Y2 0) / ofReal a2`. -/
theorem certified [IsFiniteMeasure μ]
  {Y1 Y2 : ℕ → Ω → ℝ} {X1 : ℕ → Ω → S1} {X2 : ℕ → Ω → S2}
  {B1 : Set S1} {B2 : Set S2} {a1 a2 : ℝ}
  -- [D1]+[D2]: I-bridge outputs on the composite law
  (hY1 : Supermartingale Y1 $ μ) (hY10 : 0 ≤ Y1)
  (hY2 : Supermartingale Y2 $ μ) (hY20 : 0 ≤ Y2)
  -- floor-row barriers at skeleton samples [D2]
  (ha1 : 0 < a1) (ha2 : 0 < a2)
  (hbar1 : ∀ n ω, X1 n ω ∈ B1 → a1 ≤ Y1 n ω)
  (hbar2 : ∀ n ω, X2 n ω ∈ B2 → a2 ≤ Y2 n ω)
  -- violation witnesses at skeleton samples [D3]
  (hwit1 : conform1 n sat1c ⊆ {ω | ∃ n, X1 n ω ∈ B1})
  (hwit2 : conform2 n sat2c ⊆ {ω | ∃ n, X2 n ω ∈ B2})
  -- the wiring square (κ-slack folded) and the output reading
  (hw : conform1 n sat1 ⊆ conform2)
  (hout : conform1 n sat1 n sat2 ⊆ satOut) :
  μ (conform1 n satOutc) ≤
    ENNReal.ofReal (∫ ω, Y1 0 ω ∂μ) / ENNReal.ofReal a1 +
    ENNReal.ofReal (∫ ω, Y2 0 ω ∂μ) / ENNReal.ofReal a2

```

BernsteinLogicAux.AGCompose.GshsParallel .certified

statement bound

```

/-- **C2, certified form** (n-ary, sum-form): component certificates – nonnegative
supermartingales `Y i` w.r.t. the joint skeleton filtration [D1][D2], syzygy
weights `λi ≥ 0`, and per-component floor rows giving a common barrier level `a` on
each bad set (boundary rows transport pointwise through sums, inventory §3.C
claim 1: on `Bad i` the term `λi·Yi` alone clears `a`, the other terms being
nonnegative) – yield the composite avoid bound via the landed [K1]:

`ofReal a * μ {∃ n, X n ∈ ⋃ i ∈ s, Bad i} ≤ ofReal (∑ i ∈ s, λi · ∫ Yi 0)`.

```

The right side is the syzygy-weighted initial certificate mass – the semantic reading of the composite certificate  $\sum \lambda_i V_i$  at the initial skeleton time. To recover per-component barrier levels `a<sub>i</sub>`, instantiate `a := min over s of λ<sub>i</sub>·a<sub>i</sub>`. -/
theorem certified (s : Finset ι) [IsFiniteMeasure μ]
 {Y : ι → ℕ → Ω → ℝ} {lam : ι → ℝ} {X : ℕ → Ω → St} {Bad : ι → Set St} {a : ℝ}
 (hY : ∀ i ∈ s, Supermartingale (Y i) \$ μ)
 (hY0 : ∀ i ∈ s, 0 ≤ Y i)

```

(hlam :  $\forall i \in s, 0 \leq \text{lam } i$ )
(hbar :  $\forall i \in s, \forall n \omega, X n \omega \in \text{Bad } i \rightarrow a \leq \text{lam } i * Y i n \omega$ ) :
ENNReal.ofReal a *  $\mu \{ \omega \mid \exists n, X n \omega \in \bigcup i \in s, \text{Bad } i \} \leq$ 
ENNReal.ofReal ( $\sum i \in s, \text{lam } i * \int \omega, Y i \theta \omega \partial \mu$ )

```

BernsteinLogicAux.AGCompose.GshsParallel . **certified\_pair**

statement bound

```

/-- **C2, certified form, two components** – the inventory's  $V_1 + \lambda V_2$  shape,
directly. Barrier reading: component 1's floor clears `a` on `B1`, component 2's
 $\lambda$ -weighted floor clears `a` on `B2`; the composite avoid bound is

`ofReal a *  $\mu \{ \exists n, X n \in B_1 \cup B_2 \} \leq \text{ofReal } (\int Y_1 \theta + \lambda \cdot \int Y_2 \theta)$ ` -/
theorem certified_pair [IsFiniteMeasure  $\mu$ ]
  {Y1 Y2 :  $\mathbb{N} \rightarrow \Omega \rightarrow \mathbb{R}$ } {X :  $\mathbb{N} \rightarrow \Omega \rightarrow \text{St}$ } {B1 B2 : Set St} {lam a :  $\mathbb{R}$ }
  (hY1 : Supermartingale Y1  $\mathcal{G} \mu$ ) (hY10 :  $0 \leq Y_1$ )
  (hY2 : Supermartingale Y2  $\mathcal{G} \mu$ ) (hY20 :  $0 \leq Y_2$ ) (hlam :  $0 \leq \text{lam}$ )
  (hbar1 :  $\forall n \omega, X n \omega \in B_1 \rightarrow a \leq Y_1 n \omega$ )
  (hbar2 :  $\forall n \omega, X n \omega \in B_2 \rightarrow a \leq \text{lam} * Y_2 n \omega$ ) :
ENNReal.ofReal a *  $\mu \{ \omega \mid \exists n, X n \omega \in B_1 \cup B_2 \} \leq$ 
ENNReal.ofReal ( $\int \omega, Y_1 \theta \omega \partial \mu + \text{lam} * \int \omega, Y_2 \theta \omega \partial \mu$ )

```

BernsteinLogicAux.GshsFeedbackSum . **supermartingale**

statement bound

```

/-- **The composite certificate is a supermartingale** for the joint-skeleton
filtration – the C3a conclusion's engine. Follows from `drift` by discarding the
nonnegative  $\theta$ -margin. -/
theorem supermartingale : Supermartingale F.W  $\mathcal{G} \mu$ 

```

BernsteinLogicAux.GshsFeedbackSum . **avoid\_bound**

statement bound

```

/-- **The composite avoid bound** (K1, consumed as the landed
`BernsteinLogic.Ville.reach_avoid`): a barrier `a` for `W` on a sampled bad set `B`
bounds the probability that any joint-skeleton sample reaches `B`. -/
theorem avoid_bound {S : Type*} {X :  $\mathbb{N} \rightarrow \Omega \rightarrow S$ } {B : Set S} {a :  $\mathbb{R}$ }
  (hbarrier :  $\forall k \omega, X k \omega \in B \rightarrow a \leq F.W k \omega$ ) :
ENNReal.ofReal a *  $\mu \{ \omega \mid \exists k, X k \omega \in B \} \leq \text{ENNReal.ofReal } (\int \omega, F.W \theta \omega \partial \mu)$ 

```

BernsteinLogicAux.MaxFormPDMP . **sound\_GSHS\_safety\_barrier**

statement bound

```

/-- **The T5 consumer at a general positive barrier** (Statement II's
Step 7 barrier ` $\min(1, \lambda/\lambda')$ ` needs this): with floor ` $m > 0$ ` on the avoid
region, the reach probability is bounded by ` $V x / m$ '. Proved by scaling
the certificate to barrier `1` and consuming the landed
`sound_GSHS_safety`; no new engine plumbing. -/
theorem sound_GSHS_safety_barrier [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {W : State  $\rightarrow \mathbb{R}$ } {U : Set State} {m :  $\mathbb{R}$ } (hm :  $0 < m$ )
  (hW : Measurable W) (hU : MeasurableSet U)
  (hSegD : SegmentDynkin M  $\theta$  W) (hForce : ForcedNonexpansion M W)
  (hNonneg : NonnegV W) {C :  $\mathbb{R}$ } (hC0 :  $0 \leq C$ ) (hWC :  $\forall y, W y \leq C$ )
  (hFloor :  $\forall y \in U, m \leq W y$ ) (x : State) :
Battery.pathMeasure (skel M) (SkelState.alive x)  $\{ \omega \mid \exists n, \omega n \in \text{skelBad } U \}$ 
 $\leq \text{ENNReal.ofReal } (W x / m)$ 

```

**The presentation layer, bound by display.** The System structure is the abstraction the composition theorems consume: the field-family population pattern of each mechanism (from which the doctrine profile is *derived*, never stored) and the wiring order with its causality certificate carried as data — a rank function strictly increasing along wires, which is precisely what the checker-side causality condition recomputes. System.qagTier derives the rule's tier from the same data, with both unfoldings proven; admissible\_constructs makes the admissible face's “syntactically constructible” clause a checkable statement rather than prose.

BernsteinLogicAux.QtslCompose.Presentation . **System**

statement bound

```

/-- A System presentation, at the composition-relevant abstraction: `n` charts
with their field-family population patterns, the wiring DAG (`upstream j` =
the charts whose outputs wire into `j`'s inputs), and the causality
certificate carried as **data** – a rank function strictly increasing along

```



```
wires, exactly what the checker's causality-DAG side condition recomputes.
Feedback (a directed cycle) is *unrepresentable* in this structure; the
feedback transport shape rides the landed binary cells
(`AGCompose/GshsFeedbackSum.lean`, `GaugeSmallGain.lean`, `MaxFormPDMP.lean`)
– see the AGCompose module docstring. -/
structure System (n : ℕ) : Type where
  charts : Fin n → ChartShape
  upstream : Fin n → Finset (Fin n)
  rank : Fin n → ℕ
  causality : ∀ j, ∀ i ∈ upstream j, rank i < rank j
```

BernsteinLogicAux.QtslCompose.Presentation.**System.qagTier**

statement bound

```
/-- The `QWireCompose` tier at this System, **derived from the derived
profiles** (never a stored tag): admissible iff every chart's profile admits a
syntactically constructible composite certificate. -/
def System.qagTier {n : ℕ} (S : System n) : Tier
```

BernsteinLogicAux.QtslCompose.Presentation.**System.qagTier\_admissible\_iff**

statement bound

```
/-- The split, unfolded to the profile characterization: admissible iff every
chart derives ODE or discrete. -/
theorem System.qagTier_admissible_iff {n : ℕ} (S : System n) :
  S.qagTier = .admissible ↔
    ∀ i, (S.charts i).profile = .ode ∨ (S.charts i).profile = .discrete
```

BernsteinLogicAux.QtslCompose.Presentation.**System.qagTier\_core\_iff**

statement bound

```
/-- The core face of the split: core iff some chart is stochastic (SDE or
GSHS). -/
theorem System.qagTier_core_iff {n : ℕ} (S : System n) :
  S.qagTier = .core ↔
    ∃ i, (S.charts i).profile = .sde ∨ (S.charts i).profile = .gshs
```

BernsteinLogicAux.QtslCompose.Presentation.**admissible\_constructs**

statement bound

```
/-- The core face of the split: core iff some chart is stochastic (SDE or
GSHS). -/
theorem System.qagTier_core_iff {n : ℕ} (S : System n) :
  S.qagTier = .core ↔
    ∃ i, (S.charts i).profile = .sde ∨ (S.charts i).profile = .gshs := by
  have hsplit : S.qagTier = .admissible ∨ S.qagTier = .core := by
    unfold System.qagTier
    split <=> simp
  constructor
  · intro h
    by_contra hno
    push_neg at hno
    have : S.qagTier = .admissible := by
      rw [S.qagTier_admissible_iff]
      intro i
      have := hno i
      cases hp : (S.charts i).profile <=> simp_all
      rw [this] at h
      exact absurd h (by simp)
  · intro (i, hi)
    rcases hsplit with h | h
    · rw [S.qagTier_admissible_iff] at h
      rcases h i with ho | hd <=> rcases hi with hs | hs <=> simp_all
    · exact h

/--! ## The admissible face's constructibility, pointwise

The composite certificate for the admissible tier is the weighted sum
 $\sum i, \lambda i \cdot V i$ ; its composite obligation row follows *pointwise* from the
component dissipation rows ( $e i \leq 0$  on the operating region) and the wiring
```

```

identification rows (`g k = 0` there), at nonnegative certificate weights and
free-sign wiring multipliers. This is the pointwise shadow of the landed
sequent-layer theorem (`sound_syzygy`, main package – the (T) layer of the
inventory §0 factorization); it lives here so that the tier definition's
"syntactically constructible" clause is a statement, not prose. -/
theorem admissible_construts {ι κIdx X : Type*} [Fintype ι] [Fintype κIdx]
  (R : Set X) (lam : ι → ℝ) (e : ι → X → ℝ) (nu : κIdx → ℝ) (g : κIdx → X → ℝ)
  (hlam : ∀ i, 0 ≤ lam i)
  (he : ∀ i, ∀ x ∈ R, e i x ≤ 0)
  (hg : ∀ k, ∀ x ∈ R, g k x = 0) :
  ∀ x ∈ R, (∑ i, lam i * e i x) + (∑ k, nu k * g k x) ≤ 0

```

BernsteinLogicAux.QtslCompose.AGCompose.**sat\_of\_conform**

statement bound

```

/-- **Conformance propagation** (pointwise core). If the wiring squares
transport external conformance + upstream deliveries into each component's
conformance, and each component delivers on conforming inputs (at this `ω`),
then every component delivers. Strong induction on the causality rank – the
presentation's causality certificate is exactly what makes this well-founded. -/
theorem sat_of_conform
  (hw : ∀ j, conformExt n (⋂ i ∈ S.upstream j, sat i) ⊆ conform j)
  {ω : Ω} (h : ∀ j, ω ∈ conform j → ω ∈ sat j)
  (hext : ω ∈ conformExt) : ∀ j, ω ∈ sat j

```

BernsteinLogicAux.QtslCompose.AGCompose.**qualitative**

statement bound

```

/-- **QWireCompose, qualitative form** (a.e.; the n-ary generalization of
`AGCompose.Series.qualitative`): component QSatI's as a.e. implications +
wiring squares yield the composite a.e. delivery of every guarantee, hence of
any output spec read from them (`hout`). -/
theorem qualitative [MeasurableSpace Ω] {μ : Measure Ω}
  (hw : ∀ j, conformExt n (⋂ i ∈ S.upstream j, sat i) ⊆ conform j)
  (h : ∀ j, ∀m ω ∂μ, ω ∈ conform j → ω ∈ sat j)
  {satOut : Set Ω} (hout : conformExt n (⋂ j, sat j) ⊆ satOut) :
  ∀m ω ∂μ, ω ∈ conformExt → ω ∈ satOut

```

BernsteinLogicAux.QtslCompose.AGCompose.**fail\_subset**

statement bound

```

/-- **The n-ary failure decomposition** (pure set algebra; the rule's
union-bound side of slack composition): a composite failure under conforming
external inputs localizes at some component that failed *while conforming* –
namely a rank-minimal failing one, whose upstream all delivered. -/
theorem fail_subset
  (hw : ∀ j, conformExt n (⋂ i ∈ S.upstream j, sat i) ⊆ conform j) :
  conformExt n (⋂ j, sat j)c ⊆ ⋃ j, conform j n (sat j)c

```

BernsteinLogicAux.QtslCompose.AGCompose.**fail\_bound**

statement bound

```

/-- **QWireCompose, probabilistic form: n-ary union-bound slack composition** –
the inventory §3.B1 display  $\mathbb{P}(\text{composite fails}) \leq \sum i (1 - p_i)$  at any `n`,
in failure-measure form (`ε j` plays `1 - p_j`). -/
theorem fail_bound [MeasurableSpace Ω] {μ : Measure Ω} {ε : Fin n → ℝ≥0∞}
  (hw : ∀ j, conformExt n (⋂ i ∈ S.upstream j, sat i) ⊆ conform j)
  (hε : ∀ j, μ (conform j n (sat j)c) ≤ ε j) :
  μ (conformExt n (⋂ j, sat j)c) ≤ ∑ j, ε j

```

BernsteinLogicAux.QtslCompose.AGCompose.**fail\_bound\_out**

statement bound

```

/-- `fail_bound` read at an output spec (`hout` = the output reading of the
delivered guarantees, as in the binary cells). -/
theorem fail_bound_out [MeasurableSpace Ω] {μ : Measure Ω} {ε : Fin n → ℝ≥0∞}
  {satOut : Set Ω}
  (hw : ∀ j, conformExt n (⋂ i ∈ S.upstream j, sat i) ⊆ conform j)
  (hout : conformExt n (⋂ j, sat j) ⊆ satOut)
  (hε : ∀ j, μ (conform j n (sat j)c) ≤ ε j) :
  μ (conformExt n satOutc) ≤ ∑ j, ε j

```

**The B-row (discrete-kernel) assembly.** The row the commentary's grid records as proven: B1  $n$ -ary acyclic union-bound composition through the first-failure device, B2 parallel weighted-sum on the product kernel, B3 sum-form small-gain cancellation — machine-checked end-to-end over Kernel.traj through the battery substrate, with no semantic hypothesis fields (the profile identification enters as the choice of substrate, which is the conditional mark's honest residue for this row).

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **firstFail**

statement bound

```
/-- The event that component `j` is a first violator: at some time the path
is in `j`'s violation set, no component's violation set having been visited
before. Distinct components' first-failure events may overlap (simultaneous
first violations); the union bound is indifferent. -/
def firstFail (bad : Fin m → Set S) (j : Fin m) : Set (ℕ → S)
```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **reach\_subset\_iUnion\_firstFail**

statement bound

```
/-- The first-failure decomposition (the discrete-profile face of K4's event
decomposition; kin of the event-stratum `AGCompose.fail_subset`, with
time-minimality replacing rank-minimality): a path that ever visits any
violation set has a first violation, and that violation belongs to some
component. Pure path-set algebra. -/
theorem reach_subset_iUnion_firstFail (bad : Fin m → Set S) :
  {ω : ℕ → S | ∃ n, ω n ∈ ⋃ i, bad i} ⊆ ⋃ j, firstFail bad j
```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **measure\_firstFail\_le**

statement bound

```
/-- The stopped-process device (inventory §3.B1), isolated: stopping the
composite chain at the first violation by any component, component `j`'s
certificate — with drift live off the joint violation set and barrier only
on `j`'s own violation set — bounds the probability that `j` fails first
by its initial value. K1 (Supermartingale.ville_ineq) at barrier `1`;
contrast `Battery.measure_reach_le`, whose barrier must cover the whole stop
set and which therefore cannot see a multi-component decomposition. -/
theorem measure_firstFail_le {bad : Fin m → Set S} {V : S → ℝ} {C : ℝ} {j : Fin m}
  (hA : MeasurableSet (⋃ i, bad i)) (hV : Measurable V)
  (hV0 : ∀ y, 0 ≤ V y) (hVC : ∀ y, V y ≤ C)
  (hbar : ∀ y ∈ bad j, 1 ≤ V y)
  (hdrift : ∀ y, y ∉ (⋃ i, bad i) → ∫ z, V z ∂K y ≤ V y) (x : S) :
  pathMeasure K x (firstFail bad j) ≤ ENNReal.ofReal (V x)
```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **fail\_bound**

statement bound

```
/-- QWireCompose B-row, probabilistic form (engine form; inventory §3.B1's
display `P(composite fails) ≤ ∑ i (1-p_i)` at any `n`, semantically discharged
over `Kernel.traj`): per-component bounded certificates with drift off the
joint violation set and barriers on their own violation sets bound the
composite failure probability by the sum of initial certificate values. The
ε-shaped quantities the event-stratum `AGCompose.fail_bound` consumes as
hypotheses are here established from kernel-level data. -/
theorem fail_bound {bad : Fin m → Set S} {V : Fin m → S → ℝ} {C : Fin m → ℝ}
  (hA : MeasurableSet (⋃ i, bad i))
  (hV : ∀ j, Measurable (V j)) (hV0 : ∀ j y, 0 ≤ V j y)
  (hVC : ∀ j y, V j y ≤ C j)
  (hbar : ∀ j, ∀ y ∈ bad j, 1 ≤ V j y)
  (hdrift : ∀ j, ∀ y, y ∉ (⋃ i, bad i) → ∫ z, V j z ∂K y ≤ V j y) (x : S) :
  pathMeasure K x {ω | ∃ n, ω n ∈ ⋃ i, bad i}
    ≤ ∑ j, ENNReal.ofReal (V j x)
```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **fail\_bound\_wired**

statement bound

```
/-- QWireCompose B-row, wired form (the catalogued rule's hypothesis shape):
per-component datasheet rows — drift valid only on the component's own
conformance region — plus the state-level wiring square (external conformance
folded; joint upstream delivery implies conformance, the I-bridge reading of
`QWireCert` at this stratum) yield the same union-bound conclusion. The
`System`'s wiring order supplies the square's index set; its rank/causality
data is consumed by the event-stratum assembly, not here (time-minimality
replaces the rank induction — see the module header, and `fail_bound_upstream`
```

```

for the honest-width form this now specializes). -/
theorem fail_bound_wired (Sys : System m) {conf bad : Fin m → Set S}
  {V : Fin m → S → ℝ} {C : Fin m → ℝ}
  (hA : MeasurableSet (⋃ i, bad i))
  (hw : ∀ j, (⋂ i ∈ Sys.upstream j, (bad i)c) ⊆ conf j)
  (hV : ∀ j, Measurable (V j)) (hV0 : ∀ j y, 0 ≤ V j y)
  (hVC : ∀ j y, V j y ≤ C j)
  (hbar : ∀ j, ∀ y ∈ bad j, 1 ≤ V j y)
  (hdrift : ∀ j, ∀ y ∈ conf j, y ∉ bad j → ∫ z, V j z ∂K y ≤ V j y) (x : S) :
  pathMeasure K x {ω | ∃ n, ω n ∈ ⋃ i, bad i}
    ≤ ∑ j, ENNReal.ofReal (V j x)

```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **ae\_avoid\_of\_zero**

statement bound

```

/-- **The qualitative reading** (inventory §3.B1's "a.s.-invariance-shaped"
half): started strictly inside every certificate's zero set, the composite
chain a.s. never violates any component's guarantee – the conditioning-tower
display degenerates, at the discrete profile, to the union bound at total
mass zero. -/
theorem ae_avoid_of_zero {bad : Fin m → Set S} {V : Fin m → S → ℝ}
  {C : Fin m → ℝ}
  (hA : MeasurableSet (⋃ i, bad i))
  (hV : ∀ j, Measurable (V j)) (hV0 : ∀ j y, 0 ≤ V j y)
  (hVC : ∀ j y, V j y ≤ C j)
  (hbar : ∀ j, ∀ y ∈ bad j, 1 ≤ V j y)
  (hdrift : ∀ j, ∀ y, y ∉ (⋃ i, bad i) → ∫ z, V j z ∂K y ≤ V j y) {x : S}
  (hzero : ∀ j, V j x = 0) :
  Vm ω ∂pathMeasure K x, ∀ n, ω n ∉ ⋃ i, bad i

```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **drift\_parallelComp**

statement bound

```

/-- **B2, the linearity row** (inventory §3.B2's one-paragraph argument,
machine-checked): component drift rows transport to the weighted-sum
certificate on the synchronous product – the syzygy combination at weights
 $(1, \lambda)$ , verbatim. Off the lifted joint violation set both component rows
are live; linearity of the integral over  $K_1 \parallel K_2$  does the rest. -/
theorem drift_parallelComp {bad1 : Set S1} {bad2 : Set S2}
  {V1 : S1 → ℝ} {V2 : S2 → ℝ} {lam : ℝ} (hlam : 0 ≤ lam)
  (hint1 : ∀ y, y ∉ bad1 → Integrable V1 (K1 y))
  (hint2 : ∀ y, y ∉ bad2 → Integrable V2 (K2 y))
  (hd1 : ∀ y, y ∉ bad1 → ∫ z, V1 z ∂K1 y ≤ V1 y)
  (hd2 : ∀ y, y ∉ bad2 → ∫ z, V2 z ∂K2 y ≤ V2 y) :
  ∀ p : S1 × S2, p ∉ (Prod.fst-1 bad1 ∪ Prod.snd-1 bad2) →
  ∫ q, (V1 q.1 + lam * V2 q.2) ∂(K1 ∥K K2) p
    ≤ V1 p.1 + lam * V2 p.2

```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . **avoid\_parallelComp**

statement bound

```

/-- **B2, assembled** (the discrete face of `GshsParallel.certified`'s
weighted-sum supermartingale claim, end-to-end): on the synchronous product
chain, the weighted-sum certificate bounds the probability that *either*
component ever violates by its initial value. The weight's role is visible in
the barrier hypotheses: component 2's barrier is carried at level  $1/\lambda$ 
(stated as  $1 \leq \lambda \cdot V_2$  – how the syzygy weights meet the common barrier). -/
theorem avoid_parallelComp {bad1 : Set S1} {bad2 : Set S2}
  {V1 : S1 → ℝ} {V2 : S2 → ℝ} {C1 C2 lam : ℝ} (hlam : 0 ≤ lam)
  (hbad1 : MeasurableSet bad1) (hbad2 : MeasurableSet bad2)
  (hV1 : Measurable V1) (hV2 : Measurable V2)
  (hV10 : ∀ y, 0 ≤ V1 y) (hV20 : ∀ y, 0 ≤ V2 y)
  (hV1C : ∀ y, V1 y ≤ C1) (hV2C : ∀ y, V2 y ≤ C2)
  (hbar1 : ∀ y ∈ bad1, 1 ≤ V1 y) (hbar2 : ∀ y ∈ bad2, 1 ≤ lam * V2 y)
  (hd1 : ∀ y, y ∉ bad1 → ∫ z, V1 z ∂K1 y ≤ V1 y)
  (hd2 : ∀ y, y ∉ bad2 → ∫ z, V2 z ∂K2 y ≤ V2 y) (x : S1 × S2) :
  pathMeasure (K1 ∥K K2) x
    {ω | ∃ n, ω n ∈ (Prod.fst-1 bad1 ∪ Prod.snd-1 bad2) : Set (S1 × S2))}
    ≤ ENNReal.ofReal (V1 x.1 + lam * V2 x.2)

```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . smallGain\_drift

statement bound

```

/-- **B3, the cancellation row** (inventory §3.B3; the semantic face of the
checker-level `(1, λ, 1, 1)` syzygy fold `sound_smallGain`, at the kernel
level): per-component dissipation rows with cross-supplies on the joint
closed-loop kernel, under the small-gain membership rows, yield the
`θ`-margin drift row for `W = V1 + λV2`. Only `θ ≤ λ`, `θ ≤ θ` are consumed;
the rule-level ranges `λ > 0`, `θ ∈ (0,1)` are well-formedness, never
hypotheses (the width find of the C3a cell, reproduced at this profile). -/
theorem smallGain_drift (hlam : 0 ≤ lam)
  (hint1 : ∀ y, y ∉ A → Integrable V1 (K y))
  (hint2 : ∀ y, y ∉ A → Integrable V2 (K y))
  (hd1 : ∀ y, y ∉ A → ∫ z, V1 z ∂K y ≤ V1 y - alpha1 y + sigma1 y)
  (hd2 : ∀ y, y ∉ A → ∫ z, V2 z ∂K y ≤ V2 y - alpha2 y + sigma2 y)
  (hm1 : ∀ y, y ∉ A → lam * sigma2 y ≤ (1 - theta) * alpha1 y)
  (hm2 : ∀ y, y ∉ A → sigma1 y ≤ lam * ((1 - theta) * alpha2 y)) :
  ∀ y, y ∉ A → ∫ z, (V1 z + lam * V2 z) ∂K y
    ≤ (V1 y + lam * V2 y) - theta * (alpha1 y + lam * alpha2 y)

```

BernsteinLogicAux.QtslCompose.AGComposeDiscrete . avoid\_smallGain

statement bound

```

/-- **B3, assembled** (the discrete face of the C3a conclusion, end-to-end):
on the closed loop, the small-gain data yields the composite avoid bound for
the weighted-sum certificate – Foster–Lyapunov metatheory
(`ville_ineq`/optional stopping, through `Battery.measure_reach_le`) closing
the cell exactly as the inventory's §3.B3 statement names it. The `α`-sign
hypotheses convert the `θ`-margin into a plain drift row; the barrier is the
caller's (how `1 ≤ W` distributes between components is model data). -/
theorem avoid_smallGain (hlam : 0 ≤ lam) (htheta : 0 ≤ theta)
  {C1 C2 : ℝ} (hA : MeasurableSet A)
  (hV1 : Measurable V1) (hV2 : Measurable V2)
  (hV10 : ∀ y, 0 ≤ V1 y) (hV20 : ∀ y, 0 ≤ V2 y)
  (hV1C : ∀ y, V1 y ≤ C1) (hV2C : ∀ y, V2 y ≤ C2)
  (hbar : ∀ y ∈ A, 1 ≤ V1 y + lam * V2 y)
  (halpha1 : ∀ y, y ∉ A → 0 ≤ alpha1 y)
  (halpha2 : ∀ y, y ∉ A → 0 ≤ alpha2 y)
  (hd1 : ∀ y, y ∉ A → ∫ z, V1 z ∂K y ≤ V1 y - alpha1 y + sigma1 y)
  (hd2 : ∀ y, y ∉ A → ∫ z, V2 z ∂K y ≤ V2 y - alpha2 y + sigma2 y)
  (hm1 : ∀ y, y ∉ A → lam * sigma2 y ≤ (1 - theta) * alpha1 y)
  (hm2 : ∀ y, y ∉ A → sigma1 y ≤ lam * ((1 - theta) * alpha2 y)) (x : S) :
  pathMeasure K x {ω | ∃ n, ω n ∈ A}
    ≤ ENNReal.ofReal (V1 x + lam * V2 x)

```

## Correspondence (Steele display ↔ Lean)

**a bound Lean statement CHANGED since certification — re-review and re-certify** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The display is one  $n$ -ary rule; the bound set is a *grid*, and the dictionary's first duty is to say which display constituent lands in which theorem. All statements live over the compiled behaviour space ( $\Omega$ , the composite execution law's carrier), reached from the presentation through bridge hypotheses — the events conform  $j / \text{sat } j$  (component  $j$ 's input-conformance and guarantee-delivery, interpreted over composite behaviours: the I-bridge of the kernel-semantics interface) — so the certified statements quantify over exactly the Systems whose compilation discharges those bridges.

- the E1 premises  $\Sigma_i \models (\dots)$  (the QSatI judgments)  $\mapsto$ ,  $n$ -ary face:  $h : \forall j, \forall^\# \omega \partial \mu, \omega \in \text{conform } j \rightarrow \omega \in \text{sat } j$  (qualitative) and quantitatively  $h\epsilon : \mu (\text{conform } j \cap (\text{sat } j)^c) \leq \epsilon_j$  (fail\_bound) — per-component satisfaction as a.e. implication / failure-measure bound.
- the QWireCert premise  $\mapsto$   $hw : \forall j, \text{conformExt } n (\cap i \in S.\text{upstream } j, \text{sat } i) \subseteq \text{conform } j$ : the wiring squares with the  $\kappa$ -slack *folded into the predicates*. Consequence, stated plainly: the displayed  $\kappa_{\text{out}}$  recompute row has no counterpart in the  $n$ -ary bound statements — quantitative slack composition appears instead as the union-bound  $\sum_j \epsilon_j$  (fail\_bound\_out, the inventory's  $\sum_i (1 - p_i)$  display at any  $n$ ).

- the causality certificate  $\mapsto \text{System.rank} + \text{System.causality as data}$  (the structure is bound by display): rank strictly increasing along wires is a field, the induction of `sat_of_conform` runs on it, and a directed cycle is unrepresentable — the Lean face of “recomputed.”
- W0–W3 well-posedness rows  $\mapsto$  nothing bound — schema-stratum families, honestly absent.
- the conclusion  $\text{SysWire}(w, \vec{\Sigma}) \models \text{spec}_{\text{out}} \mapsto$  qualitative's goal (a.e. delivery of `satOut` on conforming inputs) and `fail_bound_out`'s (failure measure at the output spec), with hout the output reading of the delivered guarantees; `fail_subset` is the pure set-algebra localization engine (a composite failure localizes at a rank-minimal failing component, conforming there).
- the tier split (the display's admissible/core tier split, stated in the flavor text)  $\mapsto \text{System.qagTier}$ , a *derived function of the presentation* with both unfoldings proven (`qagTier_admissible_iff`: every chart ODE or discrete; `qagTier_core_iff`: some chart stochastic), and `admissible_constructs` making “syntactically constructible” a statement: composite row of  $\sum_i \lambda_i V_i$  from component dissipation rows + wiring identification rows, pointwise, at nonnegative certificate weights and free-sign wiring multipliers — the pointwise shadow of the main package's `sound_syzygy` (witness prose, package boundary).
- the binary GSHS cells, premise shapes as bound: `Series.certified` consumes two supermartingale certificates (Supermartingale  $Y_1 \otimes \mu$ , nonneg, positive barriers, floor-row transport `hbar_1`, violation-witness inclusions `hwit_1`) + one wiring square `conform_1 n sat_1  $\subseteq$  conform_2`, concluding the composite failure bound with certificate-valued  $\varepsilon$ 's (initial mass over barrier, per component); `Parallel.certified / certified_pair` conclude the avoid bound for the weighted-sum certificate (the semantic face of the syzygy weights); `FeedbackSum.supermartingale / avoid_bound` operate on the bundled `SumFormFeedback` structure — and its binder rows carry a width find: `soundness` consumes only  $0 \leq \text{lambda}$ ,  $0 \leq \text{theta}$ ; the rule-level ranges  $\lambda > 0$ ,  $\theta \in (0, 1)$  are well-formedness, never hypotheses — with the  $(1, \lambda, 1, 1)$  cancellation entering as the two integrated a.e. membership rows `mem_1/mem_2`; `sound_GSHS_safety_barrier` is the  $\kappa = 0$  max-form PDMP instance at a general positive floor `m`, consuming measurability, kernel instances, Dynkin/non-expansion data, and a global bound `C` on the certificate.
- the B-row (discrete-kernel) assembly (`AGComposeDiscrete`; end-to-end over `Kernel.traj` through the battery substrate, with *no* D-layer hypothesis fields — the semantic hypotheses of the GSHS cells above have no counterpart here because the discrete substrate is in Lean): the E1 premises  $\mapsto$  per-component bounded certificates  $V_j$  with datasheet drift on `conf j` and barrier  $1 \leq V_j$  on `bad j` (state-level); the `QWireCert` premise  $\mapsto \text{hw} : \bigcap i \in \text{upstream } j, (\text{bad } i)^c \subseteq \text{conf } j$  — the square pre-folded at the state stratum, while rank and causality stay event-stratum: time-minimality (the first-failure device, `firstFail / measure_firstFail_le`, the K1-consumption primitive the multi-component setting needs) replaces the rank induction, so `System.rank/causality` are honestly unused in `fail_bound_wired`; the conclusion  $\mapsto$  the failure-measure bound  $\sum_j \text{ofReal}(V_j x)$ , the  $\varepsilon$ 's of the event-stratum `fail_bound established` rather than assumed. K2 (optional stopping) is consumed transitively — it is bridge step 1 inside `ville_ineq` — and no direct optional-stopping consumption is needed for avoid-shaped conclusions. B2/B3 transpose the binary GSHS cells to the kernel level: `drift_parallelComp / avoid_parallelComp` (the weighted-sum row on  $K_1 \parallel K_2$ , the discrete face of the parallel cell) and `smallGain_drift / avoid_smallGain` (the  $(1, \lambda, 1, 1)$  cancellation, consuming only  $0 \leq \text{lam}$  and  $0 \leq \text{theta}$  — the sum-form cell's width find reproduced at this stratum).

Width note: the conditional mark's content is the grid-vs-rule gap the commentary already states (the deterministic-ODE row at statement level, the sampled-data row stated but not yet assembled, feedback in gauge-form partial; the discrete-kernel row's assembly is discharged, so the gap is one row lighter) — the dictionary adds that within the *proven* cells the mappings above are exact, that the  $n$ -ary statements are acyclic by type (not by side condition), and that no bound statement consumes the  $\mathcal{K}_\infty^0$  structure of  $\kappa$  itself (that lives in `QWIRECERTDISCHARGE`'s bound set; here slack is pre-folded or union-bounded).

## Lean verifier receipts

module `BernsteinLogicAux.AGCompose.GshsSeries`; bound build `ff5cb858ef68...` (bound from receipt `911818331ed966fc`, 2026-07-20T01:42:13Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| cb0f28f9ca6b1c44 | 2026-07-19T18:23:37Z | build+decorrelated                 | current |
| c973aa90108c94d9 | 2026-07-20T01:20:47Z | build+decorrelated                 | current |
| 911818331ed966fc | 2026-07-20T01:42:13Z | build+decorrelated                 | current |
| eada292d784c9848 | 2026-07-20T20:46:24Z | build+decorrelated                 | current |
| 1a58b3dfc83a9cb5 | 2026-07-20T20:57:57Z | build+replay+hardened+decorrelated | current |
| 3358b4d03fa77d7c | 2026-07-21T00:02:20Z | build+decorrelated                 | current |
| d4f9f9c67009d52f | 2026-07-21T05:38:28Z | build+replay+hardened+decorrelated | current |
| f86a2ccbe493f911 | 2026-07-22T18:26:47Z | build+decorrelated                 | current |
| 6d17867e1f4f7cdc | 2026-07-31T21:59:37Z | build+decorrelated                 | current |
| f4d14ab85f9d9554 | 2026-07-31T22:28:31Z | build+decorrelated                 | current |



|                  |                      |                    |         |
|------------------|----------------------|--------------------|---------|
| 90881889f02ba9fe | 2026-08-02T23:30:38Z | build+decorrelated | current |
| 684b3dbc3c2e336  | 2026-08-03T16:26:42Z | build+decorrelated | current |
| 782c96e76de07ffa | 2026-08-03T22:45:26Z | build+decorrelated | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.AGCompose.GshsParallel; bound build 8b8d0ee45b1a... (hashed from the built olean at bind time). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| f455dcef7549cd27 | 2026-07-18T13:22:39Z | build+decorrelated | current |

12 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.GshsFeedbackSum; bound build a0ce53c03e8d... (bound from receipt 45ff739d5d3e0306, 2026-07-20T07:45:49Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 45ff739d5d3e0306 | 2026-07-20T07:45:49Z | build+decorrelated                 | current |
| eb766d96123685ba | 2026-07-20T20:46:31Z | build+decorrelated                 | current |
| f0ef6207375847ca | 2026-07-20T20:58:04Z | build+replay+hardened+decorrelated | current |
| cf3cd505d8c77450 | 2026-07-21T00:02:49Z | build+decorrelated                 | current |
| 51f1db6c22c527e0 | 2026-07-21T05:38:37Z | build+replay+hardened+decorrelated | current |
| 474b9edb3b9dbacd | 2026-07-22T18:26:54Z | build+decorrelated                 | current |
| b0859917a7e80e56 | 2026-07-31T21:59:48Z | build+decorrelated                 | current |
| e0b2e5fdbdb04064 | 2026-07-31T22:28:42Z | build+decorrelated                 | current |
| 57b040e29c83ab7a | 2026-08-02T23:30:50Z | build+decorrelated                 | current |
| d425bb06e18d9d85 | 2026-08-03T16:26:54Z | build+decorrelated                 | current |
| 3058333e955b957  | 2026-08-03T22:45:37Z | build+decorrelated                 | current |

3 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.MaxFormPDMP; bound build 426c38790122... (bound from receipt 0a8c057e8cbf9aab, 2026-07-20T01:42:14Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 42fadeaf4c4d0d5b | 2026-07-19T18:24:26Z | build+decorrelated                 | current |
| 81de1969c895bf9f | 2026-07-20T01:21:39Z | build+decorrelated                 | current |
| 0a8c057e8cbf9aab | 2026-07-20T01:42:14Z | build+decorrelated                 | current |
| 12151b1f2bcd3a91 | 2026-07-20T20:46:35Z | build+decorrelated                 | current |
| 5821a7ee1f92823e | 2026-07-20T20:58:08Z | build+replay+hardened+decorrelated | current |
| c1facca742c8a57f | 2026-07-21T00:03:24Z | build+decorrelated                 | current |
| b37c4c71b35d3ed7 | 2026-07-21T05:38:51Z | build+replay+hardened+decorrelated | current |
| e32a92fac2e848c6 | 2026-07-22T18:27:07Z | build+decorrelated                 | current |
| 2ed08aae31ac4f48 | 2026-07-31T22:00:02Z | build+decorrelated                 | current |
| 1c0dd63c4cc8d828 | 2026-07-31T22:28:56Z | build+decorrelated                 | current |
| 47a3e55f8aa07fc7 | 2026-08-02T23:31:07Z | build+decorrelated                 | current |
| 5ada5a799af57e12 | 2026-08-03T16:27:10Z | build+decorrelated                 | current |
| c2de6b6e1b9e676c | 2026-08-03T22:45:54Z | build+decorrelated                 | current |

2 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.QtslCompose.Presentation; bound build efd8b570583d... (bound from receipt c2304cb9372c9e10, 2026-07-21T05:39:15Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module



and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0d24201fcc020c14 | 2026-07-20T07:45:50Z | build+decorrelated                 | current |
| c6d195b51ddfe44f | 2026-07-21T00:04:00Z | build+decorrelated                 | current |
| c2304cb9372c9e10 | 2026-07-21T05:39:15Z | build+replay+hardened+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.  
module BernsteinLogicAux.QtslCompose.AGCompose; bound build 88a891d2d053... (bound from receipt b967c74819cd1516, 2026-07-21T05:39:06Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 91f9e1cce3d69343 | 2026-07-19T03:38:55Z | build+decorrelated                 | current |
| ccc58392dbd207a9 | 2026-07-19T18:24:44Z | build+decorrelated                 | current |
| 875fc85d5bcb76ca | 2026-07-20T01:21:56Z | build+decorrelated                 | current |
| 23e1df54aafc0165 | 2026-07-20T20:46:49Z | build+decorrelated                 | current |
| 02e33e58a762ea11 | 2026-07-20T20:58:22Z | build+replay+hardened+decorrelated | current |
| 9690cee413e67035 | 2026-07-21T00:03:48Z | build+decorrelated                 | current |
| b967c74819cd1516 | 2026-07-21T05:39:06Z | build+replay+hardened+decorrelated | current |

6 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.  
module BernsteinLogicAux.QtslCompose.AGComposeDiscrete; bound build d5233632f95a... (bound from receipt 88ff74180adf5b95, 2026-08-03T20:13:53Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| 88ff74180adf5b95 | 2026-08-03T20:13:53Z | build+decorrelated | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

gate-layer coverage of the bound build:

- build (lake build (elaboration)) current
- replay (leanchecker (official kernel replay)) current
- hardened (safe\_verify (hardened screen)) current
- decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

HOL4 statement

NO HOL STATEMENT YET

The satisfaction-judgment extension postdates the HOL4 port entirely: the ported cone is the base calculus's first-order check-function surface (theory bern\_catalogue, stopping before even the base proof-object layer), and no extension-theory definition exists anywhere in the HOL development. A HOL statement of this rule presupposes a HOL port of the satisfaction stratum, which is its own campaign; until then this box is the honest state of the edifice.

HOL verifier receipts

NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), with a per-profile tier split derived from the presentation. The coverage is deliberately conditional: this section proves the cells the mechanization proves, names the bridge hypotheses under which they apply, and states what remains open — never claiming past the mechanized grid.

**Setting and claim.** A system is a structured presentation: finitely many *mechanisms* (primitive dynamics declarations — drift, diffusion, jump, and guard–reset field families over typed wires), plus *wirings* (variable identifications) carrying a *causality certificate* — a rank function strictly increasing along wires, so a directed cycle is unrepresentable in the assembled structure. Each mechanism’s *doctrine profile* is derived from which field families are populated: drift only = ODE; +diffusion = SDE; +real-time jumps = general stochastic hybrid (GSHS); counter-time jumps only = discrete kernel. An interface-indexed satisfaction judgment  $\text{QSatl}(\Sigma, (I, \alpha, \gamma[\cdot, \kappa]))$  asserts: every behaviour of  $\Sigma$  whose input trajectory satisfies the assumption  $\alpha$  pointwise delivers the guarantee  $\gamma$  (with quantitative margin per the slack  $\kappa$  where declared). A wiring certificate  $\text{QWireCert}(w, (spec_i \Rightarrow spec_{out}, \kappa))$  asserts the lens square [37]: delivered upstream guarantees, degraded by at most  $\kappa$ , imply downstream assumptions through  $w$ , and the output spec reads off the delivered guarantees (the square’s discharge is  $\text{QWIRECERTDISCHARGE}$ ’s business). The rule: from  $\Sigma_i \models spec_i$  for  $i = 1, \dots, n$  and the wiring certificate, conclude  $\text{SysWire}(w, [\Sigma_1, \dots, \Sigma_n]) \models spec_{out}$ .

The warrant factors as  $\text{sound} = (T) \circ (S)$ : *transport arithmetic* — that composite obligation rows follow from component rows plus wiring rows — which is profile-generic exact polynomial arithmetic; and the *semantic reading* — that the checked row family implies the behavioural claim — which is per-profile, and is where the tier split and every open cell live.

**Lemma R78.1** (transport is profile-silent). *With residuals  $e_i := \mathcal{L}_i V_i + \alpha_i - \sigma_i$  and membership rows  $m_1 := \lambda \sigma_2 - (1-\theta)\alpha_1$ ,  $m_2 := \sigma_1 - \lambda(1-\theta)\alpha_2$ ,*

$$e_1 + \lambda e_2 + m_1 + m_2 = \mathcal{L}(V_1 + \lambda V_2) + \theta(\alpha_1 + \lambda \alpha_2),$$

*an identity of polynomials — the cross-supplies cancel by construction of the multipliers. More generally, for nonnegative certificate weights  $\lambda_i$  and free-sign wiring multipliers  $\nu_k$ : if each  $e_i \leq 0$  and each wiring identity  $g_k = 0$  holds on the operating region, then  $\sum_i \lambda_i e_i + \sum_k \nu_k g_k \leq 0$  there.*

*Proof.* The displayed identity is a two-line expansion; the only property of the obligation operator  $\mathcal{L}$  it consumes is *linearity over the wiring* — which every profile’s operator has:  $\nabla(\cdot) \cdot f$  for ODE, one-step conditional expectation minus identity for kernels, and the extended generator’s interior terms for GSHS. The pointwise fold is *admissible\_constructs*: each  $\lambda_i e_i \leq 0$  (nonnegative times nonpositive), each  $\nu_k g_k = 0$  regardless of sign, finite sums of nonpositives are nonpositive. At the sequent layer this is the PROMISE/SYZGYV machinery, and the small-gain instance is exactly the  $(1, \lambda, 1, 1)$ -weighted SYZGYV node — which is why the split of Remark R78.6 can define “admissible” as *syntactically constructible*: the composite certificate for the non-stochastic profiles is  $\sum_i \lambda_i V_i$ , and its row is this lemma. The break in uniformity, wherever it appears below, is never here: GSHS forced jumps contribute boundary rows, not  $\mathcal{L}$ -terms, so part of the composite obligation family bypasses linearity entirely and must be handled semantically.  $\square$

**Lemma R78.2** (conformance propagation — the acyclic engine). *Let a system presentation  $S$  have  $n$  components, and work in the composite behaviour space. Write  $\text{conform}_j$  for the event that component  $j$ ’s inputs conform to  $\alpha_j$ ,  $\text{sat}_j$  for delivery of  $\gamma_j$ , and  $\text{ext}$  for external-input conformance. Suppose the wiring squares transport, pointwise:  $\text{ext} \cap \bigcap_{i \in \text{upstream}(j)} \text{sat}_i \subseteq \text{conform}_j$  for every  $j$ . If at a behaviour  $\omega \in \text{ext}$  every component delivers on conformance ( $\omega \in \text{conform}_j \Rightarrow \omega \in \text{sat}_j$ ), then every component delivers at  $\omega$ .*

*Proof.* Strong induction on the causality rank ( $\text{sat\_of\_conform}$ ): if every component of rank  $< N$  delivers, then for  $j$  of rank  $< N+1$  all of  $j$ ’s upstream components have strictly smaller rank (the causality certificate), hence deliver; the wiring square then puts  $\omega \in \text{conform}_j$ , and the per-component implication closes. The causality certificate is precisely what makes the induction well-founded, and its necessity is a two-component counterexample: components in a cycle, each delivering *only if* the other does, satisfy every wiring square while both fail — the implication chain has no base case. That failure is the classical unsoundness of naive circular assume–guarantee [2, 177]; this calculus fences it structurally (cycles unrepresentable here) and readmits feedback only through the guarded small-gain cells of Lemma R78.4, where the well-foundedness device is the  $\theta$ -margin.  $\square$

**Lemma R78.3** (failure decomposition and slack composition). *Under Lemma R78.2’s wiring hypothesis, a composite failure under conforming external inputs localizes at some component that failed while conforming:*

$$\text{ext} \cap \left( \bigcap_j \text{sat}_j \right)^c \subseteq \bigcup_j (\text{conform}_j \cap \text{sat}_j^c),$$

*hence for any measure with per-component conforming-failure bounds  $\mu(\text{conform}_j \cap \text{sat}_j^c) \leq \varepsilon_j$ , the composite fails with mass at most  $\sum_j \varepsilon_j$  — the  $\sum_j (1 - p_j)$  display, at any  $n$ .*

*Proof.* Pick a rank-minimal failing component  $j_m$  (finitely many components; the failing set is nonempty). Every upstream component of  $j_m$  has strictly smaller rank, hence delivers (minimality), so the wiring square puts  $\omega \in \text{conform}_{j_m}$ : a conforming failure ( $\text{fail\_subset}$ ). Monotonicity and finite subadditivity of the measure close ( $\text{fail\_bound}$ ; read at an output spec,  $\text{fail\_bound\_out}$ ). The per-component event  $\text{conform}_j \cap \text{sat}_j^c$  — not bare failure — is forced: component  $j$ ’s certificate priced failure *only under conforming inputs*; charging it for non-conforming failures would demand a guarantee nobody certified, and the rank-minimality argument is exactly what shows the composite never needs one.  $\square$

**Lemma R78.4** (the stochastic-hybrid cells: series, parallel, feedback sum-form). *Over the composite execution law of a GSHS composite, with component certificates entering as nonnegative supermartingales along the (stopped) joint jump skeleton, with barrier values on*

their bad sets and violation-witness inclusions: (i, series) the conditioning tower composes — qualitatively, delivery a.e. on conforming inputs chains through the wiring square; quantitatively, an upstream success mass  $\geq p_1$  and a downstream conditional success probability  $\geq p_2$  — the conditional bound holding almost everywhere on the upstream success event, itself measurable in the conditioning  $\sigma$ -algebra — compose to composite success mass  $\geq p_2 \cdot p_1$  (stronger than the union-bound shape by exactly  $(1 - p_1)(1 - p_2) \geq 0$ ); and the certified pipeline bounds the composite conforming-failure mass by  $\int Y_1(0)/a_1 + \int Y_2(0)/a_2$  through the maximal inequality. (ii, parallel) for nonnegative weights, the weighted sum  $\sum_i \lambda_i V_i$  of supermartingales w.r.t. one (joint) filtration is a supermartingale — the semantic face of the syzygy weights — and the composite avoid bound follows through the same maximal inequality. (iii, feedback, sum-form) for two components in feedback whose certificates satisfy, along the joint skeleton, the between-jumps conditional decrease (pre-jump value plus integrated assumption-minus-supply increment conditionally below the current value) and non-expansive jump accounting (post-jump value a.e. at most pre-jump value), with the integrated membership rows of the small-gain pair at  $\lambda, \theta \geq 0$ : the composite  $W = V_1 + \lambda V_2$  is a nonnegative supermartingale for the joint-skeleton filtration, with the  $\theta$ -margin banked by the compensated process, and the composite avoid bound follows.

*Proof.* (i) is AGCompose.Series: the qualitative form is an a.e. implication chain through the pointwise wiring square (qualitative); the product form integrates the downstream conditional bound over the upstream event — the set-integral of the conditional expectation of the indicator over an upstream-measurable set is the joint mass (tower\_step, tower); the certified form (certified) converts each certificate to a failure bound via the maximal inequality for nonnegative supermartingales (Ville’s inequality, mechanized in the accompanying development) and composes by the failure decomposition. Exogeneity is where acyclicity earns its keep semantically: in a cascade each component’s input process is measurable w.r.t. upstream trajectories, so the component’s certificate argument runs pathwise in its input.

(ii) is AGCompose.Parallel: induction on the finite index set, scalar-multiple and sum closure of supermartingales (supermartingale\_weighted\_sum), then the barrier argument as in (i) (certified, certified\_pair).

(iii) is the file quoted above (SumFormFeedback): folding jump non-expansion into the between-jumps decrease gives the per-component skeleton step; the weighted combination of the two steps cancels the cross-supplies *under conditional expectation* — Lemma R78.1 run pointwise inside the conditional — leaving the  $\theta$ -weighted assumption margin (drift); nonnegativity and the supermartingale property assemble (supermartingale), and the avoid bound is the maximal inequality again (avoid\_bound, with the continuous-time reading conditional on the début-sampling inclusion, debut\_avoid\_bound). Two hypotheses deserve their remark. *The joint skeleton is not a convenience*: in feedback no component’s input is exogenous — the jump chain interleaves both components’ guard hits, each component’s between-jumps argument is valid only between joint jumps, and forced jumps, having no intensity, enter only through the jump-accounting field at those times. *Non-expansion is the sum-form’s load-bearing wall*: only  $\lambda, \theta \geq 0$  and the non-expansive accounting are consumed — no dwell-time or schedule hypothesis appears anywhere in the descent, which is a checkable structural fact about the statements — and the rule-level ranges  $\lambda > 0, \theta \in (0, 1)$  are well-formedness backing the published guard-modulus reading, not soundness needs.  $\square$

**Theorem R78.5** (the  $n$ -ary acyclic assembly). *Let  $S$  be an acyclic system presentation with wiring squares transporting pointwise as in Lemma R78.2. If each component delivers a.e. on conformance, the composite delivers its output spec a.e. on external conformance (QtslCompose.qualitative); and with per-component conforming-failure bounds  $\varepsilon_j$ , the composite output-spec failure mass is at most  $\sum_j \varepsilon_j$  (fail\_bound\_out) — at any  $n$ , by Lemmas R78.2 and R78.3.*

*Proof.* Immediate from the two lemmas: the a.e. form collects the countable family of per-component a.e. implications, applies the pointwise propagation on the common full-measure set, and reads the output spec off the delivered guarantees; the probabilistic form is the failure decomposition plus subadditivity. The per-profile content enters entirely through the *bridge hypotheses* — that the component-certificate events and their bounds exist over the composite law is the per-profile discharge of Lemma R78.4 (GSHS acyclic cells) or its lighter analogues; this theorem is the profile-generic event algebra on top.  $\square$

**Remark R78.6** (the tier split, derived — never a stored tag). The rule node’s tier is a function of the presentation (System.qagTier): *admissible* iff every mechanism’s derived profile admits a syntactically constructible composite certificate — proven equivalent to “every mechanism derives ODE or discrete” (qagTier\_admissible\_iff) — and *core* iff some mechanism is stochastic (qagTier\_core\_iff). On the admissible face the node unpacks: the composite certificate is  $\sum_i \lambda_i V_i$  (Lemma R78.1), transported at the sequent layer by the PROMISE/SYZYGY machinery, and the rule adds no semantic content. On the core face no unpacking exists — the warrant is Lemmas R78.2–R78.4 — which is the taxonomy’s definition of core applied honestly, one rule node either way. Deriving the tier from field-family population (rather than storing a system-level tag) is forced by mixed composites: a sampled-data plant-plus-controller would be ill-typed across its own components under any stored per-system tag.

**Remark R78.7** (what is *not* proven — the open cells, loud). The conditional mark on this rule’s dashboard row warrants exactly the cells above, and the honest remainder is: (a) the deterministic-ODE row’s semantic reading is at statement level (its literature anchor is the certified-lens composition theorem [37]; the classical comparison argument is standard but not yet written in-house); (b) the discrete-kernel row’s assembly is discharged —  $n$ -ary acyclic union-bound, parallel weighted-sum, and sum-form small-gain, end-to-end over the trajectory kernel (the width note’s dictionary names the statements) — with the *sampled-data* row the one still stated, not yet assembled (item (d)); (c) feedback in *gauge-form* — jumps that expand the certificate ( $V(g(x)) \leq cV(x)$ ,  $c > 1$ ), repaid by flow decay under schedule hypotheses — is partial: the gauge-then-sum reduction and the machine-checked max-form piecewise-deterministic instance are landed sorry-free, while the finite-index price of the diffusion term and the admissible-window pinch remain open research — though the reason the diffusion cell was *parked* is retired: within-segment supermartingale decay for gauged certificates is now available at the continuous-time martingale-problem profile (the battery family’s discharge), so what remains is the finite-index accounting and the window pinch themselves, not a missing stochastic calculus; sum-form composites provably do not cover this regime, which is exactly why the core tier exists. Non-expansive-reset models live entirely in Lemma R78.4(iii), so nothing below gates on (c); (d) the mixed sampled-data row is stated, not proven — deterministic throughout, the natural first boundary-row exercise; (e) undirected stochastic wire sharing (common noise) is *fenced*, not missing: stochastic

components sharing a wire need one determining side, and for fenced models no discharge of the joint-filtration hypothesis exists — the lemmas are correctly unusable there; (f) the  $n$ -ary *product-form* (tower) composition is binary only — its  $n$ -ary form needs a filtration chain along the wiring order and is real further work; (g) the judgment sorts the rule concludes into are not yet staged in the certificate schema, so no end-to-end checkable certificate exercises the rule yet. Each bridge hypothesis (the interface layers carrying a component certificate onto the composite law) is a named hypothesis discharged per model class — parameters, never silent assumptions.

*Remark R78.8* (relation to the literature). The acyclic discipline is the composing-specifications tradition [2], with circularity readmitted only under a well-foundedness device [177] — here the  $\theta$ -margin of the small-gain pair, whose Lyapunov formulation descends from the nonlinear small-gain theorems [125, 126]. The parallel cell is compositional dissipativity [184, 239] run at the satisfaction layer — the weighted-sum supermartingale is the stochastic face of the storage-function sum. The wiring square, its slack algebra, and the certified-lens composition law are [37]. What the mechanized grid adds over all of these is the certificate form: exact-arithmetic transport rows plus per-profile semantic lemmas with their hypotheses reified as checkable data, rather than a metatheorem quantified over models nobody replays.

| proof step                                          | formal anchor                                                                                                                                                                                        |
|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lemma R78.1: cancellation, pointwise fold           | admissible_constructs; sequent layer sound_promise, sound_syzygy, sound_smallGain (main package)                                                                                                     |
| Lemma R78.2: rank induction                         | sat_of_conform; System (fields rank, causality)                                                                                                                                                      |
| Lemma R78.3: rank-minimal localization, union bound | fail_subset, fail_bound, fail_bound_out                                                                                                                                                              |
| Lemma R78.4(i): series                              | AGCompose.Series.qualitative, tower_step, tower, fail_bound, certified                                                                                                                               |
| Lemma R78.4(ii): parallel                           | AGCompose.Parallel.supermartingale_weighted_sum, certified, certified_pair                                                                                                                           |
| Lemma R78.4(iii): feedback sum-form                 | SkeletonCertificate (fields dynkin, jump_nonexpansive), SkeletonCertificate.step, SumFormFeedback.drift, supermartingale, avoid_bound, debut_avoid_bound, compensated_supermartingale, margin_budget |
| gauge-form partial (max-form instance)              | sound_GSHS_safety_barrier (MaxFormPDMP)                                                                                                                                                              |
| Theorem R78.5                                       | QtSLCompose.qualitative, fail_bound_out                                                                                                                                                              |
| Remark R78.6: derived split                         | System.qagTier, qagTier_admissible_iff, qagTier_core_iff; ChartShape.profile                                                                                                                         |

## Rule 79. QWIRECERTDISCHARGE (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QWireCertDischarge*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} \{R_\ell, \varphi(s) \wedge \alpha(v(s), a) \vdash \text{Lift}_s \varphi(u(s, a))\}_\ell, \{R_\ell, \varphi(s) \vdash \gamma(v(s))\}_\ell \text{ (mechanism-certification legs; } s \text{ a mechanism state, } a \text{ an admissible input)} \\ \{R_\ell \vdash \alpha_2(w(o_1), a_2) + \kappa(\gamma_1(o_1)) \geq \alpha_1(w^\#(o_1, a_2))\}_\ell \text{ (assumption legs)} \\ \{R_\ell \vdash \gamma_1(o_1) \geq \gamma_2(w(o_1))\}_\ell \text{ (guarantee legs)} \quad \vdash \kappa(0) = 0 \text{ (closed evaluation)} \\ [0, \infty) \vdash \kappa' \geq 0 \text{ (derivative-nonnegativity row)} \quad \vdash \deg \kappa > 0, \text{lead}(\kappa) \geq 0 \text{ (leading-coefficient row)} \end{array}}{\text{QWireCert}(w, (\text{spec}_i) \Rightarrow \text{spec}_{\text{out}}, \kappa)}$$

every leg family indexes over its posted proof leaves  $\ell$ , and each leaf binds its own region  $R_\ell$  as the antecedent of its sequent; the three arithmetic rows certify  $\kappa \in \mathcal{K}_\infty^0$  (zero at zero, monotone on the ray, unbounded).

$\mathcal{K}_\infty^0$  is the class under which slacks add and lenses compose; the premises are pointwise implications over polynomial data, one inherited BL sequent per posted leaf, the leaf's antecedent carrying its region — no region cover is stored on the node, and the obligation that the posted regions jointly cover the consuming composition's interface domain lives one row up, at the consumption site. The two Capucci-Myers condition families are the mechanism-certification legs (né chart certification) and the wiring-lens legs. No new checker machinery: the rule is leaf assembly, and its semantic content lives one row up in the composition rule.

QWIRECERTDISCHARGE is the wiring-certificate judgment's own discharge: it assembles a QWireCert — the license QWIRECOMPOSE consumes — entirely from inherited BL sequent leaves. Four condition families are checked, each indexing over its posted proof leaves  $\ell$ , and each leaf binds its own region  $R_\ell$  as the antecedent of its own sequent. The two *mechanism-certification legs* certify the component's contract at its mechanism:  $R_\ell, \varphi(s) \wedge \alpha(v(s), a) \vdash \text{Lift}_s \varphi(u(s, a))$  — the certified invariant, together with the assumption evaluated at the mechanism's readout and an admissible input, is preserved by the update — and  $R_\ell, \varphi(s) \vdash \gamma(v(s))$  — the invariant delivers the guarantee at the readout — where  $s$  ranges over the mechanism's state carrier and  $a$  over its admissible input carrier. The two *lens legs* certify the wiring square: the *assumption leg* (delivered upstream guarantees, degraded by at most the slack  $\kappa$ , imply the downstream assumption through the wiring map) and the *guarantee leg* (the upstream guarantee dominates the composite's output reading). For polynomial data all four are pointwise implications — ordinary sequents the base calculus already checks. Three further arithmetic rows certify the slack's class membership,  $\kappa \in \mathcal{K}_\infty^0$ : the constant coefficient is zero (closed evaluation), the derivative is nonnegative on the ray (a derivative-nonnegativity row), and the leading coefficient is positive at positive degree.

What the cited theorems warrant — and the scope of this entry's conditional check: that the three row families genuinely certify class membership (a polynomial passing them is zero at zero, monotone on  $[0, \infty)$ , and unbounded), and that the class supports the composition rule's slack algebra — slacks are closed under pointwise sum, closed under composition, the two-slack contract chain composes to the summed slack, and the guarantee legs chain along a cascade. That algebra is the quantitative content of “compose by lens composition and by addition of the slacks”.

Each membership row is load-bearing for the algebra. Drop zero-at-zero and a constant slack certifies a square that degrades even exact guarantee delivery — and the nonnegativity that anchors the additive algebra (a class slack is nonnegative on the ray *because* it is zero at zero and monotone) fails with it. Drop monotonicity and composition of slacks fails outright: the outer slack of a two-lens cascade need no longer bound the chained degradation, since a non-monotone  $\kappa_1$  can reward a worse input. Drop unboundedness and bounded candidates enter ( $\kappa \equiv 0$  passes the other two rows): the discharge then certifies membership in a class the slack does not belong to, and the published reading of  $\kappa$  as a reusable comparison-function-style gain is lost. The proof section walks each closure with the field it consumes.

**notes** Core-as-glue: the rule is leaf assembly, and the semantic content of what a wiring certificate *buys* lives one row up, in QWIRECOMPOSE's semantic reading. The non-strict monotone reading of the class follows the catalogue's own row roster (a derivative-nonnegativity row certifies monotone, not strictly increasing); the zero slack is thereby not a member — a slack-free square is a square with the  $\kappa$ -term absent, not with  $\kappa = 0$ . The judgment sort itself is baked in the extension theory; the mechanized warrant was proven ahead of the bake, and the square leaves are landed main-package sequent machinery, cited rather than re-bound here.

### Lean statement

**The slack class, bound by display.** `IsClassK0Inf` is the catalogue's  $\mathcal{K}_\infty^0$ : zero at zero, monotone on the nonnegative ray, unbounded at infinity. Its closure theorems (`add` — slacks add; `comp` — lenses compose) are proven in the same module inside the class namespace; `polynomial_discharge` maps the three checked row families onto the three fields, and `slack_triangle/guarantee_leg_comp` are the quantitative and guarantee-side composition faces.

BernsteinLogicAux.QtslCompose.WireCertDischarge . **polynomial\_discharge**

statement bound



```

/-- **The polynomial discharge** – the rule note's three membership rows, each
mapped to the class field it certifies:
1. zero-at-zero = closed evaluation ( $\text{'p.coeff } 0 = 0$ ', checked by evaluating);
2. monotone = the derivative-nonnegativity row ( $\text{'p' } \geq 0$  on the ray);
3. unbounded = positive leading coefficient (with positive degree).
A polynomial passing the three BL row families is genuinely a  $\mathcal{H}^0$  slack. -/
theorem polynomial_discharge (p : Polynomial  $\mathbb{R}$ )
  (h0 : p.coeff 0 = 0)
  (hderiv :  $\forall x : \mathbb{R}, 0 \leq x \rightarrow 0 \leq \text{p.derivative.eval } x$ )
  (hdeg :  $0 < \text{p.degree}$ ) (hlead :  $0 \leq \text{p.leadingCoeff}$ ) :
  IsClassK0Inf (fun x => p.eval x)

```

BernsteinLogicAux.QtslCompose.WireCertDischarge . **slack\_triangle**

statement bound

```

/-- **The additive slack triangle** (the quantitative face of "compose by
lens composition and by addition of the slacks"): two contract inequalities
with slack compose to one with the summed slack. Stated pointwise over an
arbitrary interface carrier – the per-region reading is  $\text{'X'}$  = the region. -/
theorem slack_triangle {X : Type*} {f g h : X  $\rightarrow \mathbb{R}$ } {c1 c2 :  $\mathbb{R}$ }
  (h1 :  $\forall x, f x \leq g x + c1$ ) (h2 :  $\forall x, g x \leq h x + c2$ ) :
   $\forall x, f x \leq h x + (c1 + c2)$ 

```

BernsteinLogicAux.QtslCompose.WireCertDischarge . **guarantee\_leg\_comp**

statement bound

```

/-- The guarantee-side lens leg composes (the  $\kappa$ -free leg of the CM square:
 $\gamma_1 \geq \gamma_2 \circ w$  chains along a cascade). -/
theorem guarantee_leg_comp {A B C : Type*} {w1 : A  $\rightarrow$  B} {w2 : B  $\rightarrow$  C}
  { $\gamma_1 : A \rightarrow \mathbb{R}$ } { $\gamma_2 : B \rightarrow \mathbb{R}$ } { $\gamma_3 : C \rightarrow \mathbb{R}$ }
  (h1 :  $\forall a, \gamma_2 (w1 a) \leq \gamma_1 a$ ) (h2 :  $\forall b, \gamma_3 (w2 b) \leq \gamma_2 b$ ) :
   $\forall a, \gamma_3 (w2 (w1 a)) \leq \gamma_1 a$ 

```

BernsteinLogicAux.QtslCompose.WireCertDischarge . **nonneg**

statement bound

```

/-- A  $\mathcal{H}^0$  slack is nonnegative on the nonnegative ray. -/
theorem nonneg (h : IsClassK0Inf  $\kappa$ ) {x :  $\mathbb{R}$ } (hx :  $0 \leq x$ ) :  $0 \leq \kappa x$ 

```

BernsteinLogicAux.QtslCompose.WireCertDischarge . **add**

statement bound

```

/-- **Slacks add** ("by addition of the slacks"): the class is closed under
pointwise sum. -/
theorem add (h1 : IsClassK0Inf  $\kappa_1$ ) (h2 : IsClassK0Inf  $\kappa_2$ ) :
  IsClassK0Inf (fun x =>  $\kappa_1 x + \kappa_2 x$ )

```

BernsteinLogicAux.QtslCompose.WireCertDischarge . **comp**

statement bound

```

/-- **Lenses compose** : the class is closed under composition (the slack of a
two-lens cascade). -/
theorem comp (h1 : IsClassK0Inf  $\kappa_1$ ) (h2 : IsClassK0Inf  $\kappa_2$ ) :
  IsClassK0Inf ( $\kappa_1 \circ \kappa_2$ )

```

BernsteinLogicAux.QtslCompose.WireCertDischarge . **IsClassK0Inf**

definition bound

```

/-- Class  $\mathcal{H}^0$  membership (the  $\text{'QWireCert'}$  slack class): zero at zero,
monotone on the nonnegative ray, unbounded at infinity. -/
structure IsClassK0Inf ( $\kappa : \mathbb{R} \rightarrow \mathbb{R}$ ) : Prop where
  zero_at_zero :  $\kappa 0 = 0$ 
  mono : MonotoneOn  $\kappa$  (Ici 0)
  unbounded : Tendsto  $\kappa$  atTop atTop

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber

until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed conclusion  $\text{QWireCert}(w, (\text{spec}_i) \Rightarrow \text{spec}_{\text{out}}, \kappa)$  is concluded by *no* bound statement — deliberately: the judgment sort lives theory-side (the note), the square leaves are landed sequent machinery cited as witness prose, and what this bound set certifies is the *semantic warrant behind the arithmetic rows* plus the class algebra the rule's composition claims lean on. The class itself is bound by display:  $\text{IsClassK0Inf} = \text{zero-at-zero} + \text{Monotone0n } \kappa \text{ (Ici } 0) + \text{Tendsto } \kappa \text{ atTop atTop} \text{ — } \mathcal{K}_{\infty}^0$  as the catalogue reads it, non-strict monotone (the note's ruling, visible in the field type).

- the three membership rows  $\mapsto$  `polynomial_discharge`, one hypothesis per row per field: closed evaluation  $\mapsto$  `p.coeff 0 = 0  $\Rightarrow$  zero_at_zero`; derivative-nonnegativity on the ray  $\mapsto$  `hderiv  $\Rightarrow$  mono` (through the mean-value machinery `monotone0n_of_deriv_nonneg`); leading-coefficient row  $\mapsto$  `0 < p.degree + 0  $\leq$  p.leadingCoeff  $\Rightarrow$  unbounded`. The displayed row asks only `lead( $\kappa$ )  $\geq$  0` and so does the statement — nonstrict suffices because a degree-witnessing leading coefficient is nonzero by definition; the commentary's “positive” is the derived fact, not an extra demand.
- “slacks add, lenses compose”  $\mapsto$  `slack_triangle` (two contract inequalities chain to the summed slack, stated pointwise over an arbitrary carrier — the per-region reading is the carrier choice) and `guarantee_leg_comp` (the  $\kappa$ -free guarantee leg chains along a cascade). The class *closure* lemmas `IsClassK0Inf.add` and `IsClassK0Inf.comp` (with the nonnegativity anchor `IsClassK0Inf.nonneg`) are bound with the entry and carry the commentary's closure claims (sum, composition), alongside the triangle and the guarantee leg.
- polynomiality of  $\kappa \mapsto$  `p : Polynomial  $\mathbb{R}$`  in the discharge theorem only; the class and its algebra are stated for arbitrary  $\mathbb{R} \rightarrow \mathbb{R}$  functions — wider than the display's polynomial slacks, safe direction.

Width note: the four leaf families of the display map onto the baked node's two proof lists — the mechanism-certification legs (both shapes) ride `QWireCertDischarge/mech_rows`, and the assumption and guarantee lens legs ride `QWireCertDischarge/lens_rows` — with every leaf's own  $R_{\ell}$  as its sequent's antecedent. In the bound set the lens legs appear only through their compositional shadows (`slack_triangle`, `guarantee_leg_comp`); the per-leaf discharge of all four families is inherited BL sequents, main package, outside this page's bindings — the package-boundary rule, stated in the module header.

## Lean verifier receipts

module BernsteinLogicAux.QtslCompose.WireCertDischarge; bound build cec5cc1749e1... (bound from receipt 90dcbae3ace744d7, 2026-07-19T18:24:58Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 7aeb2249883cf96c | 2026-07-19T03:38:57Z | build+decorrelated                 | current |
| 90dcbae3ace744d7 | 2026-07-19T18:24:58Z | build+decorrelated                 | current |
| 164a0a543f6637cb | 2026-07-20T01:22:11Z | build+decorrelated                 | current |
| 100eb93aa5f473af | 2026-07-20T20:47:01Z | build+decorrelated                 | current |
| 1f74787842459411 | 2026-07-20T20:58:34Z | build+replay+hardened+decorrelated | current |
| 807e980b9db27d7a | 2026-07-21T00:04:07Z | build+decorrelated                 | current |
| efa933bf3665661d | 2026-07-21T05:39:20Z | build+replay+hardened+decorrelated | current |
| 64dad6f82e052f30 | 2026-07-22T18:27:19Z | build+decorrelated                 | current |
| 2a1434c39627845c | 2026-07-31T22:00:17Z | build+decorrelated                 | current |
| d12854bca5ce90c3 | 2026-07-31T22:29:11Z | build+decorrelated                 | current |
| b0d9534dd5d7039b | 2026-08-02T23:31:22Z | build+decorrelated                 | current |
| bf1f56225eddc2cf | 2026-08-03T16:27:26Z | build+decorrelated                 | current |
| e839dbc9244ca3d2 | 2026-08-03T22:46:10Z | build+decorrelated                 | current |

gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement



**NO HOL STATEMENT YET**

The satisfaction-judgment extension postdates the HOL4 port entirely: the ported cone is the base calculus's check-function surface (theory `bern_catalogue`), and no extension-theory definition exists in the HOL development — in particular nothing shaped like a wiring-certificate discharge. The rule's sequent *leaves* would already be expressible in the ported vocabulary, but the judgment they assemble would not; the box waits on a HOL port of the satisfaction stratum.

**HOL verifier receipts****NONE**

no HOL verification receipt exists for this rule.

**Proof**

Genre: core rule (tier 1), core-as-glue — the rule assembles inherited sequent leaves into the wiring-certificate judgment, and its own mathematical content is deliberately thin: that the checked arithmetic rows genuinely certify slack-class membership, and that the class supports the composition algebra the composition rule recomputes. The section proves exactly that and marks the assembly gap loudly.

**Setting and claim.** A wiring certificate  $\text{QWireCert}(w, (\text{spec}_i \Rightarrow \text{spec}_{\text{out}}, \kappa)$  carries two per-region families of pointwise implications — the lens square's assumption  $\text{leg } \alpha_2(w(o_1), a_2) + \kappa(\gamma_1(o_1)) \geq \alpha_1(w^\sharp(o_1, a_2))$  and guarantee  $\text{leg } \gamma_1(o_1) \geq \gamma_2(w(o_1))$  [37] — which for polynomial data are ordinary sequents the base calculus checks, plus a slack  $\kappa \in \mathcal{K}_\infty^0$ : zero at zero, monotone on  $[0, \infty)$ , unbounded at infinity (the `IsClassK0Inf` structure quoted above; the non-strict monotone reading is the catalogued one). The rule's own obligations, proven here: the three checked rows certify membership for polynomial slacks (Lemma R79.1), and the class supports “compose by lens composition and by addition of the slacks” (Lemma R79.2).

**Lemma R79.1** (the polynomial discharge). *Let  $p$  be a real polynomial with (R1) constant coefficient zero, (R2) derivative nonnegative on  $[0, \infty)$ , and (R3) positive degree with nonnegative leading coefficient. Then  $x \mapsto p(x)$  is in  $\mathcal{K}_\infty^0$ .*

*Proof.* (`polynomial_discharge`.) Zero at zero:  $p(0)$  is the constant coefficient, so (R1) is the field it certifies — and note the checker's form of (R1) is one closed exact evaluation, the ground-fact pattern. Monotone:  $[0, \infty)$  is convex and  $p$  is everywhere continuous and differentiable, so nonnegativity of  $p'$  on the interior gives monotonicity on the ray (the mean-value inequality); (R2) is a single-variable nonnegativity row of exactly the calculus's cone-checkable shape. Unbounded: a polynomial of positive degree with nonnegative leading coefficient tends to  $+\infty$  — (R3) is one syntactic degree check plus one sign row. Each row maps onto exactly one class field; none is spare.  $\square$

**Lemma R79.2** (the slack algebra).  *$\mathcal{K}_\infty^0$  is closed under pointwise sum and under composition; two contract inequalities with constant slacks chain to one with the summed slack ( $f \leq g + c_1$  and  $g \leq h + c_2$  pointwise give  $f \leq h + (c_1 + c_2)$ ); and guarantee legs chain along a cascade ( $\gamma_2 \circ w_1 \leq \gamma_1$  and  $\gamma_3 \circ w_2 \leq \gamma_2$  give  $\gamma_3 \circ w_2 \circ w_1 \leq \gamma_1$ ).*

*Proof.* Sum (`IsClassK0Inf.add`): zeros add; monotonicity is termwise; for unboundedness, eventually the second summand is non-negative (it tends to infinity), so the sum eventually dominates the first, which tends to infinity. Composition (`IsClassK0Inf.comp`):  $\kappa_1(\kappa_2(0)) = \kappa_1(0) = 0$ ; monotone-composed-monotone is monotone *provided the inner map lands in the outer's domain of monotonicity* — which is where nonnegativity of a class slack on the ray (zero at zero plus monotone) is consumed, not decoration; unbounded-composed-unbounded is unbounded. The triangle (`slack_triangle`) and the guarantee chain (`guarantee_leg_comp`) are one-line pointwise transitivity, stated over an arbitrary interface carrier so the per-region reading is instantiation, not generalization. In application the “constants” are the slack values at the delivered guarantees — the triangle is applied pointwise per region — while composing the slack *functions* of a cascade is the closure pair's job.  $\square$

**Remark R79.3** (why each membership row is load-bearing). Drop (R1) and  $\kappa(x) = x + 1$  enters the class candidates: a wiring square then certifies with degradation even on *exact* upstream delivery ( $\kappa(0) = 1$ ); and the anchor of the algebra — nonnegativity of a class slack on the ray, proved from zero-at-zero plus monotonicity — fails too, witness  $\kappa(x) = x - 1$ . Drop (R2) and  $\kappa(x) = x^2 - x$  enters (zero at zero, positive degree, positive lead): negative on  $(0, 1)$ , so the nonnegativity anchor fails, and non-monotone, so composition closure fails at the monotonicity step (Lemma R79.2's proof consumes it twice: once as the outer map's field, once through inner-map nonnegativity), and the quantitative reading — a weaker delivered guarantee never *shrinks* the licensed degradation — inverts. Drop (R3) and bounded slacks enter ( $\kappa \equiv 0$  passes (R1)–(R2) alone); the discharge then no longer certifies unboundedness, and  $\kappa$  loses its comparison-function-style gain reading. The zero polynomial is excluded by (R3) deliberately: a slack-free square is a square with the  $\kappa$ -term *absent*, never one with  $\kappa = 0$ , so the additive algebra needs no zero element.

**Remark R79.4** (the assembly gap, loud). What this section does *not* contain: the square leaves themselves (landed sequent machinery of the base calculus, cited by the certificate — pointwise polynomial implications per region, checked by the cone machinery), and the end-to-end assembly of the judgment object, which awaits the staging of the judgment sort in the certificate schema. The dashboard's conditional mark scopes the warrant to exactly Lemmas R79.1 and R79.2: the arithmetic rows mean what they claim, and the class composes. The semantic content of what a wiring certificate *buys* a composite is deliberately not here — it is the composition rule's semantic reading, one section over.

| proof step                             | formal anchor                                             |
|----------------------------------------|-----------------------------------------------------------|
| class definition (bound by display)    | IsClassK0Inf (fields zero_at_zero, mono, unbounded)       |
| Lemma R79.1                            | polynomial_discharge                                      |
| Lemma R79.2: sum, composition          | IsClassK0Inf.add, IsClassK0Inf.comp; nonnegativity anchor |
| Lemma R79.2: triangle, guarantee chain | IsClassK0Inf.nonneg                                       |
| square legs (cited, not re-proved)     | slack_triangle, guarantee_leg_comp                        |
|                                        | inherited BERNCONe-family sequents (main package)         |

## Rule 80. QWIRECOMPOSEMARKS (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QWireComposeMarks*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\{\Sigma_i \models (I_i, A_i, E_i)\}_{i=1}^n \quad \text{where } A_i \equiv \bigwedge_{j \in \text{in}(i)} \text{ev}(g_j) \text{ PathPred}, \quad E_i \equiv g_i \\ \{\ell_i \text{ latency rows}\}_{i=1}^n \quad (\text{each premise ships its } \text{QWireCert}(w, (\text{spec}_i)_i \Rightarrow \text{spec}_{\text{out}}, -))$$

MarkFlow( $w, (t_k)_k, (P_k)_{k \in \text{cyc}_0(w)}$ ) (the schedule walk, two columns: stage rows — difference-constraint feasibility at the labels; carve rows — W0 (composite properness — the existing family; the

SysWire( $u$

the  $\exists$ -face sibling of QWIRECOMPOSE: same wiring premise vocabulary, opposite polarity economics. **No causality certificate, no ranks, no guard column** — a dip under a mark survives every extension, so a cyclic wiring conspires *for* the conclusion, and the well-foundedness kit appears precisely in the rule whose conclusion face is ForallSpec, never here. In its place, one composite-level ingredient in two columns: the MarkFlow walk — the  $\exists$ -dual of the causality walk, same arithmetic species (difference constraints over the declaration closure, bounded recompute), opposite verdict role: the  $\forall$ -walk demands strict increase along every wire, refusing all cycles; this walk demands *consistency*, admitting every cycle whose schedule closes and refusing time-travel. Its stage arithmetic forces every dependency cycle to total lag zero with one shared label — so the cycles the rule admits are exactly the instantaneous self-supporting loops, and there the walk's carve column takes over: producer-supplied *carves* (one predicate per cycle mark, presented as a scalar-codomain gate hypernet cited by digest — its sign-fiber the carve; a plated aggregate gate for token-population members; never a trace, and never a bare point of a continuous state space) checked by containment arithmetic, carved-delivery citations, and one diagonal certificate per cycle asserting the instant loop closes jointly *inside* the carve product — the discriminant carve-invariance alone provably misses. Exact value pins survive as the degenerate diagonal species, honest exactly where states are finitely presentable. Off the cycles no carve is needed: the  $\forall\exists$  premises chain deliveries in topological order, producer-side — and the premises themselves are the maximal-carve rows, so the whole column is the rule's own premise form run at tighter predicates. The conclusion is root satisfaction with the demanded marks in the record's ExistsSpec list — one turnstile, no subscript; the face lives in the argument. The joint witness (record, joint law, PDE solution) is constructed once, inside the soundness metatheorem, and never appears on the wire. Guarded premises are not exempt: the composition consumer eliminates each component guard at consumption — initial membership discharged before the premise is read as a guarantee — exactly as at QWIRECOMPOSE (the mirrored elimination row).

QWIRECOMPOSEMARKS is the satisfaction stratum's composition rule for cyclic wiring at the liveness face — the polarity complement of QWIRECOMPOSE, whose causality certificate makes a directed cycle unrepresentable. The polarity fact underneath: a safety violation and a mark realization are both finite-stage events with opposite persistence. A violation poisons every extension, so circular reasoning about guarantees-as-floors needs a well-foundedness device; a dip under a mark survives every extension, so a cycle's self-support is productive — the two-relay loop that refutes the unguarded circular safety rule is, read at the mark face, exactly the witness that realizes “both outputs hit 1.” Accordingly this rule carries *no causality certificate, no ranks, no guards*: from per-component conditional-delivery judgments (every admitted resolution whose interface conforms to the assumed in-neighbour marks admits some resolution through it realizing the own mark — with a properness presupposition on the conforming sub-body), their latency rows, the wiring plumbing, one mark-entailment sequent, the MarkFlow walk, and the properness family, it concludes satisfaction of the wired composite at an output spec whose demanded content is its record's ExistsSpec list.

The one composite-level ingredient is the walk, and it runs in two columns of the same arithmetic species — bounded recompute over the declaration closure, never chase derivation, no well-foundedness anywhere. The *stage column* is the causality walk's  $\exists$ -dual: producer-supplied stage labels per mark, checked against the difference-constraint system the latency rows induce — where the  $\forall$ -walk demands strict increase along every wire (refusing all cycles), this walk demands consistency, admitting every cycle whose schedule closes and refusing time-travel (a relay against an initialized unit delay demands  $t_x = t_y$  and  $t_y = t_x + 1$  at once). Its own arithmetic forces every admitted cycle to total lag zero and one shared label — so the admitted cycles are precisely the instantaneous self-supporting loops. There the *carve column* takes over: producer-supplied carves — one *predicate* per cycle mark, presented as a scalar-codomain gate hypernet cited by digest, its sign-fiber the carve; a plated aggregate gate where the members are token-population systems; never a trace, and never a bare point of a continuous state space — checked by three row families: containment (the carve refines its mark), carved-delivery citations of the very premises the rule already holds (instantiated at the carve events), and one *diagonal* certificate per cycle, asserting the instant loop closes jointly inside the carve product. The diagonal is the load-bearing novelty: carve-invariance alone provably misses it, and exact value pins survive as its degenerate species — honest exactly where states are finitely presentable, which is the reason the column generalized. The premises themselves are the maximal-carve rows (machine-checked as an equivalence), so the whole column is the rule's own premise form run at tighter predicates — a dial from the sketch's judgments down to points. The proof section derives both columns from countermodels:

delete the stage column and the relay-delay loop concludes falsely; delete the diagonal and the swap loop with a swap-closed mark — the relay loop's evil twin, distinguishable by delivered content alone — concludes falsely while every premise holds, *including carve-invariance at every carve*. Both refutations, the refuse/admit pairs, the discriminant pair, and the subsumption of pins by carves are machine-checked.

The soundness warrant is a metatheorem whose joint object — record, joint law, or solution, per profile — is constructed once in the proof and never appears on the wire. The mechanized instance is the discrete-relational tier: the assembly engine in carved form (carve rows + schedule + seriality + diagonal closure produce a joint run realizing every mark at its label, by stage recursion — the walk's rows align every in-neighbour of a marked component at its own label, where the carve-at-label invariant holds, so the carved-delivery rows discharge the marked constraints; the pinned engine is re-derived through it at singleton carves), and the chaining bridge at both ends of the dial (on acyclic-instant networks the conditional-delivery premises alone conclude — via derived points, and equally via the maximal carve with no diagonal presented — so the carve column enters the wire only where a cycle genuinely closed). The diagonal's per-species discharge splits the labor the way the stochastic tier already does: the checker verifies finite species rows (a witness point, a bounded enumeration, Lipschitz/gain rows with cycle product below one, convexity or order-interval flags), and the fixpoint theorem that converts them into joint solvability — exhaustion, Banach, Kakutani, Tarski — lives in the metatheorem, per profile, proved once. Properness earns a tier note: the liveness construction consumes *stepwise* properness (the composite stays live at every stage, given the schedule's pins), because bare nonemptiness of the behaviour set is witnessed by runs the construction cannot use — blocking components and pin-sensitive instant subsystems are the countermodels, named in the proof section.

Two fences are part of the statement. Marks are single-dip graded-reach demands — finitely realizable, extension-persistent; recurrence-shaped liveness (Streett-class acceptance around cycles) stays outside both cyclic variants. And the rule deliberately trades completeness for finite presentation: a component delivering only on its favourite conforming input can witness a joint behaviour that no conditional-delivery premise certifies — certifying such composites without presenting behaviours is the strategy form, future work.

**notes** The classical anchors sit on the safety side: circular assume-guarantee is unsound for safety without a well-foundedness device [2], and the standard repairs induct over time [177]. This rule is the observation that the liveness-mark fragment of the specification language needs no such device at all — greatest-fixpoint territory, where the walk replaces the ranking: schedules must close around cycles (and deliveries must agree around the instantaneous ones — agreement certified over carved predicates, with exact values as the degenerate species) rather than cycles being forbidden to close. The certificate carries judgments, stage labels, gate-hypernet digests with their containment, delivery, and diagonal rows, and one sequent; nothing behaviour-sized. Mechanization posture, stated loudly: the discrete-relational tier is proven end-to-end in carved form with the pinned form subsumed (the Lean cells below, sorry-free, standard axiom closures); stochastic and PDE instances of the metatheorem, the per-species diagonal devices at continuous tiers (Banach for the contraction species, Kakutani for the convex, Tarski for the order-theoretic), the history-dependent mechanism tier, and the general difference-constraint walk are open, and the schema realization (an additive sibling node kind beside the acyclic rule's, carrying premise citations, latency rows, the two-column walk as a side condition, and the reused properness family) is staged for the extension theory's next revision, which is why this entry's dashboard row shows a catalogued rule ahead of its wire format.

## Lean statement

**The discrete-relational tier, bound whole.** The engine, the chaining bridge, and the battery — the hypotheses-necessity refutations are first-class bound statements, not test scaffolding: the rule's two walk columns each carry their forcing countermodel in the same module that proves the repaired kit sound.

BernsteinLogicAux.QtslCompose.AGComposeMarks . marks\_of\_pinned\_delivery

statement bound

```
/-- **The assembly engine** (the soundness metatheorem's discrete-relational
instance, pinned form): pin rows + schedule walk + seriality + schedule-relative
instant closure produce a joint run realizing every mark at its label. The
joint behaviour exists only under this conclusion's binder. -/
theorem marks_of_pinned_delivery (hser : Serial N) (hfeas : MarkFlowFeasible N t)
  (hpin : PinnedDelivery N mark a) (hclose : InstantClosure N a t) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i
```

BernsteinLogicAux.QtslCompose.AGComposeMarks . pins\_of\_conditional\_delivery

statement bound

```
/-- **Chaining** on an acyclic network (any topological order of the
dependency graph), the ∀ conditional-delivery premises alone produce a pin —
each component's delivered value is chosen against its in-neighbours' already-
chosen deliveries, in topological order. This is why the premise-shaped rule
needs no value column on the wire in the acyclic case: the pin is derivable,
producer-side, from the premises it already cites. -/
theorem pins_of_conditional_delivery (hloc : DepLocal N) (ord : Fin n → ℕ)
  (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (hdel : ∀ i, ConditionalDelivery N mark i) :
  ∃ a : Fin n → V, PinnedDelivery N mark a
```

BernsteinLogicAux.QtslCompose.AGComposeMarks . instantClosure\_of\_acyclic

statement bound

```

/-- Acyclicity dissolves the joint instant solve: with a topological order,
seriality alone closes every instant, pins included – coordinate by
coordinate, in order. (Holds for ANY pin vector `a`; the pinned coordinates'
own admission is the pointed rows' business, not this closure's.) -/
theorem instantClosure_of_acyclic (hser : Serial N) (hloc : DepLocal N)
  (ord : Fin n → ℕ) (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (a : Fin n → V) (t : Fin n → ℕ) :
  InstantClosure N a t

```

BernsteinLogicAux.QtslCompose.AGComposeMarks . marks\_of\_conditional\_delivery\_acyclic

statement bound

```

/-- **The premise-shaped rule, acyclic instance**: on a network with a
topological order, the ∀ conditional-delivery premises + the schedule walk +
seriality conclude – no pin appears among the hypotheses (it is derived).
This is the discrete face of the design's claim that the value column is
forced only by instant cycles. -/
theorem marks_of_conditional_delivery_acyclic (hser : Serial N) (hloc : DepLocal N)
  (ord : Fin n → ℕ) (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (hfeas : MarkFlowFeasible N t) (hdel : ∀ i, ConditionalDelivery N mark i) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i

```

BernsteinLogicAux.QtslCompose.AGComposeMarks . z1\_walk\_infeasible

statement bound

```

/-- The schedule walk REFUSES the relay-delay loop: its rows demand
`t 0 = t 1 ∧ t 1 = t 0 + 1`. -/
theorem z1_walk_infeasible : ¬ ∃ t : Fin 2 → ℕ, MarkFlowFeasible z1Net t

```

BernsteinLogicAux.QtslCompose.AGComposeMarks . allOnes\_walk\_feasible

statement bound

```

/-- The schedule walk ADMITS the all-ones loop at the shared label `0`. -/
theorem allOnes_walk_feasible : MarkFlowFeasible allOnesNet ![0, 0]

```

BernsteinLogicAux.QtslCompose.AGComposeMarks . z1\_refutes\_walkless

statement bound

```

/-- **M1 (the walk deleted): the pin-rowed rule WITHOUT the schedule walk is
unsound** – the relay-delay loop satisfies seriality, the pin rows, and the
instant closure, yet realizes no mark on any run at any stage. (The design
memo's forcing argument for MarkFlow, machine-checked; `z1_walk_infeasible`
shows the walk indeed refuses this instance.) -/
theorem z1_refutes_walkless :
  ¬ (∀ (m : ℕ) (W : Type) ( _ : Nonempty W) (N : MarkNet m W)
    (mark : Fin m → Set W) (a : Fin m → W) (t : Fin m → ℕ),
    Serial N → PinnedDelivery N mark a → InstantClosure N a t →
    ∃ ω : ℕ → Fin m → W, JointRun N ω ∧ ∀ i, ∃ k, ω k i ∈ mark i)

```

BernsteinLogicAux.QtslCompose.AGComposeMarks . z5\_refutes\_pinless

statement bound

```

/-- **M2 (the pins deleted): the premise-shaped rule without a value column is
unsound** – conditional delivery (∀) at every component, dep-locality,
a feasible schedule, unpinned joint instant closure, AND a nonempty composite
all hold for the swap loop with swap-closed marks, yet no joint run realizes
any mark: each component maps the mark set into the mark set (the premises
are true), but the loop's instant fixpoint lands outside it. The one
composite-level ingredient the stage arithmetic cannot see is the VALUE
consistency – the pin rows. -/
theorem z5_refutes_pinless :
  ¬ (∀ (m : ℕ) (W : Type) ( _ : Nonempty W) (N : MarkNet m W)
    (mark : Fin m → Set W) (t : Fin m → ℕ),
    Serial N → DepLocal N → (∀ i, ConditionalDelivery N mark i) →
    MarkFlowFeasible N t → W0Serial N → (∃ ω, JointRun N ω) →
    ∃ ω : ℕ → Fin m → W, JointRun N ω ∧ ∀ i, ∃ k, ω k i ∈ mark i)

```

BernsteinLogicAux.QtslCompose.AGComposeMarks . z5\_no\_pin

statement bound

```

/-- **The repair refuses Z5**: no pin exists – the pin rows demand a value
assignment with `a 0 = swap3 (a 1)` and `a 1 = a 0` inside the marks, and the

```

```

swap has no fixpoint there. (With `z5_refutes_pinless`: the pin rows are
exactly the ingredient that separates the swap loop from the all-ones loop,
which `allOnes_marks_realized` below certifies through the same engine.) -/
theorem z5_no_pin : ¬ ∃ a : Fin 2 → Fin 3, PinnedDelivery z5Net z5Mark a

```

BernsteinLogicAux.QtslCompose.AGComposeMarks . **allOnes\_marks\_realized**

statement bound

```

/-- **Positive control**: the assembly engine's full hypothesis kit is
satisfiable on the all-ones relay loop and concludes – both marks realized at
the shared label `0` (the design's Role-2 example, end-to-end: self-support
productive under the  $\exists$ -face). -/
theorem allOnes_marks_realized :
  ∃ ω : ℕ → Fin 2 → Bool, JointRun allOnesNet ω ∧
  ∀ i, ω (![0, 0] : Fin 2 → ℕ) i) i ∈ boolMark i

```

**The carve generalization (r5), bound whole.** Pins as predicates: the carved engine at general lags, the subsumption of value pins as singleton carves (degeneracy as a theorem), the dial identity (the rule's  $\forall\exists$  premise IS the maximal-carve row), the acyclic bridge at the maximal carve, the discriminant pair (carve-invariance holds on the swap loop; no carve inside the marks closes its diagonal), the carve-level mutants, and the fat-carve positive control.

BernsteinLogicAux.QtslCompose.AGComposeCarves . **marks\_of\_carved\_delivery**

statement bound

```

/-- **The generalized assembly engine** (C1): carve rows + schedule walk +
seriality + diagonal closure produce a joint run realizing every mark at its
label. No hypothesis names a value; the joint behaviour – and every delivered
point – exists only under this conclusion's binder. -/
theorem marks_of_carved_delivery (hser : Serial N) (hfeas : MarkFlowFeasible N t)
  (hpin : CarvedDelivery N mark P) (hclose : CarveClosure N P t) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i

```

BernsteinLogicAux.QtslCompose.AGComposeCarves . **carvedDelivery\_of\_pinned**

statement bound

```

/-- The r4 pin rows are the singleton instance of the carve rows. -/
theorem carvedDelivery_of_pinned (hpin : PinnedDelivery N mark a) :
  CarvedDelivery N mark (fun i => {a i})

```

BernsteinLogicAux.QtslCompose.AGComposeCarves . **carveClosure\_of\_pinned**

statement bound

```

/-- The r4 closure discharges the singleton diagonal. The walk is
load-bearing: every in-neighbour of a pinned instant coordinate is itself
labeled `k` (zero-lag row), so it is either an instant coordinate held at its
pin by the closure or a pipelined coordinate held at its pin by conformance –
and then the pointed row fires. -/
theorem carveClosure_of_pinned (hfeas : MarkFlowFeasible N t)
  (hpin : PinnedDelivery N mark a) (hclose : InstantClosure N a t) :
  CarveClosure N (fun i => {a i}) t

```

BernsteinLogicAux.QtslCompose.AGComposeCarves . **marks\_of\_pinned\_delivery\_via\_carve**

statement bound

```

/-- **Degeneracy is a theorem, not a coincidence**: the r4 engine's exact
statement, re-derived through the generalized engine at singleton carves. -/
theorem marks_of_pinned_delivery_via_carve (hser : Serial N)
  (hfeas : MarkFlowFeasible N t) (hpin : PinnedDelivery N mark a)
  (hclose : InstantClosure N a t) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i

```

BernsteinLogicAux.QtslCompose.AGComposeCarves . **carvedDelivery\_mark\_iff\_conditional**

statement bound

```

/-- At `P = mark`, the carve rows ARE the conditional-delivery premises: his
sketch's judgments and the r4 pin rows are the two ends of one dial. -/
theorem carvedDelivery_mark_iff_conditional :
  CarvedDelivery N mark mark ↔ ∀ i, ConditionalDelivery N mark i

```

BernsteinLogicAux.QtslCompose.AGComposeCarves . **carveClosure\_of\_acyclic**

statement bound



```

/-- Acyclicity dissolves the diagonal at ANY delivery-closed curve: with a
topological order, pinned instant coordinates choose from `F n P` one
`ord`-level at a time (their in-neighbours are all labeled `k` by the walk:
pipelined ones held in-carve by conformance, instant ones settled at earlier
levels), and unpinned coordinates close by seriality as in r4. -/
theorem carveClosure_of_acyclic (hser : Serial N) (hloc : DepLocal N)
  (ord : Fin n → ℕ) (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (hfeas : MarkFlowFeasible N t) (hpin : CarvedDelivery N mark P) :
  CarveClosure N P t

```

BernsteinLogicAux.QtslCompose.AGComposeCarves .marks\_acyclic\_via\_maximal\_carve

statement bound

```

/-- **The dial's other end, machine-checked** (the second derivation of r4's
acyclic corollary): on acyclic networks, his sketch's  $\forall \exists$  premises conclude
THROUGH the carve engine at the MAXIMAL curve `P = mark` – no point derived,
no diagonal presented. With `marks_of_conditional_delivery_acyclic` (r4,
via derived points) this brackets the dial from both ends. -/
theorem marks_acyclic_via_maximal_carve (hser : Serial N) (hloc : DepLocal N)
  (ord : Fin n → ℕ) (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (hfeas : MarkFlowFeasible N t)
  (hdel : ∀ i, ConditionalDelivery N mark i) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i

```

BernsteinLogicAux.QtslCompose.AGComposeCarves .z5\_carvedDelivery\_mark

statement bound

```

/-- **C5 – invariance is NOT the discriminant**: the swap loop's carve rows
HOLD at the maximal curve (swap maps the mark set into the mark set; the
relay copies it). What Z5 loses is the diagonal, not the delivery. -/
theorem z5_carvedDelivery_mark : CarvedDelivery z5Net z5Mark z5Mark

```

BernsteinLogicAux.QtslCompose.AGComposeCarves .z5\_no\_carve\_closure

statement bound

```

/-- **C6 – the diagonal IS the discriminant**: NO carve system inside the
marks closes the swap loop's instant diagonal – containment and closure alone
contradict (the delivery rows are not even needed). The r4 refusal
`z5_no_pin` is this fact's singleton shadow. -/
theorem z5_no_carve_closure :
  ¬ ∃ P : Fin 2 → Set (Fin 3), (∀ i, P i ⊆ z5Mark i) ∧
  CarveClosure z5Net P ![0, 0]

```

BernsteinLogicAux.QtslCompose.AGComposeCarves .z5\_refutes\_closureless

statement bound

```

/-- **M2' (the diagonal deleted): the carve rule without `CarveClosure` is
unsound** – carve rows at the maximal curve, dep-locality, a feasible
schedule, unpinned joint instant closure, and a nonempty composite all hold
on the swap loop, yet no joint run realizes any mark. The r4 mutant
`z5_refutes_pinless` restated in r5 vocabulary: what the stage arithmetic
cannot see, carve-invariance cannot see either. -/
theorem z5_refutes_closureless :
  ¬ (∀ (m : ℕ) (W : Type) ( _ : Nonempty W) (N : MarkNet m W)
    (mark P : Fin m → Set W) (t : Fin m → ℕ),
    Serial N → DepLocal N → CarvedDelivery N mark P →
    MarkFlowFeasible N t → W0Serial N → (∃ ω, JointRun N ω) →
    ∃ ω : ℕ → Fin m → W, JointRun N ω ∧ ∀ i, ∃ k, ω k i ∈ mark i)

```

BernsteinLogicAux.QtslCompose.AGComposeCarves .z1\_refutes\_walkless\_carve

statement bound

```

/-- **M1' (the walk deleted): the carve rule without `MarkFlowFeasible` is
unsound** – the relay-delay loop carries carve rows AND a discharging
diagonal at the singleton carve `{true}` (the closure's conformance
hypothesis hands the delay's base value in-carve, and the relay copies it),
yet realizes no mark on any run. The walk is forced at carve level exactly
as at value level. -/
theorem z1_refutes_walkless_carve :
  ¬ (∀ (m : ℕ) (W : Type) ( _ : Nonempty W) (N : MarkNet m W)
    (mark P : Fin m → Set W) (t : Fin m → ℕ),

```



$$\begin{aligned} & \text{Serial } N \rightarrow \text{CarvedDelivery } N \text{ mark } P \rightarrow \text{CarveClosure } N \text{ P } t \rightarrow \\ & \exists \omega : \mathbb{N} \rightarrow \text{Fin } m \rightarrow W, \text{JointRun } N \omega \wedge \forall i, \exists k, \omega k i \in \text{mark } i) \end{aligned}$$
BernsteinLogicAux.QtslCompose.AGComposeCarves . **fatCarve\_marks\_realized**

statement bound

```

/-- **The measure-zero objection, answered at theorem level**: the relay loop
realizes its marks through the engine with carve = mark = `{v ≠ 0}` - a
NON-singleton carve; no hypothesis names a value. (The diagonal closes at a
point the PROOF chooses; the kit never presents one. Z5's swap loop, with the
same carve and the same walk, has no such point - `z5_no_carve_closure`.) -/
theorem fatCarve_marks_realized :
  ∃ ω : ℕ → Fin 2 → Fin 3, JointRun relay3Net ω ∧
  ∀ i, ω (![0, 0] : Fin 2 → ℕ) i) i ∈ relay3Mark i

```

## Correspondence (Steele display ↔ Lean)

the Steele display and a bound Lean statement and the dictionary text **CHANGED** since certification — re-review and re-certify The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The display is the rule at the satisfaction stratum; the bound set is its soundness metatheorem's discrete-relational instance, over an explicit network model ('MarkNet': in-neighbour sets, per-component delivery lags, memoryless admitted-output relations over wire snapshots, initial segments; 'JointRun' the per-stage constraint satisfaction). The dictionary, display constituent by bound statement:

- the conditional-delivery premises  $\Sigma_i \models (I_i, A_i, E_i) \mapsto \text{ConditionalDelivery}$  (every mark-conforming snapshot admits an own-mark output through it) — the memoryless instantiation; the latency rows' seriality obligation  $\mapsto \text{Serial}$ ; dep-locality (a component reads only its declared in-neighbours)  $\mapsto \text{DepLocal}$ .
- the MarkFlow stage column  $\mapsto \text{MarkFlowFeasible}$  — the walk's Lean *specification* at the exact-lag row shape ( $t_i = t_j + \text{lag}_i$  per in-edge,  $\text{lag}_i \leq t_i$ ); the general difference-constraint species (windows, rational stages) is the wire format's and is honestly absent here.
- the carve column's row families  $\mapsto \text{CarvedDelivery}$  (containment + carved delivery: every snapshot with in-neighbour reads inside the carves admits an output in the own carve) and  $\text{CarveClosure}$  (the diagonal: at every stage, for every carve-conforming base, the instant subsystem closes jointly with labeled coordinates in-carve and admitted) — the latter consumed *abstractly*: its per-species certificate dischargers are the wire format's, its fixpoint devices the metatheorem's. The r4 value column  $\mapsto \text{PinnedDelivery} + \text{InstantClosure}$ , subsumed as the singleton case by  $\text{carvedDelivery\_of\_pinned} / \text{carveClosure\_of\_pinned} / \text{marks\_of\_pinned\_delivery\_via\_carve}$ ; the premises themselves are the maximal-carve rows,  $\text{carvedDelivery\_mark\_iff\_conditional}$ .
- the W0-species rows  $\mapsto \text{InstantClosure} / \text{CarveClosure}$  (schedule-relative stepwise properness) — deliberately STRONGER than bare nonemptiness, the tier discharge the construction consumes; the unpinned form  $\text{W0Serial}$  appears only in the refuted premise-only statements, to make those refutations as strong as possible.
- the conclusion  $\mapsto$  the existential in  $\text{marks\_of\_carved\_delivery} \text{ (engine)} / \text{marks\_of\_conditional\_delivery\_acyclic} + \text{marks\_acyclic\_via\_maximal\_carve}$  (both dial ends of the acyclic case): some  $\text{JointRun}$  realizes every mark at its label (the joint witness under the binder, never a hypothesis).
- the countermodel battery, as prereg'd mutants:  $\text{z1\_refutes\_walkless} / \text{z1\_refutes\_walkless\_carve}$  (stage column deleted, at value and carve level),  $\text{z5\_refutes\_pinless} / \text{z5\_refutes\_closureless}$  (pin/diagonal column deleted — against kits with  $\text{W0Serial}$  and composite nonemptiness granted), the discriminant pair  $\text{z5\_carvedDelivery\_mark}$  (invariance holds on the swap loop) /  $\text{z5\_no\_carve\_closure}$  (no carve inside the marks closes its diagonal),  $\text{z5\_no\_pin}$  (its singleton shadow),  $\text{z1\_walk\_infeasible} / \text{allOnes\_walk\_feasible}$  (the walk's refuse/admit pair),  $\text{allOnes\_marks\_realized}$  (positive control through the pinned engine) +  $\text{allOnes\_direct}$  (independent witness) +  $\text{fatCarve\_marks\_realized}$  (positive control through the carved engine at a non-singleton carve — no hypothesis names a value).

Width note: the conditional mark's content is the tier gap stated in the commentary — within the mechanized tier the mappings above are exact, the engine is proven at general lags, and the mark-entailment sequent is consumed at its identity instance (the conclusion realizes the premises' own marks; the cut against a distinct output-spec mark list is sequent-layer machinery, not this module's).

## Lean verifier receipts

module BernsteinLogicAux.QtslCompose.AGComposeMarks; no attested build yet; source-bound only. The statements above are hash-bound to the source file, but no build attestation exists for this module yet, so no receipt can be judged to attest the quoted build.

**NO ATTESTED BUILD BOUND for this module** 7 receipt(s) in the store attest superseded builds of this module and are omitted from this rendering. module BernsteinLogicAux.QtslCompose.AGComposeCarves; no attested build yet; source-bound only. The statements above are hash-bound to the source file, but no build attestation exists for this module yet, so no receipt can be judged to attest the quoted build.

**NO ATTESTED BUILD BOUND for this module** 6 receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

gate-layer coverage of the bound build:

build (lake build (elaboration)) **stale only**  
 replay (leanchecker (official kernel replay)) **none**  
 hardened (safe\_verify (hardened screen)) **none**  
 decorrelated (lean4export + nanoda (decorrelated kernel)) **stale only**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The satisfaction-judgment extension postdates the HOL4 port entirely (the ported cone is the base calculus's first-order check-function surface), and this rule's wire format is itself staged ahead of the schema — there is no wire checker for a HOL statement to mirror yet. The box inherits QWIRECOMPOSE's honest state one rule further out.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1). The soundness warrant is a metatheorem whose joint witness never appears on the wire; this section proves it at the discrete-relational tier — the tier the mechanization proves — and derives every certificate row from a countermodel that breaks the rule without it. Per-profile instances beyond this tier are stated as open, never claimed.

**Setting and claim.** The acyclic composition rule's causality certificate makes a directed cycle unrepresentable; this rule is its polarity complement, existing precisely to admit cyclic wiring — for *liveness marks only*. The polarity fact that makes this sound where the safety analogue fails: a safety violation and a mark realization are both finite-stage events with opposite persistence. A violation of a prefix-closed guarantee poisons every extension, so under a  $\forall$ -reading a cyclic wiring can conspire to admit a bad behaviour and a well-foundedness device is mandatory. A dip under a mark survives every extension, so under an  $\exists$ -reading the same conspiracy works for the conclusion: circular self-support is a post-fixpoint, productive rather than vicious. The rule concludes  $\text{SysWire}(w, [\Sigma_1, \dots, \Sigma_n]) \models \text{spec}_{\text{out}}$  where the output spec's demanded content is its record's ExistsSpec list — one turnstile, no subscript: the face lives in the argument. Premises: per-component *conditional delivery* (for every admitted resolution whose interface conforms to the assumed in-neighbour marks, some admitted resolution through it realizes the own mark — the  $\forall$ -conforming- $\exists$ -delivering reading, with a properness presupposition on the conforming sub-body), each shipping its *latency row* (delivery-stage arithmetic at its own profile tier); the wiring certificate's plumbing; one mark-entailment sequent  $\Lambda_i g_i \vdash \text{marks}(\text{spec}_{\text{out}})$ ; the MarkFlow walk in two columns (below); and the W0-species properness rows.

*Remark R80.1* (the walk, stage column: what breaks without it). Take a relay (out  $\equiv$  in, instantaneous) closed against a unit delay initialized to 0, marks “hits 1” on both wires. Both conditional-delivery premises are true — a relay fed a trace hitting 1 emits one; a delay shifts a hit by one stage — and the composite is proper. But its unique behaviour is all-zeros: the initialization propagates around the cycle forever, and the conclusion is false. Premise shape alone is therefore unsound; something composite-level is forced. The forced ingredient is *schedule arithmetic*: the producer supplies one stage label  $t_k$  per mark, and the checker verifies the finite difference-constraint system assembled from the latency rows — label equality across instantaneous wires, shifts through delay media. The relay-delay loop is refused exactly here: its rows demand  $t_x = t_y$  and  $t_y = t_x + 1$ , infeasible — while the loop of two relays passes at the shared label 0. This is the  $\exists$ -dual of the causality walk: the same arithmetic species (difference constraints over the declaration closure, bounded recompute, never chase derivation), the opposite verdict role — the  $\forall$ -walk demands strict increase along every wire, refusing all cycles; this walk demands consistency, admitting every cycle whose schedule closes. Feasibility of difference constraints is a Bellman-Ford-class check, not an induction device: no well-foundedness enters. Mechanized: `z1_refutes_walkless` (every other hypothesis of the pinned engine true, conclusion false), with the refuse/admit pair `z1_walk_infeasible / allOnes_walk_feasible`.

*Remark R80.2* (the walk, value column: what breaks without it). The stage column has a blind spot, and it is exactly where this rule lives. Around any dependency cycle the latency rows sum, so a feasible schedule forces every cycle to total lag zero and one shared label: the admitted cycles are the instantaneous self-supporting loops, whose marks all land at one instant. There the  $\exists$ -chosen delivery *values* must agree around the loop — and no stage arithmetic can see values. Witness: a pointwise swap

gate over  $\{0, 1, 1'\}$  ( $1 \mapsto 1', 1' \mapsto 1, 0 \mapsto 0$ ) closed against a relay, with mark “hits  $\{1, 1'\}$ ” on both wires — a mark set closed under the swap. Every remaining hypothesis holds: both conditional deliveries (the swap maps the mark set into itself; the relay copies it), the schedule ( $t_x = t_y = 0$ ), properness (all-zeros — 0 is the swap’s fixpoint), even joint instant seriality. But every joint run satisfies  $x = \text{swap}(x)$  around the loop, hence is all-zeros, and the conclusion is false: the swap loop is the relay loop’s evil twin, distinguished by values alone. The forced repair is the walk’s *value column*: on each zero-lag cycle the producer supplies a *pin*  $\alpha_k$  per mark — the delivered value, spec-representable and label-sized, the value analogue of  $t_k$ , never a trace — and the checker verifies mark membership per pinned wire plus *pointed delivery*: each pinned component’s conditional-delivery judgment cited at the pins (through every conforming input agreeing with the neighbouring pins, the own pin is admitted). The relay loop passes ( $\alpha_x = \alpha_y = 1$ ); the swap loop is refused ( $\alpha_x = \text{swap}(\alpha_y), \alpha_y = \alpha_x$  has no solution inside the marks). Mechanized: `z5_refutes_pinless` (the premise-only kit accepts a false conclusion) and `z5_no_pin` (the pin rows refuse the witness). Off the cycles no pin is needed — see the chaining lemma. An exact value is, however, the wrong general datum — the next remark generalizes the column and keeps this one as its degenerate species.

**Remark R80.3** (the carve column: pins generalized to predicates). An exact pin is measure-zero data: honest for a finite alphabet, unrepresentable for members whose wire states are continuous or infinite-dimensional. The column therefore generalizes: per zero-lag- cycle mark the producer supplies a *carve* — a predicate on the wire’s state carrier, presented as a scalar-codomain gate hypernet cited by digest (its nonnegativity locus the carve), or a plated aggregate gate where the member is a token-population system. The soundness question this raises is the design’s crux: a point composes trivially around an instant cycle (the joint assignment is the pin vector), while a carve leaves each delivery free within a set, so joint existence becomes a nonemptiness obligation over the carve product — the *diagonal*. Carve- invariance does not settle it: on the swap loop the carve rows hold at the maximal carve — the swap maps the mark set into itself (`z5_carvedDelivery_mark`) — yet *no* carve inside the marks closes the diagonal (`z5_no_carve_closure`): around the loop any jointly-admitted value is a swap fixpoint, and the only one lies outside the marks. So the carve rows come in three families: *containment* (carve refines mark), *carved delivery* (the rule’s own conditional-delivery judgment, cited at the carve events — the premises are literally the maximal-carve rows, `carvedDelivery_mark_iff_conditional`, so the column is one dial from the sketch’s judgments down to r4’s points), and one *diagonal certificate* per cycle, species-tagged: a witness point or bounded enumeration where states are finite (the degenerate species — and the swap loop is refused mechanically there); contraction rows with exact-rational cycle gain below one (Banach closes the diagonal — small-gain arithmetic reused at the  $\exists$ -face); convexity or order-interval flags (Kakutani, Tarski). The checker verifies only the species’ finite rows; each fixpoint theorem lives in the soundness metatheorem, per profile — exactly where Ionescu-Tulcea already lives for the stochastic assembly. Mechanized at this tier: the carved engine (`marks_of_carved_delivery`, the diagonal consumed abstractly as `CarveClosure`), the subsumption of pins as singleton carves (`marks_of_pinned_delivery_via_carve` — degeneracy as a theorem), the carve-level mutants (`z5_refutes_closureless`, `z1_refutes_walkless_carve`), and a fat-carve positive control in which no hypothesis names a value (`fatCarve_marks_realized`).

**Remark R80.4** (the reading of the premise, the properness presupposition, and W0: what breaks without each). Three further hypotheses earn their places by countermodels at the design tier. (i) Reading the component premise as co-realization (“some behaviour realizes assumptions and guarantee together”) is unsound: a component can co-realize only in a mode the closed loop never enters (a gate that cooperates only on inputs the wiring cannot produce), so the conditioning must do real work —  $\forall$ -conforming- $\exists$ -delivering is the reading under which the metatheorem is true. (ii) The properness presupposition on each premise’s conforming sub-body blocks vacuous conditioning: a component admitting no conforming input satisfies the conditional vacuously while delivering nothing. (iii) The W0-species rows block the empty composite (an inconsistent loop — a negation gate against a relay — satisfies every safety spec and no liveness spec), and for the liveness *construction* their tier discharge is stepwise: the composite must stay live at every stage given the schedule’s pins, because bare nonemptiness of the behaviour set is witnessed by runs the construction cannot use (a component may block after delivering; an instant subsystem’s solvability may flip on a pinned coordinate’s value). Stepwise properness is what the mechanized engine consumes.

**Lemma R80.5** (the assembly engine — soundness at the discrete-relational tier). *Fix  $n$  components over a wire alphabet, each with in-neighbours, a delivery lag ( $0 = \text{instantaneous}$ ), a memoryless admitted-output relation reading its lagged snapshot, and an initial segment. Suppose: (a) seriality (initial sets and output relations nonempty — the mechanism-presentation obligation of the latency rows); (b) a feasible schedule; (c) pins with their membership and pointed-delivery rows; (d) schedule-relative instant closure (at every stage, with pinned instantaneous coordinates held at their pins and pipelined coordinates at any given values, the remaining instantaneous coordinates close jointly). Then some joint run realizes every mark at its label.*

*Proof* (as mechanized, `marks_of_pinned_delivery`). *Build the run by stage recursion. At stage  $k$ , pipelined coordinates choose: their initial value before the pipe fills; the pin at their own label (legal because  $t_i \geq \text{lag}_i$  is a walk row); any admitted output otherwise (seriality). Instantaneous coordinates close jointly by (d), with the schedule’s pinned ones held at their pins. Two facts discharge the pinned constraints. First, the pin invariant: at its label every coordinate carries its pin — by construction for pipelined coordinates, by (d)’s base clause for instantaneous ones. Second, the walk’s alignment: for a marked component  $i$ , every in-neighbour  $j$  satisfies  $t_i = t_j + \text{lag}_i$ , so the snapshot  $i$  reads at delivery time is taken at stage  $t_j$  — where, by the invariant,  $j$  carries its pin. The pointed-delivery row therefore applies verbatim and admits the pin. Extension past the last label is seriality plus (d) at unpinned stages. The realized-at-label conclusion is the invariant plus the membership rows. Carved form (as mechanized, `marks_of_carved_delivery`): the same recursion with the invariant weakened from “carries its pin” to “lands inside its carve” — pipelined labeled coordinates now choose from the intersection of their admitted set with their carve (nonempty by the carved-delivery row, since the walk aligns their in-neighbours at labels where the invariant holds), and instantaneous labeled coordinates close through the diagonal certificate, in-carve and admitted at once. The pinned statement above is re-derived from the carved engine at singleton carves (`marks_of_pinned_delivery_via_carve`).*

**Lemma R80.6** (the chaining bridge — no pins where no cycle bites). *On a network whose dependency graph admits a topological order, the conditional-delivery premises alone produce pins satisfying the rows of Lemma R80.5: choose each component’s delivered value by its*

*premise, applied to the already-chosen deliveries of its in-neighbours, in topological order; dep-locality (a component reads only its declared in-neighbours) turns the chosen witness into the pointed row. Moreover acyclicity dissolves the joint instant solve: seriality alone furnishes the instant closure, coordinate by coordinate in order. Hence the premise-shaped rule with no value column is sound on acyclic-instant networks (`marks_of_conditional_delivery_acyclic`) — the pins are producer-derivable search data there, entering the wire only where the walk's own arithmetic says a cycle closed. Proof (as mechanized, `pins_of_conditional_delivery` + `instantClosure_of_acyclic`). Induction on topological levels: at each level, apply the premise at the snapshot assembling the lower levels' pins (conforming by their membership conjuncts); transport the chosen admission along dep-locality to every dep-agreeing snapshot. The instant closure by the same induction, with seriality supplying unpinned choices. The bridge holds at the dial's other end too: the same premises are the maximal-carve rows, and acyclicity dissolves the diagonal at any delivery-closed carve (`carveClosure_of_acyclic`), so the carved engine also concludes with nothing pinned and nothing carved on the wire (`marks_acyclic_via_maximal_carve`).*

*Remark R80.7 (the metatheorem beyond this tier).* The joint object lives in the soundness proof and only there: at the discrete-relational tier it is the recursion above; at stochastic tiers the joint law is assembled by Ionescu–Tulcea/Kolmogorov extension along the wiring's stagewise dependency structure, with the mark-escaping resolutions selected per the premises; for PDE profiles, whatever solution object the profile's semantic theorem provides. The rule's premise kit does not change per profile — only the bridge lemmas do, exactly as in the acyclic rule's grid. Those instances, the per-species diagonal devices at continuous tiers (Banach for the contraction species, Kakutani/Brouwer for the convex, Knaster–Tarski for the order-theoretic — each converting its species' finite rows into the joint instant solve the carved engine consumes abstractly), history-dependent mechanisms, and the general difference-constraint walk (windows, rational stages) are open; the mechanized tier is the one this section proves, and the conditional mark warrants exactly it. Two scope fences are part of the statement: marks are single-dip graded-reach demands (finitely realizable, extension-persistent) — recurrence-shaped liveness stays outside; and the rule trades completeness for finite presentation — a component delivering only on its favourite conforming input can witness a joint behaviour no conditional-delivery premise certifies.

## Rule 81. QSIMTRANSPORT (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — *QSimTransport*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{S' \models \text{spec}' \quad \text{QSim}(\sigma, S, S')}{S \models \sigma^* \text{spec}'} \text{QSIMTRANSPORT}$$

certification transport along a coarse-graining:  $\sigma$  is an exact simulation witness,  $S'$  coarse-grains  $S$  — if  $S'$  is certified and  $\text{QSim}(\sigma, S, S')$  holds ( $\sigma$  maps every behaviour of  $S$  to a behaviour of  $S'$ ), then  $S$  is certified for the pulled-back spec ( $\sigma^*$  pulls back the whole interface spec, assumptions and guarantees). Expectation specs pull back with the bound intact; region/invariance specs pull back to the preimage region. The former side condition (per-mode polynomial map + compatibility squares as BL leaves) is the kind's discharge, one node down: *QSIMDISCHARGE* consumes exactly those squares ( $\sigma$ -strict tier; the kernel tier joins with its own bounded coverage walk), *QSIMCOMPOSE* composes witnesses. Exactness is what *QSim* asserts — the approximate  $\sigma$ - $\varepsilon$  class concludes *QDist*, never this premise. One-sided functional refinement — a simulation witness is never a two-sided coupling, satisfaction of an interface-bearing spec is plain *QSat*, as at *QWIRECOMPOSE*.

*QSIMTRANSPORT* transports certification along a coarse-graining: if the abstract system  $S'$  is certified, and the simulation judgment  $\text{QSim}(\sigma, S, S')$  holds — its discharge, one node down, checks the per-mode polynomial map's compatibility squares as inherited BL leaves — then the concrete system  $S$  is certified for the pulled-back spec  $\sigma^* \text{spec}'$ . A producer uses it in the direction the rule states: certify the abstraction *once* (small state space, cheap rows), then reuse that certificate under every refinement that simulates it, paying only the simulation judgment's discharge per refinement.

What the cited theorems warrant is three strata. The *behavioural* form: over compiled behaviour classes, if  $\sigma$  maps every behaviour of  $S$  to a behaviour of  $S'$  (the semantic content of the squares) and  $S'$ 's behaviours all satisfy the spec, then  $S$ 's behaviours satisfy the pullback — two lines, which is the honest size of the rule's semantic content; everything substantive in an application is the sequent-level discharge of the squares. The *finite-kernel* instance, where the development has machine-checked simulation machinery: a functional simulation witness at the finite Markov tier is exactly a pushforward-kernel identity (per state, the pushed concrete kernel row equals the abstract kernel row at the image state), the marginal-process lemma then transports laws at every time, and both spec-transport faces follow — expectation specs pull back along  $\sigma$  with the bound intact, and region/invariance specs pull back to the preimage region. And the *general measurable-kernel* tier, mirroring the finite instance name-for-name over Markov kernels and the Giry monad: the same pushforward identity (now a *Measure.map* equation per state), the same law transport at every horizon, both spec-transport faces — with the finite tier's nonnegativity hypotheses gone, true by type.

The witness is one-sided by design: a simulation map is functional refinement, not a two-sided relational coupling. A functional simulation induces a degenerate coupling, but the transport theorems never route through it — they are the pushforward identity plus summation identities, and the relational tier's coupling machinery (joint kernels, absorption hypotheses) is deliberately not shared. What breaks without the hypotheses: without the compatibility squares the pushed law and the abstract law decouple at the first step, and the pullback claims anything; without the nonnegativity hypotheses of the region form, signed “mass” could cancel and hide probability outside the preimage region.

**notes** The general measurable-kernel tier is discharged (third stratum, bound below). The simulation premise is the judgment kind *QSim*, baked in the extension theory together with its discharge and composition nodes: the former side condition (map data + compatibility squares) is that kind's discharge, so nothing here renders as a hypothesis mark. Exactness is what the kind asserts — the approximate class concludes distance judgments instead. What remains beyond the three proven strata: the categorical strengthening under which the composition rule could cite the lens-composition theorem as a lemma *template* rather than proving its cells directly — the direct proofs never pull back along strict weakenings, which is why the strengthening is not a dependency of anything on this page. No end-to-end certificate exercises the rule yet. Related reading, not dependency: an exchangeability quotient is a simulation map, so aggregate-space reduction can be *read* as an instance of this rule — it stands on its own proof.

### Lean statement

BernsteinLogicAux.QtslCompose.SimTransport.pullback

statement bound

```
/-- **QSimTransport, behavioral form.** `Beh`/`Beh` are the compiled
behavior classes of the concrete and abstract Systems; `hsim` is the
semantic content of the simulation witness ( $\sigma$  maps behaviors to behaviors);
`hsat` is `QSatI(S', spec')`. Conclusion: `QSatI(S,  $\sigma^*$  spec')`. -/
theorem pullback { $\Omega$   $\Omega'$  : Type*} { $\sigma$  :  $\Omega \rightarrow \Omega'$ } {Beh : Set  $\Omega$ } {Beh' : Set  $\Omega'$ }
  {spec' : Set  $\Omega'$ } (hsim :  $\sigma '' Beh \subseteq Beh'$ ) (hsat : Beh'  $\subseteq$  spec') :
  Beh  $\subseteq$   $\sigma^{-1}$  spec'
```



BernsteinLogicAux.QtslCompose.SimTransport . **expect\_push**

statement bound

```

/-- Observable pullback: the expectation of `g` under a pushed law is the
expectation of the pulled-back observable `g ∘ σ` under the original law.
(The `σ` spec` of the rule display, at expectation-spec granularity.) -/
theorem expect_push (σ : α → β) (μ : α → ℝ) (g : β → ℝ) :
  ∑ b, push σ μ b * g b = ∑ s, μ s * g (σ s)

```

BernsteinLogicAux.QtslCompose.SimTransport . **sim\_expect**

statement bound

```

/-- **QSimTransport, expectation form** (finite tier). Along a simulation
witness `PushKernel σ K Q`, the time-`n` expectation of a pulled-back
observable on the concrete chain *equals* the abstract chain's expectation –
`push_law` composed with `expect_push`. -/
theorem sim_expect {σ : α → β} {K : α → α → ℝ} {Q : β → β → ℝ}
  (hσ : PushKernel σ K Q) (μ₀ : α → ℝ) (g : β → ℝ) (n : ℕ) :
  ∑ s, law K μ₀ n s * g (σ s) = ∑ b, law Q (push σ μ₀) n b * g b

```

BernsteinLogicAux.QtslCompose.SimTransport . **sim\_expect\_bound**

statement bound

```

/-- Certified expectation bounds transport: a bound proven against the
abstract (coarse) chain holds for the concrete chain at the pulled-back
observable. This is CM's rule 2 at the finite tier, in the direction the
rule states it (certify the abstraction once, reuse under refinement). -/
theorem sim_expect_bound {σ : α → β} {K : α → α → ℝ} {Q : β → β → ℝ}
  (hσ : PushKernel σ K Q) (μ₀ : α → ℝ) (g : β → ℝ) {n : ℕ} {b₀ : ℝ}
  (hb : ∑ b, law Q (push σ μ₀) n b * g b ≤ b₀) :
  ∑ s, law K μ₀ n s * g (σ s) ≤ b₀

```

BernsteinLogicAux.QtslCompose.SimTransport . **sim\_region**

statement bound

```

/-- **QSimTransport, region form** (finite tier): if the abstract law at time
`n` puts no mass outside `R'`, the concrete law puts no mass outside
`σ ⁻¹ R'` – invariance specs pull back to the preimage region. -/
theorem sim_region {σ : α → β} {K : α → α → ℝ} {Q : β → β → ℝ}
  (hσ : PushKernel σ K Q) {μ₀ : α → ℝ}
  (hK : ∀ s t, 0 ≤ K s t) (hμ₀ : ∀ s, 0 ≤ μ₀ s) {n : ℕ} {R' : Set β}
  (hR' : ∀ b, b ∉ R' → law Q (push σ μ₀) n b = 0) :
  ∀ s, σ s ∉ R' → law K μ₀ n s = 0

```

BernsteinLogicAux.QtslCompose.SimTransportMeas . **push\_law**

statement bound

```

/-- **QSimTransport, measurable-kernel tier – law transport.** Along a
simulation witness, the pushforward of the time-`n` concrete law is the
time-`n` abstract law of the pushed start. (The measurable-tier face of
`QRelChain.push_law`; both spec-transport faces below are corollaries.) -/
theorem push_law {σ : α → β} {K : Kernel α α} {Q : Kernel β β}
  (hσ : Measurable σ) (h : PushKernel σ K Q) (μ : Measure α) :
  ∀ n, (law K μ n).map σ = law Q (μ.map σ) n
| 0 => rfl
| n + 1 => by
  rw [law_succ, map_bind _ _ hσ, funext h, ← bind_map _ _ hσ,
    push_law hσ h μ n, law_succ]

/-! ## The two spec-transport faces -/

/-- Expectation specs pull back along `σ` with the bound intact –
`ℝ≥0∞`-observable face, an *equality* as at the finite tier. -/
theorem sim_expect_lintegral {σ : α → β} {K : Kernel α α} {Q : Kernel β β}
  (hσ : Measurable σ) (h : PushKernel σ K Q) (μ : Measure α)
  {g : β → ℝ≥0∞} (hg : Measurable g) (n : ℕ) :
  ∫- s, g (σ s) ∂law K μ n = ∫- b, g b ∂law Q (μ.map σ) n

```

BernsteinLogicAux.QtslCompose.SimTransportMeas . **sim\_expect\_bound**

statement bound

```

/-- Certified expectation bounds transport in the direction the rule states:
certify the abstraction once, reuse under refinement. -/

```

```

theorem sim_expect_bound {σ : α → β} {K : Kernel α α} {Q : Kernel β β}
  (hσ : Measurable σ) (h : PushKernel σ K Q) (μ : Measure α)
  {g : β → ℝ≥0∞} (hg : Measurable g) {n : ℕ} {b₀ : ℝ≥0∞}
  (hb : ∫- b, g b ∂law Q (μ.map σ) n ≤ b₀) :
  ∫- s, g (σ s) ∂law K μ n ≤ b₀

```

BernsteinLogicAux.QtslCompose.SimTransportMeas.sim\_region

statement bound

```

/-- Region/invariance specs pull back to the preimage region. The finite
tier's nonnegativity hypotheses (`hK`, `hμ₀` — "signed mass could cancel")
are absent: at this tier they hold by type. -/
theorem sim_region {σ : α → β} {K : Kernel α α} {Q : Kernel β β}
  (hσ : Measurable σ) (h : PushKernel σ K Q) (μ : Measure α)
  {R' : Set β} (hR' : MeasurableSet R') {n : ℕ}
  (hz : law Q (μ.map σ) n R'ᶜ = 0) :
  law K μ n (σ-1 R')ᶜ = 0

```

BernsteinLogicAux.QtslCompose.SimTransportMeas.pushKernel\_iff\_comp

statement bound

```

/-- A functional simulation witness is exactly commutation of the step
kernels with the deterministic kernel of `σ`: the square the lens/fibred
strengthening would functor over. -/
theorem pushKernel_iff_comp {σ : α → β} (hσ : Measurable σ)
  {K : Kernel α α} {Q : Kernel β β} :
  PushKernel σ K Q ↔
  Kernel.deterministic σ hσ ∘K K = Q ∘K Kernel.deterministic σ hσ

```

## Correspondence (Steele display ↔ Lean)

reviewer-certified against the displayed texts

The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

Three strata, as the commentary's conditional scope states; the display maps onto each at its own granularity. *Behavioural*: the displayed conclusion  $S \models \sigma^* \text{spec}' \mapsto \text{pullback's goal } \text{Beh} \subseteq \sigma^{-1} \text{spec}'$  over compiled behaviour classes; the E1 premise  $S' \models \text{spec}' \mapsto \text{hsat} : \text{Beh}' \subseteq \text{spec}'$ ; the judgment premise  $\text{QSim}(\sigma, S, S') \mapsto \text{hsim} : \sigma^{-1} \text{Beh} \subseteq \text{Beh}'$  — the kind's *semantic content* exactly: the per-mode polynomial structure of  $\sigma$  is erased ( $\sigma$  is any function), and the BL discharge of the compatibility squares lives in the kind's own discharge node, outside this bound set. The statement is axiom-free set algebra — two lines, the honest size of the rule's semantic content.

- *finite-kernel instance*: the simulation witness  $\mapsto \text{PushKernel } \sigma K Q$  — the per-state marginal identity  $\text{push } \sigma (K s) = Q (\sigma s)$ , which is what a functional simulation IS at this tier.
- expectation specs pull back “with the bound intact”  $\mapsto \text{sim\_expect}$ , and the certified statement is *stronger* than the display's reading: an *equality* of expectations at every horizon ( $\sum s, \text{law } K \mu_0 n s * g (\sigma s) = \sum b, \text{law } Q (\text{push } \sigma \mu_0) n b * g b$ ), with  $\text{sim\_expect\_bound}$  the displayed inequality direction as its corollary (certify the abstraction once, reuse under refinement).
- region/invariance specs pull back to the preimage  $\mapsto \text{sim\_region}$ : abstract law puts no mass outside  $R' \Rightarrow$  concrete law puts none outside  $\sigma^{-1} R'$  — and this face alone consumes nonnegativity ( $hK, h\mu_0$ ): the commentary's “signed mass could cancel” hypothesis, visible as exactly two binder rows.
- $\text{expect\_push}$  is the observable-pullback engine ( $\sigma^*$  at expectation granularity), fibre-summation algebra.

*Measurable-kernel tier* (second bound block, module `SimTransportMeas`, namespace `...QtslCompose.Meas`): the simulation witness  $\mapsto \text{Meas.PushKernel } \sigma K Q$  — per state,  $(K s). \text{map } \sigma = Q (\sigma s)$  over `ProbabilityTheory.Kernels`;  $\text{push\_law}$  is the law transport at every horizon over the  $n$ -fold `Giry` bind;  $\text{sim\_expect\_bound}$  and  $\text{sim\_region}$  are the two spec-transport faces in measure vocabulary (the addressability equalities  $\text{sim\_expect\_lintegral\_integral}$  live in the module, cited not bound); and  $\text{pushKernel\_iff\_comp}$  is the intertwining square —  $\text{Kernel.deterministic } \sigma h\sigma \circ_K K = Q \circ_K \text{Kernel.deterministic } \sigma h\sigma$ , push-after-step equals step-after-push — the shape the lens/fibred strengthening would functor over, so that residue names a bound *object*, not a blank. The finite tier's nonnegativity binders ( $hK, h\mu_0$ ) vanish at this tier by type, which is the “signed mass could cancel” clause discharged by construction. Width note: one-sidedness is structural in the bound statements — nothing consumes or constructs a joint kernel;  $\text{push\_law}$  (the shared marginal-process lemma) is genuinely the same lemma the relational tier rides, at both tiers, while the coupling machinery is deliberately absent from this bound set. The conditional mark is dropped: the former side condition is the baked `QSim` premise; what remains named — the lex/fibred (categorical) strengthening — is a dependency of nothing on this page.



## Lean verifier receipts

module BernsteinLogicAux.QtslCompose.SimTransport; bound build b2a09651f9a0... (bound from receipt dc0250279a3a2d0d, 2026-07-19T18:24:55Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| edfcab6a34d8774b | 2026-07-19T03:38:57Z | build+decorrelated                 | current |
| dc0250279a3a2d0d | 2026-07-19T18:24:55Z | build+decorrelated                 | current |
| 714abbd46c61ec9c | 2026-07-20T01:22:07Z | build+decorrelated                 | current |
| 28aeb5d1779c6cdc | 2026-07-20T20:46:58Z | build+decorrelated                 | current |
| 1bb39a35738bbe3  | 2026-07-20T20:58:31Z | build+replay+hardened+decorrelated | current |
| 1ae45ad709a68e43 | 2026-07-21T00:04:02Z | build+decorrelated                 | current |
| 5b96bfe1e468ce59 | 2026-07-21T05:39:17Z | build+replay+hardened+decorrelated | current |
| 016e87c2921cd31d | 2026-07-22T18:27:17Z | build+decorrelated                 | current |
| e776a27adccc860  | 2026-07-31T22:00:16Z | build+decorrelated                 | current |
| cf62752ca2ea53ab | 2026-07-31T22:29:10Z | build+decorrelated                 | current |
| 3c2ed3fc0dce1ac5 | 2026-08-02T23:31:21Z | build+decorrelated                 | current |
| 6121f42e978829a4 | 2026-08-03T16:27:25Z | build+decorrelated                 | current |
| 3a4a61220d562456 | 2026-08-03T22:46:09Z | build+decorrelated                 | current |

module BernsteinLogicAux.QtslCompose.SimTransportMeas; bound build 9530b0b3c465... (bound from receipt 3c92f1e3a2986972, 2026-07-20T08:24:45Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 3c92f1e3a2986972 | 2026-07-20T08:24:45Z | build+decorrelated                 | current |
| 06e7f179212d370d | 2026-07-20T20:47:00Z | build+decorrelated                 | current |
| c314290e315215f2 | 2026-07-20T20:58:33Z | build+replay+hardened+decorrelated | current |
| c471feb792abf799 | 2026-07-21T00:04:04Z | build+decorrelated                 | current |
| b18bbdfab8a60463 | 2026-07-21T05:39:18Z | build+replay+hardened+decorrelated | current |
| 16821f18670ae2eb | 2026-07-22T18:27:18Z | build+decorrelated                 | current |
| fb024d6489c5da75 | 2026-07-31T22:00:16Z | build+decorrelated                 | current |
| ed5047f4fc559336 | 2026-07-31T22:29:10Z | build+decorrelated                 | current |
| e618e69852b924d8 | 2026-08-02T23:31:21Z | build+decorrelated                 | current |
| 19820df5b08bd97a | 2026-08-03T16:27:25Z | build+decorrelated                 | current |
| 682c99d152518997 | 2026-08-03T22:46:09Z | build+decorrelated                 | current |

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The satisfaction-judgment extension postdates the HOL4 port entirely: the ported cone is the base calculus's check-function surface (theory bern\_catalogue), and no extension-theory definition — nothing shaped like simulation transport — exists in the HOL development. The box waits on a HOL port of the satisfaction stratum, which is its own campaign.

## HOL verifier receipts

**NONE**

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1). The rule’s semantic content is honestly small — the behavioural pullback is two lines — and the section says so rather than inflating it; the substance of any application is the sequent-level discharge of the compatibility squares, plus the finite-kernel instance where the transport is proven against actual chain machinery. The strata beyond the two walked here, and the genuine gaps, are marked loudly.

**Setting and claim.** A simulation witness  $\sigma : S \rightarrow S'$  is a per-mode polynomial map whose *compatibility squares* — commutation of  $\sigma$  with the two systems’ dynamics declarations, mode by mode — are checked as inherited sequent leaves; the rule consumes the witness as the judgment  $\text{QSim}(\sigma, S, S')$ , whose own discharge node carries those squares. The rule: from  $S' \models \text{spec}'$  and the judgment, conclude  $S \models \sigma^* \text{spec}'$  — certify the abstraction once, reuse under refinement [37, 168].

**Theorem R81.1** (behavioral pullback). *Let  $\text{Beh}$ ,  $\text{Beh}'$  be the compiled behaviour classes of  $S$ ,  $S'$ , and suppose  $\sigma$  maps behaviours to behaviours ( $\sigma[\text{Beh}] \subseteq \text{Beh}'$  — the semantic content of the compatibility squares). If every behaviour of  $S'$  satisfies  $\text{spec}'$ , then every behaviour of  $S$  satisfies  $\sigma^{-1}[\text{spec}']$ .*

*Proof.*  $\omega \in \text{Beh}$  gives  $\sigma(\omega) \in \text{Beh}' \subseteq \text{spec}'$ , i.e.  $\omega \in \sigma^{-1}[\text{spec}']$  (pullback). Two lines is the honest size: the rule’s semantic content is inclusion-chasing, and everything substantive in an application is establishing  $\sigma[\text{Beh}] \subseteq \text{Beh}'$  from the checked squares — per-mode polynomial identities, discharged by the base calculus. The hypothesis is not droppable to “ $\sigma$  is any map”: without the squares the image of a concrete behaviour need not be an abstract behaviour, and the pullback claims anything at all.  $\square$

**The finite-kernel instance.** At the finite Markov tier the witness has teeth the behavioural form leaves abstract: a *functional simulation* is per-state marginal identity — pushing the concrete kernel row at  $s$  along  $\sigma$  yields the abstract kernel row at  $\sigma(s)$  (a pushforward-kernel witness) — and the transport theorems are proven against the chain machinery.

**Lemma R81.2** (observable pullback). *For any law  $\mu$  on the concrete states and observable  $g$  on the abstract states,  $\sum_b (\sigma_* \mu)(b) g(b) = \sum_s \mu(s) g(\sigma(s))$ ; and along a pushforward-kernel witness, the time- $n$  expectation of a pulled-back observable on the concrete chain equals the abstract chain’s expectation of  $g$  at time  $n$  (started from the pushed initial law). Hence any certified expectation bound proven against the abstract chain holds for the concrete chain at the pulled-back observable.*

*Proof.* The first identity is summation over fibres of  $\sigma$  (expect\_push). The second composes it with the marginal-process lemma — the per-state marginal identities propagate, by induction on time, to equality of pushed laws at every time — giving sim\_expect; the bound transports by rewriting (sim\_expect\_bound). Note the direction: the producer proves one bound on the small chain and every refinement inherits it; nothing is re-proven per refinement beyond the witness identities themselves. The interface reading rides the same equalities: an assumption event  $A'$  pulls back with its mass intact (the pushed law at time  $n$  equals the abstract law, so  $\nu_n(\sigma^{-1} A') = \lambda_n(A')$ ), hence assumption-weighted bounds transport, and conditional expectations transport whenever the conditioning mass is positive.  $\square$

**Lemma R81.3** (region pullback). *Along a pushforward-kernel witness with entrywise-nonnegative kernel and initial law: if the abstract law at time  $n$  puts no mass outside  $R'$ , the concrete law at time  $n$  puts no mass outside  $\sigma^{-1}[R']$ .*

*Proof.* For  $s$  with  $\sigma(s) \notin R'$ : the concrete mass at  $s$  is at most the pushed mass at  $\sigma(s)$  (one nonnegative term of a fibre sum), which equals the abstract mass at  $\sigma(s)$  (the marginal-process lemma), which is zero; nonnegativity closes the squeeze (sim\_region). The nonnegativity hypotheses are load-bearing, not hygiene: with signed “mass” the fibre sum could cancel — a negative sibling hiding positive mass at  $s$  outside the preimage region — and the single-term bound fails. Nonnegativity is what makes “no mass outside  $R'$ ” transfer pointwise through a many-to-one map.  $\square$

**Remark R81.4** (refinement is one-sided; a simulation is not a coupling). The witness machinery genuinely shared with the relational (two-chain) tier is exactly the pushforward-kernel identity and the marginal-process lemma — a functional simulation *is* such a witness. Nothing else transfers: the relational tier’s certificates are joint-kernel data over a product space with absorption and drift hypotheses, and none of that is simulation transport. A functional simulation induces a degenerate coupling, but the theorems above never route through it — they are pushforward plus summation identities, full stop. Conflating the two would import hypotheses (coupling emptiness, absorption) that this rule neither needs nor checks.

**Remark R81.5** (the gaps, loud). The section walks two strata — behavioural, and finite-kernel. The general measurable-kernel tier is mechanized and bound in this rule’s entry, mirroring the finite instance name-for-name over Markov kernels (the same pushforward identity per state, now a Measure.map equation; law transport at every horizon; both spec-transport faces — with the finite tier’s nonnegativity hypotheses gone, true by type); it is cited there rather than re-walked here. Not yet mechanized: the categorical strengthening under which the composition rule could cite the lens-composition theorem of [37] as a lemma *template* — the direct composition proofs never pull back along strict weakenings, so nothing in this book depends on that strengthening; it is named so that its absence is a recorded choice, not an oversight. The judgment kinds the rule consumes and concludes into are staged in the certificate schema (the simulation judgment together with its discharge and composition nodes), but no end-to-end certificate yet exercises the rule — loader and end-to-end work, not mathematics. Related reading, not dependency: an exchangeability quotient is a simulation map, so aggregate-space reduction can be *read* as an instance of this rule; it stands on its own proof and cites nothing here.

| proof step                                   | formal anchor                                                                                                                                                                                                                                                                                                         |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Theorem R81.1                                | pullback<br>expect_push, sim_expect, sim_expect_bound; marginal-process lemma<br>push_law (chain substrate module)<br>sim_region; law nonnegativity law_nonneg (chain substrate module)<br>PushKernel (chain substrate module) — the per-state marginal identity<br>inherited sequent leaves, per mode (main package) |
| Lemma R81.2                                  |                                                                                                                                                                                                                                                                                                                       |
| Lemma R81.3                                  |                                                                                                                                                                                                                                                                                                                       |
| witness form                                 |                                                                                                                                                                                                                                                                                                                       |
| compatibility squares (cited, not re-proved) |                                                                                                                                                                                                                                                                                                                       |

## Rule 82. QFLATTEN (admissible rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QFlatten*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\text{QFlatEq}(\text{sys}, \text{flat}) \quad \mathcal{M}_{\text{flat}}, x_0 \in R \models \varphi \text{ (the closed-system pipeline's judgment)}}{\text{sys}, x_0 \in R \models \varphi} \text{QFLATTEN}$$

*flat* is the flattening of the composite *sys*; the substitution rule it always was: satisfaction transports across the flattening equation. The former side condition (polynomial substitution + guard-product bookkeeping) is the kind's discharge — oracle-shaped by construction (*qtsl\_flatten*, a bounded deterministic recompute over declaration structure), with the refusal boundary stated positively as the judgment's scope: shared-carrier confinement, and binary wiring with *n*-ary/directed cases reducing by currying = the checker's substitution recompute (refusal-shaped completeness cost, never a soundness hypothesis). The guarantee the rule was minted to state — SysWire adds structure, never expressiveness — is carried by the kind's existence: the flattened manifest's compiled semantics is the composite's (theorem at both kernel tiers on the synchronous-product face), and the projections remain free QSIMTRANSPORT witnesses under sibling well-formedness (unit row sums). The clock rows (schedule hypotheses, simultaneity, non-Zeno) are premise classes cited by every composition rule, deliberately not rows here.

QFLATTEN transports satisfaction across the flattening equation: given the judgment *QFlatEq*(*sys*, *flat*) — *flat* is the flattening of the composite presentation, discharged by the checker's substitution recompute — a verdict on the monolithic manifest is a verdict on the composite. A producer reaches for it as the bridge to every closed-system pipeline: compile nodes, certificate synthesis, and the whole battery machinery operate on one flat manifest, while modelling happens compositionally. The rule's two guarantees pull in opposite directions and are both load-bearing: wiring adds *structure*, never *expressiveness*. Never expressiveness: the flattened manifest's compiled semantics is exactly the composite's — at the compiled class this is definitional (a presentation's semantics IS its flattened object's), and at the finite-kernel tier it is a theorem: the monolithic chain's law is, at every horizon, the product of the component laws. Added structure: the flattened manifest remembers its components — the projections are exact simulation witnesses for free, so every component-level specification remains addressable on the flattened object through QSIMTRANSPORT.

What the judgment's discharge recomputes: the substitution itself — polynomial substitution plus guard-product bookkeeping — which is why the rule is admissible rather than core: it discharges against verified recomputation and the semantic witness pair, adding no new kernel content. Directed substitution wirings do *not* reduce to the independent synchronous product of the witness pair — wiring an upstream output into a downstream input creates conditional dependence the product law does not carry; they rest on the checker's substitution recompute itself (currying the upstream trajectory into a dependent downstream kernel), definitional in the compiled class.

What would break: component well-formedness (unit row sums) is what makes the projections exact witnesses — against a defective second factor, the first projection's pushed row picks up the leftover mass as a spurious scale factor, and the component-spec addressability claim fails. The scheduling hypotheses (clock rows, simultaneity, non-Zeno families) are deliberately premise classes cited by every composition rule rather than rows of this one.

**notes** The flattening premise is the judgment kind *QFlatEq*, baked in the extension theory; its discharge is the flatten recompute behind a registry seam, with the shared-carrier and binary-wiring confinements as the judgment's stated scope — refusal-shaped, never a hypothesis of this rule. The proven fragment spans two tiers: the finite-kernel tier, on the same substrate as the relational chain machinery — the tier where “law”, “product”, and “pushforward” are finite object-arithmetic objects — and the general measurable-kernel tier, where they are the Giry-monad objects (second bound block). The remaining confinement is the *independent binary product* witness pair — a design fence, not a gap: *n*-ary independent wirings iterate it, while directed wirings rest on the checker's substitution recompute (a dependent downstream kernel, not the independent product). The general compiled-class statement is substitution correctness, checker recomputation by design.

### Lean statement

BernsteinLogicAux.QtslCompose.Flatten . law\_prodKernel

statement bound

```
/-- **The flatten witness** (`QFlatten`'s "never expressiveness" face): the
flattened monolithic chain's law is, at every time, the product of the
component chains' laws — the composite presentation and its monolithic
manifest compile to the same semantics. -/
theorem law_prodKernel (K1 : α → α → ℝ) (K2 : β → β → ℝ)
  (μ1 : α → ℝ) (μ2 : β → ℝ) :
  ∀ n, law (prodKernel K1 K2) (prodDist μ1 μ2) n
    = prodDist (law K1 μ1 n) (law K2 μ2 n)
```

```

| 0 => rfl
| n + 1 => by
  rw [law_succ, law_prodKernel K1 K2 μ1 μ2 n, step_prodKernel, law_succ,
    law_succ]

```

section Projections

variable [DecidableEq α] [DecidableEq β]

```

/-- The added *structure*: the first projection is a simulation witness
(`QSimTransport.PushKernel`) from the flattened manifest onto component 1 –
for free, whenever component 2 is well-formed. -/
theorem pushKernel_fst {K1 : α → α → ℝ} {K2 : β → β → ℝ}
  (h2 : ∀ b, ∑ b', K2 b b' = 1) :
  PushKernel Prod.fst (prodKernel K1 K2) K1

```

BernsteinLogicAux.QtslCompose.Flatten . **pushKernel\_fst**

statement bound

```

/-- The added *structure*: the first projection is a simulation witness
(`QSimTransport.PushKernel`) from the flattened manifest onto component 1 –
for free, whenever component 2 is well-formed. -/
theorem pushKernel_fst {K1 : α → α → ℝ} {K2 : β → β → ℝ}
  (h2 : ∀ b, ∑ b', K2 b b' = 1) :
  PushKernel Prod.fst (prodKernel K1 K2) K1

```

BernsteinLogicAux.QtslCompose.Flatten . **pushKernel\_snd**

statement bound

```

/-- Mirror: the second projection simulates onto component 2. -/
theorem pushKernel_snd {K1 : α → α → ℝ} {K2 : β → β → ℝ}
  (h1 : ∀ a, ∑ a', K1 a a' = 1) :
  PushKernel Prod.snd (prodKernel K1 K2) K2

```

BernsteinLogicAux.QtslCompose.FlattenMeas . **law\_parallelComp**

statement bound

```

/-- **QFlatten, measurable-kernel tier – the flatten witness** ("never
expressiveness" face): the flattened monolithic chain's law is, at every
horizon, the product of the component chains' laws – the composite
presentation and its monolithic manifest compile to the same semantics.
(The measurable-tier face of `law_prodKernel`.) -/
theorem law_parallelComp (μ : Measure α) (ν : Measure β)
  (K : Kernel α α) (Q : Kernel β β)
  [SFinite μ] [SFinite ν] [IsSFiniteKernel K] [IsSFiniteKernel Q] :
  ∀ n, law (K ⋈ Q) (μ.prod ν) n = (law K μ n).prod (law Q ν n)
| 0 => rfl
| n + 1 => by
  rw [law_succ, law_parallelComp μ ν K Q n, bind_parallelComp,
    ← law_succ, ← law_succ]

/-! ## The added structure: projections are simulation witnesses -/

/-- The added *structure* ("adds structure" face): the first projection is a
simulation witness (`SimTransportMeas.PushKernel`) from the flattened
manifest onto component 1 – for free, consuming exactly the *sibling's*
Markov property (`[IsMarkovKernel Q]`: unit total mass per row) and nothing
of `K`'s own well-formedness. Margin-zero boundary, displayed by
`Measure.map_fst_prod`: against a defective sibling the pushed row is
`(Q p.2 univ) • K p.1` – the leak coefficient is the sibling's total mass.
(The measurable-tier face of the finite tier's `pushKernel_fst`.) -/
theorem pushKernel_fst {K : Kernel α α} {Q : Kernel β β}
  [IsSFiniteKernel K] [IsMarkovKernel Q] :
  PushKernel Prod.fst (K ⋈ Q) K

```

BernsteinLogicAux.QtslCompose.FlattenMeas . **pushKernel\_fst**

statement bound

```

/-- The added *structure* ("adds structure" face): the first projection is a
simulation witness (`SimTransportMeas.PushKernel`) from the flattened

```

```

manifest onto component 1 – for free, consuming exactly the *sibling's*
Markov property (`[IsMarkovKernel Q]`: unit total mass per row) and nothing
of `K`'s own well-formedness. Margin-zero boundary, displayed by
`Measure.map_fst_prod`: against a defective sibling the pushed row is
`(Q p.2 univ) • K p.1` – the leak coefficient is the sibling's total mass.
(The measurable-tier face of the finite tier's `pushKernel_fst`.) -/
theorem pushKernel_fst {K : Kernel  $\alpha$   $\alpha$ } {Q : Kernel  $\beta$   $\beta$ }
  [IsSFiniteKernel K] [IsMarkovKernel Q] :
  PushKernel Prod.fst (K  $\mathbb{I}_k$  Q) K

```

BernsteinLogicAux.QtslCompose.FlattenMeas . **pushKernel\_snd**

statement bound

```

/-- Mirror: the second projection simulates onto component 2, consuming
exactly the first component's Markov property. -/
theorem pushKernel_snd {K : Kernel  $\alpha$   $\alpha$ } {Q : Kernel  $\beta$   $\beta$ }
  [IsMarkovKernel K] [IsSFiniteKernel Q] :
  PushKernel Prod.snd (K  $\mathbb{I}_k$  Q) Q

```

BernsteinLogicAux.QtslCompose.FlattenMeas . **bind\_parallelComp**

statement bound

```

/-- One step of the flattened (synchronous-product) chain factors as the
product of the component steps. (The measurable-tier face of
`step_prodKernel`.) Proof is pure ` $\circ_m$ `-algebra:
`Measure.prod_comp_right`/`prod_comp_left` are the two one-sided
measure-level product-bind identities, fused by
`Kernel.parallelComp_comp_parallelComp`. -/
theorem bind_parallelComp ( $\mu$  : Measure  $\alpha$ ) ( $\nu$  : Measure  $\beta$ )
  (K : Kernel  $\alpha$   $\alpha$ ) (Q : Kernel  $\beta$   $\beta$ )
  [SFinite  $\mu$ ] [SFinite  $\nu$ ] [IsSFiniteKernel K] [IsSFiniteKernel Q] :
  ( $\mu$ .prod  $\nu$ ).bind (K  $\mathbb{I}_k$  Q) = ( $\mu$ .bind K).prod ( $\nu$ .bind Q)

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed judgment premise  $\mathbf{QFlatEq}(\text{sys}, \text{flat})$  — the flattened manifest has the *same* compiled semantics, each projection  $\pi_i$  a free  $\mathbf{QSIMTRANSPORT}$  witness —  $\mapsto$  the theorem pair at the finite-kernel tier:  $\text{law\_prodKernel}$  (never expressiveness) and  $\text{pushKernel\_fst}/\text{snd}$  (added structure). The kind's discharge (the substitution recompute) has *no* Lean counterpart by design: at the compiled class the presentation's semantics IS its flattened object's, so that discharge is checker recomputation, and what is theorem-shaped is exactly what is bound. The rule's own transport step (a verdict on  $\mathcal{M}_{\text{flat}}$  is a verdict on  $\text{sys}$ ) is substitution of equals along the judgment — no separate statement to bind.

- $\text{SysWire}(w, \text{Sys}_1, \dots, \text{Sys}_n) \mapsto$  the *binary synchronous product*: kernels  $K_1 : \alpha \rightarrow \alpha \rightarrow \mathbb{R}$ ,  $K_2 : \beta \rightarrow \beta \rightarrow \mathbb{R}$  over finite types, flattened as  $\text{prodKernel}$  on  $\alpha \times \beta$ . Two confinements at this block, both design fences: binary-independent (directed and  $n$ -ary substitution wirings rest on the checker's substitution recompute — currying the upstream trajectory into a *dependent* downstream kernel, which is not the independent product bound here; per the module header), and finite-kernel tier ( $[\text{FIntype } \alpha] [\text{FIntype } \beta]$ ) — the latter discharged by the measurable-tier block below, leaving the independent binary product as this bound set's remaining confinement.
- same compiled semantics  $\mapsto$   $\text{law\_prodKernel}$ :  $\text{law} (\text{prodKernel } K_1 K_2) (\text{prodDist } \mu_1 \mu_2) n = \text{prodDist} (\text{law } K_1 \mu_1 n) (\text{law } K_2 \mu_2 n)$  at every  $n$  — and its binder row is the width fact: the identity is proven for *arbitrary* kernel and law data, no Markov/distribution well-formedness consumed — pure bilinear algebra, strictly wider than the display's well-formed-system reading, safe direction.
- projections as free  $\mathbf{QSIMTRANSPORT}$  witnesses  $\mapsto$   $\text{pushKernel\_fst} : \text{PushKernel Prod.fst} (\text{prodKernel } K_1 K_2) K_1$  — consuming only  $\forall b, \sum b', K_2 b b' = 1$ : the *sibling* component's unit row sums, nothing of its own, and not even nonnegativity — the certified statement isolates exactly which well-formedness fragment the commentary's “what would break” paragraph names (a defective second factor leaks scale into the pushed row).  $\text{snd}$  mirrors.  $[\text{DecidableEq}]$  binders enter only here (the pushforward's fibre sums).

*Measurable-kernel tier* (second bound block, module FlattenMeas, namespace  $\dots\text{QtslCompose.Meas}$ ): the product is mathlib's own pair —  $K \mathbb{I}_k Q$  ( $\text{Kernel.parallelComp}$ ) over  $\text{Measure.prod}$  — and the theorem pair transposes name-for-name:  $\text{law\_parallelComp}$  (never expressiveness:  $\text{law} (K \mathbb{I}_k Q) (\mu.\text{prod } \nu) n = (\text{law } K \mu n).\text{prod} (\text{law } Q \nu n)$  at every horizon, over  $\text{SimTransportMeas}'s$

Giry-bind law) and `pushKernel_fst/_snd` (added structure: projections as `Meas.PushKernel` witnesses), with `bind_parallelComp` the one-step glue. Binder floor, the honest analogue of the finite tier's “arbitrary data” width fact: `SFinite/IsSFiniteKernel` — product-measure theory is s-finite theory — still strictly wider than the display's Markov/probability reading, same safe direction. And the “what would break” margin-zero witness is displayed by `mathlib` itself: `Measure.map_fst_prod` gives the pushed row as  $(\nu \text{ univ}) \cdot \mu$  — leak coefficient the sibling's total mass — with `pushKernel_fst` consuming exactly `[IsMarkovKernel Q]` and nothing of `K`'s own well-formedness, not even finiteness. Width note: the clock rows (schedule/simultaneity/non-Zeno) are premise classes deliberately absent from both display and bound statements; the closure facts at both tiers live in their modules but outside this bound set — cited, not certified here: `finite isMarkov_prodKernel`, `isDist_prodDist`, `push_fst_prodDist`; measurable `sfinite_law`, `isProbabilityMeasure_law`, the addressability compositions `sim_expect_fst/_snd` (component specs consumed end-to-end through `QSIMTRANSPORT`'s measurable engine), the bridge `pushKernel_iff_stochSim`, and the `mathlib` instances `IsMarkovKernel (K  $\mathbb{K}$  Q)` / `prod.instIsProbabilityMeasure`.

## Lean verifier receipts

module `BernsteinLogicAux.QtslCompose.Flatten`; bound build `30ed43793772...` (bound from receipt `8cd9a31cf76c1595`, 2026-07-19T18:24:49Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| e4496ce5107a3441 | 2026-07-19T03:38:56Z | build+decorrelated                 | current |
| 8cd9a31cf76c1595 | 2026-07-19T18:24:49Z | build+decorrelated                 | current |
| 62e3b3d615f78f37 | 2026-07-20T01:22:02Z | build+decorrelated                 | current |
| b45a913927ac5488 | 2026-07-20T20:46:52Z | build+decorrelated                 | current |
| d52960f0b18dc76a | 2026-07-20T20:58:26Z | build+replay+hardened+decorrelated | current |
| 0eabb0d2dfbc00c  | 2026-07-21T00:03:55Z | build+decorrelated                 | current |
| 1f0f373238ef2c2b | 2026-07-21T05:39:11Z | build+replay+hardened+decorrelated | current |
| 35d0f3c7c71856fc | 2026-07-22T18:27:16Z | build+decorrelated                 | current |
| 53dde5ffb738ab3a | 2026-07-31T22:00:14Z | build+decorrelated                 | current |
| 50f8cfda2d25e116 | 2026-07-31T22:29:08Z | build+decorrelated                 | current |
| edc0e93a8721d195 | 2026-08-02T23:31:19Z | build+decorrelated                 | current |
| a6fae671c06c5ebb | 2026-08-03T16:27:23Z | build+decorrelated                 | current |
| 0427fd20db259dcd | 2026-08-03T22:46:07Z | build+decorrelated                 | current |

module `BernsteinLogicAux.QtslCompose.FlattenMeas`; bound build `40d14da6aacd...` (bound from receipt `4747724a289473cd`, 2026-07-20T08:24:46Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 4747724a289473cd | 2026-07-20T08:24:46Z | build+decorrelated                 | current |
| 2a11a368eea6adc5 | 2026-07-20T20:46:54Z | build+decorrelated                 | current |
| d11be966df6a113  | 2026-07-20T20:58:27Z | build+replay+hardened+decorrelated | current |
| ba9ffebde141f18b | 2026-07-21T00:03:56Z | build+decorrelated                 | current |
| 13a1681b1015f8c2 | 2026-07-21T05:39:13Z | build+replay+hardened+decorrelated | current |
| 89eb1271b74a3f48 | 2026-07-22T18:27:16Z | build+decorrelated                 | current |
| d86334f5a6f3694d | 2026-07-31T22:00:15Z | build+decorrelated                 | current |
| 4aceb786dcde76f2 | 2026-07-31T22:29:09Z | build+decorrelated                 | current |
| e040a1347e8cd2a0 | 2026-08-02T23:31:20Z | build+decorrelated                 | current |
| 52c94a6a75909deb | 2026-08-03T16:27:23Z | build+decorrelated                 | current |
| 0a269083e637c33e | 2026-08-03T22:46:07Z | build+decorrelated                 | current |

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.



## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development checks the base calculus's wire format (theory `bern_catalogue`). QFLATTEN operates on system presentations of the extension theory — baked, System stratum and all; its semantic witnesses live in the auxiliary Lean package. No HOL development consumes the extension theory's rows; the box becomes writable when a flatten recompute joins a ported checker chain.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: admissible rule (tier 2) — a proof-tree-rewriting recipe of the reduction-to-verified-machinery kind: the node discharges against the checker's substitution recompute plus a semantic witness pair, not against catalogue nodes. Scope, stated loudly: the witness pair is proven at the finite-kernel tier, on the synchronous-product wiring; the general compiled-class statement is substitution correctness, definitional by the presentation semantics.

**Recipe R82.1** (the QFLATTEN rewrite). A QFLATTEN node transports satisfaction across the flattening judgment — *flat* the monolithic manifest of a composite presentation:

$$\frac{\text{QFlatEq}(\text{sys}, \text{flat}) \quad \mathcal{M}_{\text{flat}}, x_0 \in R \models \varphi}{\text{sys}, x_0 \in R \models \varphi} \text{QFLATTEN} \quad \text{the checker replays the substitution.}$$

The rewrite target contains zero catalogue nodes: a presentation's compiled semantics *is* its flattened object's, so at the compiled class the judgment's discharge is a recomputation — polynomial substitution plus guard-product bookkeeping — and the rule adds no kernel semantic content (the admissibility criterion). What deserves proof, and gets it below, is the slogan the rule is trusted for: **wiring adds structure, never expressiveness**. Directed substitution wirings do *not* reduce to the independent synchronous product treated below — wiring an upstream output into a downstream input creates conditional dependence the product law does not carry — so the flatten witness below attests the independent-product case, and the directed case rests on the substitution recompute (the judgment's discharge above) alone.

**Proposition R82.2** (never expressiveness: the flatten witness). *Let  $K_1, K_2$  be finite kernels with initial laws  $\mu_1, \mu_2$ , and let the flattened manifest be the product kernel  $K_{\otimes}((a, b), (a', b')) = K_1(a, a') K_2(b, b')$  with initial law  $(\mu_1 \otimes \mu_2)(a, b) = \mu_1(a) \mu_2(b)$ . Then at every horizon  $n$ ,*

$$\text{law}_{K_{\otimes}}(\mu_1 \otimes \mu_2, n) = \text{law}_{K_1}(\mu_1, n) \otimes \text{law}_{K_2}(\mu_2, n) :$$

*the monolithic chain's law is, at every time, exactly the product of the component laws — the composite presentation and its flattened manifest compile to the same semantics.*

*Proof.* One step factors, by Fubini for finite sums and bilinearity:

$$\text{step}_{K_{\otimes}}(\mu_1 \otimes \mu_2)(a', b') = \sum_{a, b} \mu_1(a) \mu_2(b) K_1(a, a') K_2(b, b') = \left( \sum_a \mu_1(a) K_1(a, a') \right) \left( \sum_b \mu_2(b) K_2(b, b') \right),$$

i.e.  $\text{step}_{K_{\otimes}}(\mu_1 \otimes \mu_2) = \text{step}_{K_1} \mu_1 \otimes \text{step}_{K_2} \mu_2$ . Induction on  $n$  closes. Closure rides along: the product of kernels is a kernel and the product of laws is a law (nonnegativity termwise; unit masses by the same bilinear factorization —  $\sum_{a, b} \mu_1(a) \mu_2(b) = 1 \cdot 1$  for the product law,  $\sum_{a', b'} K_1(a, a') K_2(b, b') = 1 \cdot 1$  per product row).  $\square$

**Proposition R82.3** (added structure: the projections are free simulation witnesses). *If  $K_2$  is row-stochastic, the first projection is an exact intertwining witness from  $K_{\otimes}$  onto  $K_1$ :  $\pi_{1\#} K_{\otimes}((a, b), \cdot) = K_1(a, \cdot)$  for every product state  $(a, b)$ ; symmetrically for  $\pi_2$  under row-stochastic  $K_1$ ; and  $\pi_{1\#}(\mu_1 \otimes \mu_2) = \mu_1$  whenever  $\mu_2$  has unit mass.*

*Proof.*  $\pi_{1\#} K_{\otimes}((a, b), \cdot)(a') = \sum_{b'} K_1(a, a') K_2(b, b') = K_1(a, a') \sum_{b'} K_2(b, b') = K_1(a, a')$ , the last step by the unit row sum. The initial-law statement is the same computation at  $\mu$ . Consequence: the flattened manifest *remembers* its components — each projection is a free exact-intertwining (PushKernel) witness, the QSIMTRANSPORT witness form, instantiating the strict (zero-degradation) column of the QDISTTRANSPORT matrix — so every component-level specification remains addressable on the flattened object with zero degradation. SysWire adds these handles and nothing else: Proposition R82.2 caps expressiveness, this proposition delivers the structure.  $\square$

**Remark R82.4** (why the well-formedness hypotheses are not decoration). Both directions consume them. In Proposition R82.3, drop the unit row sum of the *other* factor and the projection witness fails: the pushed row picks up  $\sum_{b'} K_2(b, b')$  as a spurious scale factor, so a sub-stochastic (leaky) second component would make every component-1 judgment transport against a deflated law — the component-spec addressability claim silently weakens. In Proposition R82.2, unit masses are what make the product law a probability law at every time. The scheduling hypotheses of composite dynamics — clock rows, simultaneity, non-Zeno families — are deliberately *premise classes* cited by every composition rule rather than rows of this one: flattening is about state and kernel structure, and importing the clock discipline here would duplicate its single authoritative statement.

*Remark* R82.5 (the tier boundary, honestly). The propositions above are the synchronous-product instance at the finite-kernel tier — the tier where law, product, and pushforward are finite exact-arithmetic objects, shared with the relational-chain machinery. General wirings (directed substitution, guard products) reduce by the checker’s own recompute; the reduction is definitional in the compiled class and is not re-proven as a theorem here. Nothing above claims the measure/kernel tier; its flatten-commutes statements — the same theorem pair over Markov kernels and product measures — are mechanized and bound in this rule’s entry, cited there rather than re-walked here.

| proof step                      | formal anchor                                                                                                                                                       |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Recipe R82.1 judgment discharge | the substitution recompute (polynomial substitution + guard-product book-keeping; checker-side, behind the flattening judgment’s registry seam)                     |
| Proposition R82.2               | prodKernel; prodDist; step_prodKernel; law_prodKernel; closure isMarkov_prodKernel, isDist_prodDist                                                                 |
| Proposition R82.3 consumed by   | pushKernel_fst; pushKernel_snd; push_fst_prodDist<br>QSIMTRANSPORT (the PushKernel witness form); the strict column of QDIST-TRANSPORT’s matrix (strict_cell_exact) |

## Rule 83. QLinGrowthDischarge (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QLinGrowthDischarge*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{\text{env}_i : B \vdash \pm f_i(x) \leq a + b \|x\|}{\models \text{QLinGrowth}(f, a, b, B)}$$

QLinGrowthDischarge concludes the linear-growth-envelope kind with the constants carried: membership of a polynomial vector field in the envelope  $(a, b)$  over a scope box, from per-coordinate affine-envelope sign sequents — exact-rational, checker-replayed. Downstream cells consume the pair  $(a, b)$  arithmetically, never a membership bit: the kind exists precisely so that growth constants compose.

**notes** Entered at the roster census (baked, never catalogued); no Lean statement binds it yet. The displayed norm form is schematic — the envelope rows' own sequent shape is the authority.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

theory Qtsl . QLinGrowth

quoted from the theory source at render

| QLinGrowth : Set;

theory Qtsl . QLinGrowth/\*

quoted from the theory source at render

| QLinGrowth/judgment : QLinGrowth -> QJudgment;  
| QLinGrowth/f : QLinGrowth -> PolyVec;  
| QLinGrowth/a : QLinGrowth -> Q;  
| QLinGrowth/b : QLinGrowth -> Q;  
| QLinGrowth/scope : QLinGrowth -> Box;

theory Qtsl . QLinGrowthDischarge

quoted from the theory source at render

| QLinGrowthDischarge : Set;

theory Qtsl . QLinGrowthDischarge/\*

quoted from the theory source at render

| QLinGrowthDischarge/proof : QLinGrowthDischarge -> QProof;  
| QLinGrowthDischarge/envelopes : QLinGrowthDischarge -> QProofList;

### Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

### HOL4 statement

#### NO HOL STATEMENT YET

No HOL development of the extension theory exists.

### HOL verifier receipts

#### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), the discharge species. This section states the rule's soundness obligation exactly, per the catalogue's completeness-over-polish charter (a rule that can only be named-and-gapped is still a row); the discharge is open debt on the mechanization column, and the schema quoted in the entry is the baked ground truth.

**Soundness obligation.** The per-coordinate affine-envelope sign sequents on the scope box jointly certify membership of the field in the linear-growth envelope with constants  $(a, b)$ ; the kind carries the constants so that downstream consumers compose them arithmetically. The obligation is that the envelope rows' sequent semantics entail the envelope inequality coordinatewise on the scope box — a per-row sign-sequent instantiation.

*Remark* R83.1 (status). No Lean statement binds this node. The displayed norm form is schematic; the envelope rows' own sequent shape (exact-rational, checker-replayed) is the authority.

## Rule 84. QMARGINALCOMPAT (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QMarginalCompat*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{\overline{\text{dom}_k} \text{ (per-contributor dominance rows)} \quad \text{qtsl\_stochwire\_shape}(cl, sp)}{\models \text{QMarginalCompat}(sp, cl)}$$

QMARGINALCOMPAT packages marginal compatibility for stochastic wires: a belief names convex-analytic content (the indicator of a named cell set, or a registered function class), a stochastic wire carries one with its convexity, lower-semicontinuity, and properness row kits, and this node collects the per-contributor dominance rows over a shared sample space. The contributor list's shape agreement against the space is an admission demand, discharged by the shape walk quoted below.

notes Entered at the roster census (baked, never catalogued); no Lean statement binds it yet; display is schema-faithful schematic.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

theory Qtsl.QMarginalCompat

quoted from the theory source at render

QMarginalCompat : Set;

theory Qtsl.QMarginalCompat/\*

quoted from the theory source at render

QMarginalCompat/proof : QMarginalCompat -> QProof;  
QMarginalCompat/space : QMarginalCompat -> Shape;  
QMarginalCompat/contributors : QMarginalCompat -> QStochContribList;  
QMarginalCompat/dominance\_rows : QMarginalCompat -> QProofList;

theory Qtsl.qtsl\_stochwire\_shape

quoted from the theory source at render

qtsl\_stochwire\_shape : {contribs: QStochContribList, space: Shape}  
-> Prop; // side-condition

theory Qtsl.ax/wf/stochwire\_shape

quoted from the theory source at render

report ax/wf/stochwire\_shape : forall m : QMarginalCompat,  
sp : Shape, cl : QStochContribList.  
m QMarginalCompat/space = sp,  
m QMarginalCompat/contributors = cl  
|- {contribs: cl, space: sp} qtsl\_stochwire\_shape;

### Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

### HOL4 statement

**NO HOL STATEMENT YET**

No HOL development of the extension theory exists.

### HOL verifier receipts

**NONE**

no HOL verification receipt exists for this rule.

**Proof**

Genre: core rule (tier 1), stochastic wires. This section states the rule's soundness obligation exactly, per the catalogue's completeness-over-polish charter (a rule that can only be named-and-gapped is still a row); the discharge is open debt on the mechanization column, and the schema quoted in the entry is the baked ground truth.

**Soundness obligation.** The per-contributor dominance rows, over a shared sample space whose shape agreement with the contributor list is an admission demand, jointly certify marginal compatibility of the contributors' beliefs — the packaging judgment consumed by stochastic wire composition. The obligation is that dominance per contributor entails compatibility of the marginals on the shared space.

*Remark* R84.1 (status). No Lean statement binds this node; the convex-analytic kits (convexity, lower semicontinuity, properness) ride the stochastic wire's own rows and are consumed here by citation.

## Rule 85. QAggReduce (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — *QAggReduce*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\begin{aligned}
 &(\vec{s}, \vec{k}) \in K \vdash G(\vec{s}, \vec{k}) \leq 0 \text{ (one BERNcone-family sequent over the aggregate coordinates)} \\
 &\quad s_j^P := \sum_{t:P} \phi_j(c_t), \quad k_P := \#P \text{ (the aggregate coordinates, defined)} \\
 &\quad \{\vdash l_j k_P \leq s_j^P \leq u_j k_P\}_{j,P} \text{ (coupling rows; } [l_j, u_j] \text{ a Bernstein enclosure of } \phi_j \text{ on } B_P) \\
 &\quad \{\vdash 0 \leq k_P \leq M_P\}_P \text{ (count rows)} \\
 &\{\vdash y^\top \vec{k} = y^\top \vec{k}(x_0)\}_{y \in Y} \text{ (P-invariant conservation rows: each cited witness } y \geq 0, y^\top C = 0 \text{ verified against the uncoloured skeleton's incidence)} \\
 &\quad \{\vdash (s_\alpha^P)^2 \leq k_P s_{2\alpha}^P\}_{\alpha,P} \text{ (moment rows, optional tightening)} \quad \forall x \in \text{scope}, \forall t:P \text{ in } x : c_t \in B_P \text{ (colour-box)} \\
 &\quad x \in \text{scope} \vdash G(\vec{s}(x), \vec{k}(x)) \leq 0
 \end{aligned}$$

the defined aggregate coordinates replace the certificate's feature dictionary in the display: the dictionary is certificate data (a shared reference) whose entire displayed content is the two defining equations, with  $\phi_j$  cited by the coupling rows. The enclosure box discharge is a side condition. The checker never sees a marking: the multiset quotient is compiled away producer-side, aggregate dimension independent of the marking bound  $M$ ; every row family is valid at every reachable in-scope marking under its stated premises (the moment and count-lower rows unconditionally; the count-upper row under the marking bound  $M$ ; conservation rows by their verified witness vectors, the conserved value pinned at the initial marking (the rows ride the count-row proof list — no dedicated conservation field exists on the baked node, a named schema divergence); coupling rows under their per-token bounds), so adding in-scope rows only shrinks  $K$  and never cuts a reachable in-scope aggregate point.

QAggReduce is the aggregation rule for nets whose state is a *marking* — a multiset of coloured tokens distributed over places, with variable dimension and a mandatory exchangeability quotient (tokens within a place are interchangeable, so any legitimate observable is a symmetric function of each place's colour multiset). It replaces an obligation quantified over all in-scope markings by a single polynomial obligation over finitely many *aggregate* coordinates: token counts  $k_P$  and pooled features  $s_j^P = \sum_{t:P} \phi_j(c_t)$  for a finite feature dictionary. A producer reaches for it because the alternative — flattening marking shapes into modes — explodes combinatorially in the token bound and breaks the symmetry; here obligation count and dimension are independent of the marking bound, which enters only as a constant in count rows.

What the checker recomputes: one hypothesis-carrying polynomial sequent  $G \leq 0$  over the aggregate box, whose hypothesis rows assemble the containment certificate for a polytope  $K$  covering every in-scope marking's aggregate image — coupling rows  $l_j k_P \leq s_j^P \leq u_j k_P$  from enclosures of each feature on its place's colour box, count rows  $0 \leq k_P \leq M_P$  plus linear conservation laws  $y^\top \vec{k} = y^\top \vec{k}(x_0)$  (place invariants [186], the conserved value pinned at the initial marking; each cited witness  $y \geq 0, y^\top C = 0$  checker-verified against the uncoloured skeleton's incidence matrix), and optional moment rows  $(s_\alpha^P)^2 \leq k_P s_{2\alpha}^P$  (Cauchy–Schwarz over the bag). Soundness of the reduction is evaluation: if the aggregate image lands in  $K$  and  $G \leq 0$  on  $K$ , the marking-quantified obligation holds pointwise. The whole content is the containment certificate, with each row family proven valid at its stated scope: nonnegativity and moment rows for *every* bag outright; coupling rows for every bag under their colour-scope premise (the box discharge); count bounds and conservation rows as certified reachability facts. Under those side conditions, adding a row shrinks  $K$  without ever cutting a reachable aggregate point.

What would break: a row NOT valid for every reachable marking makes  $K$  fail containment and the conclusion simply false — the central instance being the coupling rows' scope premise, since an enclosure of  $\phi_j$  on the colour box says nothing about tokens outside it; the box must be *discharged* (exit-guard transitions, an invariance proof, or stopped-process semantics), never assumed. The converse failure is honest: not every polytope point is a realizable marking, so  $K$  over-approximates — a completeness cost, refusal-shaped and never soundness-shaped, with the row menu as its dial.

**notes** The operative aggregate is the measurable-query observable  $\langle \text{marking}, f \rangle$  over bags of colours; counts and pools are the  $f \equiv 1$  and  $f = \text{id}$  degenerate queries, and a query is a semiring contraction of the count tensor against  $f$  along the colour axis. Semantically the pools are commutative-monoid folds along the tokens-to-places classifying map, with the multiset as the universal aggregate [222]; the dictionary's expressive ceiling per shape is the classical generation of multisymmetric invariants by polarized power sums [238]. The coupling rows here are deterministic pool-count arithmetic within one marking — *not* probabilistic couplings; the identically-named rows of QCOUPLEDRIFT are joint-kernel marginal data over a product of chains. Same word, disjoint mathematics, neither cites the other.

### Lean statement

BernsteinLogicAux.QtslCompose.AggReduce . **aggReduce**

statement bound

```

/-- **QAggReduce, semantic form** ("proof: evaluation"): if `G ≤ 0` on `K`
and every in-scope point's aggregate vector lands in `K`, then the

```



```

marking-quantified obligation holds. `agg` is the compiled aggregate map
`x ↦ (s(x), K(x))` — its pointwise containment (`hin`) is what the
coupling/count/moment rows + the P-invariant conservation laws certify. -/
theorem aggReduce {X A : Type*} (agg : X → A) (scope : Set X) (K : Set A)
  (G : A → ℝ) (hG : ∀ a ∈ K, G a ≤ 0) (hin : ∀ x ∈ scope, agg x ∈ K) :
  ∀ x ∈ scope, G (agg x) ≤ 0

```

BernsteinLogicAux.QtslCompose.AggReduce.coupling\_row

statement bound

```

/-- **The coupling row is valid for every marking** (NOT a probabilistic
coupling — see the module docstring's fence): if every token's `f`-value
lies in `[l, u]`, the pool-count coupling `l · k ≤ s ≤ u · k` holds. Adding
these rows to the polytope `K` never cuts a reachable aggregate point. -/
theorem coupling_row {m : Multiset C} {f : C → ℝ} {l u : ℝ}
  (hl : ∀ c ∈ m, l ≤ f c) (hu : ∀ c ∈ m, f c ≤ u) :
  l * m.card ≤ query m f ∧ query m f ≤ u * m.card

```

BernsteinLogicAux.QtslCompose.AggReduce.count\_row\_le

statement bound

```

/-- The count row's upper half, from any marking bound (the `M_P` bound is a
reachability/P-invariant fact, consumed as a hypothesis — the box-discharge
side condition [D10], not proven here). -/
theorem count_row_le {m : Multiset C} {M : ℕ} (h : m.card ≤ M) :
  (m.card : ℝ) ≤ M

```

BernsteinLogicAux.QtslCompose.AggReduce.count\_row\_nonneg

statement bound

```

/-- The count row's lower half: counts are nonnegative. -/
theorem count_row_nonneg (m : Multiset C) : (0 : ℝ) ≤ m.card

```

BernsteinLogicAux.QtslCompose.AggReduce.moment\_row

statement bound

```

/-- **The moment row is valid for every marking**: `(m, f)² ≤ k · (m, f²)` —
Cauchy-Schwarz over the bag (the rule display's `(s_α)² ≤ k · s_{2α}`,
with `s_α = (m, x^α)` instantiated at `f = x^α`). -/
theorem moment_row (m : Multiset C) (f : C → ℝ) :
  (query m f) ^ 2 ≤ m.card * query m (fun c => f c ^ 2)

```

BernsteinLogicAux.QtslCompose.AggReduce.query\_const\_one

statement bound

```

/-- Counts are the `f ≡ 1` degenerate query (the 14:20 correction's
normal form: `k_P = (m, 1)`). -/
theorem query_const_one (m : Multiset C) :
  query m (fun _ => (1 : ℝ)) = m.card

```

BernsteinLogicAux.QtslCompose.AggReduce.query\_count

statement bound

```

/-- **The contraction-row form** (the γ-trace): the query observable is the
count tensor contracted against `f` along the color axis. -/
theorem query_count [DecidableEq C] (m : Multiset C) (f : C → ℝ) :
  query m f = ∑ c ∈ m.toFinset, (m.count c : ℝ) * f c

```

## Correspondence (Steele display ↔ Lean)

reviewer-certified against the displayed texts

The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed conclusion  $\forall x \text{ in scope: } G(\vec{s}(x), \vec{k}(x)) \leq 0 \mapsto \text{aggReduce's goal } \forall x \in \text{scope, } G(\text{agg } x) \leq 0$ , with the displayed premise  $\vdash G \leq 0$  on  $K \mapsto hG$  and the entire row apparatus  $\mapsto \text{hin} : \forall x \in \text{scope, } \text{agg } x \in K$  — the containment certificate, which the coupling/count/moment/conservation rows exist to assemble. The binder row is the biggest width fact on this page:  $\text{aggReduce}$  is stated for *any* types  $X$   $A$ , any  $\text{agg} : X \rightarrow A$ , any  $K : \text{Set } A$ , any  $G : A \rightarrow \mathbb{R}$  — no polynomiality of  $G$ , no polytopality of  $K$ , no aggregate box — so the certified semantic core is strictly wider than the displayed polynomial instance, safe direction (proof by evaluation needs none of the structure the checker needs).

- coupling rows  $l_j k_P \leq s_j^P \leq u_j k_P \mapsto \text{coupling\_row}$ : hypotheses  $\forall c \in m, l \leq f c$  and  $f c \leq u$  — the enclosure quantified over tokens *in the bag*, which is precisely where the box-discharge side condition lands in the certified statement (an enclosure on the colour box reaches the hypothesis only for markings confined to the box; the flavor text carries this as the box-discharge side condition, the statement as the membership guard).
- count rows  $0 \leq k_P \leq M_P \mapsto \text{count\_row\_nonneg}$  (unconditional) +  $\text{count\_row\_le}$ , whose bound  $m.\text{card} \leq M$  is a *hypothesis* — the  $M_P$  bound is a reachability/P-invariant fact, consumed, never proven. The displayed conservation rows  $y^\top \vec{k} = y^\top \vec{k}(x_0)$  (P-invariants, the conserved value pinned at the initial marking; witnesses  $y \geq 0, y^\top C = 0$  checker-verified) have no bound Lean counterpart at all: they are reachability facts of the net, outside this module's every-bag warrants. On the baked node they ride the count-row proof list — no dedicated conservation field exists, a named schema divergence.
- moment rows  $(s_\alpha^P)^2 \leq k_P s_{2\alpha}^P \mapsto \text{moment\_row}$ , unconditional for every bag (Cauchy–Schwarz by induction), instantiated at  $f = x^\alpha$ .
- the defined aggregate coordinates  $s_j^P := \sum_{t:P} \phi_j(c_t)$  and  $k_P := \#P \mapsto$  per-query functions  $f : C \rightarrow \mathbb{R}$ , arbitrary — polynomiality of the  $\phi_j$  is producer-side;  $\text{query\_count}$  certifies the contraction-row form  $\langle m, f \rangle = \sum_c \text{count}(c) \cdot f(c)$  (the  $\gamma$ -trace) under  $[\text{DecidableEq } C]$ , and  $\text{query\_const\_one}$  pins counts as the  $f \equiv 1$  degenerate query.

Width note: every bound row warrant is per-bag arithmetic — the multiset quotient is already taken (Multiset  $C$  IS the exchangeability-quotiented object), so the display's “the checker never sees a marking” has as its Lean face that no statement here mentions places, nets, or reachability; scope containment  $\text{hin}$  is where all of that enters, as one hypothesis. The conditional mark's content, at its exact extent: the count-bound and conservation rows are reachability facts of the net — consumed as hypotheses, never proven here — and a coupling row's colour-box confinement must be discharged by the producer (exit-guard transitions, an invariance certificate, or stopped-process semantics), never assumed. Every other row family is warranted for every bag outright.

## Lean verifier receipts

module BernsteinLogicAux.QtslCompose.AggReduce; bound build be9dbd259553... (bound from receipt bd4d8aca0400b822, 2026-07-21T04:51:40Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 5f6de8c6a1ff5236 | 2026-07-21T01:51:28Z | build+decorrelated                 | current |
| bd4d8aca0400b822 | 2026-07-21T04:51:40Z | build+decorrelated                 | current |
| c37acfa586210ee  | 2026-07-21T05:39:09Z | build+replay+hardened+decorrelated | current |
| 24c5f02176c0ee6a | 2026-07-22T18:27:15Z | build+decorrelated                 | current |
| 1eaf40d323232bf8 | 2026-07-31T22:00:14Z | build+decorrelated                 | current |
| 47734200303b43f0 | 2026-07-31T22:29:07Z | build+decorrelated                 | current |
| 712878ff235d6c76 | 2026-08-02T23:31:19Z | build+decorrelated                 | current |
| 36b853e33e482b54 | 2026-08-03T16:27:22Z | build+decorrelated                 | current |
| cb8f537f8267cf3e | 2026-08-03T22:46:06Z | build+decorrelated                 | current |

6 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development checks the base calculus's wire format (theory `bern_catalogue`). `QAGGREduce` is an extension-edge semantic rule whose node is baked in the extension theory; what reaches the base checker after the compile is ordinary hypothesis-carrying `BERNCONE` sequents, which the HOL side already covers as sequents without knowing their aggregation provenance. A HOL statement *about the rule* becomes writable when a checker consuming the extension theory's rows is ported.

## HOL verifier receipts

**NONE**

no HOL verification receipt exists for this rule.

**Proof**

Genre: core rule (tier 1). Scope, stated loudly: the semantic core (the aggregate-space reduction) is proven, and the row-warrant families are proven at their stated scopes — nonnegativity and moment rows for every bag, coupling rows under their colour-scope premise, while count upper bounds and place-invariant rows enter as certified in-scope reachability facts, not bagwise identities; the marking-space compile itself — producing the aggregate map and its rows from a net — is checker and compiler recomputation, outside these statements. Every hypothesis is used and remarked on.

**Setting and claim.** The state of a coloured net is a *marking*: for each place  $P$ , a finite multiset of colours drawn from a compact rational box  $B_P \subset \mathbb{R}^{n_P}$  — so the state space is a disjoint sum, over count vectors, of symmetric powers, with variable dimension and a mandatory exchangeability quotient (a “function of the state” that reads token identities is not a function on the quotient at all). The certificate language works in *aggregate coordinates*: counts  $k_P = \#P$  and pooled features  $s_j^P = \sum_{t:P} \phi_j(c_t)$  for a finite dictionary  $\Phi = \{\phi_j\}$  of polynomials. An obligation arrives as “for every in-scope marking  $x$ :  $G(\vec{s}(x), \vec{k}(x)) \leq 0$ ” with  $G$  polynomial. The rule replaces marking quantification by one polynomial sequent over a polytope  $K$  in aggregate space, whose hypothesis rows are the containment certificate. The claim (aggReduce + the row warrants): the reduction is sound, and every row family below — each under its stated scope premise — may be added to  $K$ ’s description without ever cutting a reachable aggregate point.

**Lemma R85.1** (queries, counts, and the contraction form). *For a bag  $m$  of colours and  $f : \text{colour} \rightarrow \mathbb{R}$ , write  $\langle m, f \rangle = \sum_{c \in m} f(c)$  (with multiplicity). Then  $\langle m, 1 \rangle = \#m$ , and  $\langle m, f \rangle = \sum_{c \in \text{supp}(m)} \text{count}_m(c) \cdot f(c)$  — the count tensor contracted against  $f$  along the colour axis.*

*Proof.* Both are fibre decompositions of a finite sum: group the terms of  $\sum_{c \in m} f(c)$  by value of  $c$ . Counts and pools are thus the  $f \equiv 1$  and coordinate-projection instances of one observable class, and every axis of the contraction is either summed out (colour) or an output (the query index) — there is no un-read axis, which is why, for the chosen finite dictionary, aggregate dimension is the dictionary’s index set: set by the dictionary, never by the marking.  $\square$

**Lemma R85.2** (coupling rows are valid for every bag). *If  $l \leq f(c) \leq u$  for every colour  $c$  occurring in  $m$ , then*

$$l \cdot \#m \leq \langle m, f \rangle \leq u \cdot \#m.$$

*Proof.* A sum of  $\#m$  terms, each in  $[l, u]$  (induction on the bag). In use,  $[l, u]$  is a certified enclosure of  $\phi_j$  on the colour box  $B_P$  — polynomial-enclosure machinery the calculus already owns — so the row couples the pool wire to the count wire. Two remarks. The bounds hypothesis quantifies over colours *in the bag*: the row is therefore only as good as the guarantee that tokens stay in  $B_P$  — the scope premise of **Remark R85.6**. And the degenerate case is load-bearing: at  $k_P = 0$  the two inequalities pin  $s_j^P = 0$  — emptiness is polytope geometry, not a case split.  $\square$

**Lemma R85.3** (count rows).  *$0 \leq \#m$  always; and  $\#m \leq M$  for any certified marking bound  $M$ . Linear conservation laws of the uncoloured skeleton (place invariants [186]) are further valid count rows.*

*Proof.* Cardinalities are nonnegative integers; the bound and the invariants are reachability facts of the net, consumed as hypotheses (they are certified upstream as reachability/place-invariant facts, never re-proven per obligation). Integer-valuedness of  $k_P$  is what the counting closure discipline (QCOUNTATOMROUNDING) exploits at complemented count atoms.  $\square$

**Lemma R85.4** (moment rows are valid for every bag). *For every bag  $m$  and every  $f$ :  $\langle m, f \rangle^2 \leq \#m \cdot \langle m, f^2 \rangle$ .*

*Proof.* Cauchy–Schwarz over the counting measure on the bag:  $(\sum_{c \in m} 1 \cdot f(c))^2 \leq (\sum_{c \in m} 1^2) (\sum_{c \in m} f(c)^2)$ . Instantiated at monomial features this is the family  $(s_\alpha^P)^2 \leq k_P s_{2\alpha}^P$  — optional tightenings: each one added shrinks the relaxation  $K$ , and by this lemma never below the reachable image.  $\square$

**Theorem R85.5** (soundness of QAGGREduce). *Let  $\text{agg}$  be the aggregate map  $x \mapsto (\vec{s}(x), \vec{k}(x))$ , let  $K$  satisfy  $\text{agg}(x) \in K$  for every in-scope  $x$ , and suppose the sequent  $G \leq 0$  holds on  $K$ . Then  $G(\text{agg}(x)) \leq 0$  for every in-scope  $x$ .*

*Proof.* Evaluation:  $\text{agg}(x) \in K$  and  $G \leq 0$  on  $K$ . That the proof is one line is the honest size of the semantic core — the entire content of a QAGGREduce node is the containment certificate for  $K$ , which Lemmas R85.2–R85.4 assemble from rows each valid at every reachable marking: a conjunction of per-marking-valid constraints contains the whole reachable aggregate image, so the single polynomial sequent over  $K$  (discharged by the hypothesis-carrying cone machinery of the base calculus) covers every in-scope marking at once. Obligation count and dimension are independent of the marking bound, which enters only as a constant in Lemma R85.3’s rows — the reason this rule exists instead of a mode explosion over marking shapes.  $\square$

**Remark R85.6** (the box-discharge scope premise). Lemma R85.2’s hypothesis — every occurring colour in  $B_P$  — is a claim about the *process*, and with nondegenerate diffusion a process exits every compact box with positive probability. An enclosure row quantified over states the process leaves is unsoundness, not incompleteness; so the box must be *discharged*, by one of three named routes: model box exit as a forced transition (the boundary becomes part of the model, so no in-scope state ever lies outside the box and the exit’s successor region supplies its own rows), prove invariance of the box for every reachable state (vanishing diffusion

at the boundary plus inward drift — polynomial, box-supported conditions landing in the existing row machinery), or certify the process stopped at exit — the stopped state kept inside the box, with the exit event carried in the specification’s probability budget. Which route is a per-model declaration; that *some* route is declared is a side condition of the rule. Dropping it admits the counterexample: a generator row verified on  $B_P$  says nothing at the first token that leaves, and the concluded obligation can be false on reachable states.

*Remark R85.7* (what the relaxation costs, and what it never costs).  $K$  over-approximates the reachable aggregate image — not every polytope point is a realizable marking. The cost is *completeness*: a true obligation may fail on a spurious point of  $K$  and the sequent then refuses; it is never soundness, by Theorem R85.5. The row menu is the dial: integer rounding at count atoms, conservation laws, and moment rows each cut spurious regions. The drain example, worked to its numbers in Remark R85.8, is the sharpest small witness: its interior row is *infeasible* over the real-relaxed off-target scope  $k > 0$  (it evaluates to  $+\frac{1}{4}$  at  $k = \frac{1}{2}$ ,  $s_2 = 0$ , a point no reachable marking occupies) and discharges, with an exact linear-multiplier witness, once QCOUNTATOMROUNDING tightens the scope to  $k \geq 1$  — the tightening menu earning its keep on the first toy. Conversely a row that fails at some reachable in-scope marking (a naively-cleared mean at  $k = 0$ , say) would cut reachable points and break Theorem R85.5’s containment hypothesis — validity at every reachable in-scope marking, under the row family’s stated premises, is the admission criterion for the menu: the nonnegativity and moment rows meet it for every bag outright, the coupling rows meet it once their colour-scope premise is discharged, and the count bounds and invariants meet it as certified reachability facts consumed as hypotheses.

*Remark R85.8* (the drain example, worked to its numbers). One place, colour  $x \in [-2, 2]$ , one completion transition consuming a single token at rate  $\lambda(x) = 1 + x^2$ , no source; the deadline certificate is  $V = 1 + k$  with  $\theta = \frac{1}{2}$  (counts alone suffice for  $V$ ; the pooled feature  $s_2 = \sum_t x_t^2$  enters the interior row through the rate, so its coupling row joins the scope). The generator: the continuous part vanishes ( $V$  reads no colours), and each completion removes one token, so  $\mathcal{L}V = \sum_t \lambda(x_t)(V(k-1) - V(k)) = -\sum_t (1 + x_t^2) = -(k + s_2)$ . The interior row is

$$\theta V + \mathcal{L}V = \frac{1}{2}(1 + k) - (k + s_2) = -\frac{1}{2}(k - 1) - s_2.$$

Over the real-relaxed off-target scope  $\{k > 0, 0 \leq s_2 \leq 4k\}$  — the coupling row (Lemma R85.2) at the enclosure  $x^2 \in [0, 4]$  on the colour box — the row obligation fails:  $(k, s_2) = (\frac{1}{2}, 0)$  is a scope point but no marking (counts are integers), and there the row evaluates to  $\frac{1}{2} \cdot \frac{3}{2} - \frac{1}{2} = +\frac{1}{4}$ . Once QCOUNTATOMROUNDING rounds the off-target scope  $\text{cmpl}\{k \leq 0\}$  to  $k \geq 1$ , the rewriting displayed above is the exact linear-multiplier witness: multiplier  $\frac{1}{2}$  on the scope row  $k - 1 \geq 0$  and 1 on the coupling floor  $s_2 \geq 0$  reproduce the row identically — a polynomial identity, no residue, no numerics — and the certificate is tight at  $(k, s_2) = (1, 0)$ : the single-cold-token state, the true bottleneck, since the slowest completion rate is  $\lambda(0) = 1$ . Every number cited in Remark R85.7 recomputes from this display; the example’s other rows (floor, atom, root) are not the story here and are not staged.

*Remark R85.9* (relation to the literature). The aggregate coordinates are the classical multisymmetric power sums: per fixed shape, polarized power sums generate the full invariant algebra in characteristic 0 [238], which locates the dictionary’s expressiveness ceiling (generation degree grows with the shape — a resolution bound, priced as completeness, never claimed away). Degree-one certificates in counts alone recover classical place-invariant analysis [186]. Semantically, pools are commutative-monoid folds along the tokens-to-places classifying map, with the multiset as the universal aggregate every other one factors through [222] — exchangeability and factoring-through-the-free-commutative-monoid are the same statement from two sides. The identically-named “coupling rows” of the relational tier (QCOUPLEDRIFT) are probability couplings over a product of chains; the rows here are deterministic pool-count arithmetic within one marking. Same word, disjoint mathematics, deliberately uncited in both directions.

| proof step                 | formal anchor                                             |
|----------------------------|-----------------------------------------------------------|
| Lemma R85.1                | query; query_const_one; query_count                       |
| Lemma R85.2                | coupling_row (degenerate pin at $k = 0$ : its two halves) |
| Lemma R85.3                | count_row_nonneg; count_row_le                            |
| Lemma R85.4                | moment_row                                                |
| Theorem R85.5              | aggReduce                                                 |
| integrality at count atoms | QCOUNTATOMROUNDING section (countAtomRounding_sound/_eq)  |

## Rule 86. QCOUNTATOMROUNDING (alias rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QCountAtomRounding*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{k_P \text{ a count observable, integer-valued at every reachable marking} \quad j \in \mathbb{Z}}{\text{cmpl } \{k_P \leq j\} \rightsquigarrow \{k_P \geq j+1\} \quad (\text{hypothesis positions, reachable scope})}$$

integrality is the compile's guarantee. Exact over the reachable locus (the complement over reachable markings is  $\{k_P \geq j+1\}$ , not  $\{k_P > j\}$ ), and unconditionally never wider ( $\{k_P \geq j+1\} \subseteq \{k_P > j\}$ ) — rounding to the next integer is free and strictly tighter than the real complement, the counting analogue of the strict-atom closure discipline.

QCOUNTATOMROUNDING is the counting closure discipline: when a count observable  $k_P$  is complemented in a hypothesis position, the real-arithmetic complement of  $\{k_P \leq j\}$  — the open condition  $\{k_P > j\}$  — is replaced by the closed integer-rounded atom  $\{k_P \geq j+1\}$ . Over reachable markings the two are *equal* (counts are integers there), and everywhere the rounded atom is contained in the open one, so the rewrite is free and never wider than the real complement — strictly tighter wherever the slab is realized: substituting it in hypothesis position never widens the constraint set, and the gained slab  $j < k_P < j+1$  (unreachable, but visible to real-arithmetic relaxations) is often exactly what makes a downstream combination certificate feasible. It is the counting analogue of the strict-atom closure discipline for continuous observables, with integrality replacing topology.

There is nothing for the checker to recompute beyond the syntactic rewrite itself: producers apply it before emission, and what the wire carries is the already-tightened atom inside an aggregation rule's hypothesis rows. The one genuine hypothesis is integrality — that  $k_P$  takes integer values at every reachable marking. That is a guarantee of the marking-space compile (counts are cardinalities), not of the rewrite; in the mechanization it enters as a named hypothesis, discharged definitionally when the net layer lands. Without it the exactness identity fails (a fractional “count” at  $k_P = j + \frac{1}{2}$  satisfies the true complement but not the rounded atom), and only the containment direction survives — which is not enough in hypothesis position: the rounded scope is *smaller*, so a row verified only on it silently skips exactly such fractional points, and a misapplied rounding can silently accept rather than refuse (the proof section's integrality remark carries the witness). The integrality warrant is load-bearing for soundness, not a convenience.

**notes** Load-bearing on the first worked aggregation example: a drain net's interior row is infeasible over the real-relaxed off-target scope  $k > 0$  (it fails at  $k = \frac{1}{2}$ ) and discharges with the rounded scope  $k \geq 1$ , via an exact linear-multiplier witness. The rewrite rides with QAGGREDUCE's family: its target is an atom-denotation identity consumed by count rows, not a separate rule node — one atom in, one atom out.

### Lean statement

BernsteinLogicAux.Structural .countAtomRounding\_sound

statement bound

```
/-- **P3.3 CountAtomRounding**, the soundness side (hypothesis position):
every reachable point of the true complement of `{k ≤ j}` satisfies the
tightened atom `{j + 1 ≤ k}`. This is the side the drain example's interior
row consumes — the aggregate polytope `K` must contain every in-scope
reachable point, and it does. -/
theorem countAtomRounding_sound (k : X → ℝ) (j : ℤ) (Reach : Set X)
  (hbridge : ∀ x ∈ Reach, ∃ n : ℤ, k x = (n : ℝ)) :
  ∀ x ∈ Reach, x ∈ {x | k x ≤ (j : ℝ)}c → (j : ℝ) + 1 ≤ k x
```

BernsteinLogicAux.Structural .countAtomRounding\_tighter

statement bound

```
/-- **P3.3**, the free-tightening direction: the rounded atom is contained
in the real complement unconditionally — substituting it never widens the
constraint set, so the rewrite is strictly tighter (by the cut
`{j, j+1}`), which is what buys the drain example its Farkas
feasibility. Stated at any real threshold `j`: the containment needs no
integrality — only exactness (`countAtomRounding_eq`) and soundness
(`countAtomRounding_sound`) ride the integer-threshold bridge. -/
theorem countAtomRounding_tighter (k : X → ℝ) (j : ℝ) :
  {x | j + 1 ≤ k x} ⊆ {x | j < k x}
```

BernsteinLogicAux.Structural .countAtomRounding\_eq

statement bound

```

/-- **P3.3**, exactness on the reachable locus: over integer-valued points
the rewrite loses nothing — the catalogue's "the complement ... over
reachable markings IS  $\{k_P \geq j + 1\}$ ", verbatim as a set identity. -/
theorem countAtomRounding_eq (k : X → ℝ) (j : ℤ) (Reach : Set X)
  (hbridge : ∀ x ∈ Reach, ∃ n : ℤ, k x = (n : ℝ)) :
  Reach n {x | k x ≤ (j : ℝ)}c = Reach n {x | (j : ℝ) + 1 ≤ k x}

```

## Correspondence (Steele display ↔ Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The entry's two claims (exactness and never-wider, now the flavor text) and the displayed integrality premise map onto the three bound statements. The compile's integrality guarantee — the rule's System bridge, as the roster says in words —  $\mapsto$  `hbridge : ∀ x ∈ Reach, ∃ n : ℤ, k x = (n : ℝ)`: counts integer-valued at every reachable point, a named hypothesis because lean-aux lacks the net layer that guarantees it. Then: the hypothesis-position rewrite  $\mapsto$  `countAtomRounding_sound` (every reachable point of the true complement of  $\{k \leq j\}$  satisfies  $\{j + 1 \leq k\}$  — the side the drain example's interior row consumes); “unconditionally never wider”  $\mapsto$  `countAtomRounding_tighter`, whose binder row is the width find: NO integrality, NO reachable scope — the containment  $\{j + 1 \leq k\} \subseteq \{j < k\}$  holds for arbitrary real observables, so the substitution never *widens* anywhere; soundness of the hypothesis-position substitution itself rides the exactness side — the integrality hypothesis — certified separately as `countAtomRounding_eq` — the catalogue's “the complement over reachable markings IS  $\{k_P \geq j + 1\}$ ” verbatim as a set identity on the reachable locus. Width note:  $k : X \rightarrow \mathbb{R}$  throughout; `_tighter` now takes  $j : \mathbb{R}$  outright (the containment needs no integrality, so the anchor is stated at the proposition's full width), while `_sound` and `_eq` keep the  $j : \mathbb{Z}$  cast their integer-threshold bridge demands — the count-observable grammar (the  $f \equiv 1$  query) is upstream; divergence D2's question (which side is load-bearing) is answered by the bind structure itself: soundness rides `_sound`, the free tightening rides `_tighter`, and each is consumable independently.

## Lean verifier receipts

module BernsteinLogicAux.Structural; bound build 1222cdcc326f... (bound from receipt cb0c3820c6877f22, 2026-07-29T01:03:36Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 0b7e68907f45b27f | 2026-07-28T23:48:36Z | build+decorrelated                 | current |
| e97d5614d4f08c7e | 2026-07-29T00:00:35Z | build+decorrelated                 | current |
| 27de21f047e6b194 | 2026-07-29T00:49:05Z | build+replay+hardened+decorrelated | current |
| cb0c3820c6877f22 | 2026-07-29T01:03:36Z | build+replay+hardened+decorrelated | current |
| 3530841119a9a607 | 2026-07-31T22:00:22Z | build+decorrelated                 | current |
| b6cc23eb3e5b7f97 | 2026-07-31T22:29:16Z | build+decorrelated                 | current |
| bf3dacc3ab7f20ea | 2026-08-02T23:31:31Z | build+decorrelated                 | current |
| 384a8569d2005df7 | 2026-08-03T16:27:35Z | build+decorrelated                 | current |
| 717e07acf9acac8a | 2026-08-03T22:46:19Z | build+decorrelated                 | current |

11 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement



### NO HOL STATEMENT YET

The HOL4 development checks the base calculus's wire format (theory `bern_catalogue`). This rewrite is applied producer-side, before emission: even with the aggregation node baked, the wire is expected to carry the already-tightened atom, so the HOL statement that would matter is the aggregation sequent's — and no checker consuming the extension theory's rows is ported yet. Nothing exists for a HOL statement to be about; if a future format ever carries the un-rounded atom, the rewrite's exactness identity becomes the portable statement.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: alias rule (tier 3) — a respelling template. One variation from the alias norm, stated up front: the unpack's target is an *atom-denotation identity* consumed by QAGGREDUCE's count rows (the pattern of the strict-atom closure moves), not a separate rule node — one atom in, one atom out, producers rewrite before emission. The five alias artefacts follow in that key: the unpack, the filled-in application, the payload transport, the exactness identity, and the soundness corollary.

**Recipe R86.1** (the respelling). The unpack acts on a complemented count atom in hypothesis position:

$$\text{cml}\{k_P \leq j\} \quad \rightsquigarrow \quad \{k_P \geq j + 1\}, \quad j \in \mathbb{Z},$$

where  $k_P$  is a count observable — integer-valued at every reachable marking, because counts are cardinalities (the compile's integrality guarantee, entering the mechanization as a named hypothesis until the net layer discharges it definitionally). Payload transport, field for field:

| field       | before                          | after                         |
|-------------|---------------------------------|-------------------------------|
| observable  | $k_P$                           | $k_P$ (verbatim)              |
| threshold   | $j$                             | $j + 1$ (integer bump)        |
| orientation | complemented $\leq$ (open $>$ ) | closed $\geq$                 |
| scope       | reachable markings              | reachable markings (verbatim) |

The filled-in application is a scope row of an aggregation obligation: where a compile node scopes a row “off the target  $\{k_P \leq j\}$ ”, the row's hypothesis list carries  $k_P \geq j + 1$  in place of the real-arithmetic complement.

**Proposition R86.2** (exactness over the reachable locus). *On integer-valued points, for  $j \in \mathbb{Z}$ ,  $\text{Reach} \cap \text{cml}\{k_P \leq j\} = \text{Reach} \cap \{k_P \geq j + 1\}$ . (Both restrictions are load-bearing: at a non-integer threshold the integer  $n = \lceil j \rceil$  would satisfy  $n > j$  but not  $n \geq j + 1$ .)*

*Proof.* At a reachable point,  $k_P = n$  for some integer  $n$ ; and for integers,  $n > j \iff n \geq j + 1$ . Two lines, which is the point: the complement of  $\{k_P \leq j\}$  over reachable markings is  $\{k_P \geq j + 1\}$ , not  $\{k_P > j\}$  — nothing is lost.  $\square$

**Proposition R86.3** (free tightening, unconditionally).  $\{k_P \geq j + 1\} \subseteq \{k_P > j\}$  on all points, integer-valued or not.

*Proof.*  $j + 1 \leq k_P$  implies  $j < k_P$ . So substituting the rounded atom in hypothesis position never widens the constraint set; what it removes is exactly the slab  $j < k_P < j + 1$  — strictly, whenever that slab's preimage is nonempty: unreachable by any marking, but fully visible to a real-arithmetic relaxation, which is where the gain lands.  $\square$

**Proposition R86.4** (soundness corollary). *Every reachable point of the true complement of  $\{k_P \leq j\}$  satisfies the tightened atom; hence any row valid on  $\text{Reach} \cap \{k_P \geq j + 1\}$  is valid on the reachable part of the true complement — the direction hypothesis positions consume, and by Proposition R86.2 nothing more is being assumed than the complement itself, reachably.*

*Proof.* Immediate from Proposition R86.2, left-to-right. The asymmetry of positions is worth one sentence: in *hypothesis* position a tighter scope is free (fewer points to verify, no reachable point dropped, by exactness); in *conclusion* position a tightened atom would be a stronger claim and is not licensed by this rule — the alias is a hypothesis-position discipline, exactly like its continuous sibling.  $\square$

**Remark R86.5** (why integrality is not decoration). Drop it and exactness fails: a fractional “count” at  $k_P = j + \frac{1}{2}$  satisfies the true complement but not the rounded atom, so a row verified only on the rounded scope would silently skip it — if such points were reachable, that is unsoundness in hypothesis position. The integrality hypothesis is precisely the statement that they are not: counts are cardinalities of token multisets. Note there is *no* graceful failure mode when the rewrite is misapplied to a non-count observable: in hypothesis (scope) position the rounded atom *shrinks* the region a row must be verified on, so misapplication can silently accept — the half-integer point above is exactly the witness. The integrality warrant is load-bearing, not a convenience.

**Remark R86.6** (load-bearing on the first worked example). The drain net's deadline certificate needs its interior row off the target  $\{k \leq 0\}$ . Over the real-relaxed scope  $k > 0$  the row is infeasible — it evaluates to  $+\frac{1}{4}$  at  $(k, s_2) = (\frac{1}{2}, 0)$ , a point no marking occupies — while over the rounded scope  $k \geq 1$  it discharges with an exact linear-multiplier witness (both numbers recompute



in the QAggReduce section’s drain-example remark, where the row and its multipliers are displayed). The rounding is the counting analogue of the strict-atom closure discipline (polarity-directed rounding of strict literals), with integrality playing the role topology plays there: both convert an open condition into the closed atom the row machinery wants, at zero semantic cost on the states that matter.

| proof step         | formal anchor                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Proposition R86.2  | countAtomRounding_eq                                                                                                                                    |
| Proposition R86.3  | countAtomRounding_tighter                                                                                                                               |
| Proposition R86.4  | countAtomRounding_sound                                                                                                                                 |
| integrality bridge | the named hypothesis in countAtomRounding_sound and _eq (compile-side guarantee); deliberately absent from _tighter, whose containment is unconditional |
| consumed by        | QAggReduce scope rows (count atoms); the drain example’s interior row                                                                                   |

## Rule 87. QPALWAYSFROMFOLD (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QPalwaysFromFold*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{\text{upper} : \text{qpaf\_upper}(n, c_u) \quad \text{lower} : \text{qpaf\_lower}(n, c_l) \quad \text{qtsl\_palways\_fold}(n)}{\mathcal{M} \models \text{SProbGE}(p, \text{PAlways}(\text{SAnd}[\text{SAtomLE}(c, V), \text{SAtomGE}(c, V)]))}$$

QPALWAYSFROMFOLD ties plated-fold safety evidence to the judgment stratum: it concludes a surely-invariant claim — a conserved-charge invariant, held with probability one — over the demonic composite of a token-population subject, from certificates about a plated fold of the population. Coverage telescopes footprint rows *pathwise* across every resolution of every clock wing: the telescoping argument reads firings, not clocks, so race, demonic, and forced transitions are covered uniformly and no scheduler resolution escapes. Initial membership is checked inside the evaluator's walk (the deadline-table precedent): there is no warrant row, no region-level child, and no boxed region — the subject's marking is unbounded, and a boxed region for it would be a type error, not a convenience.

A producer cites one certificate per direction: an upper-bound citation for the  $\leq$  face, a lower-bound citation for the  $\geq$  face — at least one, at most one row each; the lower's features are recomputed by the walk, never trusted. The accepted conclusion shapes are one atom or the displayed two-sided conjunction with equal polynomials; anything wider refuses loudly (a narrow walk mis-verifies nothing).

**notes** The verification content lives in the evaluator's walk, whose semantics are normative in the plated-transfer design: conclusion resolution, spec shape, atom decomposition, cert citations, subject agreement including the plate-to-population token bindings, fold scale, polynomial correspondence against the count-wire enumeration, init membership from the bag marking, sibling-claim presence, and the polarity roster — with the pathwise-species demand on covered transitions: an expectation-species footprint row never warrants a surely-invariant. No Lean statement binds the node yet; the schema and walk are baked, the walk mutant is red in the theory battery, and the mechanization column is honest open debt.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: *chase* = the engine derives; *report* = a violation is a loud judgment; *refuse* = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The baked rows: the node, its direction citations, the walk, and the plate-population binding the walk's subject-agreement step reads.

theory Qtsl . QPALwaysFromFold

quoted from the theory source at render

| QPALwaysFromFold : Set;

theory Qtsl . QPALwaysFromFold/\*

quoted from the theory source at render

| QPALwaysFromFold/proof : QPALwaysFromFold -> QProof;

theory Qtsl . qpaf\_upper

quoted from the theory source at render

| qpaf\_upper : {node: QPALwaysFromFold, cert: HCert} -> Prop;

theory Qtsl . qpaf\_lower

quoted from the theory source at render

| qpaf\_lower : {node: QPALwaysFromFold, cert: HCert} -> Prop;

theory Qtsl . ax/wf/qpalwaysfold\_walk

quoted from the theory source at render

| report ax/wf/qpalwaysfold\_walk : forall n : QPALwaysFromFold.  
|- {node: n} qtsl\_palways\_fold;

theory Qtsl . hplate\_token

quoted from the theory source at render

```
| hplate_token : {plate: HPlate, token: Token} -> Prop;
```

## Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

## HOL4 statement

### NO HOL STATEMENT YET

No HOL development of the extension theory exists; this node's content is walk structure over baked rows.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), the plated-transfer genus (safety face). This section states the rule's soundness obligation exactly, per the catalogue's completeness-over-polish charter (a rule that can only be named-and-gapped is still a row); the discharge is open debt on the mechanization column, and the schema quoted in the entry is the baked ground truth.

**Soundness obligation.** With the walk's ten steps discharged — conclusion resolution, spec shape, atom decomposition, direction citations, subject agreement including plate-to-population token bindings, fold scale, polynomial correspondence against the count-wire enumeration, initial membership from the bag marking, sibling-claim presence, and the polarity roster with its pathwise-species demand — the cited fold certificates' footprint rows telescope pathwise across every resolution of every clock wing, so the conserved-charge invariant holds along every run: the concluded claim holds surely, and any probability threshold in  $(0, 1]$  is sound.

*Remark* R87.1 (status). No Lean statement binds this node. The walk semantics are normative in the plated-transfer design; the walk mutant is red in the theory battery, which certifies the walk's presence, not the transfer theorem — the transfer theorem's mechanization is the open debt.

## Rule 88. QFIREBOUNDFROMFOLD (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — `QFireBoundFromFold`. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{\text{cert} : \text{HCert} \quad d_{\min} > 0 \quad \text{qtsl\_firebound\_fold}(n)}{\mathcal{M} \models \text{SReturnLE}(B, \text{retFireCount}(trs))}$$

QFIREBOUNDFROMFOLD is the ranking face of the same genus: from *one* gap-rowed plated certificate it concludes a roster firing-count ceiling — the specification's warranted component is the count bound over the roster firing-count run functional. The transfer theorem's pathwise face, applied: with footprint, coverage, floor, and flow discharged, every roster firing drops the fold by at least  $d_{\min}$  in the reading scale, the fold never rises anywhere else, and the fold is floored at zero — so roster firings number at most  $\text{fold}_{\varphi}(m_0)/d_{\min}$  along *every* run, uniformly in token count and interleaving.

What the checker recomputes: two exact inequalities — each roster row's gap at least  $d_{\min}$ , and  $B \cdot d_{\min}$  at least the fold's initial value — and never an argmin:  $d_{\min}$  is pinned by the producer, the same witness species as a pinned tolerance. Counting is firing-metered per roster member — deliberately not region-entry counting (the crossing-count acceptance device) and not the instant-crossing count (the crossing-budget family's compensator-mass species): three counting species, three devices, none interchangeable.

**notes** Closed-judgment route, deliberately: no warrant row, no region-level child, no boxed certificate-family table for an unbounded marking; init arithmetic lives inside the walk. The cert citation is a function, not a relation — a ranking consequence cites exactly one certificate, and exactly-one is semantic here (totality is the sort's own obligation); contrast the safety sibling's two-direction relation shape. No Lean statement binds the node yet; the schema, the positivity report row, and the walk are baked with red mutants in the theory battery, and the mechanization column is honest open debt.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: *chase* = the engine derives; *report* = a violation is a loud judgment; *refuse* = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The baked rows: the node with its pinned decrement, the positivity report, the walk, and the roster relation the concluded run functional reads.

theory Qtsl . QFireBoundFromFold

quoted from the theory source at render

| QFireBoundFromFold : Set;

theory Qtsl . QFireBoundFromFold/\*

quoted from the theory source at render

| QFireBoundFromFold/proof : QFireBoundFromFold -> QProof;  
| QFireBoundFromFold/cert : QFireBoundFromFold -> HCert;  
| QFireBoundFromFold/dmin : QFireBoundFromFold -> Q;

theory Qtsl . ax/wf/qfirebound\_dmin\_pos

quoted from the theory source at render

| report ax/wf/qfirebound\_dmin\_pos : forall n : QFireBoundFromFold,  
| d : Q.  
| n QFireBoundFromFold/dmin = d  
| - {l: 0, r: d} q\_lt;

theory Qtsl . ax/wf/qfirebound\_walk

quoted from the theory source at render

| report ax/wf/qfirebound\_walk : forall n : QFireBoundFromFold.  
| - {node: n} qtsl\_firebound\_fold;

theory Qtsl . retFireCount

quoted from the theory source at render

| retFireCount : {trs: TrList} -> PathObservable;

theory Qtsl . qtsl\_firecount\_roster

quoted from the theory source at render

| qtsl\_firecount\_roster : {sys: System, trs: TrList} -> Prop;

## Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

## HOL4 statement

### NO HOL STATEMENT YET

No HOL development of the extension theory exists; this node's content is walk structure over baked rows.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), the plated-transfer genus (ranking face). This section states the rule's soundness obligation exactly, per the catalogue's completeness-over-polish charter (a rule that can only be named-and-gapped is still a row); the discharge is open debt on the mechanization column, and the schema quoted in the entry is the baked ground truth.

**Soundness obligation.** With footprint, coverage, floor, and flow discharged for the one gap-rowed certificate, every roster firing drops the fold by at least  $d_{\min}$  in the reading scale, the fold never rises anywhere else, and the fold is floored at zero; hence roster firings number at most  $\text{fold}_{\varphi}(m_0)/d_{\min}$  along every run, uniformly in token count and interleaving — the concluded firing-count ceiling. The checker's contribution is two exact inequalities (each roster row's gap at least  $d_{\min}$ ; the ceiling times  $d_{\min}$  at least the fold's initial value);  $d_{\min}$  is pinned by the producer and never derived.

*Remark R88.1 (status).* No Lean statement binds this node. The walk semantics are normative in the transfer-former design (twelve steps); the theory battery's mutants are red at the walk and positivity loci.

## Rule 89. QPALWAYSFROMFOLD (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QPalwaysTFromFold*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{n \text{ horizon} = T \quad \text{coverage\_lb}(c) \quad \begin{array}{c} n \text{ cert} = c \quad n \text{ goalfold} = f_g \\ \text{flow\_lb}(c) \quad \forall r. u_r > 0 \Rightarrow \text{TrMeter}(tr(r), d_r) \end{array} \quad \text{qtsl\_palwayst\_fold}(n)}{\mathcal{M} \models \text{SProbGE}(p, \text{PALwaysT}(T, \text{cml}(goal)))}$$

QPALWAYSFROMFOLD is the timed-floor face of the plated-transfer genus — the third sibling beside QPALWAYSFROMFOLD (safety) and QFIREBOUNDFROMFOLD (ranking): from one certificate whose footprint rows carry the *drop-at-most* polarity (per firing, the fold decreases by at most  $u_r$  — including  $u_r = 0$ , never decreases), together with per-row clock floors, it concludes that no run of the system reaches a designated goal region within a horizon  $T$  — a surely-satisfied time-bounded invariance claim over the goal's complement.

The soundness assembly is the transfer telescope's mirror face plus a meter law: with drop-at-most coverage of every may-touch transition, mirror flow, per-row footprint sequents, and — for conjugated reading kinds — the conjugation-parameter conformance discharged (integrality of conjugated powers and the order-preservation domain; the transformed-space telescope is unjustified without it), every run satisfies  $\text{fold}_\varphi(m_t) \geq \text{fold}_\varphi(m_0) - \sum_r u_r N_r(t)$ ; with clock floors  $d_r > 0$  on every counted row,  $N_r(t) \leq \max(0, (t - o_r)/d_r) + 1$  under every resolution. A run reaching the goal region (where the fold is at most  $f_g$ ) by time  $T$  would force  $\text{fold}_\varphi(m_0) - f_g \leq \sum_{r:u_r>0} u_r \cdot (\max(0, (T - o_r)/d_r) + 1)$  — and the checker's whole trusted contribution is ONE exact rational verification of the strict reverse inequality.  $T$ ,  $f_g$ , and every  $u_r$  are pinned by the producer; the checker never computes an argmin or an optimum.

Counting from below stays deliberately *inside* the walk: the specification language refuses a standing “at least  $n$  events occur” former (a liveness claim consuming survival mass), and this rule honors that refusal — the conclusion is safety-polarity, violated by finite prefixes. The conclusion is Zeno-robust without any crossing budget: rows with  $u_r = 0$  (instantaneous rows included) may fire unboundedly, since they cannot drop the fold; an instantaneous row with  $u_r > 0$  refuses outright (it has no time floor to meter).

**notes** The schema is baked in the extension theory: the node, the fourteen-step walk (normative in the judgment-tie design), the two mirror side-conditions, the per-wing meter demands, and the drop-at-most footprint polarity are all landed rows, quoted below from the theory source. The polarity's formation carried its guard obligations into the bake — most critically that the *existing* upper-coverage side condition counts exactly rows of the two elder polarities (safety, gap), stated as a closed positive membership rather than a prohibition: a drop-at-most row bounds nothing above, and an evaluator that counted one would unsound the two elder siblings. The mechanization ladder names the discrete-unfolding profile first (round-indexed runs); the continuous lift waits on a first formalization of the clock meter semantics, shared with QRATEFROMCLOCK and QPALWAYSFROMLINEAGE. No Lean statement binds the node yet; the mechanization column is honest open debt.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The baked rows: the node with its pinned horizon and goal-fold fields, the optional lineage-offsets citation, the walk, the two mirror side-conditions on the cited certificate, the per-wing meter demands, and the drop-at-most polarity with its displacement bound.

theory Qtsl . QPALwaysTFromFold

quoted from the theory source at render

| QPALwaysTFromFold : Set;

theory Qtsl . QPALwaysTFromFold/\*

quoted from the theory source at render

| QPALwaysTFromFold/proof : QPALwaysTFromFold -> QProof;  
 | QPALwaysTFromFold/cert : QPALwaysTFromFold -> HCert;  
 | QPALwaysTFromFold/goalfold : QPALwaysTFromFold -> Q; // f\_g, pinned  
 | QPALwaysTFromFold/horizon : QPALwaysTFromFold -> Q; // T, pinned

theory Qtsl . ax/wf/qpatf\_horizon\_nonneg

quoted from the theory source at render

```

report ax/wf/qpatf_horizon_nonneg : forall n : QPAlwaysTFromFold,
  x : Q.
  n QPAlwaysTFromFold/horizon = x
|- {l: 0, r: x} q_le;

```

theory Qtsl.qpatf\_offsets

quoted from the theory source at render

```

qpatf_offsets : {node: QPAlwaysTFromFold,
  lineage: QPAlwaysTFromLineage} -> Prop;

```

theory Qtsl.qtsl\_palwayst\_fold

quoted from the theory source at render

```

qtsl_palwayst_fold : {node: QPAlwaysTFromFold} -> Prop;

```

theory Qtsl.ax/tf/qpatf\_walk

quoted from the theory source at render

```

report ax/tf/qpatf_walk : forall n : QPAlwaysTFromFold.
|- {node: n} qtsl_palwayst_fold;

```

theory Qtsl.ax/tf/qpatf\_coverage\_lb

quoted from the theory source at render

```

report ax/tf/qpatf_coverage_lb : forall n : QPAlwaysTFromFold,
  c : HCert.
  n QPAlwaysTFromFold/cert = c
|- {cert: c} qtsl_hplate_coverage_lb;

```

theory Qtsl.ax/tf/qpatf\_flow\_lb

quoted from the theory source at render

```

report ax/tf/qpatf_flow_lb : forall n : QPAlwaysTFromFold,
  c : HCert.
  n QPAlwaysTFromFold/cert = c
|- {cert: c} qtsl_hplate_flow_lb;

```

theory Qtsl.ax/tf/qpatf\_meter\_forced

quoted from the theory source at render

```

report ax/tf/qpatf_meter_forced : forall n : QPAlwaysTFromFold,
  c : HCert, r : HFootprintRow, t : Transition, d : Q,
  k : TrClock.
  n QPAlwaysTFromFold/cert = c,
  r HFootprintRow/cert = c,
  r HFootprintRow/polarity = hrpDropAtMost,
  r HFootprintRow/tr = t,
  t Transition/clock = k,
  k = {dwell: d} tkForced
|- {tr: t, d: d} qtsl_tr_meter;

```

theory Qtsl.ax/tf/qpatf\_meter\_demonic

quoted from the theory source at render

```

report ax/tf/qpatf_meter_demonic : forall n : QPAlwaysTFromFold,
  c : HCert, r : HFootprintRow, t : Transition, d : Q,
  e : EnnRat, k : TrClock.
  n QPAlwaysTFromFold/cert = c,
  r HFootprintRow/cert = c,
  r HFootprintRow/polarity = hrpDropAtMost,
  r HFootprintRow/tr = t,
  t Transition/clock = k,
  k = {dwellMin: e} tkDemonic,
  e = {q: d} enFin
|- {tr: t, d: d} qtsl_tr_meter;

```

theory Qtsl.ax/tf/qpatf\_meter\_carved

quoted from the theory source at render

```

report ax/tf/qpatf_meter_carved : forall n : QPAlwaysTFromFold,
  c : HCert, r : HFootprintRow, t : Transition, fl : Q,
  ce : EnnRat, k : TrClock.

```



```

n QPAlwaysTFromFold/cert = c,
r HFootprintRow/cert = c,
r HFootprintRow/polarity = hrpDropAtMost,
r HFootprintRow/tr = t,
t Transition/clock = k,
k = {floor: fl, ceil: ce} tkCarved
|- {tr: t, d: fl} qtsl_tr_meter;

```

theory Qtsl . hrpDropAtMost

quoted from the theory source at render

```

| hrpDropAtMost : HRowPolarity;

```

theory Qtsl . qtsl\_hplate\_coverage\_lb

quoted from the theory source at render

```

| qtsl_hplate_coverage_lb : {cert: HCert} -> Prop;

```

theory Qtsl . qtsl\_hplate\_flow\_lb

quoted from the theory source at render

```

| qtsl_hplate_flow_lb : {cert: HCert} -> Prop;

```

## Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

## HOL4 statement

### NO HOL STATEMENT YET

No HOL development of the extension theory exists; this node's content is walk structure over baked rows.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), the plated-transfer genus (timed-floor face). This section states the rule's soundness obligation exactly, per the catalogue's completeness-over-polish charter; the rule's schema and walk are baked in the extension theory (rows quoted in this section), and the discharge below is the named obligation of the mechanization column.

**Soundness obligation (mirror telescope + meter law).** Let  $F := \text{fold}_\varphi(m_0)$ , recomputed by the walk from the declared initial marking. (Telescope.) With drop-at-most coverage of every may-touch transition, per-row footprint sequents (produced-side fold contribution at least consumed side minus  $u_r$ , in the reading scale, under the guard hypotheses, for all colors in the boxes), mirror flow (the fold does not decrease between firings), and — for conjugated reading kinds — the conjugation-parameter conformance (integrality of conjugated powers; the order-preservation domain, even powers requiring certified nonnegative feature ranges: without it the transformed-space reading of the telescope is unjustified) discharged, every run satisfies  $\text{fold}_\varphi(m_t) \geq F - \sum_r u_r N_r(t)$ , where  $N_r(t)$  counts firings of row  $r$  by time  $t$ . (Meter.) With every counted row ( $u_r > 0$ ) carrying a clock floor  $d_r > 0$  and first-firing offset  $o_r$  (from a cited lineage node, else the one-dwell default  $o_r := d_r$ ),  $N_r(t) \leq \max(0, (t - o_r)/d_r) + 1$  under every resolution. (Assembly.) The goal region carries a certified fold ceiling  $f_g$ ; a run reaching it at  $t \leq T$  forces  $F - f_g \leq \sum_r u_r N_r(t) \leq \sum_{r:u_r>0} u_r \cdot (\max(0, \frac{T-o_r}{d_r}) + 1)$ , using  $u_r \geq 0$  throughout and monotonicity in  $t$ ; the checker-verified strict inequality  $\sum_{r:u_r>0} u_r \cdot (\max(0, \frac{T-o_r}{d_r}) + 1) < F - f_g$  refutes it, so no admitted run reaches the goal region within  $T$  — the concluded time-bounded invariance, surely.

The conclusion is safety-polarity and consumes no crossing budget: rows with  $u_r = 0$  (instantaneous rows included) may fire unboundedly without touching the telescope's right side; an instantaneous row with  $u_r > 0$  refuses at the walk's meter step.

*Remark R89.1 (status).* The schema is baked: the node, its fourteen-step walk, the mirror side-conditions, the per-wing meter demands, and the drop-at-most polarity are landed theory rows (quoted in this section). No Lean statement binds the node yet: the named obligation is the discrete-unfolding profile first (round-indexed runs), with the continuous lift behind the clock meter semantics' first formalization — the keystone shared across this rule family.

## Rule 90. QPALWAYSFROMLINEAGE (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QPalwaysTFromLineage*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{\text{unit\_lineage}(c) \quad \{\text{hrpGap}(1), \text{hrpDropAtMost}(1)\} \text{ per advancing row on } \chi = -\psi \text{ (pathwise species)}}{\text{TrMeter}(tr(r), d) \text{ per advancing row} \quad \forall \tau \in m_0. \psi(\tau) \leq 0 \quad T < k \cdot d \quad \text{qtsl\_palwayst\_lineage}(n)} \\ \mathcal{M} \models \text{SProbGE}(p, \text{PALwaysT}(T, \bigwedge_{q: \text{depth}(q) \geq k} \text{count}_q = 0))$$

QPALWAYSFROMLINEAGE is a causal-depth meter: tokens whose production chain has depth  $k$  cannot exist strictly before time  $k \cdot d$ . It concludes plate-emptiness — no token of declared causal depth at least  $k$  exists at any time  $t \leq T$ , for a pinned horizon  $T$  checked STRICTLY below  $k \cdot d$  (the induction proves emptiness through round  $k - 1$  and permits a depth- $k$  token in the round- $k$  marking, so the inclusive endpoint is deliberately not concluded; on the round grid  $T = (k - 1) \cdot d$  is the tightest warranted choice, and  $k$  is a positive integer) — from four ingredients: *unit lineage* (every covered transition consumes at most one and produces at most one token across the cited plates, so per-token lineage is a path: for a row consuming a cited token and producing a cited successor, the fold-level displacement of the depth dictionary is the successor's depth minus the consumed token's; birth rows root at depth at most one, and a death row on a positive-depth token satisfies neither bracket and is refused — a completeness cost, never an unsoundness); a *bracket* of footprint rows on the negated depth dictionary pinning each chain firing's advance to exactly one (the drop-at-most row and the gap row at equal witness — both sides load-bearing, since a decrease bound alone would admit one firing advancing depth by ten — and every bracket row of the pathwise deterministic-reset species: a stochastic moment row could satisfy both sequents at expected advance one while individual outcomes jump farther, defeating the surely conclusion); clock floors  $d$  on every advancing row; and initial depth at most zero, recomputed by the walk from the declared initial marking (universally over its tokens — vacuous when empty).

The rule has a second face: a QPALWAYSFROMFOLD node may cite it as the source of its per-row first-firing offsets ( $o_r :=$  the row's consumed-tuple minimum declared depth over the guard box, times  $d$  — recomputed by the CONSUMING walk, never read from stored producer data, because an inflated offset errs on the unsound side; a row whose guard box admits no consumed cited token takes the consumer's one-dwell default instead). This face is unaffected by the strict endpoint: a firing that consumes a depth- $k$  token happens at round  $k$  or later, time at least  $k \cdot d$ , which is exactly the offset claim.

The rule mints *profile-gated*, and the gate is stated honestly rather than smoothed over: per-token chain spacing is NOT derivable from the per-transition meter law when several tokens share a guard — a transition's accrued dwell can be spent on a freshly produced token. This is the classical timed-Petri-net memory-policy fork (single-server versus per-token timers). The profile, exactly: every covered row carries the one period  $d$  as its dwell parameter (demonic and forced wings admitted; a carved wing at floor equal to ceiling  $d$ , so firings cannot fall off the round grid); the subject is declared at the round-indexed step reading — firings at round  $r$  read the round- $r$  marking and produce into round  $r + 1$ , rounds at times  $r \cdot d$ , the theory's own discrete-time-unfolding reading, whose canonical case is the all-demonic uniform-dwell net; and the cited plates' token colors do not evolve between firings (the brackets constrain firing displacement only, so between-round color drift would evade them). Under that profile the per-token spacing IS the step law's own content and the soundness induction is clean; the continuous-time lift is a genuine semantics decision the proposal prices without baking.

**notes** The schema is baked in the extension theory: the node, its walk, and its structural side-conditions are quoted below from the theory source. Profile-gated from birth, per the catalogue's precedent of rules proven at a named profile and filed beyond it; the concluded specification former itself is profile-independent — the gate lives in the walk's side conditions, never in the conclusion's meaning. A worked instance of the discrete induction (a fabrication-line model with a 370-deep production chain) is already kernel-checked; the oracle obligation that generalizes it to the profile rule remains owed, and no Lean statement binds the node yet — the mechanization column is honest open debt.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: *chase* = the engine derives; *report* = a violation is a loud judgment; *refuse* = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The baked rows: the node with its positive depth, positive period, and nonnegative horizon reports, the unit-lineage structural side-condition on the cited certificate, and the causal-depth walk.

theory Qtsl . QPALwaysTFromLineage

quoted from the theory source at render

| QPALwaysTFromLineage : Set;

theory Qtsl.QPALwaysTFromLineage/\*

quoted from the theory source at render

```
QPALwaysTFromLineage/proof : QPALwaysTFromLineage -> QProof;
QPALwaysTFromLineage/cert  : QPALwaysTFromLineage -> HCert;
QPALwaysTFromLineage/depth : QPALwaysTFromLineage -> Nat;
QPALwaysTFromLineage/period : QPALwaysTFromLineage -> Q;
QPALwaysTFromLineage/horizon : QPALwaysTFromLineage -> Q;
```

theory Qtsl.ax/wf/qpatl\_depth\_pos

quoted from the theory source at render

```
report ax/wf/qpatl_depth_pos : forall n : QPALwaysTFromLineage,
  k : Nat.
  n QPALwaysTFromLineage/depth = k
|- {l: 0, r: k} nat_lt;
```

theory Qtsl.ax/wf/qpatl\_period\_pos

quoted from the theory source at render

```
report ax/wf/qpatl_period_pos : forall n : QPALwaysTFromLineage,
  d : Q.
  n QPALwaysTFromLineage/period = d
|- {l: 0, r: d} q_lt;
```

theory Qtsl.ax/wf/qpatl\_horizon\_nonneg

quoted from the theory source at render

```
report ax/wf/qpatl_horizon_nonneg : forall n : QPALwaysTFromLineage,
  x : Q.
  n QPALwaysTFromLineage/horizon = x
|- {l: 0, r: x} q_le;
```

theory Qtsl.qtsl\_unit\_lineage

quoted from the theory source at render

```
qtsl_unit_lineage : {cert: HCert} -> Prop;
```

theory Qtsl.qtsl\_palwayst\_lineage

quoted from the theory source at render

```
qtsl_palwayst_lineage : {node: QPALwaysTFromLineage} -> Prop;
```

theory Qtsl.ax/tf/qpatl\_walk

quoted from the theory source at render

```
report ax/tf/qpatl_walk : forall n : QPALwaysTFromLineage.
|- {node: n} qtsl_palwayst_lineage;
```

theory Qtsl.ax/tf/qpatl\_unit\_lineage

quoted from the theory source at render

```
report ax/tf/qpatl_unit_lineage : forall n : QPALwaysTFromLineage,
  c : HCert.
  n QPALwaysTFromLineage/cert = c
|- {cert: c} qtsl_unit_lineage;
```

## Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

## HOL4 statement

### NO HOL STATEMENT YET

No HOL development of the extension theory exists; this node's content is walk structure over baked rows.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1), the plated-transfer genus (causal-depth face), profile-gated from birth. This section states the rule's soundness obligation exactly, gate included; the rule's schema and walk are baked in the extension theory (rows quoted in this section), and the discharge below is the named obligation of the mechanization column.

**Soundness obligation (depth induction at the discrete-unfolding profile).** At the gated profile (every covered row carrying the one period  $d$  as its dwell parameter — demonic and forced wings admitted, carved wings at floor equal to ceiling  $d$ ; the subject declared at the round-indexed step reading, firings at round  $r$  reading the round- $r$  marking and producing into round  $r + 1$ , rounds at times  $r \cdot d$ ; cited plates' token colors static between firings), by induction over rounds: unit lineage makes each token's production history a path through covered firings; the bracket rows on the negated depth dictionary — every one of the pathwise deterministic-reset species — pin each chain firing's depth advance to exactly one (the gap row bounds the advance from below, the drop-at-most row from above — both sides load-bearing); initial depth is at most zero, universally over the declared initial marking's tokens (vacuous when empty), by the walk's recomputation. Hence a token present at round  $m$  arose through at most  $m$  chain firings in strictly increasing rounds and has declared depth at most  $m$  — so plates of declared depth at least  $k$  ( $k \geq 1$ ) are empty at rounds  $0, \dots, k - 1$ : emptiness at every time  $t \leq T$  for the pinned horizon  $T < k \cdot d$ , surely. The endpoint is strict by construction — the induction permits a depth- $k$  token in the round- $k$  marking itself, so the inclusive horizon  $k \cdot d$  is deliberately not concluded.

**The gate, exactly.** Beyond the profile, per-token chain spacing is NOT a consequence of the per-transition meter law when tokens share a guard: a transition's accrued dwell can be spent on a freshly produced token, collapsing the chain's temporal spacing to one dwell for arbitrarily many chain steps. This is the classical timed-Petri-net memory-policy fork (single-server versus per-token timers); the continuous-time lift is a semantics decision the proposal prices without baking, and the rule refuses outside the profile rather than concluding from an unproven spacing premise.

*Remark R90.1 (status).* The schema is baked: the node, its walk, and its structural side-conditions are landed theory rows (quoted in this section). No Lean statement binds the node yet. A worked instance of the discrete induction (a fabrication-line model with a 370-deep production chain) is already kernel-checked; the named obligation generalizes it to the profile rule.

## 15.11 Distance Transport

### Rule 91. QCOUPLEDRIFT (core rule)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QCoupleDrift*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\begin{array}{l}
 \{ \vdash K(p, \cdot) \geq 0, \sum_q K(p, q) = 1, \pi_{i\#} K(p, \cdot) = \hat{K}_i(p_i, \cdot) \} \text{ (coupling rows, exact } \mathbb{Q}; \hat{K}_i \text{ the completed component kernels)} \\
 \{ \vdash K(p, p) = 1 \text{ for both-absorbed } p \} \text{ (freeze rows)} \\
 \{ \vdash \sum_q K(p, q) V(q) \leq V(p) \} \text{ (relational supermartingale generator row, every product state)} \\
 \vdash V \geq 0 \quad \{ \vdash V \geq 1 \text{ (disagreeing), resp. } V \geq |v_1 - v_2|, \text{ on both-absorbed states} \} \text{ (terminal domination)} \\
 \text{absorption: per-component reach criterion; } \mu_0 \text{ a probability law with } \pi_{i\#} \mu_0 = \text{the component initial laws (initial marginal rows); } \vdash \sum_p \\
 \hline
 \text{QDist}[d](\mathcal{M}_1, \mathcal{M}_2) \leq b \quad d \in \{\text{TV, gap-pseudometrics}\}
 \end{array}$$

the coupling marginals are tied to the *completed* component kernels  $\hat{K}_i$  (absorbing states frozen — the completion carries the component freezes), and the absorption criterion is recomputed by the checker (a per-component graph reach). Coupling emptiness refuses, never lies; the  $\mathbb{W}_1$  registry entry and the mixture-polynomial tier are designed, not yet realized — the finite-chain tier above is the proven fragment.

QCOUPLEDRIFT is the coupling-witnessed discharge of the distance judgment: from a certified coupling of two absorbing Markov chains and a relational supermartingale on the product, it concludes a bound on how far the two systems' final readouts can differ — total-variation distance of the final readout laws, or the gap of final expected values. A producer reaches for it to compare two models of the same system (a nominal chain against a faulted one, a fine model against a simplification): the producer chooses a coupling — a product-space kernel whose marginals are the two chains — that keeps the pair together where the chains agree, and a nonnegative function  $V$  that decreases along the coupled dynamics and dominates the disagreement indicator at absorption. The certified bound is  $V$ 's initial expectation.

What the checker recomputes, all in exact rational arithmetic: that the coupling rows form a genuine kernel (entrywise nonnegative, unit row sums) whose row marginals are exactly the two completed component kernels; that both-absorbed states are frozen; the drift inequality  $\sum_q K(p, q) V(q) \leq V(p)$  at every product state; nonnegativity of  $V$ ; the terminal-domination rows ( $V \geq 1$  on disagreeing both-absorbed states for the total-variation form,  $V \geq |v_1 - v_2|$  for the expectation-gap form); the per-component absorption criterion as a finite-graph reachability check; and the initial bound  $\sum_p \mu_0(p) V(p) \leq b$ .

Each hypothesis is load-bearing, and the proof section walks the omission counterexamples: broken marginals sever the product chain from the systems being compared; without nonnegativity the drift induction cannot be localized to the absorbing set; without terminal domination the coupling inequality has nothing to consume; without absorption the final pair law is not a coupling of the final component laws (mass that never both-absorbs escapes the accounting). A coupling that cannot be completed refuses — the coupling is an input premise, so when none exists no certificate can be formed: the rule never lies by emptiness.

**notes** The judgment-primary form ( $\text{QDist}[d] \leq \varepsilon$ , one conclusion per certificate) deliberately differs from pre/post-style relational program logics [116]: it composes at judgment level (triangle, weaken, symmetry where flagged), and expectation-gap readouts enter as gap-pseudometric registry entries rather than as a second judgment form. The proven fragment is the finite-chain certificate tier, and its conclusion now reads in general vocabulary: the certified bound dominates the Jordan-decomposition total-variation distance of the final readout laws as measures (see the width note); the mixture-polynomial tier and the  $\mathbb{W}_1$  registry entry are designed, not yet realized. A pilot study ran four exact-rational certificates against an independent checker with a mutation battery; on correlated faulting the relational route certified a total-variation bound roughly three times tighter than the best single-chain route — the reason the tier exists.

#### Lean statement

BernsteinLogicAux.TvDist.exists\_hahn

statement bound

```

/-- **The Hahn-set bridge**: for finite measures of equal total mass there
is a measurable set `i` on which the difference attains `tvDist`, and which
dominates the difference on every measurable set. This is the single lemma
through which all Jordan-decomposition bookkeeping flows; everything
downstream (metric axioms, coupling inequality, finite-tier seam) consumes
only its statement. -/
theorem exists_hahn (μ ν : Measure X) [IsFiniteMeasure μ] [IsFiniteMeasure ν]

```

```
(hmass :  $\mu$ .real univ =  $v$ .real univ) :
 $\exists i : \text{Set } X, \text{MeasurableSet } i \wedge \mu.\text{real } i - v.\text{real } i = \text{tvDist } \mu v \wedge$ 
 $\forall A : \text{Set } X, \text{MeasurableSet } A \rightarrow \mu.\text{real } A - v.\text{real } A \leq \mu.\text{real } i - v.\text{real } i$ 
```

## BernsteinLogicAux.TvDist . tvDist\_le\_coupling

statement bound

```
-- **The coupling inequality**: if  $\pi$  on  $X \times X$  has marginals  $\mu$  and
 $v$ , then  $d_{\text{TV}}(\mu, v) \leq \pi \{r \mid r.1 \neq r.2\}$ . The off-diagonal is evaluated by
the *outer* measure, so no measurability of the diagonal (and no separation
hypothesis on  $X$ ) is needed; the marginal conditions themselves force the
total masses to agree. -/
theorem tvDist_le_coupling { $\mu v : \text{Measure } X$ } [IsFiniteMeasure  $\mu$ ] [IsFiniteMeasure  $v$ ]
( $\pi : \text{Measure } (X \times X)$ ) (h $\pi_1 : \pi.\text{map Prod.fst} = \mu$ ) (h $\pi_2 : \pi.\text{map Prod.snd} = v$ ) :
tvDist  $\mu v \leq (\pi \{r : X \times X \mid r.1 \neq r.2\}).\text{toReal}$ 
```

## BernsteinLogicAux.TvDist . tvDist\_ofFn

statement bound

```
-- **The finite-tier compatibility lemma**: on a finite discrete space the
general TV distance of the induced measures is the checker's own
 $(1/2) \cdot L^1$  total variation. Requires only nonnegativity and equal totals
(sub-probability pairs of equal mass included). -/
theorem tvDist_ofFn { $p q : L \rightarrow \mathbb{R}$ } (h $p_0 : \forall l, 0 \leq p l$ ) (h $q_0 : \forall l, 0 \leq q l$ )
(h $\text{tot} : \sum l, p l = \sum l, q l$ ) :
tvDist (ofFn  $p$ ) (ofFn  $q$ ) = QRelChain.tvDist  $p q$ 
```

## BernsteinLogicAux.TvDist . sound\_QRelTV\_measure

statement bound

```
-- **`sound_QRelTV`, conclusion re-scoped to the general object**: under
exactly the hypotheses of `QRelChain.sound_QRelTV` – the checker's
obligations C1–C9, nothing added – the certified bound dominates the
*general* total-variation distance (`TV.tvDist`, the Jordan/coupling
object of this module) between the final readout laws, viewed as measures
on the readout space. `Fintype` survives only where the certificate
arithmetic lives (the chains and the readout space); the conclusion is the
unrestricted object. -/
theorem sound_QRelTV_measure { $L : \text{Type}^*$ } [Fintype  $L$ ] [DecidableEq  $L$ ]
[MeasurableSpace  $L$ ] [MeasurableSingletonClass  $L$ ] { $f : \sigma_1 \rightarrow L$ } { $g : \sigma_2 \rightarrow L$ }
{ $V : \sigma_1 \times \sigma_2 \rightarrow \mathbb{R}$ } { $b : \mathbb{R}$ }
(h $K : \text{QRelChain.IsMarkov } K$ ) (h $\mu_0 : \text{QRelChain.IsDist } \mu_0$ )
(h $K_1 : \text{QRelChain.IsMarkov } K_1$ ) (h $K_2 : \text{QRelChain.IsMarkov } K_2$ )
(h $m_1 : \text{QRelChain.PushKernel Prod.fst } K K_1$ )
(h $m_2 : \text{QRelChain.PushKernel Prod.snd } K K_2$ )
(h $\text{freeze} : \forall p \in A_1 \times^s A_2, K p p = 1$ )
(h $\text{fr}_1 : \forall a \in A_1, K_1 a a = 1$ ) (h $\text{fr}_2 : \forall a \in A_2, K_2 a a = 1$ )
(h $\text{abs}_1 : \text{Filter.Tendsto}$ 
(fun  $n \Rightarrow \sum a \in A_1, \text{QRelChain.law } K_1 (\text{QRelChain.push Prod.fst } \mu_0) n a$ )
Filter.atTop (nhds 1))
(h $\text{abs}_2 : \text{Filter.Tendsto}$ 
(fun  $n \Rightarrow \sum a \in A_2, \text{QRelChain.law } K_2 (\text{QRelChain.push Prod.snd } \mu_0) n a$ )
Filter.atTop (nhds 1))
(h $v_0 : \forall p, 0 \leq V p$ ) (h $\text{drift} : \forall p, \sum r, K p r * V r \leq V p$ )
(h $\text{term} : \forall p \in A_1 \times^s A_2, f p.1 \neq g p.2 \rightarrow 1 \leq V p$ )
(h $b : \sum p, \mu_0 p * V p \leq b$ ) :
tvDist
(ofFn (QRelChain.push  $f$  (QRelChain.finalLaw  $K_1$  (QRelChain.push Prod.fst  $\mu_0$ )  $A_1$ )))
(ofFn (QRelChain.push  $g$  (QRelChain.finalLaw  $K_2$  (QRelChain.push Prod.snd  $\mu_0$ )  $A_2$ )))
 $\leq b$ 
```

## BernsteinLogicAux.QRelChain . sound\_QRelTV

statement bound

```
-- **Soundness of `qrel-explicit-v0` `tv_le` certificates** (README
`Soundness chain`, assembled; the `sound_QCoupleDrift` obligation of the
doctrine sketch at the explicit finite tier, TV readout): under the checker's
obligations – C1/C3 (`IsMarkov`), C4 (`PushKernel`, both faces), the
completed-kernel freeze, C5 ( $0 \leq V$ ), C6 (drift), C7 ( $V \geq 1$  on
disagreeing both-absorbed states), C8 (absorption, semantic form, per
```

```

component), C9 ( $E_{\mu_0}[V] \leq b$ ) — the certified bound dominates the true
total-variation distance of the two final readout laws. -/
theorem sound_QRelTV [Fintype L] [DecidableEq L] {f :  $\sigma_1 \rightarrow L$ } {g :  $\sigma_2 \rightarrow L$ }
  {V :  $\sigma_1 \times \sigma_2 \rightarrow \mathbb{R}$ } {b :  $\mathbb{R}$ }
  (hK : IsMarkov K) (h $\mu_0$  : IsDist  $\mu_0$ ) (hK1 : IsMarkov K1) (hK2 : IsMarkov K2)
  (hm1 : PushKernel Prod.fst K K1) (hm2 : PushKernel Prod.snd K K2)
  (hfreeze :  $\forall p \in A_1 \times^s A_2, K p p = 1$ )
  (hfr1 :  $\forall a \in A_1, K_1 a a = 1$ ) (hfr2 :  $\forall a \in A_2, K_2 a a = 1$ )
  (habs1 : Tendsto (fun n =>  $\sum a \in A_1, \text{law } K_1 (\text{push Prod.fst } \mu_0) n a$ )
    atTop ( $\mathcal{N}$  1))
  (habs2 : Tendsto (fun n =>  $\sum a \in A_2, \text{law } K_2 (\text{push Prod.snd } \mu_0) n a$ )
    atTop ( $\mathcal{N}$  1))
  (hV0 :  $\forall p, 0 \leq V p$ ) (hdrift :  $\forall p, \sum q, K p q * V q \leq V p$ )
  (hterm :  $\forall p \in A_1 \times^s A_2, f p.1 \neq g p.2 \rightarrow 1 \leq V p$ )
  (hb :  $\sum p, \mu_0 p * V p \leq b$ ) :
  tvDist (push f (finalLaw K1 (push Prod.fst  $\mu_0$ ) A1))
    (push g (finalLaw K2 (push Prod.snd  $\mu_0$ ) A2)))  $\leq b$ 

```

BernsteinLogicAux.QRelChain.sound\_QRelExpGap

statement bound

```

/-- **Soundness of `qrel-explicit-v0` `expgap_le` certificates** (README
$"Soundness chain", assembled; Jensen replaces the coupling lemma): under the
same obligations with C7 in its expectation-gap form
( $|v_1 - v_2| \leq V$  at both-absorbed states), the certified bound dominates the
true gap of final expected readout values. -/
theorem sound_QRelExpGap {v1 :  $\sigma_1 \rightarrow \mathbb{R}$ } {v2 :  $\sigma_2 \rightarrow \mathbb{R}$ } {V :  $\sigma_1 \times \sigma_2 \rightarrow \mathbb{R}$ } {b :  $\mathbb{R}$ }
  (hK : IsMarkov K) (h $\mu_0$  : IsDist  $\mu_0$ ) (hK1 : IsMarkov K1) (hK2 : IsMarkov K2)
  (hm1 : PushKernel Prod.fst K K1) (hm2 : PushKernel Prod.snd K K2)
  (hfreeze :  $\forall p \in A_1 \times^s A_2, K p p = 1$ )
  (hfr1 :  $\forall a \in A_1, K_1 a a = 1$ ) (hfr2 :  $\forall a \in A_2, K_2 a a = 1$ )
  (habs1 : Tendsto (fun n =>  $\sum a \in A_1, \text{law } K_1 (\text{push Prod.fst } \mu_0) n a$ )
    atTop ( $\mathcal{N}$  1))
  (habs2 : Tendsto (fun n =>  $\sum a \in A_2, \text{law } K_2 (\text{push Prod.snd } \mu_0) n a$ )
    atTop ( $\mathcal{N}$  1))
  (hV0 :  $\forall p, 0 \leq V p$ ) (hdrift :  $\forall p, \sum q, K p q * V q \leq V p$ )
  (hterm :  $\forall p \in A_1 \times^s A_2, |v_1 p.1 - v_2 p.2| \leq V p$ )
  (hb :  $\sum p, \mu_0 p * V p \leq b$ ) :
  |( $\sum a_1 \in A_1, \text{finalMass } K_1 (\text{push Prod.fst } \mu_0) a_1 * v_1 a_1$ ) -
    $\sum a_2 \in A_2, \text{finalMass } K_2 (\text{push Prod.snd } \mu_0) a_2 * v_2 a_2$ |  $\leq b$ 

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

reviewer-certified against the displayed texts

The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed  $d \in \{\text{TV, gap-pseudometrics}\}$  is two theorems:  $\text{QDist}[\text{TV}](\mathcal{M}_1, \mathcal{M}_2) \leq b \mapsto \text{sound\_QRelTV}$ 's goal  $\text{tvDist } (\text{push } f (\text{finalLaw } K_1 \dots)) (\text{push } g (\text{finalLaw } K_2 \dots)) \leq b$  — the true total-variation distance of the two final *readout* laws (the readout maps  $f/g$  are the displayed pseudometric's observation structure, explicit in the binder) — and the gap form  $\mapsto \text{sound\_QRelExpGap}$ 's absolute-difference goal. The obligation roster C1–C9 maps row family by row family:

- coupling rows  $\mapsto hK : \text{IsMarkov } K$  (entrywise nonneg, unit sums) +  $hm_1/hm_2$  ( $\text{PushKernel Prod.fst/snd } K K_i$  — marginals equal the completed component kernels, both faces).
- freeze rows  $\mapsto hfreeze$  (both-absorbed diagonal = 1) +  $hfr_1/hfr_2$  (component freezes).
- generator row  $\mapsto hdrift : \forall p, \sum q, K p q * V q \leq V p$ , every product state, displayed vocabulary carried exactly;  $V \geq 0 \mapsto hV0$ .
- terminal domination  $\mapsto hterm$ , per distance flavour, exactly as displayed: the indicator form ( $f p.1 \neq g p.2 \rightarrow 1 \leq V p$ ) and the gap form ( $|v_1 p.1 - v_2 p.2| \leq V p$ ), both on both-absorbed states only.
- the absorption criterion carries this entry's one representation delta: the display asks the per-component reach criterion (recomputed, per the flavor text); the BOUND statements consume absorption in *semantic* form —  $habs_1/habs_2$ , absorbed mass tending to one. The `_of_reach` variants consuming the checker's own graph criterion (via `tendsto_absorbedMass_one`) exist in the module and are cited by the roster witness, but are not in this bound set — machine-checked and cited, not bound, in this edition.



- initial marginal rows  $\mapsto$  carried *by construction*: the component initial laws in the goal ARE push Prod.fst/snd  $\mu_0$  — no separate hypothesis exists because the statement never admits mismatched initials;  $\vdash \sum_p \mu_0(p)V(p) \leq b \mapsto hb$ .

Width note: the *conclusion* is the general object — `TV.sound_QRelTV_measure` (module `BernsteinLogicAux.TvDist`) restates `sound_QRelTV`'s bound against the Jordan-decomposition total-variation distance of the final readout laws viewed as measures, with zero new hypotheses, through the general coupling inequality (`tvDist_le_coupling`: any coupling, off-diagonal weighed by outer measure, no diagonal-measurability assumption) and the finite-tier seam `tvDist_ofFn` (`QRelChain.tvDist` is the general distance restricted); TV is packaged as a metric on probability measures (`TVProb`). The conditional mark therefore re-scopes to the certificate-arithmetic tier: `FIntype` chains with exact real arithmetic standing in for exact  $\mathbb{Q}$  — a fragment fence, by design. The countable tier now carries kernel-level twins (`BernsteinLogicAux.CountableTier`: the drift-induction reach bound, the optional-stopping reach lower bound, and geometric absorption, over `Countable` state spaces with row hypotheses as `tsum` series, summability supplied by the unit-row-sum and boundedness rows), machine-checked and cited, not bound, like the `_of_reach` variants — the  $\mathbb{W}_1$  registry entry and the mixture-polynomial tier remain designed, not realized, exactly as the flavor text says; coupling emptiness has no Lean face because the theorems consume a supplied coupling — refusal-shaped completeness cost lives producer-side.

## Lean verifier receipts

module `BernsteinLogicAux.TvDist`; bound build 54f7e48568aa... (bound from receipt 1c592f5cc199fad5, 2026-07-24T09:28:46Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers             | status  |
|------------------|----------------------|--------------------|---------|
| a908c952d3907ab4 | 2026-07-22T16:31:55Z | build+decorrelated | current |
| 4f8b7d7adcc8f5d2 | 2026-07-22T18:27:35Z | build+decorrelated | current |
| 1c592f5cc199fad5 | 2026-07-24T09:28:46Z | build+decorrelated | current |
| 2cc8a0435e486a23 | 2026-07-31T22:00:34Z | build+decorrelated | current |
| b78489856a2622df | 2026-07-31T22:29:28Z | build+decorrelated | current |
| 10be3d04a0098b77 | 2026-08-02T23:31:44Z | build+decorrelated | current |
| 9bcf8f97503e6168 | 2026-08-03T16:27:47Z | build+decorrelated | current |
| c41aeb2d063d9b00 | 2026-08-03T22:46:32Z | build+decorrelated | current |

module `BernsteinLogicAux.QRelChain`; bound build 64ae4b299db4... (bound from receipt efcd1abd35a1b7f0, 2026-07-19T18:24:37Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 54e0e5134c7a203f | 2026-07-19T01:11:59Z | build+decorrelated                 | current |
| 1ac5f362df2230fb | 2026-07-19T01:58:14Z | build+decorrelated                 | current |
| efcd1abd35a1b7f0 | 2026-07-19T18:24:37Z | build+decorrelated                 | current |
| 5a18ef8a1d5d481d | 2026-07-20T01:21:49Z | build+decorrelated                 | current |
| 1119d63191cbca38 | 2026-07-20T20:46:42Z | build+decorrelated                 | current |
| 53ac120ea74296cc | 2026-07-20T20:58:15Z | build+replay+hardened+decorrelated | current |
| 1189ecab6f8e092f | 2026-07-21T00:03:39Z | build+decorrelated                 | current |
| 4a715ffa478d6acd | 2026-07-21T05:38:59Z | build+replay+hardened+decorrelated | current |
| ab2a676a4de29f86 | 2026-07-22T18:27:12Z | build+decorrelated                 | current |
| a1b101422ff8770b | 2026-07-31T22:00:09Z | build+decorrelated                 | current |
| 6aed590f9ba0b241 | 2026-07-31T22:29:03Z | build+decorrelated                 | current |
| b83a01eca4bc0784 | 2026-08-02T23:31:14Z | build+decorrelated                 | current |
| 2bbbc458342cfb9e | 2026-08-03T16:27:17Z | build+decorrelated                 | current |
| 04d2eae82194b82e | 2026-08-03T22:46:01Z | build+decorrelated                 | current |

### gate-layer coverage of the bound build:

- build (lake build (elaboration)) current
- replay (leanchecker (official kernel replay)) current
- hardened (safe\_verify (hardened screen)) current
- decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development checks the base calculus's wire format (theory `bern_catalogue`, the first-order check-function cone of the core catalogue). `QCOUPLEDRIFT` concludes a distance judgment of the extension theory: its rule sort is out of scope of the current baked theory, and the soundness spine lives in the auxiliary Lean package. Nothing exists yet for a HOL statement to be about; the box becomes writable when a revision carries the relational certificate rows and its checker chain is ported.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1). Scope, stated loudly: the finite-chain tier — finite state spaces, distribution-evolution semantics, exact-rational certificate data instantiated over  $\mathbb{R}$  by cast. The mixture-polynomial tier and the  $\mathbb{W}_1$  conclusion family are designed, not proven, and nothing below claims them. Every hypothesis of the rule is used and remarked on.

**Setting and claim.** Two finite chains are compared: state spaces  $\sigma_1, \sigma_2$ , row-stochastic kernels  $K_1, K_2$ , absorbing target sets  $A_i$  (each  $a \in A_i$  frozen:  $K_i(a, a) = 1$ ), and readouts  $f : \sigma_1 \rightarrow L, g : \sigma_2 \rightarrow L$  (resp. values  $v_1, v_2 : \sigma_i \rightarrow \mathbb{R}$ ). The certificate supplies a *coupling* kernel  $K$  on  $\sigma_1 \times \sigma_2$  with initial law  $\mu_0$ , and a function  $V \geq 0$  with bound  $b$ . The chain's law evolves by  $\mu_{n+1}(t) = \sum_s \mu_n(s) K(s, t)$ ; the *final law* of an absorbing state is its monotone limit mass  $\text{fin}(a) = \sup_n \mu_n(a)$ . Hypotheses (the checker's obligation list, in the order the proof consumes them): the coupling rows make  $K$  a kernel whose row marginals are exactly  $K_1, K_2$  (*marginal identities*); both-absorbed states  $A_1 \times A_2$  are frozen;  $V$  is a supermartingale for  $K$  ( $\sum_q K(p, q) V(q) \leq V(p)$ , every  $p$ );  $V \geq 0$ ; terminal domination ( $V(p) \geq 1$  on both-absorbed  $p$  with  $f(p_1) \neq g(p_2)$ , resp.  $V(p) \geq |v_1(p_1) - v_2(p_2)|$ ); each component chain absorbs almost surely (certified by a finite-graph reach criterion); and  $\sum_p \mu_0(p) V(p) \leq b$ . The claims (`sound_QReLTV`, `sound_QReLExpGap`): the total-variation distance of the two final readout laws, resp. the gap of final expected values, is at most  $b$ .

**Lemma R91.1** (marginal-process lemma). *If every row of  $K$  pushes along  $\pi_i$  to the corresponding row of  $K_i$ , then at every time  $n$  the pushforward of the coupled law is the component law:  $\pi_{i\#} \mu_n = \text{law}_i(n)$ , where  $\text{law}_i$  evolves  $\pi_{i\#} \mu_0$  under  $K_i$ .*

*Proof.* Induction on  $n$ ; the base is definitional. For the step it suffices that the row identities intertwine the two Markov operators:

$$(\pi_{\#} \text{step}_K \mu)(c) = \sum_{\pi(t)=c} \sum_s \mu(s) K(s, t) = \sum_s \mu(s) \sum_{\pi(t)=c} K(s, t) = \sum_s \mu(s) K_i(\pi(s), c) = (\text{step}_{K_i} \pi_{\#} \mu)(c),$$

where the third equality is the pushed-row identity at  $s$  and the last regroupes the outer sum over the fibres of  $\pi$ . This is the coupling's first duty: the product chain really is a joint presentation of the two systems being compared.  $\square$

**Lemma R91.2** (supermartingale induction).  $\sum_p \mu_n(p) V(p) \leq \sum_p \mu_0(p) V(p)$  for all  $n$ .

*Proof.* One step:  $\sum_t (\text{step } \mu)(t) V(t) = \sum_s \mu(s) \sum_t K(s, t) V(t) \leq \sum_s \mu(s) V(s)$ , using the generator row at each  $s$  and  $\mu(s) \geq 0$ . Iterate.  $\square$

**Lemma R91.3** (the finite optional-stopping bound). *Let  $A \subseteq \text{states}$  be frozen, and let  $0 \leq \gamma \leq V$  on  $A$ . Then  $\sum_{a \in A} \text{fin}(a) \gamma(a) \leq b$ .*

*Proof.* At every finite time,

$$\sum_{a \in A} \mu_n(a) \gamma(a) \leq \sum_{a \in A} \mu_n(a) V(a) \leq \sum_{\text{all } s} \mu_n(s) V(s) \leq b,$$

the middle step because  $V \geq 0$  lets the sum extend off  $A$ , the last by Lemma R91.2 and the initial bound. Frozen states have monotone mass ( $\mu_{n+1}(a) \geq \mu_n(a) K(a, a) = \mu_n(a)$ ), so each term converges upward to  $\text{fin}(a) \gamma(a)$ ; the limit of the finite sum obeys the uniform bound. This is optional stopping with monotone convergence of finite sums doing Fatou's work — no measure theory enters at this tier.  $\square$

**Lemma R91.4** (coupling inequality). *For any nonnegative weight  $\gamma$  on  $\sigma_1 \times \sigma_2$ ,*

$$\text{TV}((f \circ \pi_1)_{\#} \gamma, (g \circ \pi_2)_{\#} \gamma) \leq \gamma\{p : f(p_1) \neq g(p_2)\}.$$

*Proof.* Fix a label  $l$ . Splitting each pushed mass over the agreement region  $\{f = g\}$  and its complement, the agreement contributions to the two sides are *equal* (there,  $f(p_1) = l \iff g(p_2) = l$ ), so

$$|(f \pi_1)_{\#} \gamma(l) - (g \pi_2)_{\#} \gamma(l)| \leq \gamma\{f \neq g, f(p_1) = l\} + \gamma\{f \neq g, g(p_2) = l\}.$$

Summing over  $l$  counts the disagreement mass twice, and  $\text{TV} = \frac{1}{2} \sum_l |\cdot|$  halves it. This is the classical coupling lemma [166, §I.2], in Doeblin's tradition [63]; stated for arbitrary nonnegative  $\gamma$  because normalization is never consumed — Step 2 of the theorem applies it to the final pair law, whose unit mass Lemma R91.5 supplies separately.  $\square$

**Lemma R91.5** (final marginal identities). *Suppose additionally each component chain absorbs almost surely (its absorbed mass tends to 1). Then for  $a_1 \in A_1$ :  $\sum_{a_2 \in A_2} \text{fin}_K(a_1, a_2) = \text{fin}_{K_1}(a_1)$ , and symmetrically. Consequently the final pair law on  $A_1 \times A_2$  is a genuine coupling of the two final component laws, of total mass 1.*

*Proof.* Let  $S_n = \sum_{a_2 \in A_2} \mu_n(a_1, a_2)$  and  $T_n = \sum_{s_2} \mu_n(a_1, s_2)$ ; by Lemma R91.1,  $T_n$  is the component-1 law at  $a_1$ . Then  $S_n \leq T_n$ , and

$$T_n - S_n = \sum_{s_2 \notin A_2} \mu_n(a_1, s_2) \leq \sum_{s_2 \notin A_2} (\pi_{2\#} \mu_n)(s_2) = 1 - (\text{component-2 absorbed mass at } n) \rightarrow 0 :$$

the *one-absorbed-loner* mass — pairs whose first coordinate has frozen while the second still wanders — is dominated by the second chain’s unabsorbed mass, which its absorption hypothesis kills. Squeeze; both-absorbed pairs are frozen, so  $S_n$  also converges to the final pair mass. Total mass 1 follows by summing over  $A_1$  and applying component-1 absorption.  $\square$

**Theorem R91.6** (soundness, total-variation form). *Under all hypotheses,  $\text{TV}(f_{\#} \text{fin}_{K_1}, g_{\#} \text{fin}_{K_2}) \leq b$ .*

*Proof.* *Step 1.* By Lemma R91.5 the final pair law  $\gamma$  on  $A_1 \times A_2$  has marginals  $\text{fin}_{K_1}, \text{fin}_{K_2}$ , so the two final readout laws are the two pushforwards of the single weight  $\gamma$ . *Step 2.* Lemma R91.4 bounds their total-variation distance by  $\gamma$ ’s disagreement mass. *Step 3.* Terminal domination says the disagreement indicator is  $\leq V$  on the frozen set  $A_1 \times A_2$  (and  $\geq 0$  everywhere), so Lemma R91.3 with  $\gamma = 1[f \neq g]$  bounds the disagreement mass by  $b$ .  $\square$

**Theorem R91.7** (soundness, expectation-gap form). *With terminal domination in the form  $V(p) \geq |v_1(p_1) - v_2(p_2)|$  on  $A_1 \times A_2$ ,*

$$\left| \sum_{a_1} \text{fin}_{K_1}(a_1) v_1(a_1) - \sum_{a_2} \text{fin}_{K_2}(a_2) v_2(a_2) \right| \leq b.$$

*Proof.* Rewrite both final expectations over the final pair law via Lemma R91.5; the difference becomes  $\sum_p \text{fin}_K(p) (v_1(p_1) - v_2(p_2))$  over  $A_1 \times A_2$ . The triangle inequality moves the absolute value inside, and Lemma R91.3 with  $\gamma = |v_1 - v_2|$  closes. The coupling lemma is not consumed here — domination of the value gap replaces domination of the disagreement indicator; two conclusion families, one engine.  $\square$

**Remark R91.8** (the absorption criterion is discharged, not assumed). The checker certifies absorption per component as a finite-graph fact: from every state, the frozen set is reachable through positive-probability edges. Finiteness uniformizes this into one horizon  $N$  and one margin  $\varepsilon > 0$  valid for all states; each  $N$ -step block then absorbs at least an  $\varepsilon$ -fraction of the still-loose mass (frozen mass never returns), so the unabsorbed mass decays geometrically and the absorbed mass tends to 1 (`tendsto_absorbedMass_one`). The `_of_reach` theorem forms consume the graph criterion directly.

**Remark R91.9** (why each hypothesis is needed). *Marginal identities:* drop them and  $\gamma$  in Theorem R91.6 no longer has the two systems’ final laws as marginals — the bound then constrains an unrelated pair (couple  $\mathcal{M}_1$  with a copy of *itself* instead of  $\mathcal{M}_2$ : every absorbed pair agrees, so  $V = 0$  on the reachable diagonal — with  $V = 1$  on the unreachable disagreeing absorbed pairs — passes every other row at  $b = 0$ , whatever  $\mathcal{M}_2$  is).  $V \geq 0$ : the extension step in Lemma R91.3 ( $\sum_A \leq \sum_{\text{all}}$ ) fails, and negative values can subsidize the generator row: from an initial state of value 0, branch half-half into a frozen *disagreeing* state of value 1 and a frozen *agreeing* state of value  $-1$  (domination constrains only disagreeing states). The generator row holds at every state ( $\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot (-1) = 0 \leq 0$ ), the initial bound passes at  $b = 0$  — and the disagreement mass is  $\frac{1}{2}$ . *Terminal domination:* with  $V \equiv 0$  and disagreeing finals, generator and initial rows pass at  $b = 0$  while the true distance is 1. *The freeze:* monotone convergence of frozen mass (Lemmas R91.3, R91.5) is what makes “final law” well-defined; unfrozen, the two-state swap chain puts mass 1 on the target at even times and 0 at odd —  $\sup_n \mu_n(a)$  is no longer a limit, and certified mass drains back out. *Absorption:* without it the one-absorbed-loner mass survives — take  $K_2$  the identity off  $A_2$ : the pair never both-absorbs, the final pair law is 0, and Lemma R91.4 holds vacuously about sub-laws that no longer represent the systems; the marginal identity of Lemma R91.5 is exactly what fails — and the bound itself fails with it: component 1 absorbing at once, component 2 frozen nowhere it reaches,  $V = 0$  on the reachable ray and 1 on the (unreachable) disagreeing both-absorbed pairs passes every other row at  $b = 0$ , while the final readout laws sit at total-variation distance  $\frac{1}{2}$ . *Initial bound:* the conclusion’s  $b$  is  $V$ ’s initial expectation; without it nothing quantitative survives. A coupling that cannot satisfy the marginal rows simply does not exist as a certificate — the coupling is an input premise, so its emptiness refuses (a completeness cost), never lies.

**Remark R91.10** (relation to the literature). The engine is the coupling method for Markov chains [63, 166]: total variation bounded by disagreement probability of a coupling, here in distribution-evolution form over finite spaces [160, Ch. 4–5], with a relational supermartingale supplying the quantitative disagreement bound — the certificate-side sibling of coupling-based relational program logics [116], transposed from pre/post judgments to a single distance judgment that composes by metric axioms (triangle, weakening) at judgment level. The expectation-gap form replaces the coupling lemma by direct domination — the gap-pseudometric readout. On correlated perturbations the relational route is strictly sharper than composing two one-sided certificates, which is the rule’s reason to exist: the coupling sees the correlation, unary certificates cannot.

#### proof step

Lemma R91.1  
 Lemma R91.2  
 Lemma R91.3  
 Lemma R91.4

#### formal anchor

PushKernel; push\_step; push\_law  
 expect\_step\_le; expect\_law\_le  
 law\_mono\_absorbing; finalMass; finalExpect\_le  
 tvDist; tvDist\_push\_le\_disagree

|               |                                                                   |
|---------------|-------------------------------------------------------------------|
| Lemma R91.5   | sum_finalMass_snd_eq; sum_finalMass_fst_eq;                       |
| Theorem R91.6 | sum_finalMass_prod_eq_one; push_finalLaw_fst/_snd                 |
| Theorem R91.7 | sound_QRelTV (graph-criterion form sound_QRelTV_of_reach)         |
| Remark R91.8  | sound_QRelExpGap (graph-criterion form sound_QRelExpGap_of_reach) |
|               | exists_uniform_reach; absorbed_decay; tendsto_absorbedMass_one    |

## Rule 92. QFixBall (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — *QFixBall*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} d : \text{QDistRegistry}(X_d); \quad (X_d, d) \text{ complete} \quad \vdash \rho < 1 \text{ two-point on } \overline{B}(\hat{\mu}, r) \text{ (contraction row)} \\ \vdash d(T\hat{\mu}, \hat{\mu}) \leq \delta \text{ (residual row)} \quad \vdash \delta + \rho r \leq r \text{ (radius row, division-free)} \end{array}}{\vdash \text{QDist}[d](\hat{\mu}, \text{fix } T) \leq r}$$

every hypothesis anchored at the known snapshot  $\hat{\mu}$ , never at the unknown fixed point: the contraction row discharges by operator-norm/psdCert rows at the affine tier, the residual row is one concrete sweep at the known point, and completeness is mechanized at the affine/Euclidean instantiation ( $\mathbb{W}_2$  completeness by instance on  $(\mathcal{P}_2(X), W_2)$ ). The conclusion's definite description needs no separate license premise: this rule is the only derivation that makes  $\text{fix } T$  speakable, and it discharges the description's well-formedness itself — exactly one fixed point exists in  $\overline{B}(\hat{\mu}, r)$ , manufactured by the contraction (the  $\exists!$  of the soundness statement; no existence stratum consumed). The registry typing has no baked sort yet (the promoted QDist kind awaits its row species); completeness is the instance hypothesis of the Lean statement. Refusals:  $\rho \geq 1$ , and registry entries the subject sort does not carry.

QFixBall is the contraction-witnessed discharge of the distance judgment: it certifies that a computed snapshot  $\hat{\mu}$  lies within distance  $r$  of the fixed point of an update map  $T$  — and, in the same breath, that “the fixed point of  $T$ ” is a well-defined description at all (exactly one fixed point exists in the certified ball). A producer reaches for it wherever an iterative solver has been run to approximate a fixed-point object — a stationary law, a variational posterior, a self-consistent field — and the deliverable is not the iterate but a certified *ball* around it. Its three rows realize the anchoring inversion: every hypothesis is stated at the *known* snapshot, never at the unknown fixed point — the contraction row bounds  $T$ 's two-point ratio on the closed ball around  $\hat{\mu}$ , the residual row measures one concrete application  $d(T\hat{\mu}, \hat{\mu}) \leq \delta$ , and the radius row  $\delta + \rho r \leq r$  is division-free arithmetic.

What the checker recomputes: the contraction bound  $\rho < 1$  (at the affine tier, an operator-norm row discharged by positive-semidefinite certificate machinery on  $\rho^2 I - A^\top A$ -shaped data), the residual evaluation (one sweep of the map at the snapshot, exact rational arithmetic), and the radius inequality. Everything else — ball self-mapping, existence, uniqueness, the anchored bound  $d(\hat{\mu}, \text{fix } T) \leq \delta/(1 - \rho) \leq r$  — is manufactured by the mathematics from those three rows; no existence stratum is consumed, so the judgment is existential without vacuity.

What would break: with  $\rho \geq 1$  the fixed point need not be unique or exist and the description license fails — the rule refuses (the identity map has residual 0 at every point and fixes all of them). Without the radius row the ball need not be self-mapped and can miss the fixed point entirely, even with a genuine contraction. Without the residual row nothing ties the ball to the dynamics. The completeness of the subject space is a genuine hypothesis, carried honestly — and both named instantiations now discharge it by instance: the affine/Euclidean case directly, and the Wasserstein case on  $(\mathcal{P}_2(X), W_2)$  over a complete separable base (the instantiation block below) — never an axiom smuggled in.

**notes** The rule concludes onto a definite-description subject ( $\text{fix } T$ ) and therefore carries the description well-formedness discharge inside the rule — the one new typing obligation of the distance-judgment grammar. An independent exact-arithmetic checker replays a pilot certificate's eleven obligations; its mutation battery refuses each mutilated certificate at a named obligation. Producer strategy (proposing  $\rho$  from iterate data, pinning the affine reduction) is deliberately outside the kernel.

### Lean statement

BernsteinLogicAux.QFixBall . sound\_QFixBall

statement bound

```
-- **`sound_QFixBall`** -- the rule's full conclusion: the certified
rows produce EXACTLY ONE fixed point of `f` in `closedBall x₀ r`.
Existence is manufactured by the contraction itself (the third case of
the two-direction carve: no existence stratum is consumed, and the
judgment is existential without vacuity). `[CompleteSpace α]` is the
honest boundary – see the header. -/
theorem sound_QFixBall [CompleteSpace α] (hK : K < 1)
  (hLip : LipschitzOnWith K f (closedBall x₀ r))
  (hres : dist (f x₀) x₀ ≤ δ) (hrad : δ + K * r ≤ r) :
  ∃! y, y ∈ closedBall x₀ r ∧ IsFixedPt f y
```

BernsteinLogicAux.QFixBall . sound\_QFixBall\_affine

statement bound

```

/-- **`sound_QFixBall_affine`** – the caviwball-v0 instantiation shape:
snapshot mean vector in `EuclideanSpace ℝ (Fin M)` (completeness is an
instance – the `[CompleteSpace]` boundary discharges for free here),
affine sweep map, operator-norm + residual + radius rows  $\Rightarrow$  unique
fixed point in the certified ball. -/
theorem sound_QFixBall_affine {M : ℕ}
  (A : EuclideanSpace ℝ (Fin M)  $\rightarrow$  L[ℝ] EuclideanSpace ℝ (Fin M))
  (c x₀ : EuclideanSpace ℝ (Fin M)) {K : ℝ $\geq$ 0} {δ r : ℝ}
  (hK : K < 1) (hA : ||A|| $_+$   $\leq$  K)
  (hres : dist (A x₀ + c) x₀  $\leq$  δ) (hrad : δ + K * r  $\leq$  r) :
   $\exists!$  y, y  $\in$  closedBall x₀ r  $\wedge$  IsFixedPt (fun x  $\Rightarrow$  A x + c) y

```

BernsteinLogicAux.QFixBall. **dist\_fixedPoint\_le**

statement bound

```

/-- **The anchored a-priori bound** – for ANY fixed point in the ball
(uniqueness not consumed): `dist x₀ μ*  $\leq$  δ / (1 - K)`. Proved by one
triangle inequality through the snapshot's image; the literature's
one-point-at-the-limit anchoring never appears. -/
theorem dist_fixedPoint_le (hK : K < 1)
  (hlip : LipschitzOnWith K f (closedBall x₀ r))
  (hres : dist (f x₀) x₀  $\leq$  δ) (hrad : δ + K * r  $\leq$  r)
  {y : α} (hy : y  $\in$  closedBall x₀ r) (hfy : IsFixedPt f y) :
  dist x₀ y  $\leq$  δ / (1 - K)

```

BernsteinLogicAux.QFixBall. **anchored\_le\_radius**

statement bound

```

/-- The radius row also caps the anchored bound by `r` itself
(`δ/(1-K)  $\leq$  r`) – the form the checker's ball-radius obligation uses. -/
theorem anchored_le_radius (hK : K < 1)
  (hrad : δ + K * r  $\leq$  r) : δ / (1 - K)  $\leq$  r

```

**The Wasserstein instantiation, discharged by instance.** The completeness hypothesis named above is closed on the space where this rule's stochastic clients live: the statement below runs `sound_QFixBall` on  $(\mathcal{P}_2(X), W_2)$  — second-moment laws under quadratic Wasserstein distance over a complete separable metric space — with the `CompleteSpace` side condition supplied by instance resolution rather than hypothesis. No completeness assumption survives in the statement; the affine and Wasserstein instantiations now stand on the same footing.

BernsteinLogicAux.Wasserstein.Complete. **sound\_QFixBall\_wasserstein**

statement bound

```

/-- **`sound_QFixBall`**, instantiated at the Wasserstein-2 space.** The
`[CompleteSpace]` hypothesis – the honest boundary of
`BernsteinLogicAux/QFixBall.lean` (header item 1), and the side
condition that kept the rule's coverage marked conditional – is
discharged here **by instance**: for any complete separable metric `X`,
certified contraction/residual/radius rows on  $(\mathcal{P}_2(X), W_2)$  produce
the unique fixed point in the certified ball. -/
theorem sound_QFixBall_wasserstein {T : P2 X  $\rightarrow$  P2 X} {μ₀ : P2 X} {K : ℝ $\geq$ 0}
  {δ r : ℝ} (hK : K < 1)
  (hlip : LipschitzOnWith K T (closedBall μ₀ r))
  (hres : dist (T μ₀) μ₀  $\leq$  δ) (hrad : δ + K * r  $\leq$  r) :
   $\exists!$  ν, ν  $\in$  closedBall μ₀ r  $\wedge$  IsFixedPt T ν

```

## Correspondence (Steele display $\leftrightarrow$ Lean)

reviewer-certified against the displayed texts

The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The displayed conclusion splits, and so do the bound statements: the description license — fix  $T$  well-defined (unique) in  $\overline{B}(\hat{\mu}, r)$  —  $\mapsto$  `sound_QFixBall` :  $\exists! y, y \in \text{closedBall } x_0 \ r \wedge \text{IsFixedPt } f \ y$  (existence manufactured by the contraction itself; no existence stratum consumed, exactly as the roster says); the distance clause  $\text{QDist}[d](\hat{\mu}, \text{fix } T) \leq r$  is then ball membership, with the sharper anchored bound  $\delta/(1 - \rho)$  carried by `dist_fixedPoint_le` — whose binder row shows uniqueness is NOT consumed (any fixed point in the ball obeys it) — and capped by the radius row via `anchored_le_radius`.



- contraction row, two-point on the ball  $\mapsto$  hlip : LipschitzOnWith K f (closedBall x<sub>0</sub> r) with K :  $\mathbb{R}_{\geq 0}$ , hK : K < 1 — the  $\rho \geq 1$  refusal is the strict binder; at the affine tier the operator-norm row  $\mapsto$  hA :  $\|A\|_+ \leq K$  in sound\_QFixBall\_affine (EuclideanSpace, the caviwball-v0 pinned mean map).
- residual row  $\mapsto$  hres : dist (f x<sub>0</sub>) x<sub>0</sub> ≤ δ — anchored at the snapshot, and no hypothesis anywhere mentions the fixed point: the flavor text's “never at the unknown fixed point” is a checkable property of the binder list.
- radius row  $\mapsto$  hrad : δ + K \* r ≤ r, division-free verbatim.
- registry completeness  $\mapsto$  the instance binder [CompleteSpace α] — a named hypothesis of the general statement, never an axiom, and discharged by instance resolution in both named instantiations: directly for the affine/Euclidean case, and for  $\mathbb{W}_2$  by the Wasserstein block (sound\_QFixBall\_wasserstein) below.

Width note: the subject space is an arbitrary metric space — the registry-entry structure of  $d$  is erased, so the certified reach covers every registry instantiation whose completeness discharge arrives (both named instantiations' discharges are now in hand), strictly wider than the displayed registry framing, safe direction; the means-only (variance-conclusion) refusal and the Gaussian-pinning producer strategy are kernel-external, no Lean counterpart, correctly.

## Lean verifier receipts

module BernsteinLogicAux.QFixBall; bound build 68796bc2f9db... (bound from receipt 060c7093ff4899bf, 2026-07-21T04:51:42Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| a88bac06d271526e | 2026-07-21T00:27:07Z | build+decorrelated                 | current |
| 060c7093ff4899bf | 2026-07-21T04:51:42Z | build+decorrelated                 | current |
| ae5851604e101130 | 2026-07-21T05:38:57Z | build+replay+hardened+decorrelated | current |
| 85c6bac865d222ba | 2026-07-22T18:27:12Z | build+decorrelated                 | current |
| bf722ef76b576c08 | 2026-07-31T22:00:08Z | build+decorrelated                 | current |
| 0caae5e60f19815a | 2026-07-31T22:29:02Z | build+decorrelated                 | current |
| 9780a725a44c924b | 2026-08-02T23:31:13Z | build+decorrelated                 | current |
| 490335613ee08847 | 2026-08-03T16:27:17Z | build+decorrelated                 | current |
| e68b74ba7bf3e8de | 2026-08-03T22:46:01Z | build+decorrelated                 | current |

5 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

module BernsteinLogicAux.Wasserstein.Complete; bound build 06292c010c42... (bound from receipt fc2b784b4f0c347e, 2026-07-29T08:09:21Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| f660300e93359aad | 2026-07-28T23:54:07Z | build+decorrelated                 | current |
| ddcacf582a03fe61 | 2026-07-29T05:43:06Z | build+decorrelated                 | current |
| fc2b784b4f0c347e | 2026-07-29T08:09:21Z | build+replay+hardened+decorrelated | current |
| 439956f3f0160d8a | 2026-07-31T22:00:35Z | build+decorrelated                 | current |
| d52523d643b3462b | 2026-07-31T22:29:29Z | build+decorrelated                 | current |
| a5c8597e9187735d | 2026-08-02T23:31:45Z | build+decorrelated                 | current |
| 9f4e8bbd7042154f | 2026-08-03T16:27:49Z | build+decorrelated                 | current |
| c0e6026aa1939c74 | 2026-08-03T22:46:33Z | build+decorrelated                 | current |

4 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

build (lake build (elaboration)) current

replay (leanchecker (official kernel replay)) current

hardened (safe\_verify (hardened screen)) current

decorrelated (lean4export + nanoda (decorrelated kernel)) current

comparator: none no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.



## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development's object is the base calculus's wire-format checker (theory `bern_catalogue` ports the first-order check-function cone of the core catalogue). QFIXBALL lives one extension edge above that stratum — it concludes a distance judgment of the extension theory, its rule sort is out of scope of the current baked theory, and its mechanization lives in the auxiliary Lean package. There is nothing checker-shaped for a HOL statement to be about yet; the box becomes writable when a revision of the theory carries the certificate objects and the corresponding checker chain is ported.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1) — Banach's fixed-point theorem [15] in certificate form. Scope, stated loudly: the theorems are parametric in a complete metric space; the affine/Euclidean instantiation is mechanized, and completeness of the Wasserstein space is a *named hypothesis*, not a discharged one. Every hypothesis of the rule is used below and remarked on.

**Setting and claim.** Fix a metric space  $(X, d)$ , a map  $T : X \rightarrow X$ , a snapshot  $\hat{\mu} \in X$  (the object the producer actually computed), and rational certificate data  $\rho \in \mathbb{Q} \cap [0, 1)$ ,  $\delta, r \in \mathbb{Q}$ . The three certified rows:

$$\begin{array}{ll} \text{(H-ANCHOR)} & d(Tx, Ty) \leq \rho d(x, y) \text{ for all } x, y \in \overline{B}(\hat{\mu}, r) & \text{(two-point contraction on the ball),} \\ \text{(H-RESID)} & d(T\hat{\mu}, \hat{\mu}) \leq \delta & \text{(one concrete sweep at the snapshot),} \\ \text{(H-RADIUS)} & \delta + \rho r \leq r & \text{(division-free; equivalent to } r \geq \delta/(1 - \rho)\text{).} \end{array}$$

The claim (`sound_QFixBall`): if  $X$  is complete, there is *exactly one* fixed point of  $T$  in  $\overline{B}(\hat{\mu}, r)$  — so the description `fix T` is licensed on the ball, and the distance judgment  $d(\hat{\mu}, \text{fix } T) \leq r$  holds; moreover the sharper anchored bound  $d(\hat{\mu}, \text{fix } T) \leq \delta/(1 - \rho)$  holds (`dist_fixedPoint_le, anchored_le_radius`). Every hypothesis is anchored at the *known* point  $\hat{\mu}$ ; the unknown fixed point appears only in conclusions — the anchoring inversion that makes the statement checkable at all.

**Lemma R92.1** (the rows are sign-coherent). *H-RESID forces  $\delta \geq 0$ ; together with  $\rho < 1$ , H-RADIUS forces  $r \geq 0$ , so the ball is honest.*

*Proof.*  $0 \leq d(T\hat{\mu}, \hat{\mu}) \leq \delta$ . From H-RADIUS,  $(1 - \rho)r \geq \delta \geq 0$ , and  $1 - \rho > 0$ . □

**Lemma R92.2** (self-mapping is derived, never assumed). *T maps  $\overline{B}(\hat{\mu}, r)$  into itself.*

*Proof.* Let  $d(x, \hat{\mu}) \leq r$ . Both  $x$  and  $\hat{\mu}$  lie in the ball (Lemma R92.1 for  $\hat{\mu}$ ), so H-ANCHOR applies to the pair:

$$d(Tx, \hat{\mu}) \leq d(Tx, T\hat{\mu}) + d(T\hat{\mu}, \hat{\mu}) \leq \rho d(x, \hat{\mu}) + \delta \leq \rho r + \delta \leq r. \quad \square$$

**Lemma R92.3** (in-ball uniqueness). *If  $y, z \in \overline{B}(\hat{\mu}, r)$  are both fixed by  $T$ , then  $y = z$ .*

*Proof.*  $d(y, z) = d(Ty, Tz) \leq \rho d(y, z)$  with  $\rho < 1$  forces  $d(y, z) \leq 0$ , hence  $= 0$ . Note the proof consumes only the *two-point* row, on the ball — no iteration, no completeness. □

**Lemma R92.4** (the anchored a-priori bound). *Any fixed point  $y \in \overline{B}(\hat{\mu}, r)$  satisfies  $d(\hat{\mu}, y) \leq \delta/(1 - \rho) \leq r$ .*

*Proof.* One triangle inequality through the snapshot's image:

$$d(\hat{\mu}, y) \leq d(\hat{\mu}, T\hat{\mu}) + d(T\hat{\mu}, Ty) \leq \delta + \rho d(\hat{\mu}, y),$$

so  $(1 - \rho)d(\hat{\mu}, y) \leq \delta$ . The cap by  $r$  is H-RADIUS rearranged:  $\delta \leq (1 - \rho)r$  (`anchored_le_radius`). Uniqueness is not consumed — the bound holds for *any* fixed point in the ball, which is what lets the rule state it before existence is settled. □

**Theorem R92.5** (soundness of QFIXBALL). *Let  $X$  be complete and let the three rows hold. Then  $T$  has exactly one fixed point  $\mu^*$  in  $\overline{B}(\hat{\mu}, r)$ , and  $d(\hat{\mu}, \mu^*) \leq \delta/(1 - \rho) \leq r$ .*

*Proof. Existence.* By Lemma R92.2 the closed ball is  $T$ -invariant; it is a closed subset of a complete space, hence complete, and  $T$  restricted to it is a  $\rho$ -contraction (H-ANCHOR). Banach's theorem [15] applies to the restriction: the Picard iterates  $T^n \hat{\mu}$  form a Cauchy sequence ( $d(T^{n+1} \hat{\mu}, T^n \hat{\mu}) \leq \rho^n \delta$ , summable) whose limit  $\mu^*$  lies in the ball and is fixed. Existence is thus *manufactured* by the contraction — the rule consumes no separate existence certificate, so the concluded description is existential without vacuity. *Uniqueness* is Lemma R92.3; the *bound* is Lemma R92.4. Together: `fix T` is well-defined on the ball (the description license the judgment carries) and the distance judgment holds at radius  $r$ , with  $\delta/(1 - \rho)$  as the sharper certified value. □

**Proposition R92.6** (the affine instantiation). *On a finite-dimensional Euclidean space, let  $Tx = Ax + c$  with  $A$  linear and an operator-norm row  $\|A\| \leq \rho < 1$ . Then  $H\text{-ANCHOR}$  holds globally (a fortiori on the ball), completeness holds by instance, and Theorem R92.5 applies verbatim (`sound_QFixBall_affine`).*

*Proof.*  $d(Tx, Ty) = \|A(x - y)\| \leq \|A\| \|x - y\| \leq \rho d(x, y)$ . The operator-norm row is the affine tier’s discharge of  $H\text{-ANCHOR}$ : it is certified by positive-semidefiniteness of  $\rho^2 I - A^\top A$ , which is PSDCERT-family machinery on rational data.  $\square$

*Remark R92.7* (why each hypothesis is needed). *Drop  $\rho < 1$* : the identity map has residual 0 at every snapshot and every radius row holds with  $\rho = 1$ , yet every point is fixed — the description license (uniqueness) fails, so the rule *refuses*  $\rho \geq 1$  rather than concluding. *Drop  $H\text{-RADIUS}$* : on  $\mathbb{R}$ , take  $Tx = \frac{1}{2}x + 2$ ,  $\hat{\mu} = 0$ , so  $\rho = \frac{1}{2}$ ,  $\delta = 2$ ; the unique fixed point is 4. With  $r = 1$  the ball  $[-1, 1]$  contains no fixed point — and indeed  $\delta + \rho r = \frac{5}{2} > 1$ : the radius row fails, correctly. *Drop  $H\text{-RESID}$* : nothing ties the ball to the dynamics — any  $r$  would pass on any map with small  $\rho$ , concluding a distance bound to a fixed point that can sit arbitrarily far away. *Weaken completeness*: on  $X = \mathbb{Q} \cap [1, 2]$  (incomplete),  $Tx = \frac{x}{2} + \frac{1}{x}$  maps  $X$  into itself with  $|T'| \leq \frac{1}{2}$  and contracts toward the irrational  $\sqrt{2}$  — iterates Cauchy, no fixed point in  $X$ , and the certificate rows even pass ( $\hat{\mu} = \frac{3}{2}$ ,  $r = \frac{1}{2}$ ,  $\rho = \frac{1}{2}$ ,  $\delta = \frac{1}{12}$ , with  $\delta + \rho r = \frac{1}{3} \leq \frac{1}{2}$ ); the conclusion’s existential half is exactly what completeness buys. The two-point row being required only *on the ball* is deliberate: locality is what a finite certificate can attest (globally,  $\|A\| < 1$  may even be false while the ball-restricted bound holds).

*Remark R92.8* (relation to the literature). The theorem is Banach’s [15]; the closed-ball variant with the condition  $d(T\hat{\mu}, \hat{\mu}) \leq (1 - \rho)r$  and the a-priori bound  $\delta/(1 - \rho)$  is the textbook local form [99, §I.1]. The certificate-form contribution is the *anchoring inversion*: classical statements estimate the distance from an iterate to the limit; the rule re-expresses every hypothesis as a row checkable at the known snapshot in exact rational arithmetic — contraction as an operator-norm (or two-point) row, residual as one evaluated sweep, radius as division-free arithmetic — so the analytic content survives while everything the checker touches is decidable. The completeness gap for measure-space subjects (Wasserstein distance over a Euclidean base) is carried as a named hypothesis; the pilot certificate discharges it by pinning the subject to the affine/Euclidean reduction of Proposition R92.6.

| proof step        | formal anchor                                                                              |
|-------------------|--------------------------------------------------------------------------------------------|
| Lemma R92.1       | <code>delta_nonneg, radius_nonneg</code>                                                   |
| Lemma R92.2       | <code>mapsTo_closedBall</code>                                                             |
| Lemma R92.3       | <code>fixedPoint_unique</code>                                                             |
| Lemma R92.4       | <code>dist_fixedPoint_le; anchored_le_radius</code>                                        |
| Theorem R92.5     | <code>sound_QFixBall</code> (existence via the contraction on the complete invariant ball) |
| Proposition R92.6 | <code>lipschitzWith_affine; sound_QFixBall_affine</code>                                   |

## Rule 93. QDISTTRANSPORT (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *QDistTransport*. The metanotation statement is typeset by hand for notational quality; its mechanized ground truth is the Lean statement below, extracted from the sources at every build of this book and hash-bound. The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial, checked by eye against the extracted Lean beside it.

$$\frac{\begin{array}{l} J = (\mathcal{M}_c \models s) \text{ (strict, law-level) } \text{ or } J = (\text{QDist}[\text{TV}](\mu_c, \nu) \leq \delta), \text{ on the coarse side} \\ \{ \vdash \text{tv}(\pi_{\#} K(s, \cdot), Q(\pi(s), \cdot)) \leq \varepsilon \}_{s \in S_f} \text{ (}\sigma\text{-witness rows, one per fine state; } \varepsilon = 0 \text{ is exact intertwining)} \\ \vdash \mu_c^0 = \pi_{\#} \mu_f^0 \text{ (initial row)} \quad n \in \mathbb{N} \text{ (declared horizon; } \varepsilon\text{-cell only)} \end{array}}{\begin{array}{l} \text{exact cell: } J \text{ transports unchanged} \quad \varepsilon\text{-cell: } \text{QDist}[\text{TV}](\pi_{\#}(\text{fine law at } n), \text{ coarse law at } n) \leq n \varepsilon \end{array}}$$

$\pi : S_f \rightarrow S_c$  is the state projection,  $K$  and  $Q$  the fine and coarse kernels,  $\mu^0$  the initial laws; each  $\sigma$ -witness row asserts that one row of the fine kernel pushes along  $\pi$  to within total-variation  $\varepsilon$  of the coarse kernel's row at the projected state. Unbounded horizons refuse without a contraction row. The displayed  $J$  is restricted to the built judgment classes (strict law-level specs and  $\text{QDist}[\text{TV}]$ ); the successor cells widen it as they land. Transport is a matrix (judgment class  $\times$  witness class), each built cell carrying its own lemma and refusal condition; sequential witnesses compose under the certified budget  $\varepsilon_1 + \varepsilon_2$  (an upper bound — the actual composite defect may be smaller), and the strict column is the  $\varepsilon = 0$  instantiation of the family. Built at the finite-kernel tier; the Lipschitz-pushforward ( $\mathbb{W}_p$ ) and description-subject cells are named successors.

**QDISTTRANSPORT** moves judgments across a simulation map. It is a *matrix* of transport cells — judgment class (strict spec vs. distance judgment)  $\times$  witness class (exact intertwining vs.  $\varepsilon$ -approximate intertwining) — each cell carrying its own conclusion form, its own lemma, and its own refusal condition. A producer reaches for it to certify a fine model through a coarse one: prove the spec on the abstraction, exhibit the simulation witness, and transport. The two built kernel-tier cells: under an *exact* witness (every row of the fine kernel pushes along the readout map exactly onto the coarse kernel's row) laws transport unchanged at every horizon, so any judgment about the coarse law pulls back with zero degradation; under an  $\varepsilon$ -witness (every row pushes to within total-variation distance  $\varepsilon$ ) a strict judgment does NOT transport — what transports is a *distance* judgment, with  $\varepsilon$  inherited and horizon-compounded: the pushed fine law tracks the coarse law to  $n \varepsilon$  at declared horizon  $n$ . The degrading cell is what makes approximate simulation a distance-judgment *producer*.

What the checker recomputes: the per-row intertwining identities (exact case) or per-row total-variation defects against the declared  $\varepsilon$  (approximate case), well-formedness of both kernels, and the horizon arithmetic. The compounding law  $n \varepsilon$  is proved, never guessed: one-step defect by mixture convexity, then the coarse Markov operator's total-variation nonexpansiveness (data processing — the coarse chain cannot amplify tracking error), then telescoping. Sequential witnesses compose with certified budget  $\varepsilon_1 + \varepsilon_2$  — an upper bound with no additional bookkeeping slack (plain addition is the honest lattice at this tier), though the composite's actual defect may be smaller.

What would break: without the coarse kernel's row stochasticity the operator could amplify differences and the telescoping collapses; without a genuine probability law at the start, mixture convexity fails; and without a contraction row the linear growth in  $n$  is sharp, which is why the  $\varepsilon$ -cell refuses unbounded horizons rather than extrapolating.

**notes** The strict column is the  $\varepsilon = 0$  instantiation of the family — one biconditional, so the exact cell re-exports the marginal-process lemma rather than restating it. Analytic lineage: approximate (bi)simulation with quantitative degradation [93] and behavioural pseudometrics for probabilistic systems [61]. Built at the finite-kernel tier (the scope the Lean statements carry); the Lipschitz-pushforward cells for  $\mathbb{W}_p$  registry entries, the description-subject cells, and the per-cell composition-coherence squares are named successors.

### Lean statement

BernsteinLogicAux.QDistTransport . sound\_QDistTransport\_eps

statement bound

```
-- **`sound_QDistTransport_eps` — the degrading cell's transport
theorem** (matrix cell: strict judgment  $\times$   $\varepsilon$ -witness  $\Rightarrow$  distance
judgment): at horizon `n`, the pushed fine law and the coarse law are
within TV-distance `n * ε`. Telescoping: one-step defect (`ε`, above) +
TV-nonexpansiveness of the coarse operator (§3) + triangle. This is
the theorem that makes "ε inherited, horizon-compounded" a proved
compounding law rather than a guess; with no contraction row the
linear growth is the honest rate (the refusal condition for unbounded
horizons lives spec-side). -/
theorem sound_QDistTransport_eps [DecidableEq β] {π : α → β}
  {K : α → α → ℝ} {Q : β → β → ℝ} {ε : ℝ} {μ₀ : α → ℝ}
  (hK : IsMarkov K) (hQ : IsMarkov Q) (hμ₀ : IsDist μ₀)
  (h : EpsPushKernel π K Q ε) (n : ℕ) :
  tvDist (push π (law K μ₀ n)) (law Q (push π μ₀) n) ≤ n * ε
```

BernsteinLogicAux.QDistTransport . **epsPushKernel\_comp**

statement bound

```

/-- ** $\varepsilon$ -arithmetic composes** (derived, not guessed): transport along
` $\sigma_1$ ` then ` $\sigma_2$ ` carries the certified budget ` $\varepsilon_1 + \varepsilon_2$ ` — an upper bound
(triangle through the mid-tier pushforward + data processing; the actual
composite defect may be smaller). Plain addition is the right
composition law at the TV tier. -/
theorem epsPushKernel_comp [DecidableEq  $\beta$ ] [DecidableEq  $\gamma$ ]
  { $\pi_1 : \alpha \rightarrow \beta$ } { $\pi_2 : \beta \rightarrow \gamma$ } { $K_1 : \alpha \rightarrow \alpha \rightarrow \mathbb{R}$ } { $K_2 : \beta \rightarrow \beta \rightarrow \mathbb{R}$ }
  { $K_3 : \gamma \rightarrow \gamma \rightarrow \mathbb{R}$ } { $\varepsilon_1 \ \varepsilon_2 : \mathbb{R}$ }
  (h1 : EpsPushKernel  $\pi_1$   $K_1$   $K_2$   $\varepsilon_1$ ) (h2 : EpsPushKernel  $\pi_2$   $K_2$   $K_3$   $\varepsilon_2$ ) :
  EpsPushKernel ( $\pi_2 \circ \pi_1$ )  $K_1$   $K_3$  ( $\varepsilon_1 + \varepsilon_2$ )

```

BernsteinLogicAux.QDistTransport . **sound\_QDistSelfTransport**

statement bound

```

/-- **Matrix cell (distance judgment  $\times$  strict witness), TV entry**: a
TV distance judgment transports along any simulation/readout map
unchanged (registry reading:  $TV \rightarrow TV$  at Lipschitz constant 1). -/
theorem sound_QDistSelfTransport [DecidableEq  $\beta$ ] ( $\pi : \alpha \rightarrow \beta$ )
  { $\mu \ v : \alpha \rightarrow \mathbb{R}$ } { $\varepsilon : \mathbb{R}$ } (h : tvDist  $\mu \ v \leq \varepsilon$ ) :
  tvDist (push  $\pi \ \mu$ ) (push  $\pi \ v$ )  $\leq \varepsilon$ 

```

BernsteinLogicAux.QDistTransport . **strict\_cell\_exact**

statement bound

```

/-- **Matrix cell (strict judgment  $\times$  strict/kernel witness)**: under
the exact witness, laws transport EXACTLY at every horizon
(`QRelChain.push_law` re-exported as the matrix's strict kernel cell)
— so any judgment about the coarse law pulls back with zero
degradation, and the degrading cell's ` $n \cdot \varepsilon$ ` bound specializes to ` $0$ `. -/
theorem strict_cell_exact [DecidableEq  $\beta$ ] { $\pi : \alpha \rightarrow \beta$ } { $K : \alpha \rightarrow \alpha \rightarrow \mathbb{R}$ }
  { $Q : \beta \rightarrow \beta \rightarrow \mathbb{R}$ } (h : PushKernel  $\pi \ K \ Q$ ) ( $\mu_0 : \alpha \rightarrow \mathbb{R}$ ) (n :  $\mathbb{N}$ ) :
  push  $\pi$  (law  $K \ \mu_0 \ n$ ) = law  $Q$  (push  $\pi \ \mu_0$ ) n

```

BernsteinLogicAux.QDistTransport . **pushKernel\_iff\_eps\_zero**

statement bound

```

/-- The exact intertwining is precisely the ` $\varepsilon = 0$ ` witness. -/
theorem pushKernel_iff_eps_zero [DecidableEq  $\beta$ ] { $\pi : \alpha \rightarrow \beta$ }
  { $K : \alpha \rightarrow \alpha \rightarrow \mathbb{R}$ } { $Q : \beta \rightarrow \beta \rightarrow \mathbb{R}$ } :
  PushKernel  $\pi \ K \ Q \leftrightarrow$  EpsPushKernel  $\pi \ K \ Q \ 0$ 

```

Correspondence (Steele display  $\leftrightarrow$  Lean)

**reviewer-certified against the displayed texts** The dictionary below is a reviewer-certified statement of how the mechanized statements above interpret the displayed rule: display metavariable  $\mapsto$  Lean binder, each display hypothesis  $\mapsto$  the premise or side condition carrying it, the display conclusion  $\mapsto$  the goal — with any deliberate difference in width between display and mechanization named outright. Certification is recorded as a hash triple over (the displayed cell, the extracted statement texts, this dictionary's own text); if any of the three changes, this block turns amber until a reviewer re-certifies. The correspondence claim itself remains editorial — reviewer-grade, not machine-checked; its machine-checked successor types each soundness statement against the rule as a structured object, and supersedes this block per rule as it lands.

The display is a matrix and the bound set is its built column, cell by cell. The  $\varepsilon$ -cell's conclusion  $QDist[TV](\text{fine pushed law, coarse law}) \leq n\varepsilon$  at horizon  $n \mapsto \text{sound\_QDistTransport\_eps} : \text{tvDist} (\text{push } \pi (\text{law } K \ \mu_0 \ n)) (\text{law } Q (\text{push } \pi \ \mu_0) \ n) \leq n * \varepsilon$  — the compounding law as a proved induction (one-step defect + TV-nonexpansiveness of the coarse operator + triangle), never a guess.

- the  $\sigma$ -witness rows  $\mapsto$  EpsPushKernel  $\pi \ K \ Q \ \varepsilon$  (every fine row pushes to within TV  $\varepsilon$  of the coarse row at the projected state), with the exact-intertwining column certified as its  $\varepsilon = 0$  face: pushKernel\_iff\_eps\_zero, and the strict cell strict\_cell\_exact re-exporting push\_law — laws transport *exactly* at every horizon, zero degradation.
- the initial row (coarse initial law =  $\pi_{\#}$  fine initial)  $\mapsto$  carried by construction: the coarse arguments in every goal ARE push  $\pi \ \mu_0$  — mismatched initials are unrepresentable in the statement, so no hypothesis row exists.
- the declared horizon  $\mapsto$  the explicit  $n : \mathbb{N}$  binder; the refusal condition for unbounded horizons without a contraction row is spec-side with no Lean counterpart — the honest linear rate IS the certified statement.
- sequential accumulation  $\mapsto$  epsPushKernel\_comp: certified composite budget  $\varepsilon_1 + \varepsilon_2$ , plain addition as the displayed lattice — an upper bound (the actual composite defect may be smaller); the distance-judgment  $\times$  strict-witness cell  $\mapsto$  sound\_QDistSelfTransport (TV data processing at  $L = 1$ ).

- consumed regularity: the  $\varepsilon$ -cell alone demands well-formedness (IsMarkov both kernels, IsDist  $\mu_0$ ) — the mixture-convexity and nonexpansiveness steps use it — while the strict cell and the composition law are hypothesis-free beyond their witnesses: visible in the binder rows.

Width note: finite-kernel tier (Fintype, DecidableEq for fibre sums), as the flavor text confines it; the unbuilt cells ( $\mathbb{W}_p$  Lipschitz pushforward, description subjects, per-cell coherence squares) are the conditional residue, named in the note, bound by nothing — and the judgment-class abstraction of the display's  $J$  ("or a strict spec") is realized concretely: strict specs ride `strict_cell_exact`, distance judgments ride the TV theorems; no single parameterized  $J$  exists in Lean, the matrix's rows are separate theorems.

## Lean verifier receipts

module BernsteinLogicAux.QDistTransport; bound build 4bd1c5103f38... (bound from receipt 6ff1345414960d8f, 2026-07-21T05:38:55Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 6ff1345414960d8f | 2026-07-21T05:38:55Z | build+replay+hardened+decorrelated | current |

13 further receipt(s) in the store attest superseded builds of this module and are omitted from this rendering.

### gate-layer coverage of the bound build:

|                                                           |         |
|-----------------------------------------------------------|---------|
| build (lake build (elaboration))                          | current |
| replay (leanchecker (official kernel replay))             | current |
| hardened (safe_verify (hardened screen))                  | current |
| decorrelated (lean4export + nanoda (decorrelated kernel)) | current |

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### NO HOL STATEMENT YET

The HOL4 development checks the base calculus's wire format (theory bern\_catalogue). The transport matrix's cells conclude judgments of the extension theory, whose rule sorts are out of scope of the current baked theory; the mechanized cells live in the auxiliary Lean package. Nothing exists yet for a HOL statement to be about; the box becomes writable when a revision carries the distance-judgment rows and its checker chain is ported.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: core rule (tier 1). Scope, stated loudly: the finite-kernel tier — finite state spaces, distribution evolution, total-variation entries of the registry. The Lipschitz-pushforward cells for Wasserstein entries, the description-subject cells, and the per-cell composition-coherence squares are named successors, not claimed. Every hypothesis is used and remarked on.

**Setting and claim.** A *fine* chain (states  $\alpha$ , kernel  $K$ , initial law  $\mu_0$ ), a *coarse* chain (states  $\beta$ , kernel  $Q$ ), and a readout map  $\pi : \alpha \rightarrow \beta$ . Write  $\pi_{\#}$  for pushforward,  $\text{law}_K(n)$  for the time- $n$  law, and  $\text{TV}(p, q) = \frac{1}{2} \sum_l |p(l) - q(l)|$ . The witness comes in two grades: *exact intertwining* — every kernel row pushes exactly,  $\pi_{\#}K(s, \cdot) = Q(\pi(s), \cdot)$  — and  $\varepsilon$ -intertwining —  $\text{TV}(\pi_{\#}K(s, \cdot), Q(\pi(s), \cdot)) \leq \varepsilon$  for every  $s$ . The transport matrix's built cells, which are the claims proved below: (**strict**  $\times$  **exact**) laws transport unchanged at every horizon, so any judgment about the coarse law pulls back with zero degradation (`strict_cell_exact`); (**distance**  $\times$  **strict witness**) a total-variation distance judgment moves along any readout map unchanged (`sound_QDistSelfTransport`); (**strict**  $\times$   $\varepsilon$ -**witness** — **the degrading cell**) what transports is a *distance* judgment:  $\text{TV}(\pi_{\#}\text{law}_K(n), \text{law}_Q(n)) \leq n\varepsilon$  at declared horizon  $n$  (`sound_QDistTransport_eps`); and witnesses compose with  $\varepsilon_1 + \varepsilon_2$  (`epsPushKernel_comp`).

**Lemma R93.1** (data processing: pushforward is TV-nonexpansive). *For any map  $\pi$  and any  $p, q$ :  $\text{TV}(\pi_{\#}p, \pi_{\#}q) \leq \text{TV}(p, q)$ .*

*Proof.* Per label  $b$ ,  $|\sum_{\pi(s)=b} (p-q)(s)| \leq \sum_{\pi(s)=b} |p-q|(s)$ ; summing over  $b$  regroups the right side into  $\sum_s |p-q|(s)$ . Merging states can only cancel differences, never create them [160, Ch. 4]. This lemma is the (distance  $\times$  strict-witness) cell: a certified  $\text{TV} \leq \varepsilon$  on the fine side transports to the coarse side unchanged — the registry reading of total variation as a Lipschitz-1 entry under pushforward.  $\square$

**Lemma R93.2** (the coarse operator cannot amplify). *If  $Q$  is row-stochastic with nonnegative entries, then  $\text{TV}(\text{step}_Q p, \text{step}_Q q) \leq \text{TV}(p, q)$ .*

*Proof.*  $|\text{step}_Q(p-q)(t)| \leq \sum_s |p-q|(s) Q(s, t)$  by the triangle inequality and  $Q \geq 0$ ; summing over  $t$  and swapping sums, the inner  $\sum_t Q(s, t) = 1$  leaves  $\sum_s |p-q|(s)$ . Both hypotheses on  $Q$  are consumed: nonnegativity to pull the absolute value through, unit row sums to collapse the mass.  $\square$

**Lemma R93.3** (one-step defect). *Let  $\mu$  be a probability law and let the  $\varepsilon$ -intertwining rows hold. Then  $\text{TV}(\pi_{\#}(\text{step}_K \mu), \text{step}_Q(\pi_{\#} \mu)) \leq \varepsilon$ .*

*Proof.* Both sides are  $\mu$ -mixtures of per-state rows:  $\pi_{\#}(\text{step}_K \mu) = \sum_s \mu(s) \pi_{\#} K(s, \cdot)$  and  $\text{step}_Q(\pi_{\#} \mu) = \sum_s \mu(s) Q(\pi(s), \cdot)$  (the second by regrouping the coarse step over the fibres of  $\pi$ ). Total variation is jointly convex — a mixture of pairs is bounded by the mixture of pairwise distances — so the defect is at most  $\sum_s \mu(s) \cdot \varepsilon = \varepsilon$ , using that  $\mu$  is nonnegative with total mass 1.  $\square$

**Theorem R93.4** (the degrading cell: horizon-compounded transport). *With  $K, Q$  row-stochastic,  $\mu_0$  a probability law, and the  $\varepsilon$ -intertwining rows: for every  $n$ ,*

$$\text{TV}(\pi_{\#} \text{law}_K(n), \text{law}_Q(n)) \leq n\varepsilon,$$

where the coarse chain starts from  $\pi_{\#} \mu_0$ .

*Proof.* Induction on  $n$ ; the base is  $\text{TV}(x, x) = 0$ . Step, by the triangle inequality through the hybrid term  $\text{step}_Q(\pi_{\#} \text{law}_K(n))$ :

$$\begin{aligned} \text{TV}(\pi_{\#} \text{law}_K(n+1), \text{law}_Q(n+1)) &\leq \underbrace{\text{TV}(\pi_{\#} \text{step}_K \text{law}_K(n), \text{step}_Q \pi_{\#} \text{law}_K(n))}_{\leq \varepsilon \text{ (Lemma R93.3, at the time-}n \text{ law)}} + \underbrace{\text{TV}(\text{step}_Q \pi_{\#} \text{law}_K(n), \text{step}_Q \text{law}_Q(n))}_{\leq n\varepsilon \text{ (Lemma R93.2 + induction)}}. \end{aligned}$$

The compounding law is thus *proved*, not guessed: one fresh defect per step, prior error never amplified. A strict judgment does NOT transport along an  $\varepsilon$ -witness; what transports is a distance judgment with  $\varepsilon$  inherited and horizon-compounded — precisely this cell.  $\square$

**Proposition R93.5** (witness composition). *If  $\sigma_1 = (\pi_1, \varepsilon_1)$  intertwines  $K_1$  with  $K_2$  and  $\sigma_2 = (\pi_2, \varepsilon_2)$  intertwines  $K_2$  with  $K_3$ , then  $\pi_2 \circ \pi_1$  is an  $(\varepsilon_1 + \varepsilon_2)$ -witness from  $K_1$  to  $K_3$ .*

*Proof.* At any state  $s$ , triangle through the mid-tier row  $\pi_{2\#} K_2(\pi_1(s), \cdot)$ : the first leg is  $\text{TV}(\pi_{2\#} \pi_{1\#} K_1(s, \cdot), \pi_{2\#} K_2(\pi_1(s), \cdot)) \leq \varepsilon_1$  by Lemma R93.1 applied to  $\sigma_1$ 's row defect; the second leg is  $\sigma_2$ 's row at  $\pi_1(s)$ . Plain addition records the honest upper bound at this tier —  $\varepsilon_1 + \varepsilon_2$  is what the triangle gives, with no minimality claim: data processing can gain slack (a constant second projection has composite defect 0 whatever  $\varepsilon_1$  was).  $\square$

**Proposition R93.6** (the strict column is the  $\varepsilon = 0$  instantiation). *Exact intertwining holds iff the rows are 0-intertwining; under it,  $\pi_{\#} \text{law}_K(n) = \text{law}_Q(n)$  for every  $n$ , and every judgment about the coarse law pulls back with zero degradation.*

*Proof.*  $(\Rightarrow)$  equal rows have distance 0.  $(\Leftarrow)$   $\text{TV}(p, q) \leq 0$  forces  $p = q$  entrywise (a sum of absolute values is zero only if each vanishes). The exact law transport is then the marginal-process induction (the same pushed-row-identity induction as the QCOUPLEDRIFT section's marginal-process lemma, there instantiated at product projections, here at an arbitrary readout map), or equivalently Theorem R93.4 at  $\varepsilon = 0$ .  $\square$

**Remark R93.7** (why each hypothesis is needed). *Row-stochasticity of  $Q$*  (Lemma R93.2): a non-stochastic coarse operator amplifies — with  $Q = 2 \text{Id}$  each step doubles any existing difference and the telescoping collapses; with signed entries the absolute value cannot be pulled through.  *$\mu$  a probability law* (Lemma R93.3): mixture convexity needs nonnegative weights of total mass one — weights summing to 2 double the defect. *The declared horizon*: the linear rate is sharp to first order without further rows — on the two-state fine chain that leaks mass  $\varepsilon$  toward an absorbing label each step, against a static coarse chain, the distance at horizon  $n$  is  $1 - (1 - \varepsilon)^n = n\varepsilon - O(n^2 \varepsilon^2)$ , so no  $o(n\varepsilon)$  bound holds, and over unbounded horizons the error climbs to order one — the  $\varepsilon$ -cell therefore *refuses* unbounded horizons rather than extrapolating; a contraction row is the named mechanism that would buy a uniform-in- $n$  bound, and it is a different certificate. *Per-row quantification* of the intertwining: a defect bound only at states charged by  $\mu_0$  fails at later times, since the law spreads — the rows are quantified over all states precisely so Lemma R93.3 applies at every time- $n$  law.

**Remark R93.8** (relation to the literature). The degrading cell is the kernel-tier, exact-arithmetic face of approximate simulation with quantitative degradation [93]: there, an  $\varepsilon$ -approximate (bi)simulation transports properties up to a metric slack; here the slack is total variation, the systems are finite kernels, and the horizon-compounding law is proved from mixture convexity plus data processing. Behavioural pseudometrics for probabilistic systems [61] are the semantic tradition the distance-judgment registry discretizes; the coupling toolkit is classical [160, 166]. The matrix framing — judgment class  $\times$  witness class, one lemma and one refusal condition per cell — is the rule's own contribution: it makes “what survives transport along a defective witness” a typed question with a proved answer per cell, instead of a convention.

| proof step        | formal anchor                                                    |
|-------------------|------------------------------------------------------------------|
| Lemma R93.1       | tvDist_push_le; cell form sound_QDistSelfTransport               |
| Lemma R93.2       | tvDist_step_le                                                   |
| Lemma R93.3       | EpsPushKernel; push_step_defect                                  |
| Theorem R93.4     | sound_QDistTransport_eps (triangle: tvDist_triangle)             |
| Proposition R93.5 | epsPushKernel_comp; push_comp                                    |
| Proposition R93.6 | pushKernel_iff_eps_zero; eq_of_tvDist_le_zero; strict_cell_exact |



## 15.12 Stored-Chain Certificate Species

### Rule 94. DTMCABSORBBRACKET (core rule)

#### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *DtmcAbsorbBracket*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{q : \text{DAbsQuery} \quad \text{qtsl\_dabs\_wf}(q) \quad \text{stored rows of } q.\text{chain} \text{ (dprob, dinit, dclass, dreject)}}{\text{qtsl\_dabs\_absorb\_bracket}(q, c, lo, hi)}$$

DTMCABSORBBRACKET is a certificate species rather than a proof-node rule: a checker-recomputed claim over the stored rows of a finite chain, introducing zero new judgment sorts and zero new specification formers. It certifies guaranteed two-sided brackets on the eventual-absorption measure of one class of a finite stored-row chain — and, through its normalized sibling DTMCPOSTERIORBRACKET, on normalized conditional measures (posteriors of discrete probabilistic programs, where the compilation of a program to a chain is an untrusted translation outside the trust path). The species is stated at full generality: any finite chain, any class labeling, any horizon.

The evaluation semantics, once:  $m_0$  is the chain's stored initial measure;  $m_{t+1} = m_t K$ , where  $K$  is the stored transition-row matrix COMPLETED by a self-loop at every state with no stored outgoing row (*rowless-is-absorbing*). For a query with horizon  $T$ , class labeling, and rejection labeling:  $N_c$  is the class- $c$  mass of  $m_T$ ;  $S = \sum_c N_c$ ;  $R = 1 - S$  (rejected mass of  $m_T$ ); and  $R_c$  is the *reachability-sharpened* per-class residual — the part of  $R$  sitting at states from which some class- $c$  state is reachable through strictly positive stored rows (a backward walk seeded at the class, run under the same work budget; budget exhaustion during the walk refuses the whole claim, never verdicts). The claim holds iff  $lo \leq N_c$  and  $N_c + R_c \leq hi$ . Mass that can never arrive in the class is never charged to it: a class unreachable from every alive state has  $R_c = 0$  and a width-zero bracket at any  $T \geq 1$ , with no termination assumption; and  $R_c \leq R$  always, so any claim admissible under the uniform-residual reading stays admissible.

The completion convention is the species' load-bearing design move: a truncated row set can only WIDEN brackets (unclassified mass accrues to the residual, never to any class), so row-set completeness is a tightness concern rather than a soundness obligation — there is no closure side condition, deliberately, and under-expansion is unrepresentable as an attack. Derived quantities ( $N_c$ ,  $S$ ,  $R$ ,  $R_c$ , ratio denominators) are never stored: the evaluator recomputes them from the chain rows, so forging an intermediate is unrepresentable; claims may be looser than canonical, never tighter.

Soundness has two sides. Below, absorption is monotone: classed states are demanded stored-absorbing (the well-formedness side condition), so class mass never decreases past the horizon. Above, the class's co-reachable set  $B_C$  (states with a strictly-positive-support path into the class;  $C \subseteq B_C$ ) has a forward-closed complement — no positive-support row leads from outside  $B_C$  into it — so the mass of  $B_C$  is nonincreasing and later class gain is bounded by  $\text{mass}_T(B_C \setminus C) = R_c$ . The uniform-residual bound ( $R_c$  relaxed to  $1 - \text{mass}_T(\text{Abs})$ ) is the pre-sharpening corollary. The argument is mechanized in the accompanying Lean development (SuFunctor/Adequacy/DtmcAbsorb.lean; sorry-free, standard axiom closure, axiom-audited on every build): the sharpened upper side  $\text{dtmcAbsorb\_mass\_le\_reach}$  is notably HYPOTHESIS-FREE on the class — it holds for any class of any finite stochastic chain, the well-formedness demands protecting only the monotone lower side; beside it sit  $\text{dtmcAbsorbBracket\_reach\_sound}$  (the bracket, both sides),  $\text{dtmcAbsorb\_reach\_residual\_le}$  (the never-loosens warrant  $R_c \leq 1 - \text{mass}_T(\text{Abs})$ , itself needing only that absorbing states absorb),  $\text{dtmcAbsorb\_eventual\_bracket\_reach}$  (the claim shape), the forward-closed generalization ( $\text{FinDTMC.ForwardClosed}$ ,  $\text{dtmc\_mass\_mono\_of\_forwardClosed}$ ) from which the incumbent monotonicity lemmas re-derive in one line, and a kernel-checked strict-improvement witness ( $\text{idBool\_strict\_improvement}$ : an unreachable class where the uniform-residual bracket has width one and the sharpened bracket width zero — a standing regression tooth, re-checked every build). The mechanized statements are deliberately universally quantified ( $\forall t' \geq T$ ) rather than supremum-shaped: over  $\mathbb{Q}$  the supremum of a monotone bounded mass sequence need not exist as a rational, and the universal form is the citable content — any eventual-absorption reading (sup or limit, in any completion of  $\mathbb{Q}$ ) inherits the claimed bracket.

One clause of the species is graded prose by design: the *truncation-transfer* reading — stored rows agreeing with an external program's chain on rowed states implies the bracket transfers to the program — is a coupling argument about a translation that lives OUTSIDE the trust path, and its assurance instrument is a battery (vendored source, Monte-Carlo differential, closed-form containment), not a kernel theorem. The certificate itself never claims more than the stored rows warrant.

**notes** The schema is baked in the certificate theory as a species sibling extending the chain grain, purely additive — the query sort with anchor-mediated chain binding (the query's chain IS its anchor's chain, labels bound by chase rows), the horizon field, the class and rejection labelings, and the three side-condition Props with their report, refusal, and chase rows, all quoted below from the theory source. The doubly-classed state is unrepresentable at the attribute grain (the class labeling is a keyed function row). The mechanized soundness lemmas are named in the commentary; no Lean statement is bound into this book yet — the binding column is honest open debt, and joins when the statement-binding machinery registers the development's package.

## Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: chase = the engine derives; report = a violation is a loud judgment; refuse = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The chain grain this species reads: states, stored transition rows, stored initial rows, and the row-mass admission demand. The species adds no rows here; it consumes them.

theory Dtmc . **DState**

quoted from the theory source at render

```
| DState : Set;    // one state of one chain
```

theory Dtmc . **dprob**

quoted from the theory source at render

```
| dprob : {src: DState, dst: DState} -> Q;
```

theory Dtmc . **dinit**

quoted from the theory source at render

```
| dinit : {chain: Dtmc, state: DState} -> Q;
```

theory Dtmc . **qtsl\_dtmc\_rowmass**

quoted from the theory source at render

```
| qtsl_dtmc_rowmass : {chain: Dtmc} -> Prop;    // side-condition
```

The baked rows of the species: the query sort with its anchor and horizon, the class and rejection labelings, the disjointness refusal, the labels-bind-own-chain chase rows, the well-formedness demand, and the two bracket claims with their interval-order reports.

theory DtmcAbsorb . **DAbsQuery**

quoted from the theory source at render

```
| DAbsQuery : Set;
```

theory DtmcAbsorb . **DAbsQuery/\***

quoted from the theory source at render

```
| DAbsQuery/anchor : DAbsQuery -> DState;
| DAbsQuery/horizon : DAbsQuery -> Nat;
```

theory DtmcAbsorb . **dclass**

quoted from the theory source at render

```
| dclass : {q: DAbsQuery, state: DState} -> Nat;
```

theory DtmcAbsorb . **dreject**

quoted from the theory source at render

```
| dreject : {q: DAbsQuery, state: DState} -> Prop;
```

theory DtmcAbsorb . **ax/wf/dabs\_class\_reject\_disjoint**

quoted from the theory source at render

```
| refuse ax/wf/dabs_class_reject_disjoint : forall q : DAbsQuery,
  s : DState, c : Nat.
  c = {q: q, state: s} dclass,
  {q: q, state: s} dreject
|- false;
```

theory DtmcAbsorb . **ax/wf/dabs\_class\_same\_dtmc**

quoted from the theory source at render

```
| chase ax/wf/dabs_class_same_dtmc : forall q : DAbsQuery, s : DState,
  c : Nat.
  c = {q: q, state: s} dclass
|- s DState/dtmc = q DAbsQuery/anchor DState/dtmc;
```

theory DtmcAbsorb . **ax/wf/dabs\_reject\_same\_dtmc**

quoted from the theory source at render

```
| chase ax/wf/dabs_reject_same_dtmc : forall q : DAbsQuery, s : DState.
  {q: q, state: s} dreject
|- s DState/dtmc = q DAbsQuery/anchor DState/dtmc;
```

theory DtmcAbsorb . qtsl\_dabs\_wf

quoted from the theory source at render

| qtsl\_dabs\_wf : {q: DAbsQuery} -> Prop; // side-condition

theory DtmcAbsorb . ax/wf/dabs\_wf

quoted from the theory source at render

| report ax/wf/dabs\_wf : forall q : DAbsQuery.  
|- {q: q} qtsl\_dabs\_wf;

theory DtmcAbsorb . qtsl\_dabs\_absorb\_bracket

quoted from the theory source at render

| qtsl\_dabs\_absorb\_bracket : {q: DAbsQuery, class: Nat, lo: Q, hi: Q}  
-> Prop; // side-condition

theory DtmcAbsorb . ax/wf/dabs\_absorb\_order

quoted from the theory source at render

| report ax/wf/dabs\_absorb\_order : forall q : DAbsQuery, c : Nat,  
l : Q, h : Q.  
{q: q, class: c, lo: l, hi: h} qtsl\_dabs\_absorb\_bracket  
|- {l: l, r: h} q\_le;

## Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

## HOL4 statement

### NO HOL STATEMENT YET

No HOL development of the extension theory exists; this species' content is an exact-rational evaluator recompute over baked rows, and its soundness theorems live in the Lean development named in the commentary.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: certificate species (tier 1 semantic soundness; checker recomputation, no proof node). The proof below is the mechanized argument restated at textbook detail; the named Lean declarations carry it, sorry-free at standard axiom closure.

**Setting.** A finite stored-row chain: state set  $X$ , stored transition rows  $K_0 \subseteq X \times X \rightarrow \mathbb{Q}_{\geq 0}$  satisfying the row-mass admission demand (each rowed source's stored masses total exactly one), stored initial measure  $m_0$ , horizon  $T \in \mathbb{N}$ , class labeling and rejection labeling over  $X$ . Let  $K$  be  $K_0$  completed by a unit self-loop at every state with no stored outgoing row (*rowless-is-absorbing*);  $K$  is then a stochastic matrix on  $X$ , and  $m_{t+1} := m_t K$  defines the exact-rational evolution. Write  $C$  for the set of class- $c$  states, Abs for the labeled (classed or rejected) states — each demanded stored-absorbing by the well-formedness side condition: rowless, or exactly the unit self-loop — and  $N_c := \text{mass}_T(C)$ ,  $S := \sum_{c'} N_{c'}$ ,  $R := 1 - S$  — (rejected mass of  $m_T$ ).

**Lemma 1 (monotone absorption — the lower side).** For  $t' \geq T$ :  $\text{mass}_{t'}(C) \geq \text{mass}_T(C)$ . *Proof.* Each state of  $C$  is stored-absorbing, so under  $K$  its mass never leaves; class mass is nondecreasing step by step. (Removing the well-formedness hypothesis admits the counterexample the mutant battery pins: a classed state with a stored mass-leaking outgoing row loses class mass at the very next step.)  $\square$

**Lemma 2 (forward-closed complement — the upper side).** Let  $B_C$  be the co-reachable set of  $C$ : states admitting a path into  $C$  through strictly positive stored rows ( $C \subseteq B_C$ ). Then  $X \setminus B_C$  is forward-closed — no positive-support row of  $K$  leads from outside  $B_C$  into  $B_C$  (such a row would itself witness co-reachability of its source) — hence  $\text{mass}_{t'}(B_C)$  is nonincreasing in  $t'$ . *Proof.* For a forward-closed set  $F$ ,  $1_F K^\top \geq 1_F$  pointwise on  $F$ 's complement contributes nothing, and mass on  $F$  can only stay or grow; apply to  $F = X \setminus B_C$  and complement. This is the mechanized generalization (FinDTMC.ForwardClosed, dtmc\_mass\_mono\_of\_forwardClosed), from which Lemma 1 re-derives in one line: a stored-absorbing set is forward-closed around each of its points.  $\square$

**Theorem (the bracket).** Define  $R_c := \text{mass}_T(B_C \setminus C)$  (the reachability-sharpened per-class residual, recomputed by a backward walk over strictly positive stored rows, seeded at  $C$ , under the same work budget — exhaustion refuses the whole claim). For every  $t' \geq T$ :

$$\text{mass}_{t'}(C) \in [\text{mass}_T(C), \text{mass}_T(C) + R_c].$$

*Proof.* Lower: Lemma 1. Upper: class mass at  $t'$  sits inside  $B_C$  (mass outside  $B_C$  can never arrive in  $C$ , by Lemma 2 applied along the trajectory), and  $C$ 's gain since  $T$  is at most  $B_C$ 's non- $C$  mass at  $T$ , which is  $R_c$ , since  $B_C$ 's total is nonincreasing while its  $C$ -part is nondecreasing.  $\square$

The mechanized upper side (`dtmcAbsorb_mass_le_reach`) is HYPOTHESIS-FREE on the class: it holds for any class of any finite stochastic chain — the well-formedness demands protect only Lemma 1. The never-loosens warrant (`dtmcAbsorb_reach_residual_le`) gives  $R_c \leq 1 - \text{mass}_T(\text{Abs})$ , needing only that labeled states absorb, so every claim admissible under the uniform residual remains admissible under the sharpened one; and the strict-improvement tooth (`idBool_strict_improvement`) exhibits an unreachable class where the uniform bracket has width one and the sharpened bracket width zero — a standing kernel-checked regression witness. A class unreachable from every alive state has  $R_c = 0$ : a width-zero bracket at any  $T \geq 1$ , with no termination assumption anywhere.

**Corollary (eventual readings).** The statements are universally quantified over  $t' \geq T$  deliberately: over  $\mathbb{Q}$  the supremum of the monotone bounded sequence  $\text{mass}_{t'}(C)$  need not exist as a rational. Any eventual-absorption reading — supremum or limit, in any completion of  $\mathbb{Q}$  — inherits the claimed bracket from the universal form (`dtmcAbsorb_eventual_bracket_reach`).

**Why each hypothesis.** Row-mass admission: without it, doubled stored masses inflate  $N_c$  (a battery locus). The well-formedness demand: Lemma 1's counterexample above. The budget refusal on the backward walk: a partial walk could under-build  $B_C$ , dropping states whose mass belongs in  $R_c$  — so exhaustion refuses rather than verdicts. Interval order  $lo \leq hi$  is a report row; inversion refuses. And the completion convention (rowless-is-absorbing) makes truncation *widen* brackets: omitted rows leave their states absorbing and their mass in the residual, never in a class — there is no closure side condition to attack, and claims may be looser than canonical, never tighter.

*Remark R94.1 (status).* The schema is baked (species rows quoted in this section); the argument above is mechanized sorry-free at standard axiom closure, axiom-audited on every build, in the module named in the commentary. No Lean statement is bound into this book yet: the binding column is honest open debt until the statement-binding machinery registers the development's package.

## Rule 95. DTMCPOSTERIORBRACKET (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: QTSL-rules.toml — *DtmcPosteriorBracket*. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\frac{q : \text{DAbsQuery} \quad \text{qtsl\_dabs\_wf}(q) \quad \text{stored rows of } q.\text{chain}(\text{dprob}, \text{dinit}, \text{dclass}, \text{dreject}) \quad S > 0 \text{ (recomputed; refusal otherwise)}}{\text{qtsl\_dabs\_posterior\_bracket}(q, c, lo, hi)}$$

DTMCPOSTERIORBRACKET is the normalized face of DTMCABSORB: a bracket on the *posterior* eventual-absorption measure of one class, with normalizer the total classed mass. Hard conditioning is a CLASS, not a score: rejected mass (failed conditioning) and unlabeled or nonterminating mass are excluded from the normalizer by the same labeling mechanism, so every stored row remains a probability row and the chain grain's row-mass admission demand is untouched. This is what lets posteriors of discrete probabilistic programs ride the species without any weighted-row machinery.

The claim holds iff  $lo \leq N_c/(S+R)$  and  $(N_c+R)/(S+R) \leq hi$ , with  $N_c, S, R$  recomputed by the same  $T$ -step exact-rational evaluation as the sibling; it REFUSES at  $S = 0$  (no positive-evidence witness). The mathematical content beyond the sibling's conservation bracket is the correlated-ratio lemma — pure algebra, no chain in sight: from  $N_c \leq M_c$  classwise,  $\sum_c (M_c - N_c) \leq R$ , and  $S > 0$ , the true posterior lies in  $[N_c/(S+R), (N_c+R)/(S+R)]$ . The correlation is the point: numerator and denominator draw their errors from ONE budget, so both endpoints use the single normalizer  $S+R$  rather than the looser interval quotient — never worse, often sharper (measured  $3.0\times$  tighter than the interval-quotient baseline at matched truncation on the species' first worked example). Mechanized as *dtmcPosteriorBracket\_sound*, over any finite class alphabet, in the same module and with the same receipts as the sibling.

An honesty note carried from the design: the bracket width is exactly  $R/(S+R)$ , class-uniform — the tightness of the whole judgment is one number, which is what keeps comparison-table claim shapes honest. The posterior face deliberately keeps the UNIFORM residual  $R$  even though the unnormalized sibling mints with the reachability-sharpened  $R_c$ : the single-normalizer argument requires one shared budget across all classes, and the class-uniform width claim stays honest only under it. An asymmetric refinement (uniform lower endpoint, class-specific upper endpoint  $(N_c+R_c)/(S+R_c)$ ) is sound and is deliberately NOT minted — it boards a future revision of the theory if and when a use case demands it.

**notes** The schema is baked in the certificate theory alongside the sibling: same query sort, labelings, and well-formedness demand; its own claim relation with its own interval-order report; the  $S > 0$  refusal is evaluator-side (a recompute refusal, not a stored row). Quoted below from the theory source. No Lean statement is bound into this book yet — the binding column is honest open debt, as the sibling.

### Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: *chase* = the engine derives; *report* = a violation is a loud judgment; *refuse* = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The baked rows beyond the sibling's: the posterior claim relation and its interval-order report. The query sort, labelings, and well-formedness demand are the sibling's own rows, quoted in its section.

theory DtmcAbsorb . **qtsl\_dabs\_posterior\_bracket**

quoted from the theory source at render

```
qtsl_dabs_posterior_bracket : {q: DAbsQuery, class: Nat, lo: Q,
                             hi: Q} -> Prop; // side-condition
```

theory DtmcAbsorb . **ax/wf/dabs\_posterior\_order**

quoted from the theory source at render

```
report ax/wf/dabs_posterior_order : forall q : DAbsQuery, c : Nat,
  l : Q, h : Q.
  {q: q, class: c, lo: l, hi: h} qtsl_dabs_posterior_bracket
|- {l: l, r: h} q_le;
```

### Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

## HOL4 statement

### NO HOL STATEMENT YET

No HOL development of the extension theory exists; this species' content is an exact-rational evaluator recompute over baked rows plus one algebraic lemma, mechanized in the Lean development named in the commentary.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: certificate species (tier 1 semantic soundness; checker recomputation, no proof node). The content beyond the unnormalized sibling is one algebraic lemma, proved here in full; the mechanized form is `dtmcPosteriorBracket_sound`, over any finite class alphabet.

**Setting.** As in `DTMCABSORBBRACKET`: from the  $T$ -step evaluation, per-class truncation masses  $N_c \geq 0$ , their total  $S = \sum_c N_c$ , and the residual  $R \geq 0$ . Write  $M_c$  for the true eventual class- $c$  mass (any completion-valued reading). The sibling's conservation bracket gives, classwise,  $N_c \leq M_c$  and  $\sum_c (M_c - N_c) \leq R$ .

**Lemma (correlated ratio).** If additionally  $S > 0$ , then for every class  $c$ :

$$\frac{M_c}{\sum_{c'} M_{c'}} \in \left[ \frac{N_c}{S+R}, \frac{N_c+R}{S+R} \right].$$

*Proof.* Let  $\delta_{c'} := M_{c'} - N_{c'} \geq 0$  and  $\Delta := \sum_{c'} \delta_{c'} \leq R$ . The denominator is  $\sum_{c'} M_{c'} = S + \Delta \in [S, S+R]$ , and  $S > 0$  makes every quotient well-defined. Upper endpoint:

$$\frac{M_c}{S+\Delta} = \frac{N_c + \delta_c}{S+\Delta} \leq \frac{N_c + \Delta}{S+\Delta} \leq \frac{N_c + R}{S+R},$$

the first step since  $\delta_c \leq \Delta$ , the second since  $x \mapsto (N_c + x)/(S + x)$  is nondecreasing for  $x \geq 0$  (as  $N_c \leq S$ ) and  $\Delta \leq R$ . Lower endpoint:

$$\frac{M_c}{S+\Delta} \geq \frac{N_c}{S+\Delta} \geq \frac{N_c}{S+R}.$$

□

The correlation is the content: numerator and denominator draw their errors from the ONE budget  $\Delta \leq R$ , so both endpoints use the single normalizer  $S + R$ . The uncorrelated alternative — bracket numerator and denominator separately and take the interval quotient  $[N_c/(S+R), (N_c+R)/S]$  — is never better and usually worse; at matched truncation on the species' first worked example the correlated form measured 3.0× tighter.

**Why each hypothesis.**  $S > 0$ : at  $S = 0$  the posterior is 0/0-shaped and the species REFUSES (no positive-evidence witness) — a refusal, never a verdict. Nonnegativity of the  $\delta_{c'}$  is exactly the sibling's monotone lower side; the one-budget bound  $\Delta \leq R$  is its conservation upper side, summed. Hard conditioning enters as a CLASS (rejected mass leaves the normalizer by labeling), so every stored row remains a probability row and no weighted-row machinery is consumed.

**Width honesty.** Both endpoints differ by exactly  $R/(S+R)$ , class-uniformly: the tightness of the whole judgment is one number. The posterior face deliberately keeps the uniform  $R$  (the one-budget argument above requires a single normalizer shared by all classes); the asymmetric per-class refinement with upper endpoint  $(N_c + R_c)/(S + R_c)$  is sound and deliberately not part of this revision — it boards a future one if a use case demands it.

*Remark R95.1 (status).* The schema is baked (claim relation and interval-order report quoted in this section; query sort and labelings in the sibling's). The lemma above is mechanized sorry-free at standard axiom closure over any finite class alphabet. No Lean statement is bound into this book yet: the binding column is honest open debt, as the sibling's.



## Rule 96. QGENFORMCONTRACTION (core rule)

### The rule (Steele metanotation)

as realized in the mechanized development; source row: `QTSL-rules.toml` — `QGenFormContraction`. The metanotation statement is typeset by hand for notational quality; this rule cites no Lean statements yet (the absence is stated loudly below). The rule *roster* is machine-checked (every mechanized rule tag round-trips to exactly one catalogue row); the statement *text* is editorial.

$$\begin{array}{c}
 w : \text{DGenContraction} \\
 \text{stored rows: } \text{dweight } (A = \text{supp } V, \text{ nonempty}), \text{ dprob, dinit} \quad \sum_{y \in A} K(x, y) V(y) \leq c V(x) \quad \forall x \in A \\
 K(x, y) = 0 \text{ for stored rows } x \notin A, y \in A \quad \min_A V > 0 \quad \text{qtsl\_dtmc\_genform\_contraction}(w) \\
 \hline
 \text{each claimed row of } w : \text{qtsl\_dgc\_partial\_lo, qtsl\_dgc\_ev\_bracket, qtsl\_dgc\_survival\_tail, qtsl\_dgc\_z\_one}
 \end{array}$$

QGENFORMCONTRACTION strengthens the chain-content generator-form seam from the incumbent's boolean claim — transfer of the certificate is nonpositive-drift — to a score-carrying, regional, *quantitative* contraction whose conclusions include the incumbent's and add geometric tails, first-moment tails, and a termination-mass-one claim. It is a certificate species, not a proof-node rule: checker recomputation over stored rows, zero new judgment sorts, zero new formers. The mechanized carrier is an arbitrary nonnegative weight matrix (the score factorization  $s(x)K(x, y)$  is presentation, not mathematics), so stochastic, sub-probability, and soft-score row sets ride ONE induction.

This is the fold-don't-sibling reading of the change discipline enacted: the would-be separate weighted-tail species folds into the incumbent seam, and the incumbent survives as the degenerate face (score  $\equiv 1$ ,  $c = 1$ , region = all states) — its rows, wire shape, and consuming lemmas untouched. Subsumption is mechanized twice over, as the strengthening discipline demands: the instance chain (the weighted core at  $s \equiv 1$  on a stochastic kernel re-derives the previously proven stochastic core's exact statement: `massAt_expect_le_of_contraction_of_weighted`) and the composed theorem (`qGenFormContraction_subsumes_genform`: from the incumbent's recomputed nonpositive generator form, the strengthened engine derives the incumbent consumption's exact path-law conclusion, never by citing the incumbent's own proof) — so every certificate admitted under the incumbent discharges under the strengthening with an unchanged conclusion.

The evaluator recomputes, per stored row: the region-restricted weighted contraction  $\sum_{y \in A} K(x, y)V(y) \leq c V(x)$  on the region  $A$ ; no re-entry ( $K(x, y) = 0$  for stored rows from outside  $A$  into  $A$  — equivalently, the complement is closed); and the strictly positive region minimum  $\mu = \min_A V > 0$ . Conclusions, each bound exactly where its claim row is spent: weighted alive-mass decay  $\langle m_t | A, V \rangle \leq c^t \langle m_0 | A, V \rangle$ ; for  $c < 1$ , window exit-weight truncation and the time-weighted first-moment tail (the expected-value rows' engine); the exact partial survival sum (the lower claim); and — through the DERIVED conversion  $1/\mu$  rescaling the mechanized unit-floor lemma by scale-invariance — the bare alive-mass tail  $\text{mass}_t(A) \leq c^t \langle m_0 | A, V \rangle / \mu$ , the stored-row form of a hitting-time tail  $\Pr[\text{hitting time} > t] \leq M c^t$  (a self-stabilization time's region is the not-yet-stabilized configurations, whose complement's closedness is exactly the no-re-entry recompute).

The enacted schema's deviations from the drafted proposal are each load-bearing. There is NO stored exit or conversion constant: the constant is derived ( $1/\min_A V$ , with the minimum demanded strictly positive) because a stored constant would be a forgeable derived quantity — the species' own never-store-what-you-can-recompute discipline. The region IS the weight relation's support (a stored weight row is the stored membership; no separate region relation to fall out of sync). Four NAMED claim relations replace a flavor datum — partial lower bound, expected-exit-time bracket, survival tail, and termination-mass-one — making the wire self-documenting and letting each side condition bind exactly where its conclusion is spent (the  $c < 1$  demand binds exactly at tail-claiming rows; at  $c = 1$  the incumbent decay-only conclusion remains). A PRESENT score field REFUSES at this revision (deferred-safe-fail): the mechanized core is already score-carrying, and the evaluator face for present scores boards its own future revision rather than shipping unexercised. Higher moments ( $k \geq 2$ ) are deliberately absent — the same per-term reduction with  $\sum t^k c^t$  closed forms, filed to the first leg that needs them.

The soundness engine is the weighted-transfer induction (one-step transfer, induction on  $t$ , geometric series — no stochastic analysis anywhere), mechanized sorry-free at standard axiom closure and axiom-audited on every build (`SuFunctor/Adequacy/DtmcContraction.lean`): `wmassAt` with the instance tie `massAt_eq_wmassAt`; the core `wmassAt_expect_le_of_contraction` (certificate nonnegativity DROPPED — strictly stronger than the drafted specification); the score face; the regional packaging via the region-supported observable; exit domination; the exact arithmetico-geometric identity and series tails; and the two tail assemblies, in universally-quantified partial-sum form (over  $\mathbb{Q}$  the limit need not exist; any completion-valued reading inherits the bound). The alive-mass face lives on its own module (`SuFunctor/Adequacy/DtmcAliveTail.lean`): `massAt_expect_on_le_of_regionContraction` and `dtmcAliveTail_of_regionContraction`, whose two added hypotheses — the floor and the closed complement — are each witnessed load-bearing by kernel-checked countermodels COMMITTED beside the lemmas (`aliveTail_needs_floor`, `aliveTail_needs_no_reentry`), re-checked on every build. The species' mutant battery was run as committed throwaway edits, each red at its predicted load-bearing proof step.

Two further faces are deliberately NOT part of this revision. The TEMPLATE face (geometric certificate families  $V(n) = M\gamma^n$  over offset families, collapsing region instances to single rational inequalities) and the PATTERN face (finitely-presented infinite chains and product-profile templates) are deferred to their own design work: the pattern face is underdetermined at this grain (no canonical state-coordinate binding; a pattern's generated kernel and the stored completion's rowless-is-absorbing



reading are two different kernels and must not share one chain reading; and the finite-state induction does not by itself warrant infinite-sum exchange — a carrier theorem is owed), and no admitted certificate demands the one-dimensional template at the stored-weights grain, where the finite face stores every weight. Their collapse warrants are already mechanized and banked for that future work (`genFormTemplate_collapse` for one-dimensional offset families; `aliveStep_factor`, `transferRatio_of_ge`, `productProfile_collapse` and its window form, `SuFunctor/Adequacy/DtmcProductProfile.lean`, proven at full profile generality), together with a kernel-checked necessity witness (`rampProfile_needs_window_reach`) showing the class-check family alone cannot cover ramp-resident exterior states — so the face's eventual schema must carry a profile-length demand beside window coverage. What remains for that face is the infinite-carrier bridge, the kernel-object tie, and the stored/template composition glue; until then, template- and pattern-shaped claims live outside the certified lane, labeled as such.

**notes** The FINITE FACE is baked in the certificate theory as a species sibling extending the chain grain, purely additive: the witness entity (anchor-mediated chain binding, absolute horizon, the deterministic-prologue offset datum, the contraction constant, the optional score reference whose presence refuses at this revision), the keyed weight relation whose support is the region (nonempty demanded), the engine-decided demand minted per witness by a report row, and the four named claim relations with their range and interval-order reports — all quoted below from the theory source. The incumbent generator-form rows are untouched (additive delta only), and the sibling species' deliberately-absent notes carry their pointer flips from the same revision. The mechanized soundness lemmas are named in the commentary; no Lean statement is bound into this book yet — the binding column is honest open debt.

## Theory rows (geolog)

The rule's certificate content as it is declared in the geolog theory the checker loads — payload schema, side-condition demands, and axiom rows, quoted from the theory source at every build of this catalogue (a stale quotation cannot exist). Axiom modes: `chase` = the engine derives; `report` = a violation is a loud judgment; `refuse` = a violation gates admission. Side-condition relations are discharged by bounded evaluation at the engine's ground seam, never by derivation.

The incumbent seam this species strengthens — and survives as the degenerate face of: the chain-content generator-form demand.

theory Dtmc . **qtsl\_dtmc\_genform**

quoted from the theory source at render

```
qtsl_dtmc_genform : {sys: System, cert: HypernetRef,
                    gen: HypernetRef} -> Prop;
```

The baked rows of the finite face: the witness entity, the weight relation with its same-chain chase row, the engine-decided demand, and the four named claim relations with their wf reports (constant range, interval order on expected-value brackets, tail-coefficient ranges, and the strict-contraction demand exactly where an expected-value bracket is claimed).

theory DtmcContraction . **DGenContraction**

quoted from the theory source at render

```
DGenContraction : Set;
```

theory DtmcContraction . **DGenContraction/\***

quoted from the theory source at render

```
DGenContraction/anchor : DGenContraction -> DState;
DGenContraction/horizon : DGenContraction -> Nat;
DGenContraction/offset : DGenContraction -> Nat;
DGenContraction/c : DGenContraction -> Q;
DGenContraction/score : DGenContraction -> HypernetRef;
```

theory DtmcContraction . **dweight**

quoted from the theory source at render

```
dweight : {cw: DGenContraction, state: DState} -> Q;
```

theory DtmcContraction . **ax/wf/dgc\_weight\_same\_dtmc**

quoted from the theory source at render

```
chase ax/wf/dgc_weight_same_dtmc : forall w : DGenContraction,
  s : DState, v : Q.
  v = {cw: w, state: s} dweight
|- s DState/dtmc = w DGenContraction/anchor DState/dtmc;
```

theory DtmcContraction . **qtsl\_dtmc\_genform\_contraction**

quoted from the theory source at render

```
qtsl_dtmc_genform_contraction : {cw: DGenContraction} -> Prop;
```

theory DtmcContraction . **ax/wf/dgc\_demand**

quoted from the theory source at render

```
report ax/wf/dgc_demand : forall w : DGenContraction.
|- {cw: w} qtsl_dtmc_genform_contraction;
```

theory DtmcContraction . **qtsl\_dgc\_partial\_lo**

quoted from the theory source at render

| qtsl\_dgc\_partial\_lo : {cw: DGenContraction, lo: Q} -> Prop;

theory DtmcContraction . **qtsl\_dgc\_ev\_bracket**

quoted from the theory source at render

| qtsl\_dgc\_ev\_bracket : {cw: DGenContraction, lo: Q, hi: Q} -> Prop;

theory DtmcContraction . **qtsl\_dgc\_survival\_tail**

quoted from the theory source at render

| qtsl\_dgc\_survival\_tail : {cw: DGenContraction, coeff: Q, rate: Q}  
-> Prop; // side-condition (R21)

theory DtmcContraction . **qtsl\_dgc\_z\_one**

quoted from the theory source at render

| qtsl\_dgc\_z\_one : {cw: DGenContraction} -> Prop;

theory DtmcContraction . **ax/wf/dgc\_c\_range**

quoted from the theory source at render

| report ax/wf/dgc\_c\_range : forall w : DGenContraction, c : Q.  
c = w DGenContraction/c  
|- {l: 0, r: c} q\_le, {l: c, r: 1} q\_le;

theory DtmcContraction . **ax/wf/dgc\_ev\_order**

quoted from the theory source at render

| report ax/wf/dgc\_ev\_order : forall w : DGenContraction, l : Q, h : Q.  
{cw: w, lo: l, hi: h} qtsl\_dgc\_ev\_bracket  
|- {l: l, r: h} q\_le;

theory DtmcContraction . **ax/wf/dgc\_tail\_range**

quoted from the theory source at render

| report ax/wf/dgc\_tail\_range : forall w : DGenContraction, b : Q,  
r : Q.  
{cw: w, coeff: b, rate: r} qtsl\_dgc\_survival\_tail  
|- {l: 0, r: b} q\_le, {l: 0, r: r} q\_le, {l: r, r: 1} q\_lt;

theory DtmcContraction . **ax/wf/dgc\_ev\_strict**

quoted from the theory source at render

| report ax/wf/dgc\_ev\_strict : forall w : DGenContraction, l : Q,  
h : Q, c : Q.  
{cw: w, lo: l, hi: h} qtsl\_dgc\_ev\_bracket,  
c = w DGenContraction/c  
|- {l: c, r: 1} q\_lt;

## Lean statement

**not yet mechanized** this rule cites no Lean statements: nothing on this page is hash-bound for it, and its dashboard row shows the same absence across the statement, receipt, and correspondence columns.

## HOL4 statement

### NO HOL STATEMENT YET

No HOL development of the extension theory exists; this species' content is an exact-rational evaluator recompute over baked rows, and its soundness theorems live in the Lean development named in the commentary.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: certificate species (tier 1 semantic soundness; checker recomputation strengthening an existing seam). The proof is the weighted-transfer induction plus exact series arithmetic — no stochastic analysis anywhere; the mechanized lemma roster is named inline.

**Setting.** A finite stored-row chain with completed kernel  $K$  (rowless-is-absorbing, as in the sibling species); a stored weight function  $V$  on the region  $A := \text{supp } V$  (the region IS the support of the stored weight rows — nonempty demanded); a contraction constant  $c \in [0, 1]$ ; and the recomputed side conditions: regional contraction  $\sum_{y \in A} K(x, y) V(y) \leq c V(x)$  for every  $x \in A$ ; no re-entry,  $K(x, y) = 0$  for stored rows with  $x \notin A, y \in A$  (equivalently, the complement of  $A$  is closed); and strictly positive region minimum  $\mu := \min_A V > 0$ . The carrier of the mechanized core is an arbitrary nonnegative weight matrix — the score factorization  $s(x)K(x, y)$  is presentation, not mathematics — so stochastic, sub-probability, and soft-score row sets ride one induction; certificate nonnegativity is NOT assumed by the core (`wmassAt_expect_le_of_contraction`), which is strictly stronger than the drafted specification.

**Lemma 1 (one-step transfer).** With  $w_t := \langle m_t|_A, V \rangle$  the region-supported weighted mass:  $w_{t+1} \leq c w_t$ . *Proof.* Expand  $m_{t+1} = m_t K$ ; mass entering  $A$  from outside contributes nothing (no re-entry), and each  $x \in A$  contributes  $m_t(x) \sum_{y \in A} K(x, y) V(y) \leq c m_t(x) V(x)$  by the recomputed contraction, using nonnegativity of kernel entries and of the evolving mass (the two mutant loci at this step). Sum over  $x \in A$ .  $\square$

**Theorem (decay, tails, termination).** By induction,  $w_t \leq c^t w_0$  (weighted alive-mass decay — at  $c = 1$  this is exactly the incumbent seam's nonpositive-drift conclusion). Per-step exit weight is dominated through the region values (exit domination), so for  $c < 1$  and any window  $[T, T']$ , the geometric series gives the truncation bound  $\kappa c^T \langle m_0, V \rangle / (1 - c)$  and — via the exact arithmetico-geometric identity (`sum_range_mul_pow_mul_eq`) and its tail estimates (`sum_Ico_pow_le`, `sum_Ico_mul_pow_le`) — the time-weighted first-moment tail  $\kappa \langle m_0, V \rangle c^T (T(1 - c) + c) / (1 - c)^2$ , both in universally-quantified partial-sum form: over  $\mathbb{Q}$  the limit need not exist, and any completion-valued normalizer, expected-value numerator, or moment row inherits the bound (`wmassAt_exit_tail_le`, `wmassAt_exit_moment_tail_le`). Dropping  $c < 1$  from a tail claim is unsound, not vacuous — at  $c = 1$  the rational division convention would make the assembled bound false (the battery's witness) — which is why the strict-contraction demand binds exactly where a tail-shaped claim row is spent.

**Corollary (bare alive-mass tail — the derived conversion).** Where  $\mu = \min_A V > 0$ , apply the unit-floor lemma to the rescaled certificate  $V/\mu$  (contraction and no-re-entry are scale-invariant; the floor  $V/\mu \geq 1$  on  $A$  holds by definition of  $\mu$ ):

$$\text{mass}_t(A) \leq c^t \langle m_0|_A, V \rangle / \mu$$

(`massAt_expect_on_le_of_regionContraction`, `dtmcAliveTail_of_regionContraction`) — the stored-row form of the hitting-time tail  $\Pr[\text{hitting time} > t] \leq M c^t$ . The conversion constant  $1/\mu$  is DERIVED, never stored: a stored constant would be a forgeable derived quantity. Both added hypotheses are witnessed load-bearing by kernel-checked countermodels committed beside the lemmas and re-checked every build: without the floor, one state at  $V = 1/2, c = 1$  makes the bound read  $1 \leq 1/2$  (`aliveTail_needs_floor`); without the closed complement, two states with the exterior feeding the region at probability one make it read  $1 \leq 0$  at  $t = 1$  (`aliveTail_needs_no_reentry`). With stochastic rows at every region state, an initial probability measure,  $0 \leq c < 1$ , no re-entry, and the positive minimum, weighted decay converts to bare survival decay and forces termination mass exactly one — the fourth claim relation; a sign-free weighted contraction alone would NOT suffice (the negative-weight self-loop countermodel is a battery tooth).

**Subsumption (the strengthening's extra obligation).** Mechanized twice over, as a strengthening of a landed seam must be: the instance chain — the weighted core at  $s \equiv 1$  on a stochastic kernel re-derives the previously proven stochastic core's exact statement (`massAt_expect_le_of_contraction_of_weighted`) — and the composed theorem (`qGenFormContraction_subsumes_genform`): from the incumbent's recomputed nonpositive generator form, the strengthened engine at the degenerate face (score  $\equiv 1, c = 1$ , region = all states) derives the incumbent consumption's exact path-law conclusion through the duality bridge, never by citing the incumbent's own proof. Every certificate admitted under the incumbent therefore discharges under the strengthening with an unchanged conclusion; the incumbent's rows, wire shape, and consuming lemmas are untouched.

**Prologue offsets.** The offset datum  $k$  (absent = 0) excludes a deterministic prologue from the claim window: partial survival sums index from  $k$ , and where a partial or expected-value claim spends it, the well-formedness demand recomputes prologue determinism — killed-on-exit evolution from the full stored initial measure with total surviving mass exactly one at step  $k - 1$  ( $k = 0$  vacuous). An unearned offset would shift the tail window onto mass the induction never controlled; the recompute refuses it.

*Remark R96.1 (status).* The finite face's schema is baked (entity, weight relation, demand, and the four named claim relations quoted in this section); the whole species' soundness, the subsumption pair, and the alive-mass face are mechanized sorry-free at standard axiom closure, axiom-audited on every build, with the species' mutant battery run as committed throwaway edits, each red at its predicted load-bearing step. No Lean statement is bound into this book yet: the binding column is honest open debt. The template and pattern faces are NOT part of this revision — their collapse warrants and the necessity witness for the eventual profile-length demand are mechanized and banked, and the faces board the revision that follows their own design work; until then template- and pattern-shaped claims live outside the certified lane, labeled as such.

## 15.13 The Soundness Theorem

*Its content is three conformance obligations on derivation edges, and together they are what turns the page's per-rule relative soundness lemmas — each proved on the hypothesis that the base calculus is sound — into a single verdict about a whole certificate, unconditional once those per-rule lemmas, base soundness, and the loader's conformance passes are supplied.*

### Rule 97. PROOFWELLFORMEDNESS (edge discipline (glue))

baked as *typing*, not as a node sort: E1 — derivation premises are judgment kinds XOR inherited Proof, never QProof, never a rule sort (the baked proof-node sorts' premise fields carry exactly these types); E2 — certificate-data references are not derivation edges; E3 — the admittedness + acyclicity pass over the certificate's own row graph (loader conformance, deliberately a check rather than a theory axiom: a cyclic certificate would chase-satisfy every axiom while proving nothing).

#### Lean statement

The discipline, mechanized at exactly the certificate-object level the recipes state it. The certificate shape binds by display — the four definitions ARE the E1/E2 typing claims — and the theorems bind as statements: the E3 bridge, the induction template in well-founded and finite-acyclic forms with root corollary, and the consumption-site transport pair. The induction theorem proper (towerCitation\_sound\_of\_wf) closes under no axioms at all.

BernsteinLogicAux.Acceptance . **acyclic\_wellFounded**

statement bound

```
-- **The E3 bridge**: on a FINITE certificate, the loader's checkable
acyclicity pass yields exactly the well-foundedness the induction needs.
This is why E3 can be loader conformance rather than an axiom of the
theory: well-foundedness is an induction principle the logical framework
deliberately lacks, but on finite row sets it is equivalent to the
decidable absence of citation cycles. -/
theorem acyclic_wellFounded [Finite ι] (C : ι → Row σ J ι)
  (hacyc : Acyclic C) : WellFounded (cites C)
```

BernsteinLogicAux.Acceptance . **towerCitation\_sound\_of\_wf**

statement bound

```
-- **rec:tc-template`, well-founded form** – the extension-edge
induction, in the shape the reflection theorem's induction consumes.
Hypotheses are the template's clauses: clause 1 = `hbase` (base
soundness; in deployment, the BL checker-soundness frontier – residue
R4); clause 2 = `hside` + `hstep` (the acceptance relation's local-check
clause as its own parameter, then the per-rule conditional-soundness
step in the cookie-cutter shape – the per-rule lemmas are CONSUMED here,
never proven: residue R1); clause 3 = `hwf` (the well-foundedness the E3
pass buys via `acyclic_wellFounded`). Conclusion: every admitted row's
conclusion-denotation holds – at each row the induction hypothesis
supplies exactly what the per-rule lemma consumes, and clause 1 grounds
the leaves. -/
theorem towerCitation_sound_of_wf
  (C : ι → Row σ J ι) (baseTrue : σ → Prop) (sem : J → Prop)
  (SideOK : ι → Prop)
  (hbase : ∀ i s, C i = Row.base s → baseTrue s)
  (hside : ∀ i, SideOK i)
  (hstep : ∀ i prems concl, C i = Row.ext prems concl → SideOK i →
    (∀ j ∈ prems, Row.holds baseTrue sem (C j)) → sem concl)
  (hwf : WellFounded (cites C)) :
  ∀ i, Row.holds baseTrue sem (C i)
```

BernsteinLogicAux.Acceptance . **towerCitation\_sound**

statement bound

```
-- **rec:tc-template`, headline form** – the discipline as the checker
consumes it: a FINITE certificate whose citation graph passes the E3
acyclicity check, base soundness, local checks, and the per-rule
conditional lemmas jointly warrant every admitted row's
conclusion-denotation. This is the theorem that turns the book's
per-rule RELATIVE soundness lemmas into an unconditional verdict about
a whole certificate. -/
```

```

theorem towerCitation_sound [Finite ι]
  (C : ι → Row σ J ι) (baseTrue : σ → Prop) (sem : J → Prop)
  (SideOK : ι → Prop)
  (hbase : ∀ i s, C i = Row.base s → baseTrue s)
  (hside : ∀ i, SideOK i)
  (hstep : ∀ i prems concl, C i = Row.ext prems concl → SideOK i →
    (∀ j ∈ prems, Row.holds baseTrue sem (C j)) → sem concl)
  (hacyc : Acyclic C) :
  ∀ i, Row.holds baseTrue sem (C i)

```

BernsteinLogicAux.Acceptance.towerCitation\_root

statement bound

```

/-- **Root soundness** (the acceptance relation's clause-4 shape): if
some admitted row of an accepted certificate concludes the root
judgment, that judgment's denotation holds. The abstract skeleton of
the reflection theorem's statement. -/
theorem towerCitation_root [Finite ι]
  (C : ι → Row σ J ι) (baseTrue : σ → Prop) (sem : J → Prop)
  (SideOK : ι → Prop)
  (hbase : ∀ i s, C i = Row.base s → baseTrue s)
  (hside : ∀ i, SideOK i)
  (hstep : ∀ i prems concl, C i = Row.ext prems concl → SideOK i →
    (∀ j ∈ prems, Row.holds baseTrue sem (C j)) → sem concl)
  (hacyc : Acyclic C)
  {Jroot : J} (hroot : ∃ i prems, C i = Row.ext prems Jroot) :
  sem Jroot

```

BernsteinLogicAux.Acceptance.citePos\_sound

statement bound

```

/-- **`prop:tc-consume` (a), positive citations**: the child's concluded
region `R` under-approximates its formula's denotation `D`, and any
MONOTONE consuming claim proved over `R` transfers to `D`. Which
operators are monotone in which positions is each consuming rule's own
lemma's business; this lemma fixes only the direction. A certificate can
lose completeness at a citation, never soundness. -/
theorem citePos_sound {X : Type*} {R D : Set X} {P : Set X → Prop}
  (hmono : Monotone P) (hsub : R ⊆ D) (hR : P R) : P D

```

BernsteinLogicAux.Acceptance.citeCmpl\_subset

statement bound

```

/-- **`prop:tc-consume` (b), the orientation swap**: an
under-approximating certified region's complement OVER-approximates the
denotation's complement – the conservative direction where regions
appear negatively. (Set-inclusion twin of
`BernsteinLogic.Structural.compl_over_approx`, which states the same
direction through the compiled region-judgment reading.) -/
theorem citeCmpl_subset {X : Type*} {R D : Set X} (hsub : R ⊆ D) :
  Dc ⊆ Rc

```

BernsteinLogicAux.Acceptance.citeCmpl\_sound

statement bound

```

/-- **`prop:tc-consume` (b), consumer form**: an ANTITONE claim proved on
the citable complement `Rc` covers the true violation set `Dc` – a
premise row proved on, or a probability bound proved for reaching, the
over-approximation covers the truth. -/
theorem citeCmpl_sound {X : Type*} {R D : Set X} {P : Set X → Prop}
  (hanti : Antitone P) (hsub : R ⊆ D) (hR : P Rc) : P Dc

```

BernsteinLogicAux.Acceptance.Row

definition bound

```

/-- **E1, as a type**: one row of a decoded certificate. A `base` row is a
base-calculus proof leaf concluding the sequent `s` (the `⊢` side); an
`ext` row is an extension row carrying its premise slots – references to
other rows, the derivation edges, and nothing else – and its concluded
satisfaction judgment (the `⊨` side). A premise slot can land on nothing
but another admitted row, consumed as that row's conclusion. -/

```

```

inductive Row (σ J ι : Type*) where
| base (s : σ)
| ext (prems : List ι) (concl : J)

```

BernsteinLogicAux.Acceptance . Row.cited

definition bound

```

/-- **E2, in one definition**: the derivation edges out of a row are its
premise slots and nothing else – the induction metric counts these only.
Certificate-data references (shared certificate families, lattice
objects) are unrepresented here by design; their erasure is the loader's
conformance obligation, not this module's. -/
def Row.cited : Row σ J ι → List ι
| .base _ => []
| .ext prems _ => prems

```

BernsteinLogicAux.Acceptance . Row.holds

definition bound

```

/-- The denotation of a row's CONCLUSION, under a kernel semantics
`baseTrue` for base sequents and a judgment semantics `sem` (the
judgment-denotation interface, kept abstract – zero measure theory in
this module by design). The two-case shape is E1's XOR: an edge lands on
kernel truth or on a judgment denotation; there is no third thing to
cite. -/
def Row.holds (baseTrue : σ → Prop) (sem : J → Prop) : Row σ J ι → Prop
| .base s => baseTrue s
| .ext _ concl => sem concl

```

BernsteinLogicAux.Acceptance . cites

definition bound

```

/-- Row `j` is cited by row `i`: a derivation edge from `i` down to `j`.
The well-founded induction descends along this relation. -/
def cites (C : ι → Row σ J ι) (j i : ι) : Prop :=
j ∈ (C i).cited

```

BernsteinLogicAux.Acceptance . Acyclic

definition bound

```

/-- **E3's checkable half**, as the induction consumes it: no row
transitively cites itself. On a finite certificate this is precisely
acyclicity of the citation graph – the decidable pass the loader runs. -/
def Acyclic (C : ι → Row σ J ι) : Prop :=
∀ i, ¬ Relation.TransGen (cites C) i i

```

## Correspondence (Steele display ↔ Lean)

### NO CORRESPONDENCE DICTIONARY YET

no reviewer-certified dictionary maps the displayed rule onto the mechanized statements above; until one is written and certified, that mapping rests on this section's editorial care alone.

## Lean verifier receipts

module BernsteinLogicAux.Acceptance; bound build e78746b9f98a... (bound from receipt 3d1fca564d6e11dc, 2026-07-20T03:41:07Z). A receipt records that named checking layers accepted a build: (module, compiled-artifact SHA-256, environment SHA-256, checker layers), minted when the checking gate ran; the store receipts attesting the bound build are listed with the layers each run credited. Nothing here asks the reader to trust the store: every layer is an independently runnable program over the quoted sources — rebuild the module and re-run the checkers to reproduce the verdict. Receipts attesting superseded builds credit no layer and are omitted from this rendering (the count of omissions, if any, is noted below).

| receipt          | minted               | layers                             | status  |
|------------------|----------------------|------------------------------------|---------|
| 3d1fca564d6e11dc | 2026-07-20T03:41:07Z | build+decorrelated                 | current |
| 3546fbecf9c242a2 | 2026-07-20T20:46:26Z | build+decorrelated                 | current |
| e684b09bdf01a640 | 2026-07-20T20:57:59Z | build+replay+hardened+decorrelated | current |
| fe82a04ec14e7cee | 2026-07-21T00:02:22Z | build+decorrelated                 | current |
| f0f540e41c59285c | 2026-07-21T05:38:30Z | build+replay+hardened+decorrelated | current |
| 69a2515828782208 | 2026-07-22T18:26:47Z | build+decorrelated                 | current |

|                  |                      |                    |         |
|------------------|----------------------|--------------------|---------|
| bd7c9443b7f8649a | 2026-07-31T21:59:38Z | build+decorrelated | current |
| 652943c163b17f25 | 2026-07-31T22:28:31Z | build+decorrelated | current |
| dcd3934b54c0db35 | 2026-08-02T23:30:39Z | build+decorrelated | current |
| 5d9475b9d8bbe3c2 | 2026-08-03T16:26:43Z | build+decorrelated | current |
| cbbad707861e8740 | 2026-08-03T22:45:26Z | build+decorrelated | current |

gate-layer coverage of the bound build:

build (lake build (elaboration)) **current**

replay (leanchecker (official kernel replay)) **current**

hardened (safe\_verify (hardened screen)) **current**

decorrelated (lean4export + nanoda (decorrelated kernel)) **current**

**comparator: none** no fixture in the three-implementation comparator harness (compiled / kernel-interpreted / reference-interpreted) exercises this rule.

## HOL4 statement

### OUT OF SCOPE BY DESIGN

Out of scope by structure: there is no rule statement to port — the discipline concludes nothing and has no check function of its own. Its one checkable obligation, the admittedness + acyclicity pass, is certificate-loader conformance, which would arrive with a HOL-side extension-theory loader as conformance code, not as a rule check. Independently, the HOL4 development's object today is the base calculus's wire-format checker (theory bern\_catalogue, the base catalogue's check-function cone); no extension-theory HOL development exists for such a pass to live in.

## HOL verifier receipts

### NONE

no HOL verification receipt exists for this rule.

## Proof

Genre: edge discipline (glue) — the extension page's fourth category, deliberately outside the three tiers. This entry is not a rule: it has no conclusion judgment, so “semantic soundness of the conclusion” does not type-check against it, and it has nothing to unpack, so neither rewriting genre applies — it *is* the unpacking mechanism. What it owes the book instead is the warrant every other section of the extension page silently leans on: each rule's soundness lemma is proved *relative* to base soundness, and this discipline is how those conditional lemmas compose into a single verdict about a whole certificate — unconditional once the per-rule lemmas, base soundness, and the loader's conformance passes are in hand; the closing remark names what remains conditional. The presentation apes the recipe form: the general induction template, then the discipline's obligations as the instance that makes the template apply.

**Recipe R97.1** (what the discipline licenses). A concluded child judgment  $\mathcal{M}$ ,  $x_0 \in R \models \varphi_{\text{child}}$  (the region-level satisfaction kind: every point of  $R$  satisfies  $\varphi_{\text{child}}$ , with  $R$  under-approximating the formula's denotation) makes  $R$  *citabile* as a region atom in any parent row's scope or target. The working example: a nested probability former  $\text{SProbGE}(p, \psi)$  (surface syntax  $P_{\geq p}[\psi]$ ) inside a state formula compiles its child first and cites the child's concluded region as the sublevel atom  $\{V_{\text{child}} \leq 1 - p\}$ . The license is an *edge* between rows of one certificate, never a node: the citation itself concludes nothing.

**Recipe R97.2** (the induction template). Let a certificate be a finite set of admitted rows: base-calculus proof rows (leaves, concluding sequents on the  $\vdash$  side) and extension rows (concluding satisfaction judgments on the  $\models$  side), each extension row's premise fields referencing other rows; call a row *admitted* when its local checks pass — its premise fields are well typed and its side conditions recompute true. Suppose:

1. *base soundness* — every admitted base row's sequent is true in kernel semantics;
2. *per-rule conditional soundness* — for every extension rule sort  $R$  a lemma  $\text{sound}_R$ : if the referenced child judgments' denotations hold, the base leaves are true, and the side conditions recompute true, then the concluded judgment's denotation holds;
3. *acyclicity* — the certificate's citation graph is acyclic.

Then every admitted row's judgment denotation holds: induct over the citation order, which clause 3 (with finiteness) makes well-founded; at each row the induction hypothesis supplies the child denotations, admittedness supplies the recomputed side conditions, and clause 1 grounds the leaves — together exactly what clause 2's lemma consumes. The conditional lemmas are the per-rule sections of this book; the induction is this section.

**Recipe R97.3** (the instance — the discipline's three obligations). The discipline is staged as *typing and conformance*, not as a node sort, in three obligations, each of which is one premise of Recipe R97.2 made checkable:



- **E1 (edge typing).** Every premise field of an extension row is typed as a satisfaction-judgment kind or as an inherited base-proof leaf, exclusively — never as a raw rule row of either theory. This makes the induction's case analysis exhaustive: every edge lands on something clause 1 or the induction hypothesis covers; no edge can cite an object that has no soundness account.
- **E2 (data/edge separation).** Certificate-*data* references — a certificate family reused across rows, a lattice object — are not derivation edges. The induction metric counts derivation edges only, so shared data can neither smuggle a cycle past the acyclicity pass nor impose ordering the derivation structure does not have.
- **E3 (admittedness + acyclicity).** A loader pass over the certificate's own row graph checks that every referenced row is admitted and that the citation graph is acyclic — clause 3 verbatim. This is *conformance*, deliberately not an axiom of the theory: an existence-of-concluding-proof axiom would let the theory's own saturation fabricate proof rows, and well-foundedness is an induction principle, which the logical framework deliberately lacks. A cyclic certificate would satisfy every local check while proving nothing — or worse: take two rows, each citing the other's concluded judgment as its sole premise, with per-rule lemmas of the shape  $J_B \Rightarrow J_A$  and  $J_A \Rightarrow J_B$ . In an instance where both judgments are false, both implications hold, every per-row obligation and the admittedness pass discharge, and only acyclicity objects; accepting the pair would not merely warrant nothing — it would assert falsehoods.

**Proposition R97.4** (consumption-site soundness). *Let the child conclude  $\mathcal{M}$ ,  $x_0 \in R \models \varphi$  with  $R \subseteq \llbracket \varphi \rrbracket$ . Then at every consumption site the citation is used in the sound direction:*

- (a) *a positive citation passes  $R$  exactly, and any monotone claim over  $R$  (“the state reaches / remains in  $R$ ”) entails the same claim for  $\llbracket \varphi \rrbracket$ ;*
- (b) *a complement citation swaps orientation:  $\text{cmpl}(R) \supseteq \text{cmpl}(\llbracket \varphi \rrbracket)$ , so a premise row proved on  $\text{cmpl}(R)$  — or an upper probability bound proved for reaching it — covers the true violation set.*

*Proof.* (a) is monotonicity of the consuming operator in  $R \subseteq \llbracket \varphi \rrbracket$ ; (b) is the same inclusion complemented. The discipline contributes exactly this inclusion and its swap; *which* operators are monotone in which positions is each consuming rule's own lemma's business, stated and proved in that rule's section. Under-approximation on the positive side and over-approximation on the complement side are both conservative: a certificate can lose completeness at a citation, never soundness.  $\square$

**Remark R97.5** (why this is not a rule, and what the mechanization is). The three tiers each predicate on a conclusion: core is the concluded judgment's semantic soundness; admissible and alias are verified unpackings of a concluded node. A discipline with no conclusion satisfies none of the three definitions — it is the induction step the tiers' lemmas plug into, which is why it renders under its own category head rather than wearing a tier that does not type-check. Its one *checkable* obligation (E3) lands as certificate-loader conformance code. The warrant itself is now mechanized at exactly the certificate-object level this section's recipes state it (BernsteinLogicAux/Acceptance.lean), anchor by anchor: Recipe R97.2 is `towerCitation_sound` — well-founded induction over admitted rows, consuming the per-rule lemmas as its step hypothesis — in finite-acyclic (headline) and well-founded forms, with root corollary `towerCitation_root`; the E1 typing of Recipe R97.3 is the row type itself, so the exhaustiveness of the induction's case analysis is type-checked (two consumptions exist: kernel truth of a base leaf, denotation of a concluded judgment — no third case to cite); E2 is the edge metric `Row.cited` (premise slots only); E3's sufficiency is the bridge `acyclic_wellFounded` — on a finite certificate the checkable acyclicity pass yields exactly the well-foundedness the induction needs, which is why conformance suffices where the logical framework deliberately lacks an induction axiom; and Proposition R97.4 is the transport pair `citePos_sound` / `citeCmpl_subset` / `citeCmpl_sound`. The induction theorem proper (`towerCitation_sound_of_wf`) depends on no axioms at all; the bridge and transport lemmas close under the standard three.

**Remark R97.6** (what remains conditional, named). This discipline's mechanization is conditional on named residues rather than a vague gap. **R1** — the per-rule lemmas enter the induction as a hypothesis: that is the discipline's literal content (it composes conditional lemmas; it does not prove them), but the type-level identification of that hypothesis's instances with this book's bound per-rule statement family awaits the deep embedding and the per-rule schema wrappers; until then the identification is editorial, rule by rule. **R2** — E2 is discharged by *representation*: the abstract row type carries derivation edges only, and the faithfulness of the concrete certificate format's erasure onto it (no derivation edge lost, no data reference kept) is loader/decode conformance, outside the module. **R3** — admittedness (E3's first half) is absorbed by the totality of the abstract certificate's indexing: every reference IS an admitted row, by type; the loader's admittedness check is what establishes that totality. (Clause 1, base soundness, is the base calculus's own soundness frontier and enters as the named hypothesis it always was — a premise of the template, not a residue of the mechanization.)

## References

- [1] J. Farkas. *Theorie der einfachen Ungleichungen*. Journal für die reine und angewandte Mathematik 124 (1902), 1–27.
- [2] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986. (Farkas variants and multiplier sign conventions: §7.)
- [3] J.-L. Krivine. *Anneaux préordonnés*. Journal d'Analyse Mathématique 12 (1964), 307–326.
- [4] D. Handelman. *Representing polynomials by positive linear functions on compact convex polyhedra*. Pacific Journal of Mathematics 132 (1988), no. 1, 35–62.
- [5] B. D. H. Tellegen. *A general network theorem, with applications*. Philips Research Reports 7 (1952), 259–269.
- [6] J. C. Willems. *Dissipative dynamical systems, Part I: General theory*. Archive for Rational Mechanics and Analysis 45 (1972), 321–351.
- [7] P. J. Moylan and D. J. Hill. *Stability criteria for large-scale systems*. IEEE Transactions on Automatic Control 23 (1978), no. 2, 143–149.
- [8] A. van der Schaft and D. Jeltsema. *Port-Hamiltonian Systems Theory: An Introductory Overview*. Foundations and Trends in Systems and Control 1 (2014), no. 2–3, 173–378.
- [9] Z.-P. Jiang, A. R. Teel, and L. Praly. *Small-gain theorem for ISS systems and applications*. Mathematics of Control, Signals and Systems 7 (1994), 95–120.
- [10] Z.-P. Jiang, I. M. Y. Mareels, and Y. Wang. *A Lyapunov formulation of the nonlinear small-gain theorem for interconnected ISS systems*. Automatica 32 (1996), no. 8, 1211–1215.
- [11] K. J. Åström and K. Furuta. *Swinging up a pendulum by energy control*. Automatica 36 (2000), no. 2, 287–295.
- [12] G. L. Steele Jr. *It's Time for a New Old Language*. Keynote, ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), 2017. (The “Computer Science Metanotation” this book’s rule cells are set in.)
- [13] G. Gentzen. *Untersuchungen über das logische Schließen I, II*. Mathematische Zeitschrift 39 (1935), 176–210 and 405–431. English translation in M. E. Szabo (ed.), *The Collected Papers of Gerhard Gentzen*, North-Holland, 1969.
- [14] J.-Y. Girard. *Linear logic*. Theoretical Computer Science 50 (1987), no. 1, 1–101. (Source of the additive/multiplicative vocabulary for context-sharing vs. context-splitting rule formats.)
- [15] G. Birkhoff. *Lattice Theory*. 3rd ed., American Mathematical Society Colloquium Publications 25, 1967. ( $M_3$ , the diamond, as the canonical modular non-distributive lattice.)
- [16] W. Rudin. *Principles of Mathematical Analysis*. 3rd ed., McGraw-Hill, 1976. (Taylor’s theorem with the Lagrange form of the remainder: Theorem 5.15; the nonnegative-remainder enclosure at nonnegative argument used by the TAYLORBOUND section.)
- [17] S. Banach. *Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales*. Fundamenta Mathematicae 3 (1922), 133–181.
- [18] A. Granas and J. Dugundji. *Fixed Point Theory*. Springer Monographs in Mathematics, Springer, 2003. (The closed-ball condition  $d(Tx_0, x_0) \leq (1 - \rho)r$  and the a-priori bound  $d(x_0, x^*) \leq \delta/(1 - \rho)$ : §I.1.)
- [19] W. Doeblin. *Exposé de la théorie des chaînes simples constantes de Markov à un nombre fini d’états*. Revue Mathématique de l’Union Interbalkanique 2 (1938), 77–105. (The origin of the coupling method for Markov chains.)
- [20] T. Lindvall. *Lectures on the Coupling Method*. Wiley, 1992; corrected reprint, Dover, 2002. (The coupling inequality  $TV \leq \Pr[X \neq Y]$ : §I.2.)
- [21] D. A. Levin and Y. Peres, with E. L. Wilmer. *Markov Chains and Mixing Times*. 2nd edition, American Mathematical Society, 2017. (Total-variation distance, couplings, and the data-processing/projection inequality: Ch. 4–5.)
- [22] J. Hsu. *Probabilistic Couplings for Probabilistic Reasoning*. PhD thesis, University of Pennsylvania, 2017. arXiv:1710.09951. (Couplings as certificates in relational program logics — the pre/post-style sibling of the judgment form used here.)
- [23] A. Girard and G. J. Pappas. *Approximation metrics for discrete and continuous systems*. IEEE Transactions on Automatic Control 52 (2007), no. 5, 782–798. (Approximate simulation with quantitative degradation — the analytic ancestor of the  $\epsilon$ -witness transport cell.)
- [24] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. *Metrics for labelled Markov processes*. Theoretical Computer Science 318 (2004), no. 3, 323–354. (Behavioral pseudometrics for probabilistic systems.)
- [25] H. Weyl. *The Classical Groups: Their Invariants and Representations*. Princeton University Press, 1939. (Polarized power sums generate the multisymmetric invariants in characteristic 0.)
- [26] T. Murata. *Petri nets: Properties, analysis and applications*. Proceedings of the IEEE 77 (1989), no. 4, 541–580. (Place invariants — the linear conservation laws entering as free count rows.)
- [27] D. I. Spivak, R. Garner, and A. D. Fairbanks. *Functorial aggregation*. arXiv:2111.10968. (Aggregation as commutative-monoid folds along a classifying map; the multiset as the universal aggregate.)
- [28] S. Dashkovskiy, B. S. Rüffer, F. R. Wirth. *An ISS small gain theorem for general networks*. Mathematics of Control, Signals, and Systems 19(2):93–122, 2007. DOI 10.1007/s00498-007-0014-8. (The  $n$ -component ISS small-gain theorem over arbitrary interconnection graphs whose certificate form the SMALLGAIN section catalogues; its monotone-operator fixpoint machinery lives producer-side as existence theory for the  $(\lambda, B, \theta)$  side data.)
- [29] G. Nelson and D. C. Oppen. *Fast decision procedures based on congruence closure*. Journal of the ACM 27 (1980), no. 2, 356–364.
- [30] R. Nieuwenhuis and A. Oliveras. *Fast congruence closure and extensions*. Information and Computation 205 (2007), no. 4, 557–574.
- [31] S. N. Bernstein. *Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités*. Communications de la Société Mathématique de Kharkov (2) 13 (1912), 1–2.
- [32] G. T. Cargo and O. Shisha. *The Bernstein form of a polynomial*. Journal of Research of the National Bureau of Standards 70B (1966), 79–81. (Bernstein coefficients enclose the range on the simplex — the sound direction this calculus’s kernel uses.)
- [33] R. T. Farouki. *The Bernstein polynomial basis: a centennial retrospective*. Computer Aided Geometric Design 29 (2012), no. 6, 379–419.
- [34] P. de Casteljau. *Outillages méthodes calcul*. Technical report, A. Citroën, Paris, 1959. (The founding subdivision/evaluation algorithm for polynomials in Bernstein form.)
- [35] J. M. Lane and R. F. Riesenfeld. *A theoretical development for the computer generation and display of piecewise polynomial surfaces*. IEEE Transactions on Pattern Analysis and Machine Intelligence 2 (1980), no. 1, 35–46.

- [36] J. Garloff. *Convergent bounds for the range of multivariate polynomials*. In: Interval Mathematics 1985 (K. Nickel, ed.), Lecture Notes in Computer Science 212, Springer, 1986, 37–56. (Subdivision sharpens Bernstein enclosures — the completeness partner of the enclosure soundness used here.)
- [37] R. Ehlers. *Formal verification of piece-wise linear feed-forward neural networks*. Proceedings of ATVA 2017 (Automated Technology for Verification and Analysis), 2017. (The “triangle” linear relaxation of ReLU.)
- [38] E. Wong and J. Z. Kolter. *Provable defenses against adversarial examples via the convex outer adversarial polytope*. Proceedings of ICML 2018, PMLR 80, 2018.
- [39] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. *Efficient neural network robustness certification with general activation functions*. Advances in Neural Information Processing Systems 31 (NeurIPS 2018). (CROWN: adaptive two-sided linear bounds.)
- [40] G. Singh, T. Gehr, M. Püschel, and M. Vechev. *An abstract domain for certifying neural networks*. Proceedings of the ACM on Programming Languages 3 (POPL 2019), Article 41. (DeepPoly.)
- [41] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh. *Automatic perturbation analysis for scalable certified robustness and beyond*. Advances in Neural Information Processing Systems 33 (NeurIPS 2020). (The LiRPA framework and name.)
- [42] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. *Reluplex: an efficient SMT solver for verifying deep neural networks*. Proceedings of CAV 2017 (Computer Aided Verification), 2017.
- [43] R. Bunel, J. Lu, I. Turkaslan, P. H. S. Torr, P. Kohli, and M. P. Kumar. *Branch and bound for piecewise linear neural network verification*. Journal of Machine Learning Research 21 (2020).
- [44] David “davidad” Dalrymple. *Certifying strict positive definiteness via an approximate triangular inverse* (post). X (formerly Twitter), 2026, status 2022316806094913669. (The triangular-inverse method: sign-and-pattern checks on  $A \cdot Z$  for an untrusted triangular witness  $Z$ ; no divisions, witness accuracy never load-bearing.)
- [45] G. Irving. *Answer to: Certificate for strict positive definiteness of a sparse matrix*. Computational Science Stack Exchange, 2026 (question 45365, answer 45400). (The rounding-tolerant integer variant and its certificate-size analysis.)
- [46] G. Irving. *aks* (Lean formalization repository): the Random library’s diagonal-dominance PSD bridge and a certified 65536-node spectral expander. [github.com/girving/aks](https://github.com/girving/aks), 2026.
- [47] S. Gershgorin. *Über die Abgrenzung der Eigenwerte einer Matrix*. Bulletin de l’Académie des Sciences de l’URSS, Classe des sciences mathématiques et naturelles 6 (1931), 749–754.
- [48] O. Taussky. *A recurring theorem on determinants*. American Mathematical Monthly 56 (1949), no. 10, 672–676. (The Levy–Desplanques strict-dominance nonsingularity theorem and its history.)
- [49] E. G. Boman, D. Chen, O. Parekh, and S. Toledo. *On factor width and symmetric  $H$ -matrices*. Linear Algebra and its Applications 405 (2005), 239–248. (Factor-width-2 = scaled diagonal dominance — the class the /DD tier’s congruence certificates inhabit.)
- [50] R. A. Horn and C. R. Johnson. *Matrix Analysis*, 2nd edition. Cambridge University Press, 2013. (Sylvester’s criterion, congruence, and Gershgorin discs, as cited in the certificate proofs.)
- [51] M. Abadi and L. Lamport. *Composing Specifications*. ACM Transactions on Programming Languages and Systems 15 (1993), no. 1, 73–132.
- [52] K. L. McMillan. *Circular Compositional Reasoning about Liveness*. In: Correct Hardware Design and Verification Methods (CHARME 1999), Lecture Notes in Computer Science 1703, Springer, 1999, 342–346.
- [53] N. Lynch and F. Vaandrager. *Forward and Backward Simulations, Part I: Untimed Systems*. Information and Computation 121 (1995), no. 2, 214–233.
- [54] M. Capucci and D. J. Myers. *Certified lenses and compositional system certification*. arXiv:2604.03017 (2026). (The wiring-square conditions, the  $\mathcal{K}_\infty^0$  slack discipline, and the compose-by-lens-composition-and-slack-addition law cited in the composition sections.)
- [55] J. Ville. *Étude critique de la notion de collectif*. Monographies des Probabilités, Gauthier-Villars, Paris, 1939. (The maximal inequality for nonnegative supermartingales — the probability engine of every battery rule.)
- [56] J. L. Doob. *Stochastic Processes*. Wiley, New York, 1953. (Systematic supermartingale theory; maximal inequalities and optional sampling.)
- [57] H. J. Kushner. *Stochastic Stability and Control*. Mathematics in Science and Engineering 33, Academic Press, New York, 1967. (Supermartingale safety certificates for stochastic systems.)
- [58] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. 2nd edition, Cambridge University Press, 2009. (Foster–Lyapunov drift conditions; the geometric drift form behind the deadline battery.)
- [59] S. Prajna, A. Jadbabaie, and G. J. Pappas. *A framework for worst-case and stochastic safety verification using barrier certificates*. IEEE Transactions on Automatic Control 52 (2007), no. 8, 1415–1428.
- [60] R. S. Streett. *Propositional dynamic logic of looping and converse is elementarily decidable*. Information and Control 54 (1982), no. 1–2, 121–141. (The complemented-pairs acceptance condition.)
- [61] M. O. Rabin. *Decidability of second-order theories and automata on infinite trees*. Transactions of the American Mathematical Society 141 (1969), 1–35.
- [62] L. H. Landweber. *Decision problems for  $\omega$ -automata*. Mathematical Systems Theory 3 (1969), no. 4, 376–384. (Deterministic Büchi automata recognize exactly the limit languages.)
- [63] W. Thomas. *Automata on infinite objects*. In: Handbook of Theoretical Computer Science, vol. B, 133–191. Elsevier, 1990.
- [64] F. G. Foster. *On the stochastic matrices associated with certain queuing processes*. Annals of Mathematical Statistics 24 (1953), no. 3, 355–360. (The additive-drift criterion.)
- [65] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Springer, 1993. (Foster–Lyapunov drift conditions, systematized.)
- [66] A. Chakarov and S. Sankaranarayanan. *Probabilistic program analysis with martingales*. In: Computer Aided Verification (CAV 2013), Lecture Notes in Computer Science 8044, 511–526. Springer, 2013. (Ranking supermartingales.)
- [67] R. Durrett. *Probability: Theory and Examples*. 5th edition, Cambridge University Press, 2019. (Borel–Cantelli.)

## REFERENCES

- [68] C. Conley. *Isolated Invariant Sets and the Morse Index*. CBMS Regional Conference Series in Mathematics 38, American Mathematical Society, 1978.
- [69] C. Conley and R. Easton. *Isolated invariant sets and isolating blocks*. Transactions of the American Mathematical Society 158 (1971), 35–61.
- [70] G. Birkhoff. *Rings of sets*. Duke Mathematical Journal 3 (1937), no. 3, 443–454. (The representation theorem for finite distributive lattices via join-irreducibles.)
- [71] M. Nagumo. *Über die Lage der Integralkurven gewöhnlicher Differentialgleichungen*. Proceedings of the Physico-Mathematical Society of Japan (3) 24 (1942), 551–559.
- [72] F. Blanchini. *Set invariance in control*. Automatica 35 (1999), no. 11, 1747–1767.
- [73] S. Prajna and A. Jadbabaie. *Safety verification of hybrid systems using barrier certificates*. Hybrid Systems: Computation and Control (HSCC 2004), LNCS 2993, Springer, 2004, 477–492.
- [74] W. D. Kalies, K. Mischaikow, and R. C. A. M. van der Vorst. *An algorithmic approach to chain recurrence*. Foundations of Computational Mathematics 5 (2005), no. 4, 409–449.
- [75] W. D. Kalies, K. Mischaikow, and R. C. A. M. van der Vorst. *Lattice structures for attractors II*. Foundations of Computational Mathematics 16 (2016), 1151–1191.
- [76] W. D. Kalies, K. Mischaikow, and R. C. A. M. van der Vorst. *Lattice structures for attractors III*. Journal of Dynamics and Differential Equations 34 (2022), 1729–1768. arXiv:1911.09382.
- [77] K. Mischaikow and M. Mrozek. *Conley index*. Handbook of Dynamical Systems, Vol. 2, North-Holland, 2002, 393–460.
- [78] W. D. Kalies and T. Wehbe. *Compositionality of Global Dynamics in Product and Skew-Product Systems*. arXiv:2606.30925, 2026.
- [79] R. Statman. *Lower bounds on Herbrand's theorem*. Proceedings of the American Mathematical Society 75 (1979), no. 1, 104–107. (Non-elementary lower bounds on quantifier-inference elimination; with [191], the classical witness for the cut-elimination blow-up.)
- [80] V. P. Orevkov. *Lower bounds for lengthening of proofs after cut-elimination*. Zapiski Nauchnykh Seminarov LOMI 88 (1979), 137–162; English translation in Journal of Soviet Mathematics (1982).

# Partial Completeness

Every rule in the catalogue of chapter 15 carries its own soundness story: a statement, a proof, and, where mechanization has caught up, a machine-checked realization. Soundness can therefore be audited locally. Each rule earns its keep on its own, and a user who trusts the kernel need trust nothing else.

Completeness makes a different kind of claim, and no amount of local auditing can establish it. A completeness theorem says that the rule set as a whole is not missing anything, so that every true entailment expressible in some class of specifications is actually derivable. Claims of that shape must be won fragment by fragment, by exhibiting either a derivation or a countermodel for every specification the fragment can express. The prize is a change in the meaning of refusal. Without completeness, a tool that fails to derive a specification reports only that a rule might be missing; within a complete fragment, the same failure delivers a definite verdict about the specification itself.

This chapter records what has been proved. A reader who wants only the shape of the result can stop after the completeness theorem (Theorem 16.4); the axiom basis in section 16.1.1 shows what completeness costs, and the closing sections account for the evidence and draw the boundary of the claim. The most surprising idea in the chapter, for a reader browsing, lives in Remark 16.5 and in Figure 16.1: the models that witness completeness must branch, even though nothing in the specifications mentions branching.

## 16.1 A complete fragment: hard deadlines under staggering

The fragment lives in the timed core of the calculus and uses two modalities over propositional logic.

**Definition 16.1** (clocked model). A *clocked model* has finitely many locations, each carrying a positive maximal dwell time and a fixed valuation of the atoms. A *trajectory* is a timed path through locations that respects the dwell bounds.

The two modalities read as follows.

- $\Box\varphi$ , *invariance*:  $\varphi$  holds at every point reachable from here. The formula  $D := \Box\perp$  (“deadness”) holds exactly at the states with no future, so  $\neg D$  asserts liveness.
- $A_T\varphi$ , *worst-case service*: every trajectory from here reaches a  $\varphi$ -point within time at most  $T$ . The quantifier ranges over all branches, so the operator reports a worst case rather than a per-path promise. That one word, “every”, is the source of everything difficult below.

The characteristic constraint of the fragment, and the origin of its working name, is the *gnand*<sup>1</sup>, a globally held prohibition of joint imminence:

$$\text{gnand}(a, d_a, b, d_b) := \Box\neg(A_{d_a}a \wedge A_{d_b}b),$$

read: at no reachable moment is  $a$  due within  $d_a$  while  $b$  is due within  $d_b$ . It serves as a natural mutual-exclusion primitive for services that must stay temporally staggered, two operations that may never both be imminent, in either order.

<sup>1</sup>“Graded NAND”: a prohibited conjunction, as in NAND, whose conjuncts are graded by time bounds. The name arose as a working label during the development and proved too convenient to retire.



**Example 16.2** (a single-boiler espresso machine). A single-boiler espresso machine cannot brew and steam at the same time. Suppose an espresso order  $e$  must be served within 90 seconds on every branch of the schedule, a milk order  $m$  within 120, and the boiler forbids any reachable moment at which both are due within 30 seconds. The specification comprises the pending obligations  $\neg e \wedge A_{90} e$  and  $\neg m \wedge A_{120} m$ , the staggering constraint  $\text{gand}(e, 30, m, 30)$ , and liveness  $\neg D$ . Suppose the café also advertises that the 90-second figure is tight, so that some branch really does take the full 90. That assertion,  $\neg A_{89} e$ , already forces the schedule to *branch*: a barista who always serves promptly would make the worst case better than declared, so some branch of the schedule must dawdle while another serves. This innocent observation grows into the central construction of the completeness proof (Remark 16.5).

The example's shape, in general, defines the fragment.

**Definition 16.3** (g<sub>and</sub> core). A *g<sub>and</sub> core* is a specification asserting, over  $n$  atoms:

- **pending obligations**  $\neg a \wedge A_{s_a} a$ , meaning  $a$  has not yet been served and must be served within  $s_a$  on every branch;
- optionally, **exactness**  $\neg A_{s_a-1} a$  (the deadline is tight: some branch delays past  $s_a - 1$ ) or an **avoidance fact**  $\neg A_h a$  (some branch withholds  $a$  through time  $h$ );
- any finite set of g<sub>and</sub> constraints over pairs of atoms; and liveness  $\neg D$ .

Deadlines may be taken to be integers with no loss of generality.<sup>2</sup>

The fragment deliberately excludes triggered obligations  $\Box(x \rightarrow A_d y)$ , the atom-level variant of the staggering constraint, and possibility assertions  $\neg \Box \nu$  with their associated time-locks. The boundary is principled rather than incidental; it follows a *sign*. Every constraint in the fragment either forbids joint imminence or states a fact at the initial state, and the model construction below is protected by a monotonicity lemma that holds for such negative-side checks. The excluded formers *demand* positive worst-case facts at interior states of a model. Constructions that meet such demands exist in the development record, but their completeness theory is open, and this section claims nothing about them.

### 16.1.1 The proof theory

The axiom basis was not designed in advance; it was *discovered*, by iterating between attempted completeness proofs and machine searches for countermodels. Three axiom schemata were added to the seed basis this way, each certified genuinely new by an explicit independence witness before being adopted, and several others were pruned as derivable. The final basis comprises:

- four axioms governing the invariance modality;
- reflexive service,  $\varphi \rightarrow A_T \varphi$ ;
- budget composition,  $A_{T_1} A_{T_2} \varphi \leftrightarrow A_{T_1+T_2} \varphi$ ;
- the graded absurdity law,  $A_T \perp \rightarrow D$ ;
- relativization of obligations through invariants,  $\Box \nu \wedge A_T (\chi \wedge \nu) \leftrightarrow \Box \nu \wedge A_T \chi$ ;
- a first-service law: at the first service among  $n$  co-pending obligations, the remaining obligations still bind; and
- the last and decisive addition, a *residual-split* first-service law. For any cutoff  $c$  strictly below all deadlines,

$$\bigwedge_{i \leq n} A_{T_i} \alpha_i \rightarrow A_c \left( \bigvee_i (\alpha_i \wedge \bigwedge_{i' \neq i} A_{T_{i'}} \alpha_{i'}) \vee \bigwedge_i A_{T_i-c} \alpha_i \right),$$

meaning that within  $c$ , either some first service has occurred (and the co-obligations persist), or nothing has been served and every obligation holds at its true residual  $T_i - c$ .

Derivation rules are propositional calculus, modus ponens, import of a background theory, monotonicity for both modalities, and rational rescaling.

<sup>2</sup>Rational rescaling and the decomposition of a deadline into iterated unit deadlines are derivable rewrites in the proof theory, so a fractional menu of deadlines reduces to an integer one.

### 16.1.2 The completeness theorem

**Theorem 16.4** (completeness for the gnand fragment). *For gnand cores, the following are equivalent:*

1. *the core is satisfiable in a clocked model;*
2. *an effective arithmetic on the core (a fixpoint computation on per-atom effective deadlines and service-precedence edges, with one additional branch-conditional check) fires none of its refutation conditions;*
3. *an explicit finite model of the core exists on the uniform grid with dwell  $\frac{1}{n+1}$ , where  $n$  is the number of pending atoms, and the theorem's proof constructs it;*
4. *the core is consistent:  $\nVdash D$  from the core in the basis above.*

In particular, derivability and clocked validity coincide on the fragment. Since soundness is proved over a much broader class of trajectory spaces, of which clocked models are a special case, validity over every intermediate semantic class coincides with both. Satisfiability and validity of cores are therefore *decidable*, not merely semi-decidable; the fixpoint arithmetic of item (2) decides both.

Within its fragment the theorem is unconditional; its statement carries no unproven hypotheses. Its proof, however, forms a chain with several links, and the accounting must proceed link by link. That every arithmetic refutation in item (2) is matched by a formal derivation of  $D$  is a theorem, with the residual-split law doing the work its adoption promised: the arithmetic's deadline-tightening step is derivable by iterating the law at successive cutoffs, descending step by step to the arithmetic's own value. That every arithmetic-feasible core yields a model at grid  $\frac{1}{n+1}$  combines a proved monotone-protection lemma with one combinatorial schema whose verification is per-instance and population-scale (section 16.1.3) rather than symbolic.

*Remark 16.5* (delayer wings). The model construction deserves a name of its own. The emitted models are *schedule trees*: an earliest-deadline-first spine together with *delayer wings*, auxiliary branches whose only role is to keep each declared worst case at its declared value, so that the negative graded facts remain true even though every individual path serves early (Figure 16.1). That branching structure can satisfy staggering constraints which every branching-free model provably violates. This was the central semantic discovery of the development, and it is why the fragment's models are trees rather than runs.

### 16.1.3 Evidence and assurance status

The proofs are prose mathematics, developed against an unusually dense instrument battery. The completeness chain itself is not yet mechanized in a proof assistant. The three axiom schemata the search added, however, now carry sorry-free machine-checked soundness lemmas (`sound_u10f`, `sound_firstService`, and `sound_fsResidual`, in `Qtsl/TimedCore.lean` under `lean/qtsl` in `maitria-coalition/qtsl`; their statements are reproduced in Appendix A). A machine-checked countermodel, `firstService_attainment_necessary`, accompanies them. The two first-service lemmas assume two properties of the trajectory space, fusion closure and attainment of first service; clocked models satisfy both. The countermodel shows that the attainment hypothesis cannot be dropped, by exhibiting a fusion-closed space in which the first-service

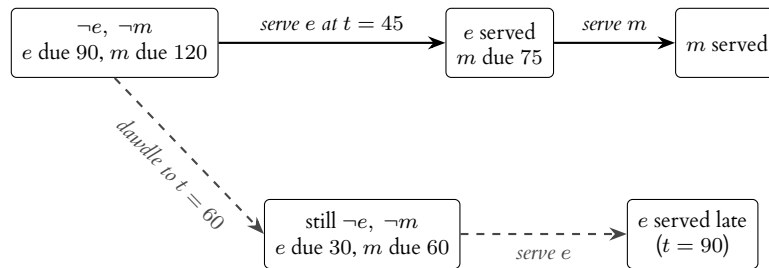


Figure 16.1: A schedule tree for the espresso core of Example 16.2. The solid spine serves early, in earliest-deadline-first order. The dashed delayer wing dawdles, so that the worst-case service time of  $e$  really attains the declared 90 seconds and the exactness assertion  $\neg A_{89} e$  comes out true. No single run can do both jobs; the tree does.



schema's premises hold and its conclusion fails. Rule-level mechanization otherwise tracks the catalogue's own realization discipline (chapter 15). The machine evidence behind the chain, all of it replayable from the development record as it migrates to `maitria-coalition/qtsl`:

| instrument         | what it checks                            | result                           |
|--------------------|-------------------------------------------|----------------------------------|
| derivation sweep   | arithmetic-refuted cores derive $D$       | 64,786 cells, 0 underivable      |
| cross-pair sweep   | the descent lemma's coverage              | 4,116/4,116 derived              |
| branch engine      | branch-conditional refutations derive $D$ | 1,000/1,000; cycle mode 200/200  |
| model emitter      | feasible cores yield grid models          | 122/122 named; 541/541 random    |
| soundness controls | no false derivations of $D$               | silent on every satisfiable core |

The controls in the last row did real work during the development. They caught two draft derivation engines asserting more than their machinery guaranteed, both retired before any proof consumed them, and the same battery falsified three earlier conjectured axiom bases outright. A rule set that has survived that regime differs, as an epistemic object, from one that has merely not yet failed.

#### 16.1.4 What is not claimed

Completeness is claimed for `g` and cores only. The composition of this fragment with triggered obligations and possibility constraints is open theory; so is the calculus's next expressive tier, in which worst-case service generalizes to a time-bounded *until* that carries safety clauses along the way. Minimality of the basis, a separate question from sufficiency, is settled for integer deadline menus (the residual-split law is genuinely underivable from the rest: there is a six-state witness, and six states is minimal) and open for arbitrarily fine fractional menus. No answer to the minimality question can weaken the theorem, since a redundant axiom would only mean that the same completeness holds from fewer assumptions.

PART III

MTK

---

# Tutorial: A Re-entrant Semiconductor Fab

---

DRAFT

The preceding chapters introduced the stack one layer at a time: mechanisms and their composition (chapter 12), the pattern inventory that composition ships as (chapter 13), the semantics that specifications are stated in (chapter 9), the plated-certificate discipline for token-population systems (section 10.4 and section 12.18), and the producer stack that finds certificates for a checker to replay (chapter 19). This chapter runs the whole stack over a single model, end to end: describe a factory line, declare it as database rows — an SDCPN, a stochastically and dynamically coloured Petri net in the modelling catalogue’s frame (section 12.18), which is the native habitat of everything that follows — recompose it along its natural seams, state what should be certified about it, and read the certificates. The certificates come in two families whose division of labour is stated rather than smoothed over. The primary family (section 17.5) certifies the discrete line itself, at its full combinatorial state space: exact statements holding along every run, at every lot count, under every resolution of the scheduling freedom — the variadic-certificate rows of section 12.18 at work on their first industrial subject, and then re-read on the wall clock, where the dispatch seat is priced end to end: a certified floor over every policy, the exact optimum and the supervisor family attaining it, a widened full-dispatch seat, an untrusted filler inside a certified envelope, and one certificate instantiated across six orders of magnitude of lot count. The second family (section 17.6) certifies the *fluid relaxation* of the same line: policy-indexed stability under sustained arrivals, at loads within a tenth of a lot per week of capacity — claims the discrete family does not yet price, which is why the fluid lane is retained in full as the comparison lane, with the contrast stated where the two families meet. A closing section then builds the fluid lane’s model on the toolkit’s TypeScript surface and drives its release grid back to checked certificates, in a listing that runs.

The model is the *Mini-Fab*: five machines in three stations, six route steps, every station visited twice per lot. It was posed by Kempf at Intel in 1994 as a deliberately minimal semiconductor-fab exercise and has circulated as a community benchmark since; the complete quantitative restatement that this chapter’s constants are witnessed against is van den Berk’s 2004 Eindhoven master’s thesis [17] (the systems engineering group of Rooda and Lefeber), whose second chapter restates the entire case. The spec digest recording that witnessing, decision point by decision point, is committed beside the certificates in the experiments repository (section 2.9).

Small as it is, the line exhibits the domain’s whole hard structure: re-entrant flow, a single-machine bottleneck, batching with mix constraints, sequence-dependent setups, personnel shared across stations, maintenance calendars, and machine failures that destroy work in progress. And its headline claim family has teeth, because it is famously falsifiable: re-entrant lines can destabilize strictly below nominal capacity under plausible dispatch policies (the Kumar–Seidman phenomenon [145]), so “this policy keeps the line stable at this load” is a claim the world is entirely capable of refuting.

## 17.1 The line

Three workstations process lots of wafers. *Diffusion* has two furnaces, a buffer for 18 lots, and consumes its input in batches of exactly three lots. *Implantation* has two machines and a buffer for 12. *Lithography* has one machine and a buffer for 12. Every lot follows the same six-step route — diffusion, implantation, lithography, implantation, diffusion, lithography — so each station is visited twice, running a “low” recipe on a lot’s first visit and a “high” recipe on its second:

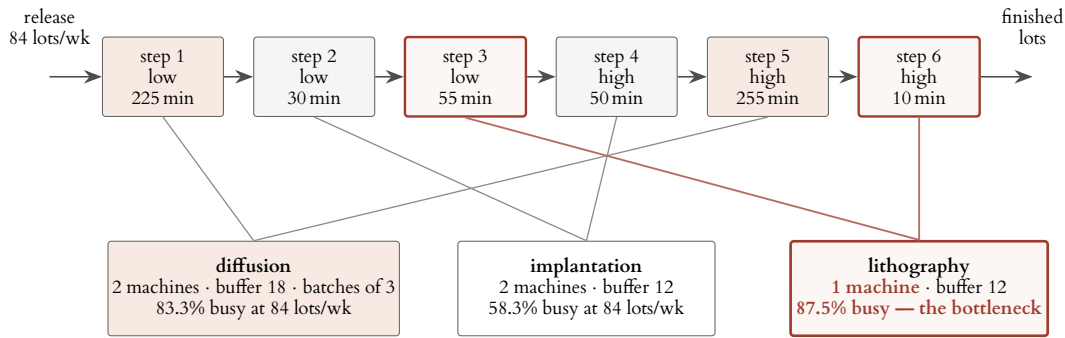


Figure 17.1: The Mini-Fab's re-entrant route. Buffers are drawn in route order (top); the three stations (bottom) each serve two non-consecutive steps, which is what *re-entrant* means — the crossing connectors are the route folding back on itself. Lithography runs both of its steps on a single machine and is the line's bottleneck. Utilizations are the deterministic machine-occupancy figures at the nominal release of 84 lots/week.

| step | station      | recipe | process (min) | load (min) | unload (min) |
|------|--------------|--------|---------------|------------|--------------|
| 1    | diffusion    | low    | 225           | 20         | 40           |
| 2    | implantation | low    | 30            | 15         | 15           |
| 3    | lithography  | low    | 55            | 10         | 10           |
| 4    | implantation | high   | 50            | 15         | 15           |
| 5    | diffusion    | high   | 255           | 20         | 40           |
| 6    | lithography  | high   | 10            | 10         | 10           |

Loading and unloading hold the machine; at diffusion they are paid per *batch* of three, not per lot. There is no preemption and no rework.

Three product types are released on constant per-type intervals: 3 test lots, 51 of type A, and 30 of type B per week of  $7 \times 1440 = 10080$  minutes — 84 lots/week in all. One release interval is worth pausing on: type A arrives every  $10080/51 = 3360/17$  minutes. The parameters are all integers, and still the event times of the system are not, so exact rational time arithmetic is a necessity rather than a nicety, and the declared model keeps every duration in  $\mathbb{Q}$ . Arrivals are deterministic; the only stochastic elements anywhere in the specification are implantation failures and repairs (below).

Around the route sit the resources that make the model interesting. One *transporter* (capacity one lot, 4 minutes per move between adjacent areas, 1 minute to load or unload) carries every lot between stations. Two *operators* perform all machine loading and unloading, non-preemptively: operator 0 serves diffusion and implantation, operator 1 serves lithography and implantation, so implantation is contested and every station's handling competes with another station's. Lithography additionally pays a sequence-dependent *setup*, performed by an operator with the machine empty, whenever its input changes against the previous job: 5 minutes for a product-type change, 10 for a step change, 12 for both. At the two-machine stations, a *test* lot's second visit must use the machine it did not use on its first (so both machines stay monitored); a diffusion batch contains at most one test lot, and a high-step batch must not mix the two commercial types. Each operator takes three 60-minute off-blocks per 720-minute shift, placed freely. One *technician* performs scheduled maintenance (75 minutes per diffusion furnace per day; 30 and 120 minutes per shift-window at lithography and implantation respectively), and implantation machines *fail* during processing — time between failures uniform on  $[1440, 4560]$  minutes of processing time, repair uniform on  $[360, 480]$  — destroying the lot in process.

The capacity arithmetic that organizes everything downstream: per lot, lithography is occupied  $(10 + 55 + 10) + (10 + 10 + 10) = 105$  minutes on its single machine; diffusion  $95 + 105 = 200$  minutes spread over two furnaces (batch handling divided by three); and implantation  $60 + 80 = 140$  minutes over two machines. At 84 lots/week the three stations run at 87.5%, 83.3%, and 58.3% respectively — before setups and maintenance, and the margin shrinks from there. The single-machine lithography station is the bottleneck, and the line's deterministic fluid capacity is  $10080/105 = 96$  lots/week, lithography-pinned (Figure 17.1).

## 17.2 The model as rows

The declared model is a set of database rows — the Geolog substrate of chapters 3–5 — over the System vocabulary of chapter 12. The ground layer is nothing but the tables of the previous section, recorded as entities with attributes, in the entity-per-row idiom:

```

Station : Set;
Station/name      : Station -> Str;
Station/machines  : Station -> Nat;
Station/bufcap    : Station -> Nat;
Station/batch     : Station -> Nat; // machine-input batch size (1 = none)

// st_diff: "diffusion" 2 18 3; st_impl: "implantation" 2 12 1;
// st_lith: "lithography" 1 12 1

RouteStep : Set;
RouteStep/idx      : RouteStep -> Nat;
RouteStep/station  : RouteStep -> Station;
RouteStep/recipe   : RouteStep -> Str; // "low" | "high"
RouteStep/proc     : RouteStep -> Q; // minutes
RouteStep/load     : RouteStep -> Q; // handling; the machine is held
RouteStep/unload   : RouteStep -> Q;

// (1, diff, low, 225, 20, 40) (2, impl, low, 30, 15, 15)
// (3, lith, low, 55, 10, 10) (4, impl, high, 50, 15, 15)
// (5, diff, high, 255, 20, 40) (6, lith, high, 10, 10, 10)

SetupRule : Set; // lithography, sequence-dependent
SetupRule/typeChange : SetupRule -> B;
SetupRule/stepChange : SetupRule -> B;
SetupRule/minutes    : SetupRule -> Q;
// (t,f,5) (f,t,10) (t,t,12) (f,f,0)

FailRule : Set; // implantation only
FailRule/tbflLo : FailRule -> Q; FailRule/tbflHi : FailRule -> Q;
FailRule/repLo : FailRule -> Q; FailRule/repHi : FailRule -> Q;
// time between failures 1440..4560 (processing time); repair 360..480

```

Rows are data rather than code: nothing about these declarations commits to a decomposition, and both of the readings below compile their dynamics from the same ground tables.

The *monolithic reading* renders the whole fab as one mechanism — an SDCPN, and the class name is a parts list for exactly this specification, worth unpacking on first contact. *Places* hold coloured tokens: waiting lots (coloured by product type, release time, and first-visit machine), machine phases (setup, loading, busy, awaiting unload, unloading, maintenance, down), personnel and transporter phases, and a sink of finished lots. *Dynamically coloured*: every duration in the tables is a token colour in motion — a timer drifting at  $\dot{\tau} = 1$  inside its phase and forcing a jump at its threshold — so time lives in the colours, never in a side-calendar. *Stochastically*: implantation’s failure and repair durations are random firing measures on the reset, drawn from the declared uniform family — and nothing else in this specification is random. The transitions carry the guards and firing laws, and the thesis’s own release idiom — per-type signal generators firing at constant intervals — is the timer-mechanism idiom, so the model’s front door needs no vocabulary at all beyond what chapter 12 already provides. The net’s interface exposes *aggregate* coordinates only: counts and pools — work in progress, per-step queue lengths, busy-machine counts per station, a completed-lot count, and a pool summing realized flow times.

Three features of the specification look, at first sight, like they should resist an aggregate-count rendering. None of them does, and each dissolution is informative.

**Batching survives count coordinates.** A diffusion load consumes three same-step lots subject to joint colour constraints (at most one test lot; high-step batches commercially pure). At the aggregate stratum, “a feasible batch exists” is a finite disjunction of count inequalities — for the high step, with  $k_A, k_B, k_{\text{test}}$  the

per-type queue counts,

$$k_A \geq 3 \vee k_B \geq 3 \vee (k_{\text{test}} \geq 1 \wedge k_A \geq 2) \vee (k_{\text{test}} \geq 1 \wedge k_B \geq 2),$$

and the reset decrements the counts by the chosen composition. *Which* three lots form the batch is invisible to every aggregate observable, by exchangeability. What the guard does add is a *choice* among feasible compositions — which belongs to dispatch, below, not to a new row species. Machine dedication for test lots is likewise just a colour coordinate (the first-visit machine), with a mismatch guard on the second visit.

**FIFO dissolves into class-level dispatch.** The count grammar deliberately has no expression for “the  $i$ -th token,” so first-in-first-out order within a colour class is structurally invisible — and for this model the invisibility is semantically exact, not a compromise: service law depends only on a lot’s type and step, so permuting same-colour lots moves no count trajectory, no utilization or throughput observable, and (by Little’s law, which relates only means) not even mean flow time. What FIFO *does* determine that matters is which colour class a freed machine serves next — and that residue is a dispatch policy: the index of every certificate in the fluid lane (section 17.6), and a freedom the combinatorial certificates of section 17.5 instead quantify over — adversarially, every policy at once. The dissolution does have a boundary: per-lot flow-time *distribution tails* do depend on within-class order, so a specification demanding per-lot quantile bounds would reopen the question (an age coordinate — still a colour — is the natural extension).

**Simultaneity is the norm, not a corner case.** Every duration in the tables is an integer number of minutes measured from a common origin, so distinct timers reach their thresholds at the same instant constantly — every shift boundary fires off-block guards on three resources at once. The declared model therefore carries explicit serialization data for simultaneous forced jumps (a priority order on transition families, with commutation as the alternative discharge), and the reference simulator’s deterministic tie-break — time, then declared family order, then entity identity — is that serialization made executable. Two correct simulators that disagree under tie-reordering are answering the commutation question the serialization row exists to settle.

## 17.3 The compositional reading

The monolithic reading is complete and faithful, and it deliberately gives up the one thing the composition axis exists to buy: decomposition. A per-station argument, a per-resource invariant, a certificate that transports across a boundary — all of these want the fab as a *composite*: release generators, three station mechanisms, two operators, a technician, a transporter, and a sink, wired together. The seams come in four kinds — directed lot handoffs, resource-pool memberships, fused event ports, and the dispatch freedom the fusions create — and the pattern inventory they draw on is chapter 13’s.

What the seams are, mathematically, pattern by pattern:

**Directed lot flow.** A station’s completions, as a counted output, wired into the next station’s arrival input — token flow carried as signal flow, with the lot’s colours riding as a tuple. This is ordinary directed wiring. It has one limitation worth naming: conservation (a lot leaves diffusion if and only if it arrives at implantation) is a semantic property the wire itself does not state; in the compositional reading it is recovered where it matters, as the pool pattern’s invariant.

**Resource pools.** Operator 1 must be exclusively held by either a lithography task or an implantation task, never both; so must the technician, across three stations, and the transporter, across every handoff. Continuous signal-sharing carries none of this — there is no signal, only a conserved unit. The resource-pool pattern of chapter 13 renders each resource as a pool with *take/give* ports and a crossing discipline (each user declares which of its modes count as holding the unit), and its payoff is a theorem rather than a convention: for a well-formed pool share, every jump of the composite preserves

$$k_{\text{pool}} + \sum_{u \in \text{users}} \#H_u = C,$$

the count-row conservation invariant — free to the checker, since it is a linear invariant of the count coordinates. For this model,  $C = 1$  four times over: each of the four pools is a single human or vehicle.

**Event fusion.** “Load starts” is one event touching three parties: the machine (empty, set up), the buffer (a feasible batch present), and the operator (idle, at the station). The fusion pattern makes it one *joint jump*: enabling is the conjunction of the parties’ local guards evaluated on the pre-jump snapshot, each party writes only its own state (so reset commutation within the fusion is a theorem), and one party — here the station’s timer — owns the timing. Without fusion, every synchronized cluster collapses into a single mechanism, which is the monolithic reading by forced march; fusion is what lets the composite reading exist at all.

**Dispatch.** Fusion creates contention: a freed lithography machine with both step-3 and step-6 lots waiting; an idle operator with two stations bidding; a feasible-batch guard satisfiable by several compositions. Undispatched, the composite is nondeterministic, and the semantics resolves the freedom *demonically* — which is the correct default precisely because of Kumar–Seidman: “stable at this load, whatever resolves the contention” is typically false below capacity, and should be unprovable when it is false. A dispatch row installs a named policy — a chooser over the enabled-conflict set, reading aggregate observables. The two certificate families divide over exactly this freedom: each fluid certificate of section 17.6 is indexed by the (system, policy) pair, while the combinatorial certificates of section 17.5 leave the freedom demonic and hold under every resolution of it. The off-block placement freedom (operators schedule their own breaks) sits in the same slot: it is control data, not plant data.

**Distribution-valued resets.** Implantation’s failure and repair durations are uniform on rational intervals — noisy timer thresholds, neither deterministic resets nor exponential clocks. The declared form is a reset kernel from a named family (uniform[ $a, b$ ] here), consumed by the certification interface only through its exact rational moments. Fixed and uniformly-distributed durations make the count process semi-Markov; the timer-as-state discipline (the supplementary-variable construction) is what keeps the presentation Markovian in the extended state, so this is vocabulary, not new metatheory.

Here is where the two readings stand. The monolithic reading uses mechanism vocabulary alone — timers, coloured counts, guarded jumps; the combinatorial certificates of section 17.5 are stated against its token-population refinement (places as plates, lots as tokens), and the fluid certificates of section 17.6 against its aggregate projection. The compositional reading exercises the fusion, pool, dispatch, and reset-kernel patterns exactly as chapter 13 catalogues them; the fusion and pool patterns had been reserved as a future axis until this model arrived as their first customer and ended the deferral, and the composite row set is mid-flip from design to artefact — the fusion stratum’s producer surface has landed, the pool packaging has not, and section 17.7 draws that boundary where the code makes it concrete. Equally informative is what the model does *not* demand: no nesting, no mode-switched macro-locations (shifts are timer guards inside mechanisms), no credal wire content, and no delayed trace — until some future dispatch chooser reads *lagged* observables (yesterday’s throughput, smoothed work in progress), at which point the loop becomes observation-delayed feedback and lands in the delayed-trace pattern. A discrete-event benchmark exercises a largely disjoint slice of the vocabulary from the robotics and hybrid-dynamics exemplars the patterns were designed against, and it still lands on the catalogued inventory; that is evidence the inventory’s shape is holding.

One number quantifies what the composite seams own. The reference discrete-event simulation committed beside the model tables (semicon/minifab/sim in the experiments repository) implements the deterministic layers only — machines, batching, setups, dedication, first-in-first-out within colour — on an exact  $\frac{1}{17}$ -minute integer tick, and measures a deterministic-core throughput ceiling of 87 lots/week. The thesis’s full model — adding the personnel, transport, maintenance, and failure layers — measures a maximal FIFO throughput of 57.5 lots/week. The roughly thirty lots per week between those two numbers is carried by *exactly* the layers that live on the pool, fusion, and reset-kernel seams: a quantitative reading of how much of this line’s behaviour the composition patterns are responsible for.

## 17.4 The specification

What should be certified about a fab divides into two claim families, and the division is structural rather than stylistic: the two families quantify differently, are checked by different machinery, and fail differently.

The first family is *exact and pathwise*: claims about the discrete line itself, holding along every run, at



every lot count, under every resolution of the scheduling freedom. Conservation — machines neither minted nor lost; a lot census that nothing but a completed lot can decrease; a congestion measure that no firing increases; and hard progress — from any work-in-progress profile, a stated budget of transition firings that no scheduler can exceed. These are the apparent infinity “one claim per marking, one row per transition instance” collapsed to finitely many symbolic rows by the plate discipline (section 10.4), and section 17.5 certifies them. In the released specification vocabulary (chapter 14) every member of this family now renders. The safety members render as PAIways atoms over the model’s aggregate wires, and a worked render appears beside its certificate below. The hard firing-count member renders through the count-ceiling spelling,  $\text{SReturnLE}(B, \text{retFireCount}(R))$ : a sure bound on an event count was a catalogued deliberate absence of the language until the transition-count functional landed (section 14.7.7), and the witness class that entry names for the demand — structurally bounded resource-sharing net families — is instantiated by this model. The judgment stratum arrived in the same motion: the Rule Catalogue (chapter 15) now carries one plated-transfer rule per face, each concluding a formed  $S \models \varphi$  row — a surely-held invariant from one certificate citation per direction, and a roster firing-count ceiling from one gap-rowed certificate. What still separates this chapter’s rows from formed judgments is narrower than a missing former, and it is stated with the certificates (section 17.5.8).

The second family is *long-run*, and the claim class matters (section 2.1: these are the infinite-horizon-average rows of the taxonomy, and the class label travels with every result). Over the monolithic reading’s aggregate interface, with  $\text{wip}(t)$  the total work in progress and  $N(T)$  the completed-lot count, the specification atoms are:

- **stability**: there is a finite  $W$  with  $\limsup_{T \rightarrow \infty} \frac{1}{T} \int_0^T \text{wip}(t) dt \leq W$  — the line does not drown;
- **throughput**:  $\liminf_{T \rightarrow \infty} N(T)/T \geq \delta$  — the release rate is actually sustained;
- **cycle time**: mean flow time at most  $\varphi$  — which Little’s law,  $\bar{\varphi} = \bar{W}/\lambda$ , derives from the other two rather than asking anyone to certify it separately. Certify two, derive the third.

In the released vocabulary this family now splits down its middle, and the split is a fact about the language worth stating precisely. The stability atom *renders*: the long-run average is an accumulator of the specification vocabulary —  $\text{poAvg}$ , the Cesàro average of a state observable (section 14.5.6) — and the atom is its expectation cap,

$$\text{SReturnLE}(W, \text{poAvg}(\text{wip})),$$

the upper prevision of the averaged work in progress over every law the dispatched composite admits, with the residual freedoms (setup sequencing, allocation slack, tie-breaks) resolved adversarially — exactly the quantifier this family wants; the level faces (section 14.6.3) put threshold probes on the same functional, and the occupancy siblings render the same way: the long-run occupied fraction of a station is the average of an indicator observable, so “lithography is busy at least seven-eighths of the time, in the long run” is  $\text{SReturnGE}(7/8, \text{poAvg}(\cdot))$  over the busy-count indicator — the vocabulary’s own compositional spelling of occupation fractions. These renders form and build today (section 17.7 carries the live witness), and what stands behind a formed long-run claim is stated with the same care as everything else in this chapter: the average former’s certificate species — the gain/bias pair, a Poisson-equation drift row — is named by the rule catalogue and not yet claimed, so these are formed claims awaiting their battery, where the safety renders of the first family already sit on discharged certificates.

The throughput atom is the other half, and it now renders — as an envelope, which is the only honest shape for it.  $\liminf_{T \rightarrow \infty} N(T)/T$  as a bare long-run limit is deliberately not the vocabulary’s object — a limit claim carries zero finite-window content, so a consumer sized against it can be overwhelmed on any finite horizon — and the event-rate family’s primitive is instead the affine envelope from start (section 14.6.4). The throughput claim the list above records is the envelope’s lower face over the completion-transition roster, at the release rate  $\delta$  and a certified burst budget  $\sigma$ ,

$$\text{SRateGE}(\delta, \sigma, R_{\text{sink}}) :$$

along every admitted law, the expected completed-lot count dominates  $\delta t - \sigma$  at *every* horizon — which yields the displayed limit reading as a corollary ( $\underline{E}[N(T)]/T \geq \delta - \sigma/T$ ), never the other way around. What stands behind the formed claim is stated with the same care as the stability half’s: its certificate species — the lower-envelope drift rows, the Foster-Lyapunov shape whose potential’s forced descent pays out completions — is named by the rule catalogue and not yet claimed for this exhibit, so the throughput

render joins the long-run family's formed claims awaiting their battery. The argument the earlier edition recorded beside the vocabulary — census certified (section 17.5.4), drain rates bounded by the machine roster, a finite average-work-in-progress bound forcing  $wip(T)/T \rightarrow 0$ , the census pinning completion rate to release rate — is now the recipe for that certificate's potential rather than a measurement of an absence. And the cycle-time atom never wanted a former at all: Little's law derives it, as the list above already said.

Every claim of this second family is about a (system, policy) pair, never about the system alone. The undischarged claim — stable under *every* resolution of contention — is demonic, and for re-entrant lines it is typically *false* below capacity; the specification's refusal to let per-component stability compose across an undischarged boundary is correctness rather than caution. The first family, by contrast, needs no policy index at all — its claims are demonic by construction — and that is precisely why it can say nothing yet about sustained arrivals: a released lot *raises* every potential of section 17.5, and pricing arrivals is a different certificate species (the crossing budgets of the hybrid-process correspondence, section 12.16), deliberately not interchangeable with ranking rows. Today the arrivals story is the fluid lane's, the exactness story is the combinatorial lane's, and the frontier itself is the seam between them.

## 17.5 The combinatorial certificates

What has been certified about the discrete line, with no fluid scaling and no aggregate projection anywhere in the statement: the lot-flow core of the Mini-Fab — the exact Kempf process times, the machines at their true capacities (2, 2, 1), the full re-entrant route — as an SDCPN of the modelling catalogue's frame, a token-population System at its full combinatorial state space, carrying twelve plated certificates whose 150 footprint sequents are discharged in exact rational arithmetic, one row set covering *every* lot count, *every* scheduler, *every* run. The model is small enough to print whole, and this section does — the net on its own page in Petri notation (Figure 17.2), the classification tables beside it, and the certificate hypernets in plate notation (Figure 17.3); every constant below re-derives from the campaign committed beside the toolkit's semiconductor strategy, whose test battery replays the entire verification (chapter 19 for the producer stack; the strategy source and its battery ship in the toolkit repository).

### 17.5.1 The Mini-Fab as an SDCPN

The exhibit is declared in the SDCPN frame's own compositional reading — *places are Bags, arcs are patterns* (section 12.18) — and this subsection presents it the way the container carries it. Sixteen places, each a Bag of coloured tokens; as certificate subjects they are sixteen *plates* — symbolic token populations in the sense of section 10.4:

| places (as plates)                                  | count | tokens are                  | colour                          |
|-----------------------------------------------------|-------|-----------------------------|---------------------------------|
| $q_0$                                               | 1     | lots not yet released       | none                            |
| $q_1, \dots, q_6$                                   | 6     | lots waiting at step $k$    | none                            |
| $p_1, \dots, p_6$                                   | 6     | lots in process at step $k$ | remaining work $w \in [0, T_k]$ |
| $m_{\text{diff}}, m_{\text{impl}}, m_{\text{lith}}$ | 3     | idle machines               | none                            |

with  $T = (225, 30, 55, 50, 255, 10)$  the process times of section 17.1, read as unit-work quanta: a lot in process at step  $k$  carries the integer minutes of work it has left, in the declared colour box  $[0, T_k]$ . Nineteen Petri transitions, in five families, each declaring its clock species from the frame's own transition vocabulary:

| transition                    | count | firing                                                  | clock                          |
|-------------------------------|-------|---------------------------------------------------------|--------------------------------|
| release                       | 1     | $q_0 \rightarrow q_1$                                   | carved $[1, 1]$ , demonic seat |
| $\text{start}_k$              | 6     | $q_k + m_{\text{st}(k)} \rightarrow p_k$ at $w = T_k$   | carved $[1, 1]$ , demonic seat |
| $\text{tick}_k$               | 6     | $p_k$ at $w \geq 1 \rightarrow p_k$ at $w - 1$          | forced, dwell 1                |
| $\text{finish}_k$ ( $k < 6$ ) | 5     | $p_k$ at $w = 0 \rightarrow q_{k+1} + m_{\text{st}(k)}$ | carved $[1, 1]$ , demonic seat |
| ship                          | 1     | $p_6$ at $w = 0 \rightarrow m_{\text{lith}}$            | carved $[1, 1]$ , demonic seat |

The clock declarations are the SDCPN's own. Processing quanta ride the *forced* species at unit dwell (deterministic unit work); every scheduling choice — when to release, which waiting lot a free machine starts, when a finished lot moves on — rides a clock carved to the must-fire window  $[1, 1]$ , with the choice itself left to the demonic scheduler seat: timing owned by the clocks, choice by the seat. Nothing

else is pinned — no dispatch policy, no tie-break, no priority; both stochastic transition species stand available and unexercised, every firing law is Dirac, and no arc reads without consuming: the all-demonic, unit-dwell, Dirac corner of the frame, whose unfolding is the faithful ordering-and-counting embedding of an untimed net. Two derived time coordinates run through everything that follows, both read off these same clock rows: *counting time* — the unfolding’s step index, makespans in firings — carries every certificate statement from here to the carved clock, and *metric time* — the tick meter the carved windows define — re-prices the same rows on the wall clock (section 17.5.10). Whatever a certificate asserts below, it asserts against every scheduler at once.

Figure 17.2 draws the net whole, in Petri notation, initial marking and supervisor seat included; the tables that follow it print the classification: sixteen places as Bag populations, nineteen Petri transitions with every arc typed against the founding taxonomy of Everdij and Blom [71], the arc-type census with the pattern each element flows through, and the twelve certificates with their folds and transfer polarities. The rows of these tables are the rows of the emitted container — one table line per declared row, in the container’s own species vocabulary — so the isomorphism is checkable by diffing the page against the bytes, and the places where a declared row home is still a named successor are marked on the page rather than smoothed over. The pattern inventory the classification lands on is chapter 13’s: the destroy and create edges are the mode-switching machinery of section 13.3 in rendezvous with the transition’s timing seat (section 13.10), and the enabling-arc row — unexercised by the plant, exercised by the supervisor seat’s read-and-return wires (section 17.5.7) — is the per-tuple parameter-setting wire of section 13.1. The statics square behind the “mechanized” status entries is Problem 12.14, closed in the semantic-universe development.

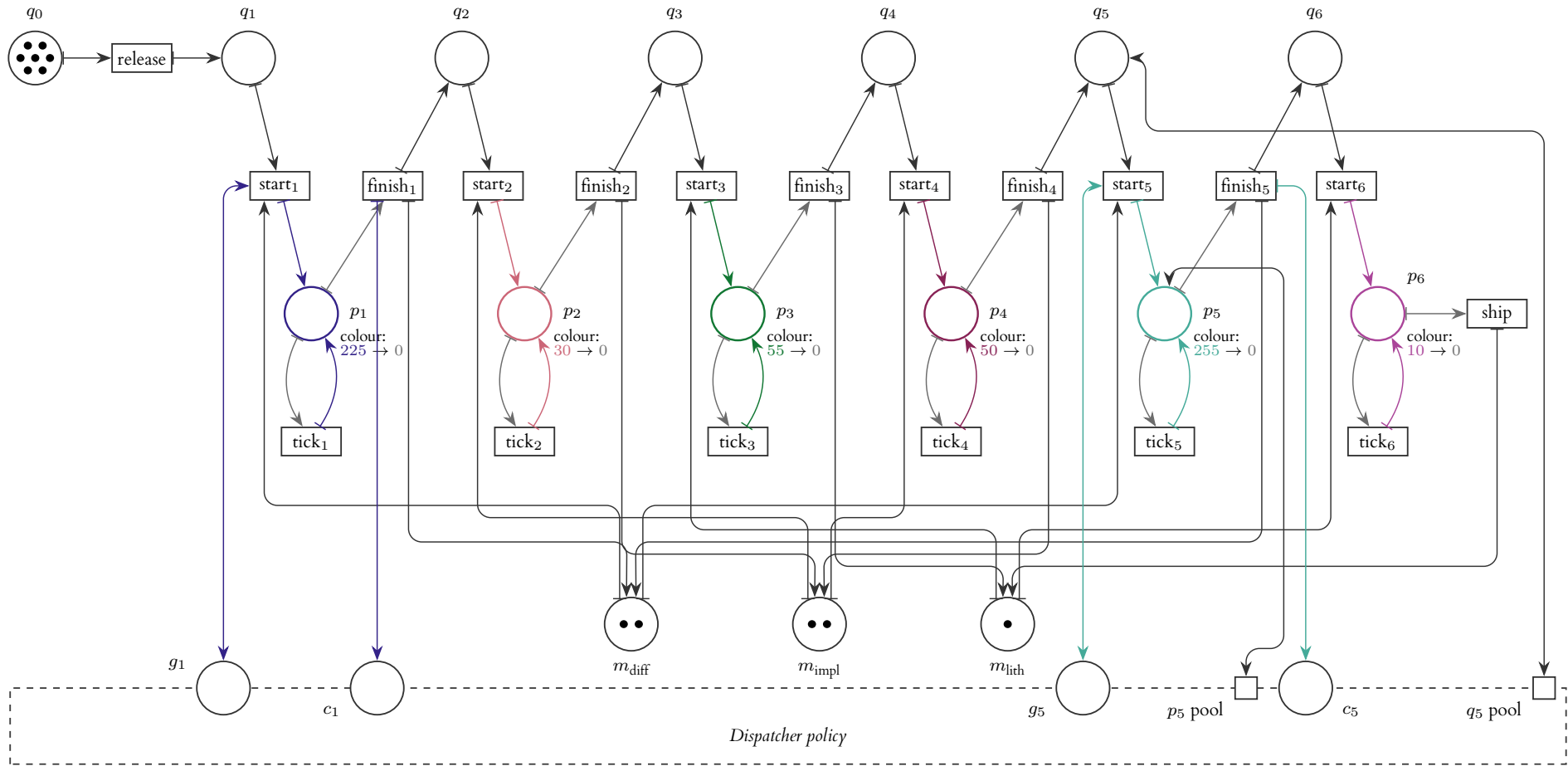


Figure 17.2: The Mini-Fab exhibit of this section, in Petri notation: circles are places, rectangles Petri transitions; every arc runs place-to-transition or transition-to-place, bar at its source, head at its target — 16 places, 19 transitions, 49 arcs, the census of the classification tables of section 17.5.1. The initial marking is drawn:  $N = 7$  lots pending in  $q_0$  (the hexagonal cluster;  $N$  appears nowhere else in the model), and  $(2, 2, 1)$  idle machines. Colour coding: each in-process place  $p_k$  declares its colour type, tagged *colour*:  $T_k \rightarrow 0$  with  $T_k$  the step's work quantum; arcs *producing* into  $p_k$  (its start; its tick's return, paying one unit of  $w$  per firing) wear that type's hue, and every arc *consuming* from an in-process place wears the shared grey of the terminal colour 0 — the value each countdown drains to, and the guard at which finishes and shipment fire; the tag numerals wear the same coding. Machine take and give-back ride the rectilinear arcs to the pool places at the bottom — the two-steps-per-machine-group wiring (1,5 diffusion; 2,4 implantation; 3,6 lithography) is the re-entrancy. The dashed box at the foot is the open system's supervisor hole — the one seat the plant leaves open, labelled by the policy that fills it — drawn at the boundary the synthesized dispatch policy of section 17.5.11 exercises, one grammar with Figure 17.4, which draws that filler seated: two gate places  $g_1, g_5$ , whose read-and-return enabling arcs (a head at both ends) gate the two diffusion starts — the seat's act wires; two firing counters  $c_1, c_5$ , one token per  $\text{finish}_1/\text{finish}_5$  firing; and two pooled read-and-return taps on  $q_5$  and  $p_5$  — four observables; with the gates, the strict subset of the full-dispatch interface (section 17.5.12) that  $\pi^*$  needs. Boundary wires wear the gated step's hue; the boundary is drawn unfilled — gate tokens are exactly what a filler mints. The seat's *first* filler (section 17.5.7) sealed this same seat at a narrower interface — one weight- $N$  cumulative-arrival tap on  $q_6$ , a gate on  $\text{start}_6$ , and the armed plate's one-shot key, its conservation pair  $k + g = 1$  the certified Boolean gate discipline. The hole apparatus is the comb's, beyond the plant census above.

**Places** (species HPlate)

| #  | plate             | colour     | Bag carrier                      | init |
|----|-------------------|------------|----------------------------------|------|
| 0  | $q_0$             | —          | $\text{Bag}(1) \cong \mathbb{N}$ | $N$  |
| 1  | $q_1$             | —          | $\text{Bag}(1)$                  | 0    |
| 2  | $q_2$             | —          | $\text{Bag}(1)$                  | 0    |
| 3  | $q_3$             | —          | $\text{Bag}(1)$                  | 0    |
| 4  | $q_4$             | —          | $\text{Bag}(1)$                  | 0    |
| 5  | $q_5$             | —          | $\text{Bag}(1)$                  | 0    |
| 6  | $q_6$             | —          | $\text{Bag}(1)$                  | 0    |
| 7  | $p_1$             | $[0, 225]$ | $\text{Bag}([0, 225])$           | 0    |
| 8  | $p_2$             | $[0, 30]$  | $\text{Bag}([0, 30])$            | 0    |
| 9  | $p_3$             | $[0, 55]$  | $\text{Bag}([0, 55])$            | 0    |
| 10 | $p_4$             | $[0, 50]$  | $\text{Bag}([0, 50])$            | 0    |
| 11 | $p_5$             | $[0, 255]$ | $\text{Bag}([0, 255])$           | 0    |
| 12 | $p_6$             | $[0, 10]$  | $\text{Bag}([0, 10])$            | 0    |
| 13 | $m_{\text{diff}}$ | —          | $\text{Bag}(1)$                  | 2    |
| 14 | $m_{\text{impl}}$ | —          | $\text{Bag}(1)$                  | 2    |
| 15 | $m_{\text{lith}}$ | —          | $\text{Bag}(1)$                  | 1    |

**The arc-type census, classified**

| SDCPN element                 | #  | pattern instance                             |
|-------------------------------|----|----------------------------------------------|
| ordinary arc, input face      | 25 | Bag destroy edge (rendezvous)                |
| ordinary arc, output face     | 24 | Bag create edge <sup>†</sup> (spawn)         |
| enabledness count             | 19 | monotone scope rows                          |
| colour guard slab             | 12 | guard content on the edge                    |
| enabling arc                  | 0  | per-tuple parameter-setting                  |
| inhibitor arc                 | 0  | excluded by formation                        |
| delay (rate-raced) transition | 0  | race clock, finite majorant                  |
| forced (deterministic delay)  | 6  | forced clock, dwell 1                        |
| immediate transition          | 0  | race clock, infinite majorant                |
| demonic scheduling            | 13 | carved clock $[1, 1]$ ; choice stays demonic |
| firing measure                | 19 | Dirac reset legs                             |
| initial marking               | 1  | root init rows                               |

**Petri transitions** (species Transition: grain, clock, attachment)

| #  | transition          | consumes [guard slab]     | produces [colour law]  | clock           | reset rows              |
|----|---------------------|---------------------------|------------------------|-----------------|-------------------------|
| 0  | release             | $q_0$                     | $q_1$                  | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 1  | start <sub>1</sub>  | $q_1, m_{\text{diff}}$    | $p_1 [w := 225]$       | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 2  | start <sub>2</sub>  | $q_2, m_{\text{impl}}$    | $p_2 [w := 30]$        | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 3  | start <sub>3</sub>  | $q_3, m_{\text{lith}}$    | $p_3 [w := 55]$        | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 4  | start <sub>4</sub>  | $q_4, m_{\text{impl}}$    | $p_4 [w := 50]$        | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 5  | start <sub>5</sub>  | $q_5, m_{\text{diff}}$    | $p_5 [w := 255]$       | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 6  | start <sub>6</sub>  | $q_6, m_{\text{lith}}$    | $p_6 [w := 10]$        | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 7  | tick <sub>1</sub>   | $p_1 [1 \leq w \leq 225]$ | $p_1 [w := w - 1]$     | forced          | tr.scale                |
| 8  | tick <sub>2</sub>   | $p_2 [1 \leq w \leq 30]$  | $p_2 [w := w - 1]$     | forced          | tr.scale                |
| 9  | tick <sub>3</sub>   | $p_3 [1 \leq w \leq 55]$  | $p_3 [w := w - 1]$     | forced          | tr.scale                |
| 10 | tick <sub>4</sub>   | $p_4 [1 \leq w \leq 50]$  | $p_4 [w := w - 1]$     | forced          | tr.scale                |
| 11 | tick <sub>5</sub>   | $p_5 [1 \leq w \leq 255]$ | $p_5 [w := w - 1]$     | forced          | tr.scale                |
| 12 | tick <sub>6</sub>   | $p_6 [1 \leq w \leq 10]$  | $p_6 [w := w - 1]$     | forced          | tr.scale                |
| 13 | finish <sub>1</sub> | $p_1 [w = 0]$             | $q_2, m_{\text{diff}}$ | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 14 | finish <sub>2</sub> | $p_2 [w = 0]$             | $q_3, m_{\text{impl}}$ | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 15 | finish <sub>3</sub> | $p_3 [w = 0]$             | $q_4, m_{\text{lith}}$ | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 16 | finish <sub>4</sub> | $p_4 [w = 0]$             | $q_5, m_{\text{impl}}$ | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 17 | finish <sub>5</sub> | $p_5 [w = 0]$             | $q_6, m_{\text{diff}}$ | carved $[1, 1]$ | exact tier <sup>†</sup> |
| 18 | ship                | $p_6 [w = 0]$             | $m_{\text{lith}}$      | carved $[1, 1]$ | exact tier <sup>†</sup> |

**Certificates** (species HCert; 150 footprint rows, species HFootprintRow)

| certificate          | claim family                    | fold             | features                          | footprint rows                                       |
|----------------------|---------------------------------|------------------|-----------------------------------|------------------------------------------------------|
| work-potential       | hard progress (ranking)         | sum              | $v = F$                           | $19 \times \text{gap } 1$                            |
| lot-count            | census upper bound              | sum              | 1 on lot plates                   | 18 safety + gap at ship                              |
| machines-diff-ub     | place invariant, $\leq$         | sum              | +1 on $m_{\text{diff}}, p_1, p_5$ | 6 safety                                             |
| machines-diff-lb     | place invariant, $\geq$         | sum              | -1 on $m_{\text{diff}}, p_1, p_5$ | 6 safety                                             |
| machines-impl-ub     | place invariant, $\leq$         | sum              | +1 on $m_{\text{impl}}, p_2, p_4$ | 6 safety                                             |
| machines-impl-lb     | place invariant, $\geq$         | sum              | -1 on $m_{\text{impl}}, p_2, p_4$ | 6 safety                                             |
| machines-lith-ub     | place invariant, $\leq$         | sum              | +1 on $m_{\text{lith}}, p_3, p_6$ | 6 safety                                             |
| machines-lith-lb     | place invariant, $\geq$         | sum              | -1 on $m_{\text{lith}}, p_3, p_6$ | 6 safety                                             |
| congestion- $\ell_2$ | congestion non-increase         | conj. pow. $p=2$ | $v = R$                           | $6 \times \varphi\text{-gap } 1 + 13 \text{ safety}$ |
| work-ceiling-max     | ceiling safety                  | max              | $v = R$                           | 19 safety (gap refused)                              |
| arrival-potential    | deadline ranking (reach $q_6$ ) | sum              | $v = D$                           | $16 \times \text{gap } 1 + 3 \text{ safety}$         |
| step6-budget         | post-arrival budget             | sum              | $v = F - D$                       | $3 \times \text{gap } 1 + 16 \text{ safety}$         |

**Places.** A place is a Bag of coloured tokens — remaining work  $w$  in its declared box for the in-process places, unit tokens (so: a counter) everywhere else. The machine plates are the resource-pool pattern in SDCPN dress: take and give are ordinary arcs, and the pool's conservation law is certified (the machine pairs above), not assumed.

**Transitions.** Every arc is an *ordinary* arc: each consumed entry is one input-face arc, each produced entry one output-face arc, every firing law Dirac — a deterministic colour polynomial in the consumed colours. There are no enabling arcs (every emitted selection row carries  $\text{read} = 0$ , a fact a reader can grep the container for) and no inhibitor arcs: enabledness everywhere is a monotone count-and-slab condition, so adding tokens never disables a step. Guard slabs are the footprint sequents' hypothesis slabs; every dwell is 1. <sup>†</sup>The six same-place ticks carry their reset as a declared scale row ( $\text{tr.scale}$ : image and reset polynomial in the container). The thirteen place-crossing transitions discharge their produced side directly in the exact-rational tier; the declared reset-row home arrives with the Bag reset-row presentation, staged for the next theory revision.

**The census.** Counts are arcs, transitions, or slabs. The zero rows read two ways: enabling arcs and both stochastic transition species are *available and unexercised* — this exhibit is the demonic-choice, carved-unit-dwell, Dirac corner of the frame (every clock carved to the window  $[1, 1]$ : timing owned by the clocks, choice by the seat), the discrete-time embedding of an untimed net, faithful for ordering and counting — while the inhibitor row is *unavailable by design*, warranted by the upward-closure theorem. Since every certificate quantifies over the demonic scheduler, its claims dominate every rate-raced refinement of the same scheduling freedom: the stochastic species, when they arrive, only resolve freedom the certificates have already priced. The founding taxonomy's conflict-resolution element is likewise priced rather than modelled: no priority or randomization rows exist, and every consequence holds by dominance over all of them at once.

**Certificates.**  $F$  is the remaining-firings dictionary ( $F_k = \sum_{j \geq k} (T_j + 2)$ ;  $v(q_0) = F_1 + 1 = 638$ ,  $v(q_k) = F_k$ ,  $v(p_k, w) = w + 1 + F_{k+1}$ , machines uncited) and  $R$  the remaining-work dictionary ( $R_k = \sum_{j \geq k} T_j$ ;  $v(q_0) = R_1 = 625$ ,  $v(q_k) = R_k$ ,  $v(p_k, w) = w + R_{k+1}$ ). Transfer polarity reads off the row profile: an all-gap sum certificate is a ranking (every firing pays, so runs are budgeted — 638 firings per lot, tight); a safety-rowed pair in both signs is a conserved place invariant ( $\#m_s + \#p_i + \#p_j = C_s$ ); the conjugated-power gap rows budget the ticks in  $\varphi$ -space; and the max fold carries safety only — a gap row on an idempotent fold is a formation error the vocabulary load-rejects, exercised by the mutant battery.

**Deadline.**  $D$  is the arrival dictionary:  $F$  with the step-6 remainder  $F_6 = T_6 + 2 = 12$  subtracted on every position before  $q_6$ , and 0 from  $q_6$  on —  $v_D$  vanishes exactly where a lot has entered  $q_6$ , so the fold's zero-set is the reach goal. The pointwise split  $D + (F - D) = F$  retells the ranking as a deadline argument: pre-arrival firings pay  $D$  (at most  $626N$  from  $N$  pending lots), post-arrival firings pay  $F - D$  (at most 12 per arrived lot), so along every maximal run every lot has entered  $q_6$  within  $T(N) = 638N - 12$  steps — counting time, the discrete-time unfolding's step index (the metric-time note below prices the same bound on the tick meter). Read as a Spec this is the deadline lower bound  $\bigwedge_{N \geq 1} [F \leq T(N)]$  (all  $N$  lots have entered  $q_6$ ) with demonic timing: in the Dirac corner every scheduler resolution is a point mass, the dictionaries quantify over the whole demonic seat, and the bound is tight — the schedule that defers the last finish<sub>5</sub> behind all other work attains it exactly (and typically-scheduled runs sit at it too: the step-6 tail is 12 firings against step 5's 255-tick pole). Three honesty lines the arithmetic enforces: the goal is *cumulative* arrival, never simultaneous occupancy ( $\#q_6 = N$  at once is demonically avoidable — start step 6 as lots arrive and  $q_6$  stays drained); the claim reads at the deadline wall (a record that stops existing is not late; the completion conjunct that upgrades the wall to full worst-case reach is the liveness face, priced separately); and the counting-time reading is primary but no longer alone — every clock in this encoding is carved to a finite must-fire window, and the metric-time note below prices the same bound with steps replaced by window widths. The judgment row binding this Spec to the container is the completion face's declared row home — not smoothed over: the deadline content shipped here is the two certificates' 38 footprint rows.

**Deadline, metric time.** With every clock carved to a positive finite must-fire window (dwell exactly one at these declarations), the counting deadline becomes a wall-clock statement: all  $N$  lots enter the finishing queue within  $638N - 12$  time units along every maximal schedule ( $626N$  under the latest-safe unlock) — probability 1, uniformly over the scheduler's remaining freedom. The certificate content is exactly the dictionary pair already presented; the transfer multiplies by the largest window ceiling (here 1), read off the model bytes by a must-fire census that refuses any idling member. Three clauses carry the reading, each at its stated tier: the dictionary rows are geolog-checked in-loop; the window-top law (a finite ceiling is a deadline) is the specification's declared clock semantics, its meter arithmetic owed at the proof tier; and maximality with deadlock-freedom-below-goal remain the same two argued clauses the counting statement already carries — the checkable species for the latter exists (the angelic-selection witness family) and awaits its active-region staging. The bound is firing-tight but not time-tight: simultaneously due windows fire in batches, and the round replay brackets the attained wall well below the certified one — structure, not slack in the rows.



**Open system and synthesis.** The plant opens into a comb: one supervisor seat whose declared interface reads the cumulative-arrival observable and drives the start<sub>6</sub> gate wire. A supervision-pattern filler closes it inside the model class — per gate, a gate place  $g$ , a one-shot key  $k$  (initial 1), and an unlock jump consuming  $k$ , reading  $q_6$  at weight  $N$  (returning what it reads) and minting  $g$ ; the gated Petri transition gains a read-and-return enabling arc, the census row the plant alone leaves at zero. Closed roster: 18 places, 20 Petri transitions, 14 certificates, 160 footprint rows — the plant's twelve certificates extend (the ranking pays the unlock's single firing through the key: total completing-run firings =  $638N + 1$ ), and the Boolean gate discipline is *certified*, not declared, by the conservation pair  $k + g = 1$ . Joint policy+certificate synthesis runs lexicographically: certified completion first (the gate that never opens ties the arrival optimum, then deadlocks with every lot in  $q_6$  shipping nothing — refused, starved plate named; so is the over-gated policy whose claimed horizon  $370N$  beats the optimum by starving arrival work out of its own constraint set), then minimize the certified horizon, where the least dictionary-certifiable  $T$  is an exact-rational linear program over dictionary space (gap rows on the admitted pre-goal roster, boxwise nonnegativity, goal pins, goal dominance) whose deterministic optimal vertex is the arrival dictionary  $D$  restricted to the closed roster. The winner is the latest-safe unlock (reopen exactly at the  $N$ th arrival): under it no pre-goal firing exists but the  $626N$  arrival-paying ones, so every schedule of the closed composite reaches all-arrivals at exactly step  $626N$  — against the open plant's  $638N - 12$ , a certified throughput-floor gain of  $12(N - 1)$  steps (+1.6% at  $N = 7$ ; +1.9% at the weekly  $N = 84$ : 52584 vs 53580). Honesty lines: the geolog-checked rows carry the *within*- $626N$  direction (gap rows in the loop); the *not-before* direction is the producer tier's exact-drop identities (each gap row's displacement equals its gap identically, both directions by Bernstein) plus the replay bracket, its in-loop row species a named absence; the  $626N$  floor over *all* fillers is plant arithmetic (626 lot-specific firings per arrival, one firing per composite step, policy firings only add); completion in every encoding is two-tier (ranking geolog-checked; enabledness below the ship goal argued); the seat's weight- $N$  observation read arc is the one arc-separation clause the flattened encoding cannot pass — carried as the staged observation stratum's named debt, never smoothed; and this armed-plate encoding has a native-read sibling (17 places, 20 Petri transitions, no key) whose completion carries one clause more — its unlock consumes nothing, so no ranking row can gap it, and unlock-fires-finitely-often rides as a stated convention — each variant citing its own roster and tier, neither borrowing the other's numbers.

Two structural findings the classification surfaces, both informative rather than embarrassing. First, the deliberate corner the clock table pins — all freedom demonic, all processing deterministic unit work, every firing law Dirac — is load-bearing for transport: the certificates are stated against that corner's adversary, which is the *largest* one, so every rate-raced or policy-refined elaboration of the same line — the stochastic species the frame holds ready, the failure and repair kernels of the fluid lane's fuller model — only resolves freedom these rows have already priced, and the classification doubles as a transport argument. Second, the machine plates exhibit the resource-pool pattern and the inhibitor exclusion in one stroke: mutual exclusion of a conserved unit needs only ordinary arcs plus the certified invariant (the machine-conservation pairs of section 17.5.4), and nowhere in a re-entrant line with true capacities does an absence test arise — the model never misses the excluded primitive, which is the exclusion argument of section 12.18 read off a working industrial subject.

### 17.5.2 One row per Petri transition

A certificate here is a plated certificate exactly as section 12.18 catalogues the species: a per-token *feature* — a polynomial in the representative token's colour block — on each cited plate; a commutative-monoid *fold* (sum, max, or the conjugated sums between them); and one *footprint sequent* per (certificate, Petri transition) pair, asserting that the firing's produced contribution is bounded by its consumed contribution — minus a pinned rational gap, on the rows that claim progress. Grammar equivariance is what makes the row finite: no expression names “the  $i$ -th token,” so the transition-level row is the family, at every marking, at every lot count. The twelve certificates below carry 150 footprint sequents in all — 19 each for the six whole-net certificates, 6 for each of the six machine-conservation certificates — and every sequent is a polynomial inequality of degree at most two over the declared colour boxes, discharged in the producer's exact tier by Bernstein-coefficient nonnegativity, in exact  $\mathbb{Q}$ , with no floating-point number representable in the pipeline. Coverage (every transition that may touch a cited plate carries a row), floor rows (gap-carrying folds bounded below), gap positivity, and scale-parameter conformance are checked in the same pass. The certificates themselves are finite drawn objects — plated feature cells, folds, four shapes carrying the whole roster — and Figure 17.3 draws them in plate notation.

### 17.5.3 The work potential, and the run-length budget

The flagship certificate is a shaped potential over the sum fold. Its feature dictionary is “remaining firings”: with  $F_k = \sum_{j \geq k} (T_j + 2)$  the firings a lot still owes from queue  $k$  onward (one start,  $T_j$  ticks, one finish per remaining step), assign

$$v(q_0) = F_1 + 1, \quad v(q_k) = F_k, \quad v(p_k, w) = w + 1 + F_{k+1}, \quad v(m) = 0.$$

Every one of the nineteen transitions carries a gap row with gap 1, and each is one line of arithmetic: a release moves a token from  $F_1 + 1$  to  $F_1$ ; a start consumes  $F_k = T_k + 2 + F_{k+1}$  and produces  $T_k + 1 + F_{k+1}$ ;



a tick pays one unit of  $w$ ; a finish consumes  $1 + F_{k+1}$  and produces  $F_{k+1}$ ; the final shipment consumes 1 and produces nothing the dictionary counts. Coverage is total, the floor is nonnegativity of the features, and the transfer consequence (section 12.18) is the specification's hard progress claim: **along every run from any marking  $m_0$ , the total number of transition firings is at most the fold  $V(m_0) = \sum v$  —** uniformly in the lot count and in the scheduler. At the Kempf constants the per-lot budget is

$$1 + \sum_{k=1}^6 (T_k + 2) = 1 + 625 + 12 = 638$$

firings, so  $N$  pending lots drain in at most  $638N$  firings — 53,592 at the nominal weekly release of 84 lots. And because this particular dictionary telescopes *exactly* — every transition drops the fold by exactly one — the bound is tight: every complete drain attains it, which the independent replay below measures at every tested lot count. This is the claim the fluid relaxation structurally cannot state (a continuous level supports no firing count), just as the arrivals-and-load claims of section 17.6 are the ones this certificate deliberately does not make.

The budget now has its render in the released vocabulary: for a run from marking  $m_0$ , the claim is the count ceiling  $\text{SReturnLE}(V(m_0), \text{retFireCount}(R))$  with  $R$  the full nineteen-transition roster (section 14.7.7), and the catalogue's ranking-face plated-transfer rule concludes exactly that judgment from one gap-rowed certificate — this one. What stands between this render and a formed row is recorded in section 17.5.8.

### 17.5.4 Conservation

Three certificate groups carry the bookkeeping claims, each over the sum fold with constant features.

*Lot census.* Feature 1 on every lot population (pending, queues, in-process; machines uncited): eighteen safety rows assert that no transition mints a lot, and the shipment transition carries a gap row — the census strictly drops exactly there. The transfer consequence: shipments along any run number at most the initial lot count. The dual lower bound is deliberately *not* certified, because it is false — lots genuinely leave at shipment — and the refusal is the honest content: this vocabulary cannot be talked into asserting conservation of a quantity the model does not conserve.

*Machine populations.* For each station, a pair of sum certificates — features  $+1$  and  $-1$  on the idle plate and the station's two in-process plates, safety rows on the six incident transitions — pins both directions of

$$\#m_s + \#p_i + \#p_j = C_s, \quad (C_{\text{diff}}, C_{\text{impl}}, C_{\text{lith}}) = (2, 2, 1) :$$

the classical place invariant, as certificate rows: a start takes an idle machine and opens an in-process slot, a finish does the reverse, a tick touches neither, and no other transition can touch the three plates at all — machines are neither minted nor lost, under any scheduler, at any lot count. In the specification vocabulary this pair renders as  $\text{PAlways}(\text{SAnd}(\text{SAtomLE}(C_s, \kappa_s), \text{SAtomGE}(C_s, \kappa_s)))$  for the station's population observable  $\kappa_s$  — a const-1 aggregate wire over the three cited plates, the same row species as the queue-length wires below; the judgment row that consumes the render is the catalogue's safety-face plated-transfer rule, whose conclusion is this conjunction under the sure cap  $\text{SProbGE}(1, \cdot)$  and whose premise shape — one certificate citation per direction — is this pair's own (section 17.4).

### 17.5.5 The bottleneck pair: two folds, one dictionary

The next two certificates share one feature dictionary — *remaining total work*: with  $R_k = \sum_{j \geq k} T_j$  (so  $R_1 = 625$ ), assign  $v(q_0) = R_1$ ,  $v(q_k) = R_k$ ,  $v(p_k, w) = w + R_{k+1}$  — and differ only in the fold, which is exactly the point.

*Congestion,  $\ell_2$ .* The conjugated-sum fold at  $p = 2$  (the scale-parametric family of section 12.18, strictly monotone at every finite  $p$ , interpolating from sum toward max). Read in  $\varphi$ -space, per that section's normative discipline — the rows constrain  $\sum_t v_t^2$ , and the checker never computes a root: every tick transition carries a gap row with  $\varphi$ -space gap 1 (from  $w \geq 1$ , the drop  $(w + R)^2 - (w - 1 + R)^2 = 2(w + R) - 1 \geq 1$ ), and every other transition carries a safety row, because the dictionary is arranged so that moves preserve remaining work exactly: a start converts  $R_k = T_k + R_{k+1}$  into  $w + R_{k+1}$  at  $w = T_k$ , a finish converts  $0 + R_{k+1}$  into  $R_{k+1}$ , and the shipment retires  $R_7 = 0$ . The even-power order-preservation

side condition is the nonnegativity of the feature range on the declared boxes — discharged, not assumed. Consequences: the  $\ell_2$  work-congestion measure never increases along any run, and processing quanta number at most the initial  $\varphi$ -mass  $\sum v(m_0)^2$ . Stated with the modulus honesty the modelling chapter demands: the  $\varphi$ -space budget is loose in aggregate units (at the weekly release the 52,500 ticks of the drain sit far below the initial  $\varphi$ -mass of  $84 \times 625^2 = 32,812,500$ ), which is exactly why the  $\varphi$ -space statement — not a root of it — is the certified one; the sharp tick count is the work potential's job, one certificate over.

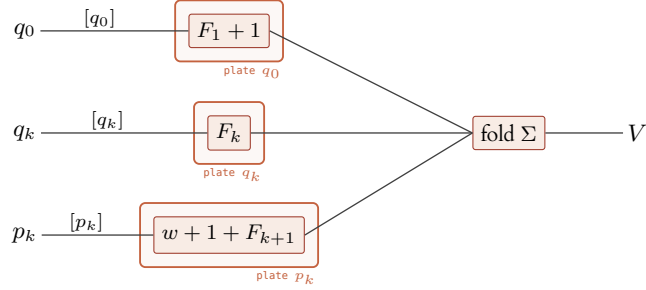
In the released specification vocabulary the non-increase claim has a worked render today. The emitted aggregate wires include the  $\varphi$ -space work masses of the six in-process populations and the six queue lengths, so the congestion observable is the linear read-out

$$s_2 = \sum_k \text{l2work}_{p_k} + \sum_k R_k^2 \cdot \text{len}_{q_k} + R_1^2 \cdot \text{len}_{q_0},$$

and for a run from a marking with congestion value  $S_0$  the render is  $\text{PAlways}(\text{SAtomLE}(S_0, s_2))$  — formation per chapter 14, the atom's polynomial canonical over the wire coordinates.

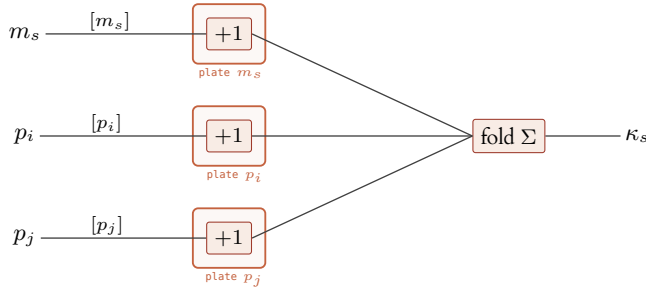
*Ceiling, max.* The same dictionary under the max fold: every transition a safety row, no gap row anywhere — the maximum remaining total work over lots never increases. The asymmetry with the congestion certificate is enforced, not stylistic: section 12.18 establishes that idempotent folds are progress-incapable in every scale (an untouched dominating token absorbs any local drop), so a gap row on a max certificate is a *formation error*, and the vocabulary load-rejects it — the constructor refuses the row, and the mutant battery below exercises the refusal. Safety is what a max fold can carry, and safety is what this certificate claims.

(a) work-potential — hard progress (ranking); fold  $\Sigma$ ; dictionary  $F$



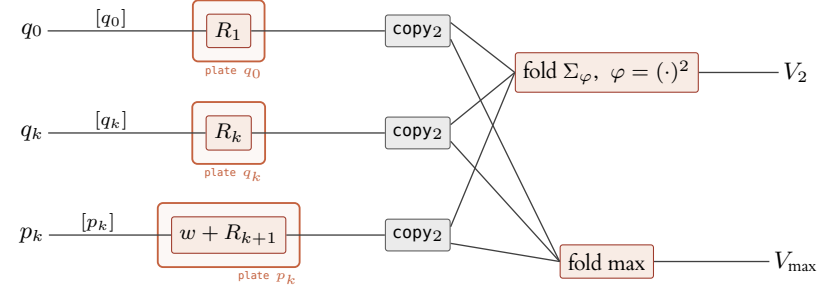
(one  $q_k$  and one  $p_k$  row per step,  $k = 1, \dots, 6$ ; all plate indices bound by the fold)  
 $F_k = \sum_{j \geq k} (T_j + 2)$ , the firings a lot still owes;  $v(m) = 0$ : machines uncited. One gap row per transition, gap 1 — 19 rows; along every run, from any marking  $m_0$ , total firings  $\leq V(m_0)$ , tight at 638 per lot.

(b) machine conservation — place invariants; fold  $\Sigma$ ;  $\pm$  twins



the  $-1$  twin certifies the reverse inequality, pinning  $\kappa_s = \#m_s + \#p_i + \#p_j = C_s$  under every scheduler, at every lot count;  $(s; i, j) \in \{(\text{diff}; 1, 5), (\text{impl}; 2, 4), (\text{lith}; 3, 6)\}$  with  $(C_{\text{diff}}, C_{\text{impl}}, C_{\text{lith}}) = (2, 2, 1)$  — 6 safety rows per certificate, 36 in all.

(c) the bottleneck pair — two folds, one dictionary  $R$



$R_k = \sum_{j \geq k} T_j$ , the work a lot still owes ( $R_1 = 625$ ); rows repeat per step as in (a). Congestion- $\ell_2$  (upper fold): obligations stated in  $\varphi$ -scale —  $\sum v^2$ , no root ever computed, verdicts transporting monotonely — 6  $\varphi$ -gap rows (the ticks) + 13 safety rows. Work-ceiling (lower fold): safety rows only (19); its gap row is *refused at load* — a gap row on an idempotent fold is a formation error, not a lint (section 12.18).

Figure 17.3: The exhibit's certificate hypernets, in plate notation. Each tinted enclosing box is a functorial box declared per node: the enclosed feature cell is applied once per token of the plate named at the box corner (its whiskered lift, section 10.4), and its wires carry the plate's symbolic extent, marked  $[q_k]$ ,  $[p_k]$ ; a fold node consumes the plated wires into one scalar. The drawn artefact is therefore a single finite object denoting a function of the whole variable-dimensional marking — the grammar has no expression for “the  $i$ -th token,” which is why one footprint row per (certificate, transition) pair covers every instance at every marking (section 17.5.2). The sum fold is contraction with the plate indices bound; constant features compile to the standard count wire, so (b) reads  $\kappa_s = \#m_s + \#p_i + \#p_j = C_s$ . Four shapes carry all twelve certificates of the classification tables (section 17.5.1): (a) the ranking; (b) the six machine invariants; (c) the two folds sharing the remaining-work dictionary  $R$ ; the lot census is (b)'s wiring with feature 1 on the nine lot populations ( $q_0, q_k, p_k$ ; machines uncited), 18 safety rows and a gap row exactly at shipment; and the deadline pair (section 17.5.6) is (a)'s wiring twice more, its dictionary split pointwise into  $D$  and  $F - D$  (16 gap + 3 safety, and 3 gap + 16 safety). Row total:  $19 + 19 + 36 + 19 + 19 + 38 = 150$  footprint rows, the tables' census.

### 17.5.6 The deadline pair: the flagship dictionary, split pointwise

The last two certificates price a deadline, and they cost no new dictionary: they split the work potential's  $F$  pointwise. Let  $D$  be the *arrival dictionary* —  $F$  with the step-6 remainder  $F_6 = T_6 + 2 = 12$  subtracted on every position before  $q_6$ , and 0 from  $q_6$  on — so  $v_D$  vanishes exactly where a lot has entered  $q_6$ : the fold's zero-set is the reach goal. *Arrival-potential* carries  $D$  over the sum fold: sixteen gap rows with gap 1, and safety rows on the three transitions at or past arrival, which pay nothing. *Step6-budget* carries the complement  $F - D$ : three gap rows on the post-arrival transitions, sixteen safety. The pointwise identity  $D + (F - D) = F$  retells the run-length budget as a deadline argument — pre-arrival firings pay  $D$  (at most  $626N$  from  $N$  pending lots), post-arrival firings pay  $F - D$  (at most 12 per arrived lot) — so along every maximal run, every lot has entered  $q_6$  within

$$T(N) = 638N - 12$$

steps: counting time, the discrete-time unfolding's step index. As a specification this is the deadline lower bound  $P_{\geq 1}[F_{\leq T(N)}]$  (all  $N$  lots have entered  $q_6$ ) with demonic timing — in the Dirac corner every scheduler resolution is a point mass, and the dictionaries quantify over the whole demonic seat. The bound is tight: the schedule that defers the last finish<sub>5</sub> behind all other work attains it exactly, and typically-scheduled runs sit at it too, the step-6 tail being 12 firings against step 5's 255-tick pole — the independent replay below witnesses both dynamically.

Three honesty lines the arithmetic enforces, stated rather than smoothed over. The goal is *cumulative* arrival, never simultaneous occupancy:  $\#q_6 = N$  at once is demonically avoidable (start step 6 as lots arrive and  $q_6$  stays drained), so no certificate of this pair claims it. The claim reads at the deadline wall — a record that stops existing is not late; the completion conjunct that upgrades the wall to full worst-case reach is the liveness face, priced separately. And the deadline above is stated in counting time; its metric-time twin rides the carved windows the clock rows already declare — a dwell floor with a finite must-fire ceiling — under which these same two dictionaries price the same bound with steps replaced by ticks (section 17.5.10). The judgment row binding the deadline specification to the container is the completion face's declared row home — a named successor; the deadline content shipped today is the pair's 38 footprint sequents.

### 17.5.7 The open system: the policy synthesized jointly with its certificate

The deadline pair prices the line against every dispatch at once, and that universality is also the bound's ceiling:  $638N - 12$  is what no scheduler can exceed, not what a good one achieves. The specification section already named the shape of the next claim family — everything long-run is about a (system, policy) pair, never the system alone (section 17.4) — and here the exact pathwise family acquires the same index, as *syntax*: the plant opens into a comb. Same sixteen places, same nineteen Petri transitions, plus one *hole* — the dashed box at the foot of Figure 17.2, a first-class term of the System grammar carrying nothing but a declared interface: read the cumulative-arrival observable, drive the start<sub>6</sub> gate wire. The hole is not a figure of speech; the emitted container carries the comb's own constructor rows, and the claim that exactly one hole occurs is decided in the production loop by an occurrence walk over those rows. What fills it is the supervision pattern of section 13.11, arriving exactly as that entry promised: the filler is an ordinary jump-only System owning a gate place, the gated transition carries a weight-1 read-no-consume enabling arc — the census row the plant alone leaves at zero — and the closed composite is again a coloured net of the same class. Supervision is composition inside the model class, so filling the hole is no semantic ceremony: it is one relation between three System terms, and the closed composite re-enters the same certificate machinery as the plant itself.

The synthesis is joint: policy and certificate are searched for together, the certificate machinery serving as the search instrument rather than a post-hoc audit. The objective is lexicographic, completion first. A candidate policy must first carry a certified completion claim — a ranking over the closed composite whose rows the checker re-derives — and only then competes on the certified arrival horizon, where the least dictionary-certifiable  $T$  is an exact-rational linear program over dictionary space: gap rows on the transitions the policy admits pre-goal, boxwise nonnegativity, zero pinned on the goal region, dominance everywhere a pending lot sits. Refutation is the improvement signal: exact replay of the closed composite under adversarial schedules either attains the certified horizon — closing the bracket pointwise — or returns a trajectory whose step ledger shows where the adversary spends its firings, which is precisely the gate the next candidate should own. On this model the search terminates at a theorem rather than a

timeout. The winner is the *latest-safe unlock* — hold  $\text{start}_6$  closed until the  $N$ th lot enters  $q_6$ , then open — and under it no pre-goal firing exists except the  $626N$  that pay the arrival dictionary, so every schedule of the closed composite reaches all-arrivals at exactly step  $626N$ . The linear program agrees from the other side: its deterministic optimal vertex is the arrival dictionary  $D$  of the deadline pair, restricted to the closed roster. Against the open plant's  $638N - 12$  the synthesized policy saves  $12(N - 1)$  steps — a certified throughput-floor gain of +1.6% at  $N = 7$  and +1.9% at the weekly  $N = 84$  (52584 against 53580) — and  $626N$  is the floor over *all* fillers: each arrival needs 626 lot-specific plant firings, firings interleave one per composite step, and policy firings only add, so no filler of this comb does better in counting time. The synthesized policy is not merely better; it is optimal, with both endpoints of the bracket certified objects.

The lexicographic order is not pedantry, and the mutant battery proves it the vivid way. Drop the completion clause and the “best” policy is the gate that never opens: it ties the arrival optimum — with the post-arrival work locked out, every firing pays the arrival dictionary, and all  $N$  lots reach  $q_6$  at exactly step  $626N$  — and then deadlocks with every lot in  $q_6$ , shipping nothing, a policy the deadline alone would score perfect. Its sharper sibling gates  $\text{start}_5$  instead and claims a horizon of  $370N$ , “beating” the true optimum by starving the arrival work out of its own constraint set: gap rows are demanded only on transitions the policy admits, so a policy that forbids the work escapes the rows that would price it. Both ride the battery with expectations committed in advance, and both are refused at the completion certificate, the starved queue named in the refusal. That is the completion-first discipline earning its place in the objective, and it is the synthesis story in miniature: the certificate engine does not merely bless the winner — it is what catches the degenerate candidates a deadline-only score would crown.

The campaign ships the closed composite in two encodings, and the difference between them is a certificate-tier fact, stated rather than smoothed. The *native-read* encoding adds one gate place and one unlock jump that reads  $q_6$  at weight  $N$  and returns what it reads — seventeen places, twenty Petri transitions, the roster the comb container carries — but its unlock consumes nothing, so no ranking row can gap it: a maximal run may spin the unlock forever instead of shipping, and “the unlock fires finitely often” rides as a stated convention, argued rather than checked, one clause more than the completion claim otherwise needs. The *armed-plate* encoding spends one more place to delete the clause: a one-shot key  $k$  with initial marking 1, consumed by the unlock — eighteen places, twenty Petri transitions — makes the extended ranking all-gap including the unlock (the completing run pays  $638N + 1$  firings, the key buying the single unlock) and turns the Boolean gate discipline from a declaration into a certified conservation pair  $k + g = 1$ , the machine-pair idiom aimed at the policy's own state. Each variant cites its own roster and its own tier; neither borrows the other's numbers. In both, the completion claim itself is two-tier — the ranking is geolog-checked; enabledness below the ship goal (some transition stays live while lots remain: in-process lots always tick or finish, machine conservation supplies an in-process lot whenever a station's idle plate is empty, queues are unbounded) is argued, the liveness face the deadline subsection already priced separately. The remaining honesty lines are the deadline pair's, inherited and extended. The geolog-checked rows carry the within- $626N$  direction; the not-before direction is the producer tier's exact-drop identities — each gap row's displacement equals its gap identically, both directions by Bernstein — together with the replay bracket, pointwise  $626N$  at every schedule tested, its in-loop row species a named absence (the lower-drift polarity the row vocabulary does not yet carry). The supervisor's weight- $N$  observation read arc fails exactly one clause of the supervision census — a policy arc reading a plant place; the packaged observation legs are the staged stratum that owns it — carried as a named debt in the container's own notes. And the filler-interface tie on a wired comb is refused producer-side, the substitution binding behind it argued until the checker-side evaluator lands. The two successors earlier editions of this chapter named here have both since landed against this comb, and the closing subsections of this section walk them: the metric rung — the carved clock, under which these same dictionaries price wall-clock deadlines and the dispatch seat is priced end to end (sections 17.5.10–17.5.12) — and the envelope arm, an uncertified dispatcher run inside a certified switching envelope whose downside the same family-level safety rows bound (section 17.5.13). That both were designed against this comb is the architectural point: the open system is one comb, and everything after it is fillers.

### 17.5.8 What checks these rows — today, and next

The trust story is staged, and each stage is named rather than blurred.

- **Discharged today, re-runnable from the public tree:** all 150 footprint sequents, coverage, floors,

gap positivity, and scale-parameter conformance, in the producer's exact tier (Bernstein-coefficient nonnegativity over the declared colour boxes, exact  $\mathbb{Q}$ ). The campaign — model, dictionaries, rows — is committed beside the toolkit's semiconductor strategy, and its test battery replays the whole verification, so these numbers are a cargo test away rather than a transcript's word.

- **Discharged today, in the production loop:** emitted certificate containers are sealed deterministically (double emission is byte-identical), and the engine-side theory provider discharges the canonicity claims — one per distinct feature polynomial, forty at this model — while refusing containers that claim nothing (a vacuous accept is structurally unavailable, and the battery below includes the probe). Since the evaluator family reached the coloured tier, the same in-loop walk also discharges the plated side-condition claims the container now carries: footprint on the thirty-six tick rows — the flagship's six, whose colour-typed feature dictionaries are read and decided from the container's own bytes, the lot count's six, the machine conservations' twelve, and the deadline pair's twelve — flow on all twelve certificates, and the five floor claims the sign analysis reaches, with zero refusals and zero failures on the real emission; a standalone checker binary consuming the same sealed bytes reports identical tallies.
- **Landed at a named fragment, refused loudly beyond it:** the theory demands its own side-condition families for every plated certificate — footprint, coverage, floor, flow, and parameter conformance — and the engine-side evaluator family that discharges them in-loop is now decided at the coloured place/transition fragment: same-place colour dictionaries, sum folds, displacement settled where it is constant or of single-signed even shape over the colour space, exact  $\mathbb{Q}$ . What still lies beyond is named per row rather than blurred: coverage and the thirteen place-crossing transitions' footprint rows await the produced-side reset-row presentation; the congestion ticks, the flagship's floor, and the conjugated power's parameter conformance await the declared colour-box rows (the Bernstein-over-box core that will decide them is written and exercised against exactly these shapes); the machine lower bounds' floors await a coupling route for negative features; and the max fold is an idempotent kind, outside the polynomial walk by design. A claim placed on any of these is refused with the unrealized case named — the battery below exercises two such refusals — so their discharge remains the exact tier's, replayed independently below, and the demand rows travel with the container either way: nothing is silently waived.
- **The warrant, and the judgment:** the run-length consequence rests on the transfer argument of section 12.18 — elementary telescoping, each dropped hypothesis refuted there by a finite countermodel. The judgment stratum has landed since this campaign first ran: the catalogue's plated-transfer pair concludes formed  $S \models \varphi$  rows for the safety and ranking faces, with the verification content normative in the evaluators' walks (section 17.4). The catalogue states what remains open, loudly: no Lean statement binds either rule node yet, so the formal lemma pack is still the named successor. The gap is not hidden in the artefact: the certificate carries rows and demands, and states its consequence as a consequence.
- **Jurisdiction, measured:** extraction toward the certified-binary checker reports this family's containers as carrying no content in that checker's fragment — correctly, and by design: plated rows are certificate-stratum vocabulary outside the fragment the machine-code soundness theorem binds, and their discharge route is the evaluator family and the lemma pack above, not the certified binary. The measured refusal is the correct answer, and it is the division of labour the stack's consumers were built to make explicit: one set of bytes, checkers that fail differently, none asked to vouch beyond its fragment.
- **The reproduction spine:** this campaign's committed home in the experiments repository, with a make target per number, arrives with the spine's uniform regeneration — the motion that has already reached the fluid lane next door (section 17.6) — and the chapter's numbers move in lock-step with it.

### 17.5.9 The mutant battery and the independent replay

Twelve deliberately corrupted campaigns, expectations committed before any run, each aimed at a specific gate and each in fact refused there: a broken gap (a tick that returns its token unchanged — the exact tier's footprint discharge fails); a gap row on the max fold (refused at construction — the formation error



above, exercised); a dropped footprint row (the coverage gate); a nonpositive gap (the gap-positivity gate); a conjugated power at  $p = 0$  (the parameter floor); a non-canonical feature polynomial (killed in the production loop by the engine provider's canonicity check — a real in-loop kill, not a producer-side courtesy); a container claiming nothing (refused in-loop as a vacuous accept); now that the claims themselves ride in-loop, three aimed at the coloured walk: a broken gap in the emitted bytes (its footprint claim *fails* in the loop — lying bytes under a live claim); a footprint claim placed on a row the walk cannot yet decide (refused, the missing colour-domain stratum named); and a floor claim on a negative-feature certificate (refused, the unbuilt coupling route named); and, with the deadline pair landed, two aimed at its rows: a corrupted arrival dictionary (the exact tier's footprint discharge fails at the mutated transition, the locus named), and a deadline gap inflated in the sealed bytes under a live claim (the in-loop footprint claim fails on the corrupted row — the checker refuses a lying deadline).

Beside the battery, an independent replay closes the artefact pair: an exact-rational drain simulation — written against the model tables, not the certificate code, with its expectations committed before the run — executes seeded demonic schedules at lot counts 1, 2, 5, and 84, and confirms: every drain terminates cleanly with all lots shipped and machine populations restored to (2, 2, 1); total firings equal  $638N$  *exactly* at every lot count and every schedule tested (the budget is tight, as the telescoping predicts); every firing satisfies every touched certificate row in its reading scale; the place invariants pin; the congestion fold is monotone with ticks inside the  $\varphi$ -space budget; every seeded schedule meets the deadline, with the arrival fold reaching zero exactly at the  $N$ th arrival; the adversarial schedule that defers the last finish<sub>5</sub> attains  $638N - 12$  *exactly* at every tested lot count (the deadline bound is tight dynamically, as the split predicts); and the broken-gap mutant violates its row at the *first* mutated firing — the dynamic kill agreeing with the static refusal. The replay rides the toolkit's workshop today and lands in the reproduction spine with the vendoring motion above.

### 17.5.10 The carved clock: the same rows in real time

The clock rows of section 17.5.1 support two readings, and everything so far in this section has used the first: *counting time* — makespans measured in firings, the discrete-time unfolding's step index — where total work is scheduler-invariant ( $638N$  firings, always: the work potential's telescoping) and *when* the work happens is the only nontrivial question. This subsection and the four that follow read the same declared clocks the other way — *metric time*, the tick meter — and push the open-seat story to its natural end: what the wall clock rewards, exactly how much, a policy that attains the optimum, a machine-checked sense in which “optimal” is the decision-theoretically right word, a deliberately untrusted policy run inside a certified envelope, and one certificate that spans lot counts from 7 to  $10^7$ . The subject is unchanged — the lot-flow core, with the model boundary stated at this section's end inherited whole.

The metric-time semantics is already in the declarations: every transition's clock is carved to the window  $[1, 1]$ , so each row dwells exactly one tick and fires at most once per tick — the per-row meter law; simultaneous instances resolve through the demonic scheduler seat, so the only freedom left is genuinely dispatch freedom. Both closed-composite encodings of section 17.5.7 reseal under this reading with row tallies identical to the counting-time containers — the certificates themselves never change.

Two transfers move the counting-time results onto the wall clock. First, upper bounds transfer by a *no-idle* argument: before the goal, a tick with an empty due set — no enabled transition instance whose one-tick dwell has elapsed — would freeze the state forever, so along every goal-reaching schedule each tick fires at least one row, and ticks-to-all-arrivals is bounded by pre-goal firings: at most  $626N$  ticks under the synthesized gate and  $638N - 12$  open, carried by the *same* geolog-checked rows plus two named glue clauses, argued — the meter/no-idle lemma and enabledness below the goal. Second, there is a wall-clock *floor* over all dispatch policies of any kind, supervisory or never-idle:

$$\text{ticks to all arrivals} \geq 255N + 371 \quad (21,791 \text{ at the weekly } N = 84),$$

because step 5's service tick is one row — at most one firing per tick, 255 firings per lot — behind an uncompressible 370-event first-lot pipeline fill, plus the final transfer into the arrival queue. The bound is Z3-verified below  $N = 8$ , and its machines-deleted variant is refused too (checked at  $N \leq 2$ ): machine capacity is not the binding constraint; the clock law and per-lot precedence are. (Tight at  $N = 1$ : the singleton trajectory takes exactly 626 ticks.)



Measured against that floor at  $N = 84$ , the dispatch levers separate cleanly. The synthesized arrival gate, whose counting-time value is certified (section 17.5.7), moves wall-clock arrivals 0.0%: greedy dispatch reaches all-arrivals in 22,019 ticks with the gate on or off — the gate’s value was always the certified worst-case floor, a guarantee over every schedule, not a typical-case speedup. Pull order is a  $1.83\times$  lever (an adversarial pull priority needs 40,406 ticks, +83.5%); and greedy itself sits 228 ticks above the floor. Real-time value lives almost entirely in the dispatch seat — which is why the optimum is a policy story, next, and why the seat then widens (section 17.5.12).

### 17.5.11 Optimal throughput, exactly

The exact optimum is known in closed form at small  $N$  and attained at every tested  $N$ :

$$T_{\text{opt}}(N) = 255N + 371 + d(N), \quad d = 0 \ (N \leq 2), \quad 2 \ (N = 3), \quad 4 \ (N \geq 4),$$

certified two-sided for  $N \leq 7$  by exhaustive Z3 search — satisfiability with an independently re-validated witness schedule at  $T_{\text{opt}}$ , unsatisfiability at  $T_{\text{opt}} - 1$  — and the +4 plateau is attained by an explicit supervisor family  $\pi^*$ , drawn seated in its interface as Figure 17.4 (hold the first few diffusion starts on one machine with two-tick handovers, then alternate pre-positioned; run the step-1 services back to back) at every tested lot count ( $N \leq 8, 12, 30, 84$ ): 21,795 ticks at the weekly count, within 4 ticks (0.018%) of the machine-free floor above, and *exactly* optimal in per-lot throughput at all  $N$  — 255 ticks per lot, matched from both sides. The attainment side is *container replay*, geolog-checked:  $\pi^*$ ’s explicit firing table replays against the sealed exhibit bytes to arrival at exactly 21,795, under the meter law, every hold accounted. The floor side now meets it exactly. The machine-free floor is *not* attainable for  $N \geq 3$  (Z3 refuses it outright), and the reason is mechanized — Lean-checked: a five-case machine-contiguity analysis of the first four diffusion entrants, showing any schedule faster than  $255N + 375$  leaves the third or fourth lot’s service without a contiguous machine window — and it *collapsed* under mechanization: the tick-meter forces two pigeonhole machine handovers, three of the five cases dying inside the first forcing. That lifts the certified floor to  $255N + 375$  for  $N \geq 4$  ( $255N + 373$  at  $N \geq 3$ ; the mechanized anchors match the Z3 optima exactly at  $N \leq 7$ , and the  $N = 3$  value  $d = 2$  is Z3-certified two-sided), which the replayed attainment meets exactly — the four-tick daylight between floor and attainment is closed,  $N$ -uniformly, and the mechanization explains why the plateau is +4: it is the double diffusion handover  $\pi^*$  saturates. In the same pass the counter/meter bookkeeping behind the machine-free floor landed as geolog-checked rows — this container family’s first checker-side clock-law rows.

Two consequences deserve their own sentences. *Deliberate idling is necessary for optimality*: for  $N \geq 3$  the best never-idle (work-conserving) policy needs  $255N + 455$  ticks — at the weekly count, 80 ticks behind  $\pi^*$ , with its never-idle law verified at every pre-goal tick of the container replay — and a measured greedy baseline runs 224 ticks behind; gating the *last* step moves arrivals 0.0%, while holding the *diffusion starts* recovers the full gap. The necessity is also mechanical: replaying  $\pi^*$  with holds disallowed is *refused* by the engine at its first delayed diffusion start. Where the supervisor holds is what the last 80 ticks are made of. And *the decision-theoretic reading is discharged, not gestured at*: with deterministic closed dynamics the belief state is a point mass, and imprecise-decision-theoretic selection at the widest-uncertainty locus provably collapses to makespan minimization — the reduction Lean-checked against the mechanized imprecise-decision-theory development, with a concrete refutation cell showing the collapse fails without determinism. “The synthesized policy is decision-theoretically correct” therefore means optimal throughput — the optimality receipts above discharge it, each at its stated grade, and the seat-regret face of the specification vocabulary (section 14.6.5) is where the language states exactly this shape: the regret of a filler against its own seat, here 0 at every tested lot count — two-sided by Z3 at  $N \leq 7$ , the Lean-checked floor met exactly beyond.

Figure 17.4 closes the subsection by drawing  $\pi^*$  as what it is: a filler for the supervisor seat, the Petri page’s dashed box grown to nearly the whole page, one enormous Petri transition inside — the policy’s firing kernel, its interior a plate-notation hypernet over the seat’s four observables. The page and its caption carry the policy’s entire content: no code, no lookup table, no appeal to the replay — two threshold cells and two piecewise-constant schedule decoders, readable in one look.

the plant of the Petri page, elided to the diffusion re-entrancy (steps 1 and 5 share the machine pool); everything ungated is omitted

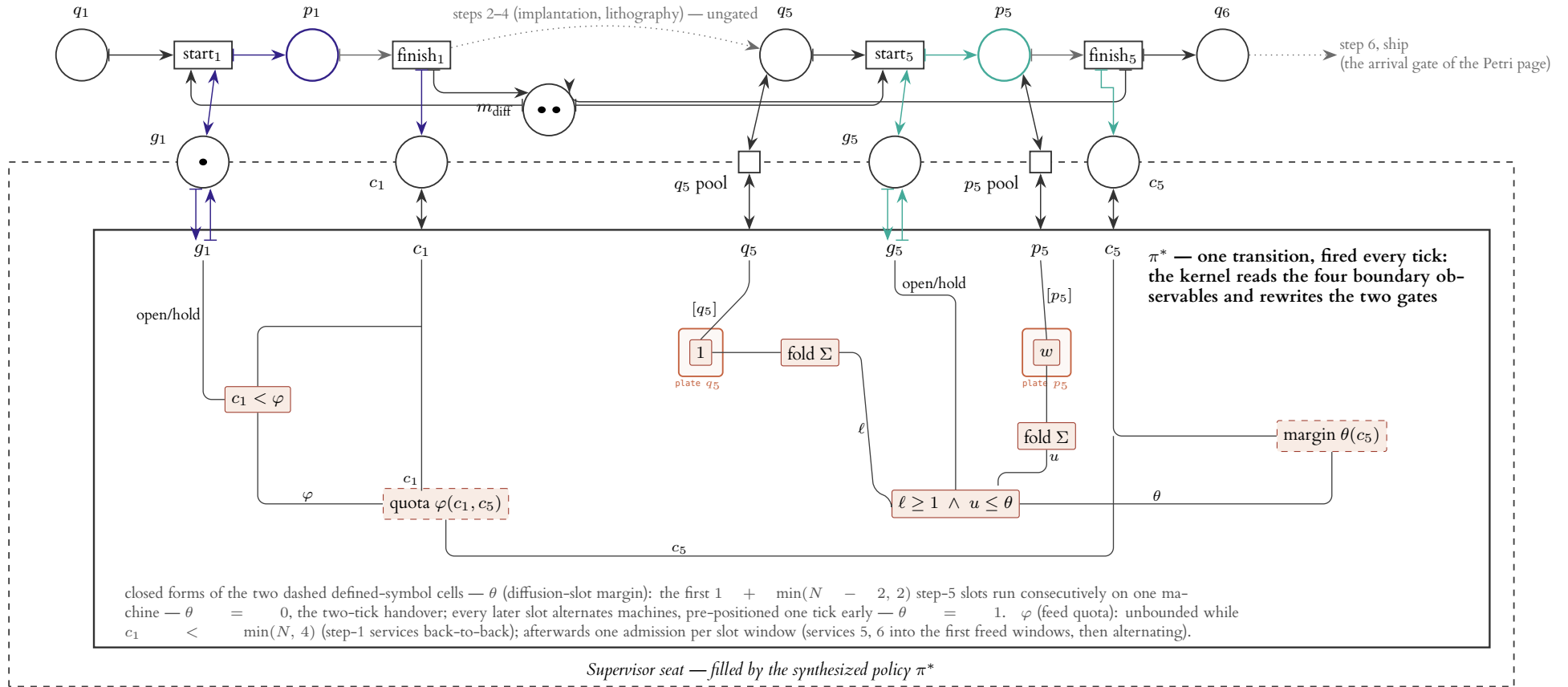


Figure 17.4: The synthesized optimal policy  $\pi^*$ , drawn as what it is: a filler for the supervisor seat. The dashed seat of Figure 17.2 is here nearly the whole page, and the plant survives only as the gated neighbourhood along the top — the diffusion re-entrancy, where  $\text{start}_1$  and  $\text{start}_5$  compete for the same two machines. Straddling the boundary, the seat's declared interface: the two pooled reads, the two firing counters, and the two gate places — a strict subset of the fifteen-observable, four-act full-dispatch interface of section 17.5.12, and the only wires  $\pi^*$  needs. Inside sits one enormous Petri transition: the policy's firing kernel, applied once per tick, reading the four observables and rewriting the two gates. Its interior is a hypernet in the plate notation of Figure 17.3: the pooled observables are two plated folds ( $\ell = \sum 1$  over  $[q_5]$ , the queue length;  $u = \sum w$  over  $[p_5]$ , the in-service residual — pre-positioning keeps at most the incumbent in  $p_5$ , so the sum reads its residual), the counters decode the schedule phase through the two dashed cells ( $\theta, \varphi$  — defined symbols, not primitive generators: piecewise-constant functions of the counters, closed forms in the box), and two threshold cells (region-membership) emit the gate bits. Holding  $\text{start}_5$  until  $u \leq \theta$  pre-positions each diffusion slot; holding  $\text{start}_1$  at the quota keeps a diffusion machine free for step 5 — deliberate idling, the one capability the never-idle class lacks. What this filler attains, each claim at its tier (section 17.5.11): all arrivals in  $255N + 371 + d(N)$  ticks,  $d = 0, 0, 2$ , then  $4$  — 21,795 at the weekly  $N = 84$ , on the certified all-policy floor exactly (attainment geolog-checked by container replay, every hold accounted; the floor itself Lean-checked); exactly optimal for  $N \leq 7$  by exhaustive Z3 search, two-sided. The best never-idle policy needs  $255N + 455$  — these two gates are worth the last 80 ticks, and replaying  $\pi^*$  with holds disallowed is refused at its first delayed diffusion start. Under deterministic closed dynamics, imprecise-decision-theoretic selection provably collapses to makespan minimization (Lean-checked) — so this page is also the decision-theoretic optimum. Initial marking drawn:  $g_1$  open ( $c_1 = 0 < \varphi$ ),  $g_5$  shut ( $\ell = 0$ ).

### 17.5.12 Widening the seat: full dispatch as a filler

The supervisor seat of section 17.5.7 reads one observable and drives one gate wire. That narrowness was an encoding shim, not a formalism limit: an interface always was a pair of port lists, and the same comb construction re-seals the same single seat with a *full-dispatch* interface — fifteen observables (six queue lengths, three machine-idle flags, six in-process work counts) and four act ports (three per-station pull decisions plus token selection) — around the unchanged plant. The closed net is the plant net, so the exhibit's twelve-certificate family binds every filler *a fortiori*, geolog-checked: the sealed full-dispatch container re-verifies with the full row roster, and the certificate never mentions the policy.

Two findings from exercising it. Token selection is a *second* dispatch freedom that determinism requires: a pull priority alone does not close the system; fixing both yields one trajectory per policy, verified by replaying residual-order permutations to identical traces. And across the 16-policy deterministic grid (8 pull priorities  $\times$  2 selection rules,  $N \in \{1, 7, 84\}$ ; Table 17.1), only the diffusion-station pull preference binds the outcome — the implantation and lithography columns of the table never move a number — while token selection swings roughly 7,000 ticks and *crosses* with pull preference; the best cell lands on  $255N + 455$  exactly at  $N \in \{7, 84\}$  — the work-conserving optimum of the previous subsection, consistent with the separation proved there, since grid policies never deliberately idle. Threshold-style gate supervisors provably cannot express per-station pull preference — argued: a gate place maintained by fixed-weight fused arcs is an affine functional of the marking, read through upward-closed thresholds, and a pull preference's acceptance region is a union no affine threshold carves. Dispatch is a genuine seat species, not sugar over gates — enablement carving and seat selection are different composition operations.

### 17.5.13 An untrusted policy in a certified envelope

The envelope certificates — safety, floors, the deadline — never read the policy: they quantify over every filler of the seat, which is why the policy may be *anything*, including a black box nobody should trust. (The per-policy rows of Table 17.1 do read the policy — but only as a replayed *trace*, re-checked row by row against the sealed container, so even the performance claims ask for no trust in the box, only in the replay.) A third closure of the same seat demonstrates exactly that: the open comb filled by an uncertified lookahead heuristic (a stand-in for a learned or LLM dispatcher) composed with a certified switching envelope — a decision-budget bench with a certified recovery region. In replay: the heuristic ran free, was never benched, completed, and attained the envelope's arrival bound  $638N - 11$  *exactly* at the tested  $N = 84$  — one tick above the open plant's  $638N - 12$ , the closed composite's own unlock firing — so the bound is witnessed tight, not merely sound; a deliberate idler hit the budget boundary and was benched, with recovery at exactly 1,009 firings; a rogue policy was benched after eight named in-loop refusals with the plant untouched; and disabling the bench stalls the line — the switch is load-bearing, not decorative. After all runs the sealed container re-verified *byte-identical*: the certificate is carried entirely by the envelope, geolog-checked, and no behaviour of the filler — competent, idle, or hostile — can touch it.

### 17.5.14 One certificate, every lot count

The deadline certificates of this section are stated once and instantiated at any lot count  $N$ : the sealed container's row obligations (footprint, flow, floor, canonicity) never mention  $N$  — they quantify over *all* markings — and  $N$  enters only as three integers in the container's own data (the initial lot count, the unlock guard's threshold multiplicity, and hence the instantiated horizon, which the reader recovers from the marking row by one dot product with the arrival dictionary). Table 17.2 instantiates, reseals, and re-verifies the pair of exhibits at lot counts spanning six orders of magnitude. The certificate's byte size moves only with the decimal width of those integers (six bytes across the whole sweep); the checker's wall time is flat at  $\sim 2$  ms; and the objects the claims quantify over grow combinatorially — at  $N = 10^6$  the model has at least  $2 \times 10^{38}$  reachable markings and at least  $10^{2 \times 10^7}$  distinct maximal schedules, and the checker never visits any of them.

**What is  $N$ -parametric and what is not — the honest split.** The *certificate* and the *checker* are  $N$ -parametric in the strong sense: the same 36 footprint rows (this container's share of the exhibit family's 150 footprint sequents), the same dictionaries, the same claim roster discharge at every  $N$  (identical tally vectors across the sweep), and at the landed weekly count the resealed container reproduces the landed

| dispatch policy                                                        |       |                   |       | ticks per arrival |       |                |       | certified optimality (%) |        |            |        |          |
|------------------------------------------------------------------------|-------|-------------------|-------|-------------------|-------|----------------|-------|--------------------------|--------|------------|--------|----------|
| $m_{\text{diff}}$                                                      | pulls | $m_{\text{impl}}$ | pulls | $m_{\text{lit}}$  | pulls | selection      | $N=1$ | $N=7$                    | $N=84$ | $N=10^6$   | $N=84$ | $N=10^6$ |
| ours: $\pi^*$ (synthesized holds — not never-idle)                     |       |                   |       |                   |       |                | 626.0 | 308.6                    | 259.5  | 255.000375 | 100.00 | 100.00   |
| $q_5$                                                                  | $q_2$ | $q_3$             |       |                   |       | most-advanced  | 626.0 | 320.0                    | 260.4  | —          | 99.63  | —        |
| $q_5$                                                                  | $q_2$ | $q_6$             |       |                   |       | most-advanced  | 626.0 | 320.0                    | 260.4  | —          | 99.63  | —        |
| $q_5$                                                                  | $q_4$ | $q_3$             |       |                   |       | most-advanced  | 626.0 | 320.0                    | 260.4  | —          | 99.63  | —        |
| $q_5$                                                                  | $q_4$ | $q_6$             |       |                   |       | most-advanced  | 626.0 | 320.0                    | 260.4  | —          | 99.63  | —        |
| $q_5$                                                                  | $q_2$ | $q_3$             |       |                   |       | least-advanced | 626.0 | 384.7                    | 343.2  | —          | 75.61  | —        |
| $q_5$                                                                  | $q_2$ | $q_6$             |       |                   |       | least-advanced | 626.0 | 384.7                    | 343.2  | —          | 75.61  | —        |
| $q_5$                                                                  | $q_4$ | $q_3$             |       |                   |       | least-advanced | 626.0 | 384.7                    | 343.2  | —          | 75.61  | —        |
| $q_5$                                                                  | $q_4$ | $q_6$             |       |                   |       | least-advanced | 626.0 | 384.7                    | 343.2  | —          | 75.61  | —        |
| $q_1$                                                                  | $q_2$ | $q_3$             |       |                   |       | least-advanced | 626.0 | 445.0                    | 397.1  | —          | 65.34  | —        |
| $q_1$                                                                  | $q_2$ | $q_6$             |       |                   |       | least-advanced | 626.0 | 445.0                    | 397.1  | —          | 65.34  | —        |
| $q_1$                                                                  | $q_4$ | $q_3$             |       |                   |       | least-advanced | 626.0 | 445.0                    | 397.1  | —          | 65.34  | —        |
| $q_1$                                                                  | $q_4$ | $q_6$             |       |                   |       | least-advanced | 626.0 | 445.0                    | 397.1  | —          | 65.34  | —        |
| $q_1$                                                                  | $q_2$ | $q_3$             |       |                   |       | most-advanced  | 626.0 | 448.6                    | 477.4  | —          | 54.35  | —        |
| $q_1$                                                                  | $q_2$ | $q_6$             |       |                   |       | most-advanced  | 626.0 | 448.6                    | 477.4  | —          | 54.35  | —        |
| $q_1$                                                                  | $q_4$ | $q_3$             |       |                   |       | most-advanced  | 626.0 | 448.6                    | 477.4  | —          | 54.35  | —        |
| $q_1$                                                                  | $q_4$ | $q_6$             |       |                   |       | most-advanced  | 626.0 | 448.6                    | 477.4  | —          | 54.35  | —        |
| certified floor, all policies (255N+375, $N \geq 4$ , met by $\pi^*$ ) |       |                   |       |                   |       |                | 626.0 | 308.6                    | 259.5  | 255.000375 | 100    | 100      |

Table 17.1: All seventeen policies of these subsections, benchmark-style (the synthesized policy itself is drawn as the seated filler in Figure 17.4): the sixteen never-idle full-dispatch cells (three binary pull preferences  $\times$  token selection; deterministic replay, verified by residual-order permutations) and the synthesized  $\pi^*$ , which is not a row of the grid because it deliberately idles — the one capability the never-idle class lacks, worth exactly the last 80 ticks. Read the parameter columns: only the diffusion column ever moves a number; the implantation and lithography preferences are provably irrelevant at this roster. Token selection crosses with pull preference (most-advanced wins under  $q_5$ , loses under  $q_1$ ). Units: ticks per arrival ( $T/N$ , the throughput cost of one lot; the asymptotic ideal is 255); at  $N = 1$  every policy ties the floor at 626.0 (the floor is  $255N + 371$  there, and tight: no contention). “Certified optimality” is the certified floor divided by the policy’s attained ticks: the fraction of certifiably-unbeatable performance the policy *provably* delivers.  $\pi^*$ ’s column reads 100.00 at the weekly count — earned, not rounded: the four-tick daylight between the machine-free floor and the attainment closed when the machine-contiguity tree was Lean-checked (section 17.5.11), and the certified floor now meets the replayed attainment at  $255N + 375$  exactly. The closure is  $N$ -uniform, so the  $N = 10^6$  column reads 100.00 as well. The sixteen never-idle cells are deterministic replays and do not extend to  $10^6$  (simulation is the bottleneck there by design, Table 17.2); only rows whose attainment has a certified closed form carry those columns — which is itself the point. The single-supervisor lanes of section 17.5.10 show the same pull-order lever at the same magnitude (40,406 ticks under an adversarial greedy pull there).

exhibit exactly. Re-instantiation at a new  $N$  does re-run the *producer*: it reveals the container (0.002 s at  $N = 84$ , 106 s at  $N = 10^7$  — the producer’s in-memory form expands the weight- $N$  guard unarily, a linear-cost implementation artifact of the untrusted side that never touches the sealed bytes or the checker), and re-derives the exact-rational producer receipts (the  $=$ -direction’s drop identities; 0.002 s at  $N = 84$ , 9.8 s at  $N = 10^6$ ). Dynamic attainment witnesses (replays arriving at exactly  $626N$ ) were re-run through  $N = 10^4$ , i.e. 6,260,000-step runs; beyond that the replay grinder’s quadratic cost makes simulation the bottleneck — which is the point of certifying.

**Claim grades at every  $N$ .** The within- $T^*$  direction rides geolog-checked rows at every  $N$  (the row obligations are  $N$ -free; the horizon instantiates from the container’s marking row). The not-before direction and exact attainment remain producer-tier receipts, and completion keeps its graded split (ranking geolog-checked, enabledness glue argued), exactly as stated with the certificates above — instantiation at large  $N$  upgrades nothing and degrades nothing.

**The contrast columns, derived not simulated.** Both bounds are conservative combinatorics on the net. *Markings*: lots distribute over pending, the six queues, and shipped; with machines idle every such distribution is reachable (advance one lot at a time) and distinct, giving  $\binom{N+7}{7}$  reachable markings before counting any in-process or machine state. *Schedules*: two lots processing concurrently on distinct stations

| $N$    | $T^* = 626N$  | $638N - 12$   | cert. (B) | check (ms) | markings $\geq$      | schedules $\geq$       |
|--------|---------------|---------------|-----------|------------|----------------------|------------------------|
| 7      | 4,382         | 4,454         | 3,105     | 1.9        | $3.4 \times 10^3$    | $10^{123}$             |
| 84     | 52,584        | 53,580        | 3,105     | 1.9        | $8.1 \times 10^9$    | $10^{1,719}$           |
| $10^3$ | 626,000       | 637,988       | 3,107     | 1.9        | $2.0 \times 10^{17}$ | $10^{20,469}$          |
| $10^6$ | 626,000,000   | 637,999,988   | 3,109     | 1.8        | $2.0 \times 10^{38}$ | $10^{2.0 \times 10^7}$ |
| $10^7$ | 6,260,000,000 | 6,379,999,988 | 3,111     | 1.9        | $2.0 \times 10^{45}$ | $10^{2.0 \times 10^8}$ |

Table 17.2: The synthesized-policy deadline certificate instantiated at lot count  $N$ . Columns: the certified all-arrivals horizon under the synthesized gate ( $T^* = 626N$  steps, exact on every completing run); the no-policy worst case ( $638N - 12$ ); the sealed container’s size in bytes (supervised exhibit; the no-policy container runs 2,963–2,966 B over the same sweep); the checker’s wall time (median of 7 re-discharges of the sealed bytes by the production engine’s in-loop evaluator walk, single core of an NVIDIA GB10 (Grace, aarch64) under concurrent load; all samples across all rows fell in 1.7–2.0 ms); a reachable-marking lower bound; and a lower bound on distinct maximal schedules the claims quantify over. Intermediate lot counts  $10^4$  and  $10^5$  were measured identically (containers 3,107 and 3,109 B; medians 1.9 ms).

(one in step 1, one in step 2) admit  $\binom{259}{32} \approx 10^{40.9}$  interleavings of their firing chains, all distinct as transition words; scheduling  $\lfloor N/2 \rfloor$  such pairs in blocks gives  $\binom{259}{32}^{\lfloor N/2 \rfloor}$  distinct maximal runs. Every one of them is covered by the same  $\sim 3$  kB of certificate.

**Reproduction.** Every certificate in these subsections is available as a sealed container, together with a one-command reproduction kit that rebuilds the independent checker at a pinned revision, re-verifies each container from its bytes alone — with per-relation claim counts as a load-bearing gate and committed integrity mutants that must refuse — and re-runs the producer-tier mutant batteries. The dispatch grids, the Z3 optima (each satisfiability/unsatisfiability pair re-derives in one command), and the  $N$ -scaling sweep ship the same way (section 2.9).

**The model boundary, honestly.** Everything in this section — counting time and carved clock alike — certifies the lot-flow core, and each exclusion is a named refinement rather than fine print: diffusion’s batching-of-three and its mix rules, product types and test-lot dedication, the declared buffer capacities, the transporter, the operators and technician, the maintenance calendars and failures, and the timed weekly release schedule. The certificates are arranged to survive the refinements that only restrict runs — a conservation or safety row over all schedulers holds *a fortiori* over fewer — while refinements that add Petri transitions re-run the same discharge over the extended roster; and arrivals-as-births are, as stated in section 17.4, a different certificate species with its own budget vocabulary, not a missing row here. Nothing in this section claims throughput under sustained release; that claim lives, for now, one section over — about the relaxation.

## 17.6 The fluid certificates: the comparison lane

The second certificate family predates the first and certifies a different object: the *fluid model* of the Mini-Fab’s deterministic core skeleton, under two named dispatch policies, with every residual freedom resolved adversarially. It is retained in full — as the comparison lane — because it answers exactly the question the combinatorial family does not yet price: sustained arrivals, at stated loads, policy by policy. The contrast, stated once: a fluid certificate asserts stability of the *relaxation* — continuous levels, aggregate drain rates, indexed by a dispatch policy — at releases within a tenth of a lot per week of capacity; a combinatorial certificate (section 17.5) asserts exact statements about the *discrete system* — every run, every lot count, scheduler-uniform — with no continuum limit anywhere. Neither family subsumes the other today, and the seam between them is measured rather than smoothed over: the granularity conversion still open on the fluid side is named at this section’s end, and the arrivals species still open on the combinatorial side is named at section 17.5’s. All numbers in this section re-derive from the committed artefacts in the experiments repository — the certificates and their mutant battery under `semicon/out/minifab`, replayed by the independent checker through the repository’s semiconductor make target — and are displayed



under the conventions of section 2.2. The campaign summary lives in section 2.7; this section walks the same artefact at reading depth.

### 17.6.1 The fluid model, with its decisions priced

Fluid scaling replaces lot counts by continuous levels  $q \in \mathbb{R}_{\geq 0}^6$ , one per route step, drained at the stations' machine-occupancy rates. From the tables of section 17.1, with handling folded into occupancy (the machine is held during load and unload, per-batch at diffusion):

| step | station (machines) | occupied min/lot | per-machine rate (lots/min) |
|------|--------------------|------------------|-----------------------------|
| 1    | diffusion (2)      | 285/3 = 95       | 1/95                        |
| 2    | implantation (2)   | 60               | 1/60                        |
| 3    | lithography (1)    | 75               | 1/75                        |
| 4    | implantation (2)   | 80               | 1/80                        |
| 5    | diffusion (2)      | 315/3 = 105      | 1/105                       |
| 6    | lithography (1)    | 30               | 1/30                        |

The declared-model ledger is explicit, in the spirit of the cookbook's declared-prior pattern (chapter 19): operators are folded into machine occupancy rather than modelled as a contended resource (that layer is the pool seam of section 17.3); products share a single class per step (the mix enters only through total release; dedication and batch-mix rules are granularity effects that vanish in fluid scaling); batching becomes a rate (3 lots per 285 or 315 machine-minutes); arrivals are deterministic at exact rate  $\lambda$ ; and failures and maintenance are excluded from the core skeleton — their sound entry point is capacity intervals from availability moments, named successor work, not fine print smuggled into these certificates. Setups get two sound variants: a core variant with no setups (matching the capacity arithmetic), and a setup-adversarial variant in which every lithography job may pay up to the full worst-case 12-minute setup, encoded as service-rate intervals  $\mu_3 \in [1/87, 1/75]$  and  $\mu_6 \in [1/42, 1/30]$  resolved adversarially at every instant — so the certificate quantifies over every setup sequencing with no monotonicity assumption anywhere (queueing networks are not monotone in service rates; intervals sidestep the issue rather than arguing it). Worst-case-inline setups put lithography's per-lot occupancy at 129 minutes, an adversarial ceiling of  $10080/129 \approx 78.1$  lots/week.

The two named policies are the classical static priorities for re-entrant lines: *last-buffer-first-served* (LBFS — later steps preempt earlier ones for their station's capacity: step 5 over step 1, 4 over 2, 6 over 3) and *first-buffer-first-served* (FBFS, the reverse). These are the disciplines the Kumar–Seidman destabilization literature is about, which is what gives “stable at  $\rho$ ” its falsifiable content. FIFO is not directly certified — its fluid content is class-level dispatch, per section 17.2 — but the setup-adversarial rows soundly cover FIFO's setup burden for *any* within-class order at loads up to 78 lots/week. Everything the policy does not pin — allocation slack on empty classes, idling below the top positive priority, rate realization inside the declared intervals — is inside the adversary.

### 17.6.2 The certified statement and its checker

Each certificate exhibits a linear Lyapunov function  $V(q) = \langle w, q \rangle$  with  $w \in \mathbb{Q}_{>0}^6$  and an exact rational  $\varepsilon > 0$  such that along *every* admissible fluid trajectory under the named policy,  $\frac{d}{dt}V(q(t)) \leq -\varepsilon$  for almost every  $t$  with  $q(t) \neq 0$ . Consequently any work-in-progress profile  $q_0$  drains to the empty line within  $V(q_0)/\varepsilon$  minutes, stays drained, and the line sustains its release — the stability and throughput atoms of section 17.4 at once, with a quantitative drain rate as the bonus.

The checker (`fluidcheck`, a small Rust program) trusts nothing about how the certificate was found. It rebuilds every polytope itself from the declared network data — routes, rates, capacities, policy — and replays one exact-rational Farkas dual per *sign mode*: for each of the  $2^6 - 1 = 63$  nonempty patterns of which queues are positive, the admissible drift vectors form a polytope, and the certificate's dual for that mode is a nonnegative combination of the polytope's constraints witnessing  $\langle w, \dot{q} \rangle \leq -\varepsilon$  over all of it. Soundness rests on weak linear-programming duality and one measure-theory lemma — the next subsection — and on nothing else; in particular, no solver, no floating-point arithmetic, and no completeness claim about the producer participates in the trust story. Each certificate carries 819 obligations and re-verifies in 1.5–3.0 milliseconds on one CPU core. The producer that found the certificates (floating-point proposal,

exact-rational certification, self-replay before emission) took about 0.2 seconds per certificate — and is, per the conventions, freely sloppy: being wrong costs retries, never soundness.

The bespoke container is, since this campaign first shipped, no longer the only committed form. Ten of the twelve certificates below are re-emitted as sealed geolog-commit containers on the unified certificate schema, in two committed lanes — one against the reproduction vocabulary, one pinned to the baked theory pair — each lane carrying its own mutant battery and re-verified, sign mode by sign mode, through the engine’s exact-rational ground seam; section 2.9 records the transition’s terms, and the two tree-cover certificates await their species’ re-emission. The original containers remain the record of the campaign that produced this section’s numbers, and the checker above still replays them — a committed lane never dies.

### 17.6.3 The boundary-equality lemma, the hinge

One semantic decision carries the entire result. What does the fluid model say about an *empty* buffer? The lazy encoding says only  $\dot{q}_k \geq 0$  when  $q_k = 0$  (levels stay nonnegative). The right encoding says more:

**Lemma 17.1** (boundary equality). *Let  $q_k : [0, \infty) \rightarrow \mathbb{R}_{\geq 0}$  be absolutely continuous. Then at almost every  $t$  with  $q_k(t) = 0$ , the derivative exists and  $\dot{q}_k(t) = 0$ .*

(At almost every point of the level set  $\{q_k = 0\}$  — discard the null set where  $q_k$  is not differentiable — the point is a Lebesgue density point of the set, hence an accumulation point of other zeros, and a derivative along a sequence of zeros is zero.)

The measured difference between the two encodings is not a refinement at the margin. Under the inequality-only semantics, the adversary may pour phantom flow into empty buffers whose station idles, and *no linear Lyapunov function exists at any tested load* — the certificate family is empty. Under the equality, linear certificates exist to within a tenth of a lot per week of fluid capacity. The equality also makes many sign modes vacuous: at the flagship configuration, 41 of the 63 mode polytopes are empty (a pattern like “only  $q_5$  positive” forces full diffusion output through an empty downstream line that lithography cannot pass). Vacuity needs no case split and no emptiness branch in the checker: an infeasible polytope has an unbounded dual, so the same Farkas row shape that certifies a drift bound on a nonempty mode certifies any bound on an empty one. The checker has one code path, on purpose.

### 17.6.4 The ladder

Twelve certificates and two deliberate failures, all replayed by the checker from the committed artefacts:

| certificate                     | policy | variant   | lots/week     | $\rho_{\text{lith}}$ | $\epsilon$ (V-units/min)                  |
|---------------------------------|--------|-----------|---------------|----------------------|-------------------------------------------|
| lbfs_core_84wk                  | LBFS   | core      | 84            | 0.875                | 1/3600                                    |
| lbfs_core_90wk                  | LBFS   | core      | 90            | 0.938                | 1/7200                                    |
| lbfs_core_94wk                  | LBFS   | core      | 94            | 0.979                | 1/21600                                   |
| lbfs_core_95wk                  | LBFS   | core      | 95            | 0.990                | 1/43200                                   |
| lbfs_core_191_2wk <sup>t</sup>  | LBFS   | core      | 95.5          | 0.995                | 1157/10 <sup>8</sup>                      |
| lbfs_core_959_10wk <sup>t</sup> | LBFS   | core      | 95.9          | 0.999                | 231/10 <sup>8</sup>                       |
| — probe                         | LBFS   | core      | 96 = capacity | 1.0                  | none (correctly)                          |
| fbfs_core_84wk                  | FBFS   | core      | 84            | 0.875                | $\approx 2.8 \times 10^{-4}$ <sup>a</sup> |
| fbfs_core_95wk                  | FBFS   | core      | 95            | 0.990                | 1/38880                                   |
| lbfs_setupadv_72wk              | LBFS   | setup-adv | 72            | 0.921*               | 11/74760                                  |
| lbfs_setupadv_76wk              | LBFS   | setup-adv | 76            | 0.973*               | 23/448560                                 |
| lbfs_setupadv_78wk              | LBFS   | setup-adv | 78            | 0.998*               | 1/299040                                  |
| — probe                         | LBFS   | setup-adv | 79 > ceiling  | > 1*                 | none (correctly)                          |
| fbfs_setupadv_78wk              | FBFS   | setup-adv | 78            | 0.998*               | 1/287280                                  |

(\*) setup-adversarial utilization is measured against worst-case-inline lithography occupancy, 129 min/lot. (a) exact value 6923209477/24747380706600; the display is rounded in the claim-weakening direction. (t) emitted as fully-refined sign-split *tree covers* — the certificate format’s refinement species, adjudicated by the checker’s tree walk rather than the mode-mask walk — with the weight search and all cell duals solved as one sparse linear program.

V-units are normalized so  $\sum_k w_k = 1$ :  $V$  is a weighted lot count. At the flagship configuration the optimizer’s own weights came out as  $w = (7, 7, 7, 3.5, 3.5, 2)/30$  — remaining-lithography-work flavoured,



found by the linear program, not designed. A worked reading: a ten-lot backlog at step 1, at the nominal 84 lots/week release, has  $V = 7/3$  and is certified to clear within 8400 minutes — under six days — against every adversarial resolution of the residual freedoms.

The table carries three findings. First, the certified load meets capacity: certificates exist at 95.9 of 96 lots/week (99.90% of fluid capacity) and at 78 of the 78.1-lot adversarial ceiling (99.8%); producer-side bisection brackets the last certifiable release within  $6 \times 10^{-5}$  lots/week of the fluid capacity (the one-lot gap in the original release grid was grid granularity, not lost capacity); and the probes at 96 and 79 fail *because the load is genuinely at or over capacity* — so the demonic generosity (everything unpinned handed to the adversary) is measurably free here, costing less than a tenth of a lot per week of certified capacity. Second, the exact drain rates separate the policies: FBFS edges LBFS near capacity (1/38880 versus 1/43200 at 95 lots/week; 1/287280 versus 1/299040 at the adversarial 78) — both stable everywhere tested, consistent with the classical fluid-limit picture for re-entrant lines, and the kind of exact policy comparison the claim class was chosen to produce. Third, the certificates and the reference simulation triangulate: certified setup-adversarial stability up to 78 lots/week  $<$  simulated FIFO stability at 84  $<$  the simulated deterministic-core ceiling at 87  $<$  certified zero-setup capacity at 96. The gap between 78 and 87 is the certificate’s conservatism against worst-case setups (24 minutes of setup per lot charged by the adversary, against 9.4 measured for FIFO in simulation) — and closing most of it is a policy question (run-length policies amortize setups), not a checker limitation.

The mutant battery, per the standing convention (section 2.2): seven deliberately corrupted certificates — inflated  $\epsilon$ , zeroed weight vector, negated dual, broken policy identity, dropped mode, swapped modes, over-capacity release — each designed to be refused at a specific check, and each in fact refused.

**What this section does not claim.** These are claims about the declared fluid model of the deterministic core, not about the discrete line — whose own certificates are section 17.5’s, and are of a deliberately different species: exact run-length budgets and invariants, not rate stability under arrivals. Of the granularity work that converts a fluid drain rate into discrete-line guarantees for the count chain, the Little’s-law (exact-moments) half has since been discharged: a discrete-lot certificate species replays lot-level cycles deterministically, certifying rate stability at the fluid capacity itself — a statement no fluid certificate can make — together with exact steady work-in-progress and cycle-time moments from declared initial families, with its own designed-locus mutant battery. The Foster–Lyapunov half — universal coverage over all mixed work-in-progress states — remains open, as does the sound entry of failures and maintenance via availability intervals, and the personnel, transport, and batching-granularity layers live on the composite seams of section 17.3 until those patterns carry certificates of their own. Nothing here claims either policy is *optimal* — only that its stability region is certified, to within a tenth of a lot per week of the model’s capacity, by an artefact a reader can re-refuse and re-accept on their own machine in milliseconds.

## 17.7 The construction in TypeScript

One orientation before the listing: the construction below is the fluid lane’s (section 17.6) — the model it assembles is the aggregate reading, and the certificates it drives are the release-grid drain certificates of that section’s ladder. The combinatorial campaign of section 17.5 is produced today on the toolkit’s Rust strategy surface; its TypeScript twin awaits the production-strategy surface on the toolkit’s TypeScript side (the specification surface’s revision has since landed, and this section’s listings ride it), and this section will grow that second listing when the surface lands, rather than simulate one now.

This section was staged until the toolkit’s TypeScript construction surface shipped, with its acceptance test fixed in advance: a single executable listing entering the ground tables of section 17.2 as rows, declaring the mechanisms as manifests, assembling the composite, forming specification content against the aggregate interface, handing the obligation set to the session workspace of chapter 8, and — driven over the same release grid — producing certificates the independent checker accepts at every load certified in section 17.6 and none at the two probe loads that correctly failed there. The surface shipped; the listing runs; the test passed, on the terms recorded at the end of this section. The complete file is committed as `qtsl/book/listings/minifab.ts` (with its emission seam `fluidseam.ts` beside it), against the `@maitria/mtk` and `@maitria/mtk-session` packages of section 7.6 and section 8.10; the excerpts below are verbatim slices of it, and the transcripts are the run’s own output.

The ground tables enter exactly as section 17.2 declared them — stations, route steps, release generators, the lithography setup rules, the implantation failure brackets — as data the rest of the file compiles from, so they are not reprinted here. The mechanisms are: each station at the aggregate stratum (per-visit queue levels, a busy-machine count, and the one device the platform renders every duration with — a timer drifting at rate one), with the declared buffer and machine capacities carried as domain constraints:

```
function station(st: number): system.System {
  const visits = ROUTE_STEP.filter((r) => r.st === st); // two per station
  const n = visits.length;
  const width = n + 2; // queues, busy, timer
  return system.buildSystem({
    name: STATION[st].name,
    state: [
      ...visits.map((r, v) => ({ id: v, name: `q${r.idx}`, sort: "Nat" })),
      { id: n, name: "busy", sort: "Nat" },
      { id: n + 1, name: "tau" },
    ],
    interface: {
      inputs: visits.map((r, v) => ({ id: v, name: `arrive${r.idx}` })),
      outputs: visits.map((r, v) => ({ id: v, name: `complete${r.idx}` })),
    },
    modes: [
      {
        id: 0,
        name: "serving",
        domain: [
          ...visits.map((_, v) => ({ poly: mono(width, v), op: "ge", rhs: 0 })),
          ...visits.map((_, v) => ({
            poly: mono(width, v),
            op: "le",
            rhs: STATION[st].bufcap,
          })),
          { poly: mono(width, n), op: "ge", rhs: 0 },
          { poly: mono(width, n), op: "le", rhs: STATION[st].machines },
        ],
      },
    ],
    drift: [{ mode: 0, var: n + 1, poly: constant(width, "1") }], // tau' = 1
    init: { kind: "point", coords: Array.from({ length: width }, () => 0) },
  });
}
```

The release mechanism is three timers (the thesis's signal-generator idiom is the timer-mechanism idiom, as section 17.2 observed), the sink counts finished lots and pools their flow times, and each conserved unit — operators, technician, transporter — is a unit-capacity pool mechanism whose conservation range is likewise a domain constraint. None of these manifests matches a catalogued profile, and none needs to: the free constructor never refuses, and the preflight's `noKnownProfile` notes in the transcript below are classification, not complaint — a mechanism whose populated family set matches no preset is a first-class citizen of chapter 12.

The composite wires the re-entrant lot flow, completion wire into arrival wire along the route:

```
const fab = system.composite({
  name: "minifab",
  components: [release, diff, impl, lith, sink, ...pools],
  wiring: [
    flow([0, 0], [1, 0]), // release      -> diffusion    (step 1)
    flow([1, 0], [2, 0]), // diffusion    -> implantation (step 2)
    flow([2, 0], [3, 0]), // implantation -> lithography  (step 3)
    flow([3, 0], [2, 1]), // lithography -> implantation (step 4)
    flow([2, 1], [1, 1]), // implantation -> diffusion    (step 5)
    flow([1, 1], [3, 1]), // diffusion    -> lithography  (step 6)
    flow([3, 1], [4, 0]), // lithography -> finished lots
  ]
});
```

```
1,
});
```

One boundary deserves stating here, where the code makes it concrete rather than abstract: the directed lot-flow seams of section 17.3 are constructible today, as the wiring rows above; the *pool-membership* and *event-fusion* seams are the event-port stratum, whose fusion primitive now carries a producer surface (typed fusion rows over the stations' population edges; section 13.10) while the pool-membership packaging is still design-tense, which is why the pools stand here as constructed mechanisms whose seize and release bindings await the packaged rows. And the dispatch freedom enters precisely where section 17.6 put it: as the policy index of each certificate, carried below by the release-grid driver, not as a row on this composite.

The specification content formable in the released vocabulary is the certificates' own quantitative consequence — the worked reading of section 17.6, as a typed record over the six-level aggregate interface:

```
const drainSpec = spec.build({
  name: "minifab drain: LBFS core 84/wk, ten lots at step 1",
  forall: [spec.probGe(1, spec.deadline(8400, spec.polyLe(0, wip)))] ,
  iface: {
    invars: [],
    outvars: ROUTE_STEP.map((r) => ({ name: `q${r.idx}`, shape: [] })),
  },
});
```

with *wip* the sum of the six coordinate polynomials. The long-run family of section 17.4 enters the same file at its formable half: the stability atom, through the average former. For the fluid skeleton under this policy at this release the honest cap is zero — the line drains and stays drained, so the transient washes out of the time average:

```
const longRunStable = spec.build({
  name: "minifab long-run: LBFS core 84/wk, average wip",
  forall: [spec.returnLe(0, spec.poAvg(spec.oPoly(wip)))] ,
  iface: {
    invars: [],
    outvars: ROUTE_STEP.map((r) => ({ name: `q${r.idx}`, shape: [] })),
  },
});
```

The record preflights, builds, encodes, and round-trips on the released surface (the receipt below carries the run); its resting state is the one section 17.4 assigns it: a formed claim whose certificate species is named and not yet claimed, so in the workspace's books the obligation would stand open — which is where a claim without its battery belongs. The throughput atom stays out of this listing for a narrower reason than it once had: its former is staged in the theory (section 14.6.4), but the producer surface quoted here does not construct it yet, and this listing does not simulate constructions the toolkit does not have.

The producer core is the Cookbook's propose-then-dispose discipline (chapter 19) on the checker's own vocabulary: the listing rebuilds the same constraint rows the checker rebuilds, asks one floating-point robust linear program for candidate weights (purely a proposal), then certifies one exact dual per sign mode and replays every obligation in exact arithmetic before emitting a byte. The equality rows are where two of this chapter's decisions become five lines of algebra — the boundary-equality lemma (Lemma 17.1) as flow conservation on each empty buffer, and the policy's entire fluid content as saturation above each nonempty priority level:

```
function eqRows(net: Net, mask: number): Row[] {
  const k = net.k;
  const rows: Row[] = [];
  for (let i = 0; i < k; i++) {
    if (!(mask >> i) & 1) {
      // buffer i empty: inflow = outflow (a.e. zero derivative)
      const r = zeros(2 * k);
      r[i] = Q.from(-1);
      net.route.forEach((succ, j) => {
        if (succ === i + 1) r[j] = r[j].add(Q.from(1));
      });
    }
  }
}
```

```

        rows.push({ r, rhs: net.arrivals[i].neg() });
    }
}
net.prio.forEach((order, s) => {
  for (let pos = 0; pos < order.length; pos++) {
    if ((mask >> (order[pos] - 1)) & 1) {
      // a nonempty buffer at priority: everything at or above it
      // saturates the station
      const r = zeros(2 * k);
      for (const j of order.slice(0, pos + 1)) r[k + j - 1] = 0.from(1);
      rows.push({ r, rhs: net.caps[s] });
    }
  }
});
return rows;
}

```

Finally the obligation set, handed to the session workspace: one obligation per grid point, probes included, under a statement kind whose single move runs the producer, emits through the canonical encoder, and lets the checker adjudicate. The kind's failure mode is part of its design: at an infeasible load it returns *inconclusive* — “the load is at or over the variant's capacity; absence of a certificate is not a refutation” — because no certificate is not a disproof, and the workspace's books are only as good as the outcomes recorded in them.

```

const GRID: [string, string, number][] = [
  ["LBFS", "core", 84],
  ["LBFS", "core", 90],
  ["LBFS", "core", 94],
  ["LBFS", "core", 95],
  ["LBFS", "core", 96], // probe: fluid capacity
  ["FBFS", "core", 84],
  ["FBFS", "core", 95],
  ["LBFS", "setup-adversarial", 72],
  ["LBFS", "setup-adversarial", 76],
  ["LBFS", "setup-adversarial", 78],
  ["LBFS", "setup-adversarial", 79], // probe: over the adversarial ceiling
  ["FBFS", "setup-adversarial", 78],
];

const ws = new Workspace([fluidStable]);
for (const [policy, variant, lotsPerWeek] of GRID) {
  const ob = ws.pose({ kind: "minifab.fluidStable", policy, variant, lotsPerWeek });
  const r = ws.apply(ob.id, "farkas");
  const w = r.attempt.witness as { eps?: string; checker?: string } | undefined;
  console.log(
    `${ob.id} ${ws.describe(ob.id)}\n -> ${r.attempt.outcome}` +
    (w ? `eps=${w.eps} (${w.checker})` : ` (${r.attempt.note})`),
  );
}

```

The run, whole — construction, specification, and the twelve obligations — takes under four seconds:

```

$ npx tsx minifab.ts
composite: minifab (composite), schema maitria.system.manifest.v0;
  9 preflight notes, kinds: noKnownProfile
spec: built ("minifab drain: LBFS core 84/wk, ten lots at step 1")
ob1 fluid skeleton drains under LBFS (core) at 84 lots/week
  -> discharged eps=1/3600 (ACCEPT modes=63 obligations=819 eps=1/3600)
ob2 fluid skeleton drains under LBFS (core) at 90 lots/week
  -> discharged eps=1/7200 (ACCEPT modes=63 obligations=819 eps=1/7200)
ob3 fluid skeleton drains under LBFS (core) at 94 lots/week
  -> discharged eps=1/21600 (ACCEPT modes=63 obligations=819 eps=1/21600)
ob4 fluid skeleton drains under LBFS (core) at 95 lots/week

```

```

-> discharged eps=1/43200 (ACCEPT modes=63 obligations=819 eps=1/43200)
ob5 fluid skeleton drains under LBFS (core) at 96 lots/week
-> inconclusive (the load is at or over the variant's capacity;
  absence of a certificate is not a refutation)
ob6 fluid skeleton drains under FBFS (core) at 84 lots/week
-> discharged eps=6923209477/24747380706600 (ACCEPT modes=63
  obligations=819 eps=6923209477/24747380706600)
ob7 fluid skeleton drains under FBFS (core) at 95 lots/week
-> discharged eps=1/38880 (ACCEPT modes=63 obligations=819 eps=1/38880)
ob8 fluid skeleton drains under LBFS (setup-adversarial) at 72 lots/week
-> discharged eps=11/74760 (ACCEPT modes=63 obligations=819 eps=11/74760)
ob9 fluid skeleton drains under LBFS (setup-adversarial) at 76 lots/week
-> discharged eps=23/448560 (ACCEPT modes=63 obligations=819 eps=23/448560)
ob10 fluid skeleton drains under LBFS (setup-adversarial) at 78 lots/week
-> discharged eps=1/299040 (ACCEPT modes=63 obligations=819 eps=1/299040)
ob11 fluid skeleton drains under LBFS (setup-adversarial) at 79 lots/week
-> inconclusive (the load is at or over the variant's capacity;
  absence of a certificate is not a refutation)
ob12 fluid skeleton drains under FBFS (setup-adversarial) at 78 lots/week
-> discharged eps=1/287280 (ACCEPT modes=63 obligations=819 eps=1/287280)

12 obligations: 10 discharged, 2 open (96wk, 79wk)
flagship V weights: (7/30, 7/30, 7/30, 7/60, 7/60, 1/15)

```

And the acceptance test's other half, against the standalone checker binary over the emitted files:

```

$ for f in out/*.cbor; do fluidcheck $f > /dev/null && pass=$((pass+1)); done
$ echo "standalone fluidcheck: $pass/10 ACCEPT"
standalone fluidcheck: 10/10 ACCEPT

```

Read the transcript against the ladder of section 17.6. The acceptance criterion was checker-acceptance, deliberately — a reimplemented producer may find different valid certificates — and what happened is a report, not a promise: this producer landed the *same* certificates. Every drain rate matches the table's, down to the FBFS exact value whose display there is rounded; both probes fail for the right reason; and the weight vector  $w = (7, 7, 7, 3.5, 3.5, 2)/30$  — which that section reported as found by the linear program, not designed — has now been found twice, by two implementations proposing over the same polytopes. The independence boundary is drawn where it always is in this book: the *producer* above is a fresh implementation on the TypeScript surface, while the emission seam links the checker crate itself, so that the bytes written and the bytes judged pass through one canonical encoder (at this preview the seam is a thin binding around the checker crate the reproduction spine imports; its migration is tracked in section 2.9) — and the standalone binary replays every certificate from the bytes alone, trusting nothing about how they were made.

# Implementation Pearls

---

DRAFT

This chapter is written in the tradition of Bentley’s *Programming Pearls* [16]: short, self-contained engineering essays, one per design decision that earned its place in the producer stack. The genre has a discipline of its own — a pearl is not advice; it is a technique with the measurement that decided it — and this chapter keeps to it. Each essay states the problem, the technique, and the receipt: numbers from committed artefacts, reproducible from the public repositories where the artefact has migrated, and identified as workshop-record measurements where it has not yet (per the preview note at the front of this volume). The common thread is that *certified-producer engineering* — code whose every output faces an unforgiving checker — quietly inverts several classic engineering trade-offs, and each pearl records one such inversion. Where the agents’ cookbook (chapter 19) states the moves of a good untrusted producer, this chapter states the decisions that hold up the machinery those moves run on.

## 18.1 The Certifying Tier Is a GPU Workload Too

**The problem.** The producer stack’s central split — a fast float tier proposes, an exact tier certifies (chapter 19, *Sloppy Proposer, Exact Disposer*) — is usually read as a hardware split: floats on the device, exact rationals on the host. The economics that force floats out of the inner search loop seem, by the same logic, to force exactness off the device. In the workshop prototype that seeded the toolkit’s kernels, one per-leaf basis-change GEMV at a realistic shape cost  $715\ \mu\text{s}$  in exact- $\mathbb{Q}$  arithmetic on the CPU, against  $4.65\ \mu\text{s}$  in directed  $\text{f64}$  on the same CPU and 191 ns on the device — three and a half orders of magnitude between the exact path and the device path. Leave exactness host-side and serial, and the moment search becomes device-fast, certificate assembly becomes the pipeline’s ceiling.

**The technique.** Put the exact tier on the device as an *integer* workload. The certifying kernels in `crates/mtk` represent a batch of rational tables as sign-magnitude integer numerators in base- $2^{64}$  limb planes over *one* positive shared denominator per cell (the schoolbook common-denominator move, industrialized), so that every transform is an integer recurrence with an explicit denominator-bookkeeping rule: no division, no gcd, no floating point anywhere in the certifying path. Reduction to canonical form happens host-side at the boundary. Fixed-capacity limb arithmetic is sound only if capacity exhaustion is *detected*, never wrapped, so every product writes into scratch of compile-time capacity with stated headroom lemmas, every store back is fit-checked, and the carry-out flag is captured and treated as overflow anyway — the lemma and the flag would both have to fail. On overflow the kernel poisons the cell: outputs zeroed, routing flag set, the cell’s work abandoned. A poisoned cell’s outputs are never read; the promotion ladder re-derives it from its *inputs* at the next tier — a wider device width, up to sixteen limbs, then unbounded host integers. Being wrong about a width costs a retry one rung up, never a wrong table. The kernels themselves ship as embedded CUDA source, compiled at runtime for whatever device is present, which makes the development loop one cargo run on any machine; an ahead-of-time path can join later behind the same module seam (and when it does, the pipeline-of-record law of section 18.3 applies to it).

**The receipt.** The exact-tier selftest runs eleven cases spanning the census shapes, fused transform chains, a  $\sim 130$ -digit-coefficient regime, and engineered overflow-boundary fixtures, demanding bit-exact equality against the host big-integer reference for every reduced table entry, every verdict, and every first-violation address — with the promotion ladder’s rungs each demonstrably exercised. Replaying a committed certificate from its bytes reproduces its certified threshold — a 55-digit reduced rational — bit-for-bit across three substrates: Python fractions, Rust `num-bigint`, and CUDA `u64` limb planes.

The full soundness burden, including what “bit-exact” means here and the residual risks, is stated in `crates/mtk/BERNX-SOUNDNESS.md` in the `maitria-coalition/mtk` repository.

## 18.2 Round Outward — to the Verifier’s Resolution

**The problem.** The base law of float participation in a certified system is directed rounding: on any verdict-relevant path, every operation rounds toward the safe side — toward  $-\infty$  for lower bounds, toward  $+\infty$  for upper bounds — so enclosures stay enclosures. The law is necessary, and it is not sufficient. A certifier campaign in the workshop lineage banked the counterexample: its checker decides transcendental sign questions by bisection on a dyadic lattice, deepening to a resolution of  $2^{-256}$ , and a producer emission relying on a  $10^{-70}$  relative guard alone can land a correctly-rounded margin *below one lattice unit*. Every rounding outward, every claim true in  $\mathbb{R}$  — and undecidable at any depth the checker will ever spend: a correct refusal of a true claim, forever.

**The technique.** Treat the emission grid as part of the interface. The producer quantizes every emitted bound outward onto a dyadic grid coarse enough that each claimed margin clears the verifier’s decision lattice by whole grid steps — in the campaign above, one full  $2^{-96}$  step outward keeps every margin decidable at bisection depth 16–64. The doctrine gains a clause: *exactness lives in the direction of rounding, and the magnitude of rounding must respect the verifier’s resolution*. The cost is  $2^{-96}$  of tightness, unmeasurable at certificate scale; the benefit is that a refusal now carries information — it indicts the claim, not the resolution. Two smaller dividends follow. Dyadic endpoints are exact rationals — an `f64` is a dyadic rational — so the checking side compares them in exact arithmetic, and a certificate that confines itself to the dyadic grid never smuggles a rounding into its own statement. And grid-aligned emissions keep denominators uniform across a certificate’s rows, which the exact tier’s shared-denominator representation (section 18.1) rewards directly.

**The receipt.** Workshop record: the continuous-latents certifier behind the counterexample above landed with thirteen artefacts accepted under truth containment, thirteen mutants refused at their designed loci, and a twenty-six-case property layer — on the grid-clearing emission discipline. The lattice constants above ( $2^{-256}$  checker resolution,  $2^{-96}$  emission step, decision depth 16–64) are that campaign’s numbers; its artefacts migrate to the public repositories with their campaign, per the preview note.

## 18.3 A Lane May Change Cost, Never Verdicts

**The problem.** Multi-architecture fast paths rot in a characteristic way: not silent breakage but *silent divergence*, an optimized lane that is almost always equal to the reference. Hand-vectorized code that must be faster by assumption — and equal by assumption — is how fast-path crates decay. In a system whose outputs face a checker, a divergent producer-side lane wastes campaigns; a divergent screening lane on the checking side is a soundness incident in waiting.

**The technique.** The shared-kernels repository (`maitria-coalition/kernels`) binds every kernel to one law: *an acceleration lane may change cost, never verdicts*. Three structural consequences. *Reference first*: every kernel ships a portable scalar reference, written for obviousness, always compiled — the reference is the semantics; lanes are the economics. *Receipts before dispatch*: a property-based differential battery gates every lane in CI, but passing the battery earns a lane only existence; a lane becomes its architecture’s default only with a measured receipt on real hardware, committed to the repository — and a lane that loses to the reference on its own target stays available, un-dispatched, and marked. *Fit-detection, never approximation*: fixed-width paths detect inputs they cannot represent exactly and refuse them to the caller’s promotion ladder; no rounding exists on any verdict-relevant path. For JIT-compiled device lanes there is a fourth: the same kernel source can compile branch-free under an offline pipeline and branchy — an order of magnitude slower — under a runtime pipeline, so every device lane names its *pipeline of record*, acceptance benches run that pipeline, and the disassembly branch count is printed at build time and recorded in the receipt as a watched number.



**The receipt.** The founding receipts of the coefficient-sweep family, committed under receipts/ in the kernels repository: the NEON lane at  $1.47\text{--}3.00\times$  the reference across shapes on a Grace-class aarch64 workstation, the AVX2 lane at  $1.25\text{--}3.96\times$  on a Skylake-W Xeon — with the receipt’s own candid reading attached, that the wins are modest at the largest sizes because a streaming i64 scan leaves cache and becomes memory-bandwidth-bound. The dispatch code cites those numbers, not the intuition they corrected. And the batteries earn their keep in both directions: the GPU interval family’s *first* battery run falsified the oracle, not the kernels — a naive exact-rational reference had missed IEEE 754’s signed-zero rules [120] (an exactly-cancelling round-toward- $-\infty$  addition yields  $-0.0$ ; an exact-zero product carries the XOR of the operand signs). The device was right; the oracle now spells the exact-zero cases out, and the repaired family ran sixteen-for-sixteen green at a hundred property examples each — a battery its committed receipts have since grown to thirty-three properties, still all green at the same hundred. Exact rationals carry no signed zero, so an exactness oracle for float behaviour must state IEEE’s sign semantics explicitly — a differential battery is symmetric, and the symmetry is a feature.

## 18.4 The Babysitter Thread

**The problem.** An exact producer grinding a large obligation frontier can be legitimately silent for minutes: no intermediate result is trustworthy enough to print, and the hot loop has better things to do than I/O. But at an operator’s terminal — increasingly, an operating *agent’s* terminal — “no output for minutes” is indistinguishable from “hung,” and the rational response to a hung-looking grinder is to kill it, wasting exactly the work the silence was hiding. The doubt is the cost: it can be resolved only by waiting longer or killing sooner, and both resolutions are expensive. Classic optimization-solver CLIs solved this decades ago: an iteration line, about once a second.

**The technique.** The toolkit’s convention: any hot loop expected to run beyond about a second reports progress to standard output at least once per second — iteration count plus whatever else is cheaply at hand (objective, residual, cells retired). The implementation keeps the reporting out of the loop’s way: the hot loop maintains *only counters*, written with relaxed atomic stores — no I/O, no clock reads, no timing branches on the hot path — while a *babysitter thread* wakes on a system timer about once a second, reads the counters, and prints. Where the loop body is already chunky, the degenerate inline form suffices: consult the clock only every  $N$ th iteration and print when a second has elapsed — the event-loop engine in `crates/mtk-eventloop` does exactly this in its replay path (every 64th application, one elapsed check) and rate-limits its per-pass progress lines to the same once-per-second cadence, with the final line always printed. Where a verb’s standard output is a machine-consumed data channel, the collision is flagged and the progress line moves — never silently dropped.

**The receipt.** The convention entered the toolkit the usual way: a long exact-tier run drew a “no sign of progress” report, and settling that doubt cost more than the instrumentation ever would have. The fix is as close to free as instrumentation gets — a relaxed store per iteration and one thread sleeping on a timer — and both forms are landed machinery in the mtk repository. The exact solver loops carry the full form (`crates/mtk/src/progress.rs`: two relaxed counters, threaded as an optional argument so that opting out costs nothing, with the `lp-solve` verb’s exact path spawning the once-per-second monitor); the event-loop engine’s replay path carries the inline form (`crates/mtk-eventloop/src/engine.rs`). The surface also crosses the language boundary. The toolkit’s TypeScript bindings expose the same counters as a polling handle over asynchronous solve variants; JavaScript gives the babysitter and the baby one thread to share, so the solve must leave it before any timer can read a counter mid-grind. The printing monitor stays on the consumer’s side of the boundary, where a collision with a machine-consumed output channel is fixable at the call site rather than at a process boundary. One detail is worth copying: a test in each ticking module, on both sides of the language boundary, asserts both that the counters move and that their presence changes no outcome — the same run, bit-identical, with and without the observer. The pearl is the asymmetry: the counters cost nanoseconds per iteration; their absence is priced in killed runs.

## 18.5 A Certificate Is Database Content, Not a File Format

**The problem.** Every bespoke certificate file format breeds an ecosystem tax: one parser per consumer, version skew between them, vendored readers drifting from the spec, and an interchange layer that slowly comes to carry semantics nobody proved anything about. The toolkit’s development lineage paid this tax once — an early binary certificate format grew four independent readers before it was deliberately retired — and the replacement is a structural decision, not a better format.

**The technique.** Certificates are ordinary rows in the logical database: columns of exact scalars, tensor-valued content where the payload is genuinely tensor-shaped, stored under the certificate theory that the `qtsl` repository defines as data (`theories/Qtsl.geolog`, extending the Bernstein Logic proof tower of `theories/BernsteinLogic.geolog`), in the same content-addressed commits as everything else. There is no intermediate certificate file format to version, so there is none to drift; a checker’s input is a sealed commit body, and the same bytes serve every consumer — the production engine’s incremental monitoring, the certified-checker route, the reflection route into a proof assistant’s kernel — with no translation step left to audit. Merge is set union plus re-derivation, and provenance questions (*who asserted this row, and did anyone assert its contradiction?*) are structural properties of signed content-addressed commits rather than per-format afterthoughts. What remains outside the database is deliberately thin: the untrusted tiers exchange proposals and refusals as CBOR shapes in a dependency-free types crate (`CONTRACTS.md §1.2` in the `mtk` repository draws the boundary: witness interchange is *not a proof format*), so the producer’s conveniences can evolve freely without a byte of them leaking into the checked surface’s semantics.

**The receipt.** The interchange layer’s golden vectors decode a committed certificate whose payload carries 55-digit big-integer fractions and round-trip it losslessly through the `v0` shapes (`crates/witness-types/tests/golden.rs` in the `mtk` repository) — exactness surviving serialization is a test, not a hope. And the next pearl depends on this one: canonical content is what makes byte-identity a meaningful instrument in the first place.

## 18.6 Byte-Identity Is a Conformance Instrument

**The problem.** Cross-platform reproducibility for numerical systems is conventionally stated in tolerances — “agrees to  $10^{-9}$  across machines” — which reintroduces judgment exactly where an audit wants none: who chose the tolerance, and what drifted inside it? A certificate-producing system can demand more. If the certifying path is exact and the emission is canonical, the correct cross-platform claim is not agreement within tolerance; it is *the same bytes*.

**The technique.** Make the producer deterministic by construction — exact arithmetic on the certifying path, so no platform’s float behaviour can enter (section 18.1); fixed iteration and emission orders, so no hash-map enumeration nondeterminism survives; canonical serialization (section 18.5) — and cross-silicon conformance collapses to `cmp`. The split this buys is exactly the split the system’s trust story wants: the *verdict* column of any re-run table must be identical across machines, while the *time* column is allowed to vary and thereby becomes a genuine hardware measurement instead of a confound.

**The receipt.** The reproduction spine’s semiconductor-fab case study (the worked example of chapter 17; battery and receipts in the `semicon/` tree of `maitria-coalition/qtsl-experiments`) was re-run end-to-end on two boxes of different instruction sets — a Grace-class `aarch64` workstation and an `x86-64` EPYC cloud instance. Every emitted certificate came back byte-identical to the committed fixtures on both: ten of ten fluid-scheduling certificates, with the certified slack values matching as exact rationals; six of six station-cover certificates by SHA-256; the policy-separation numbers reproduced as identical exact rationals ( $\epsilon = 1/43200$  against  $1/38880$  at the near-capacity operating point); and the deliberate failure probes — the honest-stop cases — still failing correctly on both machines, producer face and checker-mutant face alike. The checker battery ran all sixteen positive fixtures and all thirteen mutants to their expected verdicts on both boxes. The timing columns, meanwhile, differed by a uniform  $1.7\text{--}1.9\times$  single-core factor across producer walls and check kernels — a bridge measurement that classifies every cross-box timing delta as

hardware economics, with evidence, rather than leaving it to argument. Determinism turned an entire re-validation campaign into a column of *identical* plus one real hardware number.

Six decisions, one shape: in a system where every claim must survive an independent checker, the engineering wins go to whoever moves judgment out of the pipeline and into artefacts — exactness where floats were assumed, grids where guards were assumed, receipts where intuition was assumed, bytes where tolerances were assumed. The measurements above are what those trades cost and what they paid; the repositories carry the artefacts to re-run them.

# Cookbook for Agents

---

DRAFT

This chapter is the toolkit’s proof-production cookbook: moves in the Pólya shape — the move, its recognition conditions, a worked example with receipts, the failure mode it prevents, exercises — re-derived from *How to Solve It* [200] for minds that produce machine-checkable certificates. The intended reader is a large language model (or any capable agent) doing formal verification and heavy engineering; a human reads it profitably as the toolkit’s operational philosophy stated concretely. The register follows from the founding intent: a meta-insight that could once only be pattern-matched through years of accumulated practice can now simply be *stated* to an agent, in the agent’s native language, and applied faster for being explained.

Every pattern carries the same five rubrics. *The move*, stated once at full generality. *When it applies*: the recognition conditions — how you notice, mid-task, that this is the situation you are in. *Worked example from the record*: a real incident from the workshop record that produced this toolkit, receipts included, so the reader can go verify rather than take the chapter’s word — itself one of the cookbook’s moves: a claim worth acting on ships with an artefact positioned to contradict it. *The failure mode it prevents*: what happens, concretely, on the path where the move is not made. And *exercises*: open legs from the producing campaigns’ own backlogs at this preview, so that working an exercise produces value, not homework. Artefacts named in the examples migrate to the public repositories as they graduate; per the preview note at the front of this volume, treat a mention as a statement of what exists, not yet a link. Status words (“shipped,” “proposed,” “in design”) are as of this preview.

The workshop’s cookbook has already grown past this chapter: two further moves graduated after this preview’s cut (one on charters as falsifiable predictions, one a proof-assistant transparency lesson), and a strong candidate — adversarial review with planted doubts — stays deliberately unwritten here, because its authoritative statement is a live operating document in the workshop, with measured hit rates across the review campaigns that shaped it, and a book copy of a live norm would drift from the norm. Each will be written from its receipts when they migrate. Behind them, a harvest queue of roughly ninety further candidate moves waits its turn.

## 19.1 Sloppy Proposer, Exact Disposer

**The move.** Split every computation whose result must be trusted into two roles with an explicit boundary:

- a **proposer** — fast, approximate, unverified, and structurally *untrustable*: a float library, a validated-numeric package, an ML search, an SMT heuristic, your own quick script. It produces candidates: a region, a bound, a witness, a decomposition, a step size.
- a **disposer** — exact, minimal, independently implemented. It re-derives, from the candidate data alone, everything the certificate asserts (exact- $\mathbb{Q}$  arithmetic, from-scratch identity checks) and accepts or rejects. Nothing the proposer computed appears in the trusted conclusion; only what the disposer re-derived.

Then police the boundary **in both directions**:

1. Proposer arithmetic never enters the certificate. The trusted computing base contains the disposer and nothing else; the proposer is trusted only *as a search heuristic*, where being wrong costs retries, never soundness.
2. Disposer arithmetic never enters the proposer’s inner loop. Exact computation inside a search loop is pure waste: the loop’s job is to find a candidate worth checking, and it should be as cheap as

correctness-irrelevance allows. The division of labour is the point — a proposer made rigorous is as much a design error as a disposer made sloppy.

This is the book’s assurance stance (chapter 1) applied at the scale of a single computation, and the producer stack is shaped by it end to end: every GPU kernel row returns witness-typed output built for the exact tier’s disposal (chapter 18), and the checker at the bottom (chapter 15) is the disposer of last resort.

### When it applies.

- Any time a fast-but-unverified computation must feed a certificate whose soundness has to be airtight. The recognition question: “does the assurance story currently require this library/model/script to be bug-free?” If yes, restructure so it only has to be *useful*.
- Any time you notice exact or expensive computation inside a loop that exists to search. The recognition sign is a profiler line: the search dominating wall-clock with arithmetic whose precision the final verdict never uses.
- When choosing whether to “just trust” an established rigorous-numeric or solver library. The relevant question is not whether the library is good (it usually is) but whether its *output representation* already matches something your checker can re-verify — in which case de-trusting it is nearly free.

**Worked example from the record.** The validated-ODE de-trusting ladder: take enclosure claims from CAPD — a well-regarded C++ validated-numeric library for rigorous ODE integration — and re-found them on an exact checker, so that the assurance story never rests on trusting C++ interval arithmetic. The architecture is the move verbatim: three certificate species in which the proposer of every candidate — CAPD, an in-house float Picard iterator, anything — is untrusted by construction, and the checker re-derives all sequent polynomials from the candidate data by exact  $\mathbb{Q}$  arithmetic. Validity of a proposed flow tube is a checkable fixed-point inclusion (the Picard operator maps the candidate box into itself — the a-posteriori validation architecture formalized, kernel and all, by Bréhard, Mahboubi, and Pous [1]), so the checker never integrates anything — it verifies inclusion-invariance of what the untrusted integrator proposed, with zero per-instance quantitative side conditions: no Lipschitz constant, no contraction factor, no step-ratio condition in any certificate.

Both directions of boundary-policing have receipts:

- **Direction 2, learned the expensive way.** The first proposer implementation ran exact Bernstein computations inside its candidate-search loop over radius and error-matrix parameters — measured at roughly forty seconds per pass of pure waste, since the disposer re-derives everything anyway. The campaign’s own retrospective names the division of labour as one of its surprises: the lesson was not “be rigorous” (the pipeline already was, at the disposer) but “stop being rigorous in the wrong place.”
- **Direction 1, cashed out when the real library arrived.** CAPD, vendored and driven through the frozen proposer interface, checker untouched: the exact- $\mathbb{Q}$  end box agreed with CAPD’s uncertified claim to three decimals with a tube radius about  $50\times$  tighter — the cost of de-trusting was approximately zero enclosure quality (wall-clock about  $5\times$ , spent in the checker, which is the one place spending is correct). And the certified envelope *extended*: a Van der Pol step size the in-house float proposer could not certify became certifiable with CAPD proposing. An isolation probe attributed the remaining failure wall to the in-house proposer’s own search loop, not to CAPD — a diagnosis only available because the interface made proposers interchangeable.

Note what interchangeability bought beyond soundness: upgrading the proposer from quick in-house script to best-in-class library was a pure plug-in, no re-verification of anything, because the trust boundary never moved.

### The failure mode it prevents.

- **Trusted-base bloat:** the assurance story “the theorem is only as trustworthy as the library plus the compiler run” — which is CAPD’s own openly stated architecture, and precisely the thing being escaped. Every trusted component is a standing liability that must be re-audited whenever it updates; a proposer never needs auditing at all.

- **The dual waste:** search loops that run at certificate precision, making exploration the bottleneck for zero soundness benefit. This one is insidious because it *feels* virtuous — rigour everywhere — while actually signalling that the boundary was never drawn.
- **Un-upgradeable pipelines:** when proposer and verifier share arithmetic (or code), swapping in a better search engine means re-validating the world. With the boundary drawn, proposers compete and the checker stands still.

### Exercises.

1. **The jet-transport adapter.** CAPD's `Cn0deSolver` computes derivatives of the flow map (jets). The adapter interface is frozen and the ladder's first-rung sequent layer is jet-ready; wire `Cn0deSolver` through the same seam, checker untouched. Success condition: goldens accept with CAPD-proposed jet enclosures; the checker diff is empty.
2. **Settle the flagged hypothesis.** The adapter's arrival recorded that one benchmark's certifiable-step wall moved *inward* under CAPD, with a stated-but-unverified mechanism (centred forms losing a cancellation under the checker's crude absolute-value bound). Design the two-cell experiment that confirms or refutes the mechanism; if confirmed, the fix belongs in the disposer's bound, not the proposer.
3. **Generalize the isolation probe.** The probe that attributed the residual wall to the in-house search loop was built bespoke. Extract the pattern — swap proposers across the frozen interface, hold the disposer fixed, bisect which side owns a failure — into a reusable harness verb for any proposer/disposer pipeline.

## 19.2 Certify the Honest Stop

**The move.** When a line of attack ends — the rate cannot be improved, the constant cannot be sharpened, the method genuinely runs out — do not shrug and move on, and do not quietly ship the weaker thing as if it had been the goal. **Prove the stop.** Convert “we couldn't push further” into one of three certified endings, and record which one you are in:

1. **The claim past the stop is false.** Produce the witness (a concrete counterexample instance) *and*, where the machinery permits, a certificate-side proof that no valid certificate of the stronger claim exists (e.g. any candidate decomposition is forced into an inadmissible component). Then the stop is a theorem, not a fatigue report.
2. **The claim is true but this method cannot reach it.** Name the missing ingredient as a first-class object — a new prior family, a stronger bridge lemma, a representation the format lacks — so the stop becomes a scoped successor leg rather than a vague wall.
3. **The claim is true at a boundary value.** Certify the boundary from both sides (the bound holds at  $c$ ; a witness attains  $c$ ), so the constant ships as sharp rather than as “the best we found.”

In all three cases the artefact records the stop with the same rigour as the positive results — ideally *inside* the same certificate and test battery, so a future reader (or a future you, or a cheaper model retrying the problem) cannot re-open the question without meeting the proof.

### When it applies.

- Whenever a sweep, ladder, or optimization plateaus and you are about to write “we achieved  $X$ ” for the  $X$  you reached. The recognition question: *is the plateau a fact about the problem or about your effort?* If you don't know, that is ending 1 to chase — the answer is usually cheaper than the next ladder rung would have been.
- Whenever you are tempted to soften a claim's statement so that what you proved matches it. Softening is sometimes right — but only with the un-softened claim's status recorded (false? open? method-blocked?).



- When banking machinery or priors for future use: an input that turned out to be *unconsumable* (its only consumer is a false claim) is a result about the problem’s structure, worth the two sentences it costs to record.

**Worked example from the record.** The Kuramoto–Sivashinsky energy certificate, fourth target of a campaign producing exact-rational, independently replayed dissipation certificates for PDEs. The campaign’s earlier targets had certified decay *rates* — viscous Burgers at the sharp rate  $\frac{d}{dt}E \leq -2\nu E$ . For Kuramoto–Sivashinsky on the marginal domain  $\mathbb{T}_{2\pi}$  the analogous hope — any positive rate  $\alpha > 0$  — is **false**, and the artefact treats that falsity as a deliverable (ending 1, with ending 3 nested inside):

- **The witness:** at mode  $k = 1$  ( $u = \varepsilon \sin x$ ) the dissipation form  $k^4 - k^2$  vanishes and the nonlinearity is energy-neutral, so  $\frac{d}{dt}E \leq -\alpha E$  fails for every  $\alpha > 0$ . One line, checkable by hand.
- **The certificate-side impossibility:** the producer carries an assert that any nonnegative-weight decomposition of the  $\alpha$ -remainder forces weight  $-\alpha$  on the  $u_2^2$  square — unconditional in the free weight. The stronger claim is not merely un-found; it is un-certifiable in the calculus, and the artefact proves that.
- **The battery enforcement:** a mutant recomputes *everything* consistently at a positive rate — every jet identity, every replay green — and is rejected *only* by the negative weight. The stop is enforced in the executable spec (section 19.3): a future certificate claiming a positive rate on this domain will be refused by machine, not by memory.
- **The named residue** (ending 2): the artefact’s own stopping-point section states what the composed result is — energy monotonicity at the marginal domain, sharp, the strongest true statement for this functional on this domain, with the stop *provable* — and then names what the real literature benchmark ( $L > 2\pi$  mean-energy bounds, where energy genuinely grows through the unstable band) additionally requires: a background functional and a genuinely new prior family (Gagliardo–Nirenberg/interpolation rows). Its closing words are this section’s thesis in miniature: “named next leg, not tonight’s overclaim.”
- **The unconsumed input, recorded:** a banked spectral prior ( $\int u_2^2 \geq \int u_0^2$ ) is left unconsumed, on the record — the campaign notes its only would-be consumer is the false rate claim. Two sentences, and no successor will wonder why it sits unused.

A sibling specimen, same campaign, showing ending 3 in both directions: the Fisher–KPP certificate proves the invariant region  $[0, 1]$  is maximal by mutants — enlarged above to  $[0, \frac{9}{8}]$ , all bookkeeping recomputed consistently, rejection comes only from a genuinely negative Bernstein coefficient that no subdivision can repair; extended below, the endpoint row itself fires. “The certificate cannot be strengthened past the true boundary in either direction” is a tested property of the checker, not a sentence in prose.

### The failure mode it prevents.

- **The shrugged dead-end:** an unrecorded stop is a question that reopens. Someone (often a cheaper agent, often you-next-week) retries the false claim, burns the same compute, and — worse — may “succeed” via a subtly unsound route precisely because nothing in the machine refuses it.
- **The laundered stop:** shipping the weaker claim in the stronger claim’s rhetorical clothing (“we certify decay” when the hoped-for rate silently became 0). The reader’s trust is spent on the discrepancy when they find it — and in adversarial review they will.
- **Effort-plateau mistaken for truth-plateau:** without the falsity/impossibility half, “sharp” means “we got tired.” With it, sharpness is a two-sided certificate and the constant is settled.

### Exercises.

1. **The symbolic-domain carriage.** Carry the domain length  $L$  symbolically through the Kuramoto–Sivashinsky certificate: for  $L < 2\pi$  the same form yields a *positive*-rate nonlinear decay; for  $L > 2\pi$  it yields nothing — the energy method genuinely ends there. Certify both sides of that boundary, making the method’s own limit an ending-3 artefact.



2. **The background-functional leg.** The named ending-2 residue: build the interpolation prior family (the first genuinely new prior species in the campaign) and attack the  $L > 2\pi$  mean-energy benchmark. The honest-stop record is the specification of exactly what is missing.
3. **Settle a flagged wall.** The CAPD adapter's arrival (section 19.1) left one certifiable-step wall's inward move as a stated-but-unverified hypothesis. Determine which ending it is: a true wall of the bound (certify it), or an artefact of a crude inequality (fix the disposer and move it back). Either way the stop stops being a guess.

## 19.3 The Mutant Battery Is the Spec

**The move.** A checker's meaning is not what it accepts — accepting the golden certificate is table stakes — but what it **rejects**. So specify the checker by building its rejections: a battery of *mutants*, each a certificate wrong in one deliberate, named way, each asserted to be rejected **at its intended check with its intended message**. The battery is then the executable specification of the trust boundary: every mutant is a theorem of the form “a certificate lying in *this* particular way cannot pass,” and the battery as a whole delimits what an ACCEPT verdict actually asserts.

Two disciplines make the battery a spec rather than a pile of negative tests:

1. **Consistency mutants are the genre that carries the spec.** The strongest mutants recompute *everything downstream of the lie consistently*, so that every identity, every replay, every side computation is green, and rejection can only come from the single row that carries the claim's truth. A mutant rejected by accidental earlier breakage (a malformed field, an arithmetic slip you didn't intend) tests nothing about the check you meant to test — which is why each rejection's *location and message* are asserted, not just the rejection.
2. **Cover the lie taxonomy, not just arithmetic.** Wrong numbers are the easy genre. The battery must also include: sharpness violations in **both** directions (claim strengthened past the true boundary / input weakened below it), laundered caveats (an obligation's status upgraded to one no receipt backs), provenance breaks (data whose justifying source is removed, dangling, or unlisted), structural smuggles (declaring only a slice of the term the side condition should see), and — where claims are mechanism- or context-scoped — transport violations (a true-in-context-A claim rewired into context B).

The same discipline runs at every tier of this toolkit. The rule catalogue (chapter 15) is its proof-theory face: the roster is machine-checked, every mechanized rule carries receipts, and the coverage dashboard up front is the candid map of what the assurance edifice does and does not yet refuse. The composition-pattern inventory (chapter 13) is its modelling face: the refusals do real work there too, encoded structurally rather than left to the modeller's vigilance.

### When it applies.

- Whenever you write a checker, validator, decoder, or replay tool whose ACCEPT someone will trust. If the mutant battery does not exist, the trust boundary exists only in your intentions.
- Whenever a checker gains a new check: the check's first artefact is the mutant it refuses. A check without a firing mutant is unfalsifiable — you cannot distinguish “this check works” from “this check is dead code.”
- When reviewing someone else's verification claim. “10/10 checks passed” is uninformative until you know what the 11th — the one built to fail — would have done. Ask for the battery before the benchmark.

**Worked example from the record.** The PDE certificate campaign of the previous section: exact-rational dissipation certificates for Burgers, Fisher-KPP, and Kuramoto-Sivashinsky, each with a Python producer and an independent exact-fraction checker sharing zero polynomial code, each shipping a battery with per-mutant rejection messages asserted. The taxonomy, one specimen each, all on disk:

- **Consistency + sharpness (above):** a Burgers mutant claims decay rate  $\alpha = 3\nu$  (true rate:  $2\nu$ ) and recomputes all bookkeeping consistently — all arithmetic passes and rejection comes only from the negative combination weight: the checker enforces the sharpness boundary, not mere self-consistency. The Kuramoto–Sivashinsky twin pushes a linear rate of 24 to 25 and is refused only by the forced weight  $-\frac{1}{16}$ .
- **Sharpness (below) + unrepairability:** the Fisher–KPP mutant widens the certified invariant region  $[0, 1]$  to  $[0, \frac{9}{8}]$ ; the endpoint rows still pass (any  $[0, b]$  with  $b \geq 1$  is inward at the ends) and the refusal comes from one genuinely negative Bernstein coefficient equal to  $f(\frac{9}{8}) < 0$  — no subdivision can repair it, so the battery proves the boundary is the true one rather than a resolution artefact.
- **Structural smuggle:** advection-smuggle mutants add a term the semilinear/jet-linearity side condition must see; every arithmetic identity passes and only the structure check fires — establishing that the side condition, not the arithmetic, is what excludes that class of false instance.
- **Provenance edges:** the Fisher–KPP deficit-decay remainder is the campaign’s first that genuinely needs a bounding box, and the box is supplied *by another claim in the same certificate* with an explicit provenance edge. Three mutants break the edge three different ways — source removed, dangling, supplier unlisted — and each is refused by the provenance check specifically.
- **Laundered caveat:** both laundering flavours — a hypothesis quietly dropped from the declared list, and an obligation’s status promoted to “proven” with no discharge receipt — are refused by the ledger discipline (section 19.4), tested both at the ledger and inside the discharge object.
- **Transport (the campaign’s centrepiece):** a mutant takes the linear-mechanism spectral-gap rate claim — true, certified, sharp *for the linear mechanism* — and rewires it onto the nonlinear flow, where mode-1 pumping makes it false. Every jet identity, weight, and spectral replay is green; rejection comes **only** from the mechanism-scoped premise-transport table. In the artefact’s own words: “the checker refusing a false theorem whose every checkable identity is true.” That single mutant is the spec of what the transport table means.

Scale receipts, for calibration: the three batteries run 31, 18, and 29 tests (29, 17, and 28 mutants respectively), messages asserted, with a combined regression replaying all three checkers green — and each battery was built within its target’s single working session: hours, not weeks.

### The failure mode it prevents.

- **The golden-only checker:** passes its examples, accepts a subtly-false certificate the day an adversary (or an optimizer, or a bug) produces one. Without mutants you have tested the happy path of a tool whose entire purpose is the unhappy path.
- **Dead checks:** a check that nothing exercises can be broken by refactor and no test notices. The firing mutant is the check’s liveness monitor.
- **Accidental-rejection illusions:** a battery whose mutants die of malformedness rather than of their intended lie certifies nothing — the dangerous mutant is the *well-formed* one, and only location-asserted rejections distinguish the two.
- **Unfalsifiable sharpness and scope claims:** “sharp,” “maximal,” “linear mechanism only” as prose adjectives, versus as mutants the machine refuses. The former decay into folklore; the latter are enforced.

### Exercises.

1. **The spectral wire rows.** When the jet certificate schema’s spectral wire family ships, write its battery *before* its first consumer: wrong-constant discharge, missing witness mode, transport onto an unlicensed mechanism — the taxonomy above, instantiated at the wire tier.
2. **Retrofit audit.** The neural-supermartingale certificate suite grew its mutant receipts incrementally across its campaign. Audit its batteries against the taxonomy: which genres are missing (transport? provenance? both-direction sharpness?), and add the cheapest missing genre with rejection locations asserted.

3. **The battery-of-the-battery.** The location-assertion discipline currently lives in test code by convention. Build the meta-check: a harness pass that runs each mutant, captures *which* check fired, and fails the battery if any mutant is refused earlier than its declared target — turning discipline 1 of this section from prose into a tool.

## 19.4 Declared-Prior Ledgers

**The move.** Every analytic fact your certificate **consumes but does not prove** — a functional inequality, a comparison principle, a semantic bridge between the certificate’s algebra and the theorem’s meaning — is carried by an explicit **ledger** inside the certificate, under a **closed status vocabulary**. Concretely:

1. **Declare, never absorb.** An unproven ingredient appears as a named entry (assumed, named-unproven, ...), not as an invisible premise folded into the prose or — worse — into the checker’s silence. The certificate’s ACCEPT then means: *these claims hold, conditional on this list*.
2. **Close the vocabulary and back every status.** The checker enforces that statuses come from a fixed set and that any status implying a proof has its receipt present and replaying green. An obligation promoted to “proven” with no receipt is a *rejected certificate* — laundering is a machine-refused move, not a style violation.
3. **Discharge as an arc, with a tie.** Because the ledger is explicit, an assumption can be discharged *later, incrementally*: a new certified object replaces assumed with a proof-backed status, and a **tie check** verifies the discharged object is the same fact the consuming row uses (same integrand, same constant) — so an internally valid proof of a *neighbouring* fact cannot silently license the claim.

The ledger converts “is this pipeline rigorous?” from a judgement call into a query: read the list. And it converts rigour-improvement from a rewrite into a diff: shrink the list, one discharge at a time. The rule catalogue’s coverage dashboard (chapter 15) is the same move applied to the assurance edifice itself: a declared, per-rule account of which correspondences are certified and which remain pending, in data rather than in reputation.

**When it applies.**

- Whenever a formal artefact meets analysis (or any outside theory): the bridge facts — Parseval, Grönwall, a comparison principle, a solution concept’s energy inequality — are exactly the ingredients a polynomial/arithmetic checker cannot re-derive. They must go *somewhere*; the only choice is ledger or shadow.
- Whenever you catch yourself writing “standard” or “well-known” next to a fact your checker never sees. Standard facts have names; name them in data, with a status.
- When planning a campaign: rungs you cannot yet prove are not blockers if they can be *declared* — the pipeline ships sound-modulo-ledger on day one, and the ledger is the campaign’s remaining-work list, machine-readable.

**Worked example from the record.** The Wirtinger discharge arc in the Burgers energy certificate. The sharp-rate claim  $\frac{d}{dt}E \leq -2\nu E$  needs the Wirtinger/Poincaré inequality  $\int u_1^2 \geq \int u_0^2$  for mean-zero data on  $\mathbb{T}_{2\pi}$  — an analytic fact the exact- $\mathbb{Q}$  jet checker cannot see. It shipped **declared-assumed**: a named prior in the certificate, with the mean-zero hypothesis’s *invariance* itself certified (a pure divergence row), and the declaration stated out loud in the campaign record. The pipeline was already refusing laundering at this stage: the Fisher-KPP sibling’s battery included both laundering mutants (hypothesis quietly dropped; status upgraded receipt-free), machine-refused.

A later session, the same day, discharged the prior *in-certificate*: on the torus it diagonalizes exactly, so the checkable kernel is the mode polynomial  $q(k) = k^2 - 1 \geq 0$  on  $|k| \geq 1$  — certified by an exact- $\mathbb{Q}$  shift row ( $q(1+t) = t^2 + 2t$ , nonnegative coefficients, covering every mode uniformly), with sharpness as a width-zero eigenvalue enclosure  $\lambda_1 \in [1, 1]$  (lower side the shift row, upper side the witness mode  $k = 1$ ). The ledger flips: *no assumed priors remain*. Three details carry the section:

- **The tie.** The checker verifies the consuming row's integrand is  $u_1^2 - c u_0^2$  at the discharged constant  $c$  — an internally valid discharge of a different constant cannot silently license the claim, and a consistency mutant enforces exactly that. Discharge without a tie is a new laundering surface; the arc closed it the same session it opened.
- **The residue is itself ledgered.** The discharge consumes four analytic bridge facts of its own (Parseval, derivative-coefficient, mean-is-coefficient-zero, termwise summation). They did not vanish into prose: each is a named bridge: entry citing its mathlib lemma at a pinned commit, citations checked live at writing time, with one remaining gap named (the  $H^1$  extension of the derivative form, which holds as cited at  $C^1$ ). The ledger got *shorter and sharper*, not empty by fiat.
- **Laundering mutants re-targeted post-discharge.** With a discharge object present, the new lie is tampering *inside* it, or claiming spectral-certified status without it; the battery grew nine mutants aimed at the ledger and the discharge both. The status vocabulary stayed closed the whole way.

Cost calibration, because trained priors on verification cost run years stale: the whole arc — declare, ship sound-modulo-ledger, discharge with sharpness, tie, re-target the battery — completed within one working day across three sessions. Declaring first did not slow the campaign; it is what *let* the campaign ship while the discharge waited its turn. And the banked bonus paid forward the same day: the discharge artefact pre-certified the higher-order siblings ( $q = k^4 - k^2, k^4 - 1$ ), which the Kuramoto–Sivashinsky target then consumed as its spectral priors.

#### The failure mode it prevents.

- **The silent assumption import:** a “verified” pipeline resting on an unnamed analytic fact — the reader believes the checker saw everything; the gap surfaces in adversarial review (or in production) as “what exactly does ACCEPT mean here?”, and the answer costs the pipeline its credibility even where it was sound.
- **Laundering under deadline:** statuses drifting upward (“assumed” → “known” → “proven”) across retellings, with no machine in the path to refuse the drift. This is the certificate-level twin of the documentation failure in which a distilled guess hardens into quoted law: the repair in both cases is provenance enforced at the artefact, not vigilance in the prose.
- **The wrong-theorem discharge:** proving a true neighbouring fact (different constant, different function space) and letting it license the claim — refutable only if the tie between discharge and consumer is checked, which is why the tie is part of the move and not a refinement of it.
- **All-or-nothing rigour:** without a ledger, “not fully proven” and “not shippable” collapse into one, and campaigns stall at their hardest lemma. With one, soundness-modulo-a-shrinking-list is a publishable, checkable state.

#### Exercises.

1. **The Lean bridge lemmas.** The four bridge: entries in the discharge (plus the linear-mechanism spectral-support obligation in the Kuramoto–Sivashinsky artefact) are named with mathlib citations at a pin. Formalize the statements as Lean lemmas against that pin, flipping each ledger entry from cited to proof-backed — and add the battery mutant that tampers with the new receipts.
2. **The named gap.** The derivative-coefficient bridge holds at  $C^1$  data; the  $H^1$  extension is an explicitly named gap in the ledger. Close it — part of the exercise is determining whether the gap closes at the pin or needs an upstream mathlib contribution.
3. **A new prior species, ledger-first.** The  $L > 2\pi$  Kuramoto–Sivashinsky benchmark needs interpolation (Gagliardo–Nirenberg-type) priors — the first genuinely new prior family since Wirtinger. Introduce them the ledger-first way: declared species with closed statuses and laundering mutants *from birth*, so the discharge arc has its rails before anyone attempts the discharge.

## 19.5 Cheap Before Expensive

**The move.** Before committing the expensive resource — a build, a rented-GPU run, a finer subdivision, a proof campaign, a frontier-model token stream — spend the cheap instrument that could make the expensive spend unnecessary, radically smaller, or better-aimed. The move has three forms, in ascending machinery:

1. **The cheap question.** Does the record, or a free qualitative argument, already settle it? Read the project log before the build; check for the monotonicity observation before assembling the sharp ladder. The cheapest ending to an expensive plan is discovering it already happened, or was never needed.
2. **The cheap pass.** When cost scales with a count (cells, claims, pieces, obligations), run the inexpensive consolidation *before* paying the per-unit price — and demand the consolidation be loss-free where the theory permits it (same bound, fewer pieces), so the saving is pure.
3. **The cheap measurement.** When two routes to the same certified quantity compete, race them at *equal budget* on a real instance before scaling either. “Which route wins” is an empirical question with a cheap experiment, not a doctrine — and the fair comparison is at matched spend, since anything beats anything given more budget.

Two boundary clauses guard the move. First, it orders instruments; it never licenses deferral — a plan that stops at the cheap instrument needs a stated mechanism for when the expensive one fires, not a vibe; and when the expensive machinery gets built anyway, it is *banked, not wasted*: retained explicitly for the sibling case where the cheap argument does not apply. Second, the currency is latency and aim, not primarily money: in a regime of abundant implementation capacity, what the move conserves is the time the expensive spend takes, the attention it occupies, and the risk of aiming it at a target the cheap instrument would have moved.

### When it applies.

- About to dispatch or build something whose success condition might already be satisfied. The project record is the cheapest instrument available, and the check extends to every dispatch, not only the ones that feel uncertain.
- About to turn a budget dial — subdivision depth, piece count, degree, machine-hours, review rounds — because the current answer is not tight enough. First ask: is there a cheap restructuring (a coarsen pass, a representation change) that dominates dial-turning? The recognition sign is a cost that scales like  $\mathcal{O}(h^{-k})$  against an error shrinking like  $\mathcal{O}(h)$ : the dial is the losing side of an exponent gap.
- Holding a certified artefact whose slack is visible (margins, lift values, mutant-refusal exponents): slack is prepaid budget, and a cheap pass can often spend it on consolidation with zero loss.
- Routing work to minds: at any date there is a quality/cost/speed frontier of available models, and selection is by context, not by rank — the cheap capable point first, the frontier model where context-understanding or taste binds. The same logic governs how much deliberation to request from a given model: effort is a dial, and the cheap setting is the right default for work that does not need the expensive one.

**Worked examples from the record.** **The consumed dispatch.** An overnight agent was dispatched toward what its brief framed as a live build: an exact certifying-tier GPU kernel family, on a rented machine. The agent’s first instrument was one read of the project log — which showed the tier had shipped *that morning* (fused exact Bernstein slot kernels, 11/11 device-versus-host-reference bit-exact, with two successor tasks already consuming them). Verdict: dispatch adjudicated consumed; a candidate follow-on was named without being fired; total agent runtime 132 seconds. What a rented-machine campaign would have re-built, a two-minute read retired.

**The slack-funded coarsen pass.** A family of fixed-shift projection-kernel certificates was born-certifiable with lift 0 — the theorem’s manufactured slack absorbed all float noise. That slack is prepaid budget, and a cheap coarsen pass spent it: one representative solution went from 640 uniform pieces to 26 nonuniform dyadic pieces (89 claims), *bound bit-identical*, lift still 0 — the receipts’ own headline was “coarsening pays

14 $\times$ .” Every downstream cost of that row (certify, reference-check, emit, fast-check, per claim) shrank by the same factor before any expensive tier ran. One detail worth copying: the coarsen pass’s excess ledger initially under-billed (an interpolant’s convexity overshoot went unbilled), and it was the pinned test suite that caught it — the cheap pass runs under the same verification discipline as everything else.

**Tube versus hull at equal claim budget.** A design note posed its own pricing question — a curved-boundary “tube” upgrade “pays exactly when hull conservatism is the binding loss — measure first” — and the measurement ran both routes to a certified enclosure of the same switching integral, same exact tier, budget counted in claims. At about 10 claims the tube already matched the hull’s 112-claim width; at 59 claims the tube was  $\sim 1300\times$  tighter than the hull at 237; matching the tube’s width by dial-turning the hull would extrapolate to subdivision depth  $\sim 17$ ,  $\mathcal{O}(10^5)$  claims. The receipts’ verdict — on this benchmark the tube wins by two to four orders of magnitude at every matched budget — was scoped to the benchmark, with both scaling laws measured. One cheap head-to-head, and the budget dial was retired as the losing instrument for this loss class *before*, not after, a  $10^5$ -claim campaign.

A sibling specimen from the proof side: a closed-loop encounter certificate needed “stays separated forever after exit,” and the author built the full ray-clearance LP ladder — then found the certified exit condition makes the exit face *recede*, so post-exit eternity is free model algebra on already-certified facts. The LP machinery was demoted as unnecessary for that class and explicitly retained for the non-receding sibling classes: the cheap question asked late still pays, and the expensive machinery, banked, is not waste.

### The failure mode it prevents.

- **The duplicate build:** re-doing shipped work because the record went unread. The cost is not only the spend — a re-built sibling diverges from the shipped one, and reconciling them later costs more than either.
- **Dial-turning into an exponent gap:** paying  $\mathcal{O}(h^{-k})$  for  $\mathcal{O}(h)$  relief because scaling the incumbent felt like progress and restructuring felt like a detour. The receipts above put real numbers on how bad this gets: five orders of magnitude of claims to match a route change.
- **Doctrine substituting for measurement:** “the sophisticated route wins” (or “the simple route suffices”) adopted from taste, when the equal-budget race costs minutes. The tube-versus-hull brief explicitly allowed the inverse outcome — the measurement declined it, and only the measurement could.
- **Thrift theatre:** the inverted failure — reading this section as “spend less.” The expensive spend is often correct; if you are an agent whose cost intuitions were trained on older tooling, your priors on verification cost in particular run stale and high (the receipts throughout this chapter say hours where the prior says months). The move is about what you learn *before* aiming the big spend, never about flinching from it.

### Exercises.

1. **The agreement-matrix verb.** The consumed dispatch above named a follow-on without firing it: a consolidated float-versus-exact agreement-matrix verb over existing machinery, its content currently distributed across two campaigns’ receipts. Build it as a standing cheap instrument (one CLI verb) so the cheap question “do the tiers agree, where, at what exponent?” costs one invocation before any future expensive exact-tier run. Success condition: the verb runs against the committed receipts and emits the matrix; its output for one known row matches the published receipt.
2. **Coarsening at degree 2.** The projection-kernel receipts state the boundary plainly: the degree-2 fixed-shift row is a-priori sizing only — no monotone theorem at degree 2 (the Korovkin boundary, measured in the same receipts as a step change in monotonicity violation). The coarsen pass is theory-backed only at degree 1. Determine whether a-posteriori coarsening pays at degree 2: coarsen, re-certify, compare bound and claim count against the uncoarsened row. Success condition: a new receipts row with pieces/claims/bound/lift columns, whichever way the verdict falls — an inverse outcome would be a finding rather than a failure.
3. **The next binding-loss measurement.** The tube route won where hull conservatism was the binding loss on a two-dimensional benchmark with one curved boundary. Before extending tubes



(higher dimension, multiple switching surfaces), run the pricing question forward: construct the next benchmark, measure the hull baseline's scaling first, and only build the tube generalization if the exponent gap is present there too. Success condition: a receipts-shaped table for the new benchmark with both routes at matched budgets — or the measured absence of the gap, banked as the reason the generalization waits.

## 19.6 Scope and Frame

**The move.** Every claim lives in a frame — a mechanism, an interface, a semantics under which its sentences are read — and most catastrophic errors in certificate work are not wrong identities but *right identities read in the wrong frame*. The move is to pin the frame as a first-class, machine-checked object before building, and then enforce scope in three places:

1. **Premises transport only where licensed.** A fact proven in one frame (one mechanism, one solution regime, one data class) does not follow the notation to a neighbouring frame. Record, per premise, the frames where it holds and the mechanism that carries it there — an invariance row, a bridge obligation — as *checker data*, so that using a premise outside its license is a refusal, not an oversight.
2. **Changes enter only through the seam.** When an untrusted component is upgraded or swapped, the modification's entire footprint is the declared interface; everything on the trusted side runs byte-for-byte unchanged, and that claim is grep-verifiable, not asserted.
3. **Out-of-scope reads go vacuous, never false.** When the frame itself is discovered ambiguous — the same certificate admitting two readings, one of them unsound — the repair is semantic before it is mechanical: declare the conditional reading normative (the certificate asserts its conclusion *given* its declared environment), so a certificate whose hypotheses cannot be jointly met claims nothing, rather than something false. Then add the mechanical check that makes the declaration well-formed.

The common discipline: the frame is stated where the machine can refuse against it. A frame that lives in prose (“this only applies to the linear case,” “don’t touch the checker”) decays into folklore; a frame that lives in a transport table, a frozen function signature, or a normative semantics section with its checker pass is enforced by machine. The composition-pattern inventory (chapter 13) runs the same discipline at the model tier: which certificates transport across a composite boundary, and which are refused, is part of each pattern's definition.

### When it applies.

- Any artefact with more than one mechanism, regime, or model of the same system — linearizations beside nonlinear flows, discrete beside continuum, per-piece beside global. The recognition question: “could every *identity* in this proof be true while the *theorem* is false, because a premise crossed a mechanism boundary?” If the answer is not obviously no, the transport structure is missing.
- Any planned upgrade of a component inside a verified pipeline. The recognition sign is the phrase “and we’ll just also need to tweak the checker” — the seam is about to leak; freeze the interface first and make the swap's footprint checkable.
- Any review finding of the shape “under reading A this is sound, under reading B it accepts a lie.” That is not a bug in a check; it is an unpinned semantics, and patching the check without declaring the reading leaves the ambiguity for the next consumer.
- At design time, before requirements: the frame question (“what kind of thing is this, really?”) outranks every detailed requirement, because a wrong frame silently poisons all of them. The workshop's own worst-case specimen was a requirements document written against an implicit wrong frame — the modelling substrate read as an expressive logic rather than as a logical framework — whose review verdict was that most of the work needed re-doing. The frame question costs a paragraph; its absence costs the document.

**Worked examples from the record.** **Mechanism-scoped premise transport.** The Kuramoto–Sivashinsky certificate of sections 19.2 and 19.3 certifies two decay claims on two mechanisms: the nonlinear flow (energy monotone, sharp at  $\alpha = 0$ ) and the linear mechanism restricted to modes  $|k| \geq 2$



(rate  $-24$ , sharp at mode 2). The spectral-gap premise is licensed for the linear mechanism *only* — under the nonlinear flow, mode 1 is pumped from pairs like  $(3, -2)$  and the premise is not invariant — and that license is not prose: every mode-exclusion premise carries a transport table keyed by mechanism (invariance-row kind for mean-zero, bridge-obligation kind for spectral support). The battery's centrepiece rewires the gap-sector rate claim onto the nonlinear mechanism: every jet identity, weight, and spectral replay is green, and rejection comes only from the transport table. Without the transport structure, nothing else in the certificate would have objected.

**The frozen seam under a proposer swap.** When the real validated-numeric library replaced the in-house float proposer (section 19.1, seen here from the scope side), the entire change entered through one frozen interface: the adapter builds its Taylor-model polynomial and feeds it through the frozen proposal entry point, and the exact fixed point, split-tree certification, and checker run byte-for-byte unchanged. The claim is grep-verifiable and was re-verified at this chapter's writing: the override parameter appears in the proposer module (the seam's definition) and the adapter's own files, and nowhere in the checker — the checker's text is untouched by construction, not by promise. The payoff compounds: because the swap's footprint was the seam, an isolation probe could later attribute a residual certification wall to the in-house proposer's search loop rather than to the library — a diagnosis that only type-checks when proposers are interchangeable behind a frame that never moved.

**Vacuous, not false.** An adversarial review round found that a new integral-monotonicity rule in the jet certificate schema, read under plain all-pre-valuations sequent semantics, admitted a false ACCEPT: a precondition constraining a bound integral wire, plus free-sign equality cancellation, licensed a conclusion no trajectory satisfies. The repair came in the order this move prescribes — semantic half first: the contract declares the binding-environment reading normative (a certificate asserts its sequents over environments satisfying its preconditions *and* every binding it declares — the declared-hypothesis pattern), under which the attacking certificate makes the hypothesis set unsatisfiable and the certificate *vacuous* — never a false assertion, because under this semantics no environment is claimed. Then the mechanical half: a new certificate-global binding-functionality pass (duplicate bindings of one wire reject; the negative control fires in isolation) makes the declared environment a well-formed function of wires. The residue is scoped rather than hidden: until the registered-environment lane ships, a consumer composing these certificates owns the hypothesis-set satisfiability check the same way it owns assumed rows — stated in the normative section, where a composer will meet it.

### The failure mode it prevents.

- **Premise smuggling:** a true lemma licensing a false theorem across a mechanism boundary. This is the failure that green batteries cannot catch from below — every constituent identity replays — which is why the transport table has to exist as its own refusal channel.
- **Seam leak:** an “upgrade” that touches the trusted side “just a little,” after which nothing about the pipeline is a plug-in and every future swap re-opens verification of the world. The byte-for-byte-unchanged claim, grep-backed, is what keeps upgrade cost at zero on the trusted side.
- **The silently-false certificate:** an artefact sound under its author's intended reading and unsound under the reading a consumer will actually use. Declaring the semantics normative in the contract — and making the unsound reading *forbidden* rather than merely undiscussed — converts a latent soundness hole into a documented conditional.
- **Frame-blind requirements:** detailed specification against an implicit wrong frame, discovered at review, fixed by “most of this needs re-doing.”

### Exercises.

1. **The registered environment.** The jet schema names the successor leg for its own residue: carry the declared binding set as first-class certificate data (a registry object) with a sound existential-elimination/composition rule, so consumers compose certificates without carrying the conditional semantics by convention — the frame promoted from normative prose to checked data, this section's move applied to its own example. Success condition: two certificates composed through the rule, with a mutant pair (a composition whose environments conflict) refused at a named check.

2. **Transport tables as shared machinery.** Mechanism-scoped premise transport currently lives in the Kuramoto–Sivashinsky artefact’s checker. Extract it into the PDE campaign’s shared checker machinery so the Burgers and Fisher–KPP artefacts consume the same transport module — the fan-out built it once where it was needed; the unification makes it the family’s frame discipline. Success condition: all three batteries green through the shared module, plus one new transport mutant per retrofitted artefact.
3. **Unrepresentable out-of-frame states.** The modelling substrate’s tensor-contraction definition (chapter 5) was deliberately ratified in the form whose side condition — every un-hit index is an output — makes the value-relevant free choice (a dropped axis silently scaling results) unrepresentable rather than discouraged; the side condition is exactly the proof-irrelevance theorem. Realize contraction rows with that condition enforced by construction, and write the property test that witnesses proof-irrelevance (no auxiliary choice affects any value). Success condition: the lowering rejects ill-framed index maps at the type/row level, and the property test runs green against the reference semantics.

## 19.7 Separate the Timescales

**The move.** When a system’s rates split into well-separated bands, do not pay the stiffness tax in the checker — restructure the *proposal*. Timescale separation is a **producer strategy**: everything below lives in the untrusted lane and in certificate *structure*; the checker never learns what “stiff” means, and the separation parameter  $\varepsilon$  enters certificates only ever as an exact rational coefficient in rows, never as a concept — no new rule family, no new trust surface. Four sub-moves, run in order:

1. **Diagnose** (cheap, float): decide whether you are in the situation at all (recognition signs below). The diagnosis routes; it is never a claim.
2. **Factor** — reparameterize into fast/slow coordinates, as rungs of the toolkit’s reparameterization ladder. The first rung (shipped) is eigenmode coordinates: the modal matrix of the linearization is float-proposed and dyadic-snapped — the proposer/disposer pattern again — with the declared inverse verified by exact multiplication and the pullback identities replayed, so the frame travels with the certificate as an interface hypothesis whose identities the checker replays. The eigenvector matrix is the standard fast/slow splitter, and it costs nothing to de-trust. Higher rungs cover patch-local slow sets (per-patch affine; chart maps with certified-positive denominators) up to the ladder’s ceiling — hypernet maps for genuinely nonlinear slow manifolds, where the reparameterization is itself certificate content cited by digest. The ceiling is frontier at this preview: named, wire-format slot reserved, semantics awaiting their first consumer.
3. **Witness** — make “separated” a *checkable claim*, not an assumption. Two readings, used per what your matrix admits. The quadratic-form/Lyapunov reading (shipped): a fast-block rate floor  $\alpha_f$  via positive-semidefiniteness rows ( $P_f \succ 0$ ,  $Q_f - \alpha_f P_f \succ 0$ ), a slow-block rate ceiling, and the **gap row**  $\alpha_f \geq 2r\alpha_s$  as one exact comparison — and this reading works for nonsymmetric linearizations, which is the generic case. The eigenvalue reading (proposed, not shipped): certified spectral windows and an inertia rule — eigenvalue *counting* across shifts, the one conclusion shape the shipped roster does not span. When those ship, “the spectrum splits into  $(-\infty, -1/\varepsilon] \cup [-c, 0)$ ” becomes a specific certified sentence.
4. **Compose** — certify the blocks separately, then buy the coupled conclusion with the classical two-timescale composite  $W = w_s^T P_s w_s + \lambda w_f^T P_f w_f$  (Khalil, *Nonlinear Systems* [136], ch. 11, in linear-quadratic form). For polynomial and linear blocks the composite’s rows are ordinary conclusions on the full coupled system — the weighted sum *is* the certificate, and the weight  $\lambda$  is the producer’s small-gain balancing act, float-searched and exactly disposed. The general rule family is in the catalogue: SMALLGAIN (chapter 15), the calculus’s feedback-composition rule, which elaborates to a single SYZGY node whose exact conclusion identity at weights  $(1, \lambda, 1, 1)$  makes the cross-supplies cancel by construction — nothing fixpoint-shaped survives to checking time. Timescale separation is the special case in which the gain asymmetry makes the small-gain product easy: the fast block’s gain to the slow one scales like  $1/\alpha_f$ , so a real gap buys the composition its slack. And carry **the full error term**: whatever the slow-manifold or modal-frame approximation leaves over is a *certified remainder* inside the composite rows, never a dropped term. In the linear case this remainder is

literally visible — the exact off-diagonal entries the dyadic snap leaves in the modal frame. In the nonlinear case it is a certified enclosure of the approximation error: Taylor models today, with a fancier-approximants ladder (Chebyshev models [42], Remez with a-posteriori error certification, Padé with certified-positive denominators) in design at this preview.

**When it applies.** Recognition signs, each a cheap float-lane probe:

- **Eigenvalue-ratio probe:** eigenvalues of the linearization at a few representative points; a ratio  $\gtrsim 10\times$  between rate bands is the signature (one eigvals call).
- **Enclosure anisotropy:** cover refinement concentrating splits along particular directions everywhere — cells forced small in two axes at once by a skewed feature is exactly what a reparameterization fixes.
- **Step-size collapse:** an explicit integrator's stable step is orders of magnitude below the timescale of the observables you care about — classical stiffness, and the reason the certificate's cover would otherwise inherit the fast timescale globally.
- **Search-space conditioning:** certificate searches starving because coordinates live at wildly different scales. A measured instance from the workshop record: an Adam-driven certificate search starving on value ranges spanning 1 to  $10^6$ , fixed by a coordinate-scaling knob — the same pathology one level up, in the proposal lane's own dynamics.
- **The domain says so:** filters, controllers, chemical networks, neural dynamics — fast transient plus slow drift is the modal architecture of engineered and biological systems both. The workshop's worn witness is a deployed interactive-tracking system whose fast loop runs at 72–120 Hz against a 0.03–1 Hz slow loop, three orders of magnitude apart, named in its own documentation as the decision everything else rests on.

**Worked example from the record.** A deliberately linear toy, taken end to end through all four moves on shipped machinery: a two-fast/one-slow coupled linear system — slow rate  $\sim 1$ , fast rates  $\sim 64, 80$ , all-dyadic entries — exact throughout:

- float eig proposes the modal matrix; dyadic snap at  $2^{-12}$ ; exact inverse declared; the checker replays  $T \cdot T^{-1} = I$  and  $\tilde{A} = T^{-1}AT$  in  $\mathbb{Q}$ ;
- the snap's leftover off-diagonal coupling ( $\sim 7 \times 10^{-3}$ , exact) is the certified remainder, carried inside the composite;
- fast rate floor  $\alpha_f = 117097/1024$  through the *unmodified* checker, slow rate  $\alpha_s$  exact (a  $1 \times 1$  block's rate is one number), gap row  $\alpha_f \geq 2 \cdot 32 \cdot \alpha_s$  — certified per-state ratio  $57.9\times$ ;
- the two-timescale composite with  $\lambda = 8$  from a 12-point float search: composite rate  $\alpha_c = 455/256 \approx 90\%$  of the slow quadratic bound — the slow mode governs, which is the separation story's signature;
- 0.81 s end to end, a 554-byte certificate, 3/3 producer refusal probes and 8/8 checker mutants refused. The probe worth naming: cranking the back-coupling flips the slow mode *unstable* (the Schur-complement correction crosses zero) and the pipeline refuses at the exact modal row — the classical quasi-steady-state trap, caught by arithmetic rather than by vigilance.

The toy is deliberately linear — every claim rests on rows that exist at this preview. What the linear case cannot exhibit, and the frontier behind it: hypernet slow-manifold reparameterization (frontier), the eigenvalue-reading witness (proposed), and nonlinear certified remainders (in design). Each item names work in flight.

**The failure mode it prevents.**

- **Paying the stiffness tax in the cover:** without the factoring, a certificate over the coupled system inherits the fast timescale everywhere — cell budgets explode along directions the slow claim never needed. The reparameterization moves the tax into one exact similarity replay.

- **The quasi-steady-state trap:** classical singular-perturbation practice *assumes* the fast subsystem settles and substitutes its equilibrium — sound only under conditions (attractivity, weak back-coupling) that the classical workflow rarely certifies. The witness sub-move makes exactly those conditions the checkable content; the toy’s refusal probe is the trap sprung on purpose.
- **Monolithic non-reuse:** one big Lyapunov certificate for the coupled system is re-derived from scratch whenever either block changes. The composite factors the work: the fast block’s certificate is a reusable component, and only the small-gain arithmetic re-runs when the slow side varies — the same economy the assume-guarantee layer exists to buy (chapter 13), arriving here through the gap.

### Exercises.

1. **Land the eigenvalue reading:** the inertia and spectral-window rules are scoped with priced proof obligations in the workshop record. When they ship, upgrade the toy’s gap row to a certified spectral-window sentence and re-run the mutant battery.
2. **Certify a worn witness:** take a production system you operate that already runs a fast/slow split as engineering practice (the tracking-filter shape above). State the two loops as a two-mechanism system, witness the rate gap, and compose — the first deployed-system instance of this section.
3. **Van der Pol at large  $\mu$ :** the classic nonlinear two-timescale system. The slow manifold is a cubic — mid-ladder territory, with the approximants ladder’s certified remainder as the error term once it ships. Success condition: a certified relaxation-oscillation enclosure whose cover is coarse along the slow manifold.
4. **Timescale  $\times$  parameter box:** compose this section with the campaign’s parameter-boxed oscillator artefact (uniform decrease over a damping box) — a fast/slow system whose *rates* live in boxes, certifying separation uniformly over the box. One new fixture, no new rules.

## 19.8 Anchor at the Known Point

**The move.** When you must certify something about an object you cannot compute — a fixed point, a limit, an optimum — do not analyse the iteration that approaches it. **Invert the anchoring: state every hypothesis at the concrete artefact you hold** (the snapshot, the candidate, the last iterate), and let one contraction-style row transport the conclusion to the unknown object. Concretely, for a fixed point  $\mu_*$  of an operator  $T$  and a snapshot  $\hat{\mu}$ :

1. certify a **two-point contraction** on a ball around  $\hat{\mu}$ :  $d(Tx, Ty) \leq \rho d(x, y)$  with  $\rho < 1$ , as checkable rows;
2. certify the **one-step residual** at the snapshot:  $d(T\hat{\mu}, \hat{\mu}) \leq \delta$  — one application of  $T$  to a concrete object, always computable;
3. conclude  $d(\hat{\mu}, \mu_*) \leq \delta/(1 - \rho)$ , with existence *and* uniqueness of  $\mu_*$  in the ball manufactured by the contraction itself (Banach) — no convergence analysis, no iteration count, no history of how  $\hat{\mu}$  was produced.

The inversion matters because the literature’s theorems are usually anchored at the unknown point (“the iteration contracts *toward*  $\mu_*$ ” — a one-point statement at a place you cannot evaluate anything), which is useless as a certificate hypothesis. A certificate’s hypotheses must be checkable, and only the known point supports checking. The two-point form costs more analysis once, in the untrusted lane; every certificate after that is arithmetic at  $\hat{\mu}$ . This section is the producer half of a seam: the checker half is the catalogue’s QFIXBALL rule (chapter 15), whose hypotheses are anchored at the known snapshot by construction and whose refusals include  $\rho \geq 1$ .

The producer-side companions, from the same worked example:

- **Reduce before you certify.** If the operator provably maps everything into a small family (a pinning lemma: any output of  $T$  is Gaussian with known precision, whatever the input), the measure-space problem collapses to a finite-dimensional one — and the reduction is *forced*, not assumed, so the certificate still speaks about the full space.

- **Guess-and-check the spectral row.** The contraction factor  $\rho$  needs a norm bound on a nonnegative matrix; propose a Perron row vector with floats, snap to rationals, and let the checker verify the row inequality. When bare power iteration oscillates (period-2 on a two-block system), iterate  $(R + I)/2$  instead — same fixed vector, damped rotation. Both proposals are sound to check; the fix buys *sharpness*, not soundness — the checker is indifferent, and that indifference is the seam working.

#### When it applies.

- the claim is “the true object is within  $\varepsilon$  of *this*” — distance-to-fixpoint, distance-to-optimum, a posteriori error bounds — rather than “the algorithm converges”;
- every hypothesis you have tried to state mentions the unknown object’s location (a Lipschitz constant “near  $\mu_*$ ,” a curvature “at the optimum”) — the tell that the literature’s anchoring is backwards for your purpose;
- one application of the operator to a concrete object is cheap, even though finding its fixed point is not (one coordinate-ascent sweep versus the converged posterior; one Newton step versus the root; one Bellman backup versus the value function);
- a self-mapped ball is available: contraction plus residual imply  $T(\overline{B}(\hat{\mu}, r)) \subseteq \overline{B}(\hat{\mu}, r)$  for  $r \geq \delta/(1 - \rho)$ , so the hypotheses only need to hold on a set you can name in advance.

Interval-Newton and Krawczyk verification are this move’s classical ancestors in  $\mathbb{R}^n$ ; the move generalizes them to any complete metric space a reduction can reach.

**Worked example from the record.** Certified Wasserstein balls for variational inference. The task: certify that a CAVI (coordinate-ascent variational inference) snapshot is within  $\mathbb{W}_2$ -distance  $r$  of the true self-consistent fixed point.

- The literature’s theorems (contraction of the CAVI iteration [36]) are one-point statements at the unknown  $\mu_*$ . The certificate inverts: all hypotheses at the snapshot  $\hat{\mu}$ .
- The pinning reduction: every CAVI output is exactly Gaussian with precision  $Q_{ii}$  regardless of input, so the fixed-point problem in measure space collapses to an affine map on means — and the certificate quantifies over *all* product measures in  $\mathcal{P}_2$ , not just Gaussian candidates.
- Receipts: ACCEPT with certified  $\rho = 96/181$  against true contraction factor  $3/(4\sqrt{2}) \approx 0.53033$ ; ball radius  $3.6 \times 10^{-7}$ ; ground truth (computed independently, never consumed by the certifier) inside every ball with slack  $\times 1.9$ ; 11/11 mutants refused at named obligations.
- The period-2 Perron oscillation appeared live: the first run’s bare power iteration proposed a loose  $\rho = 3/4$ ; the  $(R+I)/2$  fix recovered sharpness. Both fixture generations were accepted by the checker — a live specimen of the proposer being fallible and the trust boundary not caring (section 19.1, observed end to end).

**The failure mode it prevents.** Without the inversion, the certificate route dead-ends in one of two ways. Either the producer imports the literature theorem as-is and the certificate’s hypotheses quantify over a neighbourhood of a point nobody can locate — unfalsifiable rows, which a checker must either trust (trust-surface creep) or refuse (no certificate at all). Or the producer certifies the *iteration history* (“ $n$  steps, each contracting”) — which couples the certificate to the trajectory that produced the snapshot, breaks on restarts and warm starts, and still needs the contraction constant somewhere. The anchored form is history-free: any snapshot from any source, certified on its own merits — a corrupted snapshot simply fails the residual row (refusal, not a lie).

#### Exercises.

1. **Block-Gaussian instance** — the worked example’s blocks were scalars; run the same pattern with matrix blocks, exercising operator-norm rows proper. Everything is staged; only the fixture generator changes.



2. **Discrete instance (2-spin Ising)** — the total-variation contraction is free at bounded couplings; this produces the first *discrete* Wasserstein-ball fixture and tests whether the move survives leaving the Gaussian reduction.
3. **The covariance fold-in** — fold covariance error into the ball (dropping the variance-exactness refusal); the disanalogy to price is that the mean-map affinity no longer collapses the whole problem.
4. **The transfer question**, worth a page before any build: which of your own stack’s implicit objects (stationary laws of samplers, Bellman fixed points in value-iteration producers) already have a cheap one-step residual and are one two-point contraction row away from an anchored ball?

## 19.9 Safety Walls and Liveness Tilts

**The move.** A specification that asks a system to be *robustly safe while optimistically achieving a goal* has two halves of opposite character, and most threshold-reading errors in probabilistic specification come from applying one half’s natural reading to the other. The move is to give the two halves opposite **polarities**, explicitly:

1. **Safety is a probability claim, read violation-form: a wall.** Budget the mass on the bad event —  $P(\text{bad}) \leq \beta$  — and hold violation infinitely inconsistent: the belief (the level-of-inconsistency functional of chapter 9) is 0 on the half-space where the budget holds and  $\infty$  where mass on the bad event exceeds it, with nothing in between. In particular, the run that halts, idles, or leaks mass incurs no inconsistency — *a system that stops has not hit the wall*. This is not leniency but correctness: safety is closure-invariant — adding stopping behaviours to a safe system cannot make it unsafe, and leaked mass can only lower the probability of reaching the bad event (chapter 9 carries this asymmetry as a theorem of the partial-model semantics).
2. **Liveness is an expectation claim, read as a tilt: a hinge.** Price the shortfall of the good event’s mass below its target  $\tau$  at a finite rate —  $\lambda \cdot (\tau - \mu(\text{goal}))_+$  — and charge nothing at or above the target. Beliefs take values in  $[0, \infty]$  (the Lawvere quantale: there are no negative levels of inconsistency), so the shortfall is a saturated difference, written  $(\cdot)_+$ , never a signed one. The hinge’s two sides carry the reading: the flat side means overachievement is free and buys nothing — in particular it cannot be traded against the wall — and the sloped side means a near-miss is *priced*, not merely refused, which is what “optimistically” means as mathematics.

The two readings are one family, not two species: as  $\lambda \rightarrow \infty$  the hinge steepens around its kink at  $\mu(\text{goal}) = \tau$  and converges — in the epigraph sense, the right sense for beliefs — to the hard threshold: 0 at or above the target,  $\infty$  below. A *hard* liveness threshold is therefore the  $\lambda = \infty$  endpoint of the tilt, satisfaction-form as it must be: the run that does nothing fails it, and should — that is what a hard liveness threshold *means*. Choosing  $\lambda$  is choosing where on the family the claim lives; the trap (below) is only ever reading a threshold at the wrong polarity, never choosing a finite  $\lambda$  on purpose.

Compose by maximum:  $\max(\text{wall}, \text{hinge})$  is the conjunction of beliefs, and it is **convex** — the wall is the  $\{0, \infty\}$ -indicator of a half-space, the hinge is a maximum of an affine function with zero, and a pointwise maximum of convex beliefs is convex. Certification inherits the cheapness: half-space walls, FARKAS combination, and the ReLU row family (ReLU-Pos, ReLU-Above and kin, chapter 15) — whose native object is exactly  $\max(\text{affine}, 0)$  — already span the composite. The payoff sentence: the standard reach-avoid specification, in its robustly-safe-while-optimistically-live form, costs nothing the toolkit does not already ship.

### When it applies.

- Any specification with a “never / at most” half and an “eventually / at least” half: reach-avoid, stabilize within bounds, achieve a return under a resource budget. The two halves are both thresholds, and they need opposite readings.
- Whenever the modelled system can halt, diverge, exit scope, or otherwise leak mass — partiality — so that the two readings of a threshold genuinely differ. The recognition question is: “what does this specification say about the run that does *nothing*?” If safety and liveness give the same answer, a polarity has been misassigned.

- Whenever you meet a bare threshold — yours or someone else’s — that does not carry its reading. The natural default flips sign between the two halves, so “which polarity did the author mean?” is a question to settle before consuming the row, not after.
- When composition or certification cost binds: the wall-plus-tilt composite stays convex, so the polarity discipline is also what keeps the composed specification in the cheap, Farkas-certifiable class.

**Worked example.** Three outcomes suffice: REACH (the goal), AVOID (the bad set), INERTIA (the run that never commits — halts, idles, diverges). A distribution  $\mu = (\mu_R, \mu_A, \mu_I)$  over the three is a point of the probability 2-simplex, and the specification reads: *reach with probability at least  $\frac{3}{4}$ , never more than  $\frac{1}{4}$  on the bad set*. The safety wall is the violation half-space  $\mu_A > \frac{1}{4}$  penalized at  $\infty$ ; the liveness tilt is the hinge  $\lambda \cdot (\frac{3}{4} - \mu_R)_+$ . Four distributions — the three vertices and one genuine mixture — against the wall, the hinge at two rates, and the hinge’s hard limit:

|                        | $(\mu_R, \mu_A, \mu_I)$                     | wall     | $\lambda = 4$ | $\lambda = 40$ | $\lambda = \infty$ |
|------------------------|---------------------------------------------|----------|---------------|----------------|--------------------|
| commits to the goal    | (1, 0, 0)                                   | 0        | 0             | 0              | 0                  |
| commits to the bad set | (0, 1, 0)                                   | $\infty$ | 3             | 30             | $\infty$           |
| never commits          | (0, 0, 1)                                   | 0        | 3             | 30             | $\infty$           |
| the mixed run          | $(\frac{7}{10}, \frac{1}{20}, \frac{1}{4})$ | 0        | $\frac{1}{5}$ | 2              | $\infty$           |

Read the columns against the doctrine. The **wall** distinguishes the two failures the hinge cannot: committing to the bad set hits it, never committing does not — failing dangerously and failing idly are different verdicts, and only the violation-form reading can tell them apart. The **hinge** carries the orthogonal information: the mixed run, with  $\mu_R = \frac{7}{10}$  against the target  $\frac{3}{4}$ , certifies at *every* finite  $\lambda$ , paying the priced shortfall  $\lambda \cdot \frac{1}{20}$  — the optimistic reading, a near-miss ranked rather than refused — while any distribution at or above the target pays exactly 0 on the flat side. And the  $\lambda = \infty$  **column** is the hinge steepened into the step: the mixed run and INERTIA now both fail, which is correct precisely when the claim intended is the hard one. The composite  $\max(\text{wall}, \text{hinge})$  needs both columns’ information: drop the wall and dangerous failure is merely priced (a large enough achievement elsewhere could dominate it); drop the hinge and every near-miss collapses into undifferentiated refusal.

Where this stands in the toolkit at this preview. The probability-threshold former SPROBGE (chapter 14) is a mass floor — lower probability at least  $p$  — i.e. the  $\lambda = \infty$  liveness endpoint, and the credal semantics makes it feel every lost unit of mass, exactly as a hard liveness claim should (chapter 9: a liveness claim cannot be laundered through a model that is allowed to quietly stop). Safety already runs at the other polarity internally: the safety batteries’ certified object is an upper bound on the violation mass ( $\overline{P}(\text{bad}) \leq V(x_0)$ , the threshold row of QSafetyBATTERY and kin), which is why those certificates remain sound under leak — the wall reading is already how the machinery works. The expectation formers (SRETURNGE/SRETURNLE, chapter 14) are the shipped expectation-claim species that carry the tilt: a hinge belief over an indicator channel is an expected-shortfall bound. And the polarity itself is declared in the grammar — as a sort, which is stronger than a field: the specification cap (chapter 14) types every threshold component by its face, so a mis-faced premise is a formation error, refused before any semantics runs, and each former’s reading is definitional in the certificate theory rather than a habit of its author. One disambiguation earns its keep here, because that chapter also says “safety face” and means a different axis by it. The cap’s polarity answers *who quantifies over the admitted laws*: a ForallSpec component binds every law, an ExistsSpec component asserts some law attains. This section’s polarity answers *which way a single threshold faces*: wall on the violation mass, floor on the goal mass. The two axes are independent, and one specimen shows how far apart they sit: SPROBGE — this section’s hard *liveness* endpoint — concludes ForallSpec, the credal *safety* face, because demanding the goal’s mass from every admitted law is a guarantee about a floor. What remains the author’s is this section’s own discipline: the grammar refuses a mis-faced component, and no grammar can refuse a wall aimed at the wrong event.

**The failure mode it prevents.** One trap, two signs, plus the composite’s own failure:

- **Liveness read violation-form forgives doing nothing.** Two outcomes; a system that succeeds with probability  $\frac{2}{5}$  and otherwise halts; the claim “reach success with probability at least  $\frac{1}{2}$ .” The violation-form misreading — refuse only if mass on the *failure* event exceeds  $\frac{1}{2}$  — sees zero mass



on failure and passes, but  $\mu(\text{success}) = \frac{2}{5} < \frac{1}{2}$ : the mass floor fails, and the correct verdict is refusal. Liveness laundered through leak — the quietly-stopping system certified for a goal it does not deliver.

- **Safety read satisfaction-form punishes halting as if it were violation.** Spell “avoid the bad set” as a floor on the safe-completion event and the safely-idling system fails, though it never approaches the bad set. The downstream rot is worse than the false refusal: an author who knows the system may stop weakens the floor to tolerate it, and the weakened floor now also tolerates genuine violation mass. The repair was never a looser constant; it was the other polarity.
- **The negotiable wall.** Price safety at a finite rate — treat it as one more tilt — and conjunction-by-maximum degrades into trade: enough achievement elsewhere dominates a bounded safety belief, and the optimizer will find that trade. “Robustly” means the wall is not for sale at any exchange rate, and only the violation form at infinite inconsistency says so.

Two polarities, no trap: liveness formers are satisfaction-form by definition, hard at  $\lambda = \infty$  and tilted at finite  $\lambda$ ; the safety machinery is violation-form by construction; and the reading travels with the claim as data instead of living in the reader’s habits.

### Exercises.

1. **The face mutants, now that the sort has shipped.** The cap’s polarity discipline — a mis-faced premise is a formation error — is a refusal, and this chapter’s own doctrine applies to it (section 19.3): a refusal without a firing witness is indistinguishable from dead code. Build the formation battery, each mis-faced construction fed to the grammar with its refusal location asserted — and include the one cell no formation check can own: the laundering-through-leak specimen above, a floor swapped for a wall on the complementary event, which is well-formed at both faces and wrong only about what the author meant. That last mutant’s home is the semantic tier (the flipped reading asserted against the partial-model semantics and refused there), and placing it correctly is part of the exercise.
2. **The epi-limit, certified.** The one-family claim — the hinge at rate  $\lambda$  converges in the epigraph sense to the hard threshold as  $\lambda \rightarrow \infty$  — is stated here as belief mathematics. Formalize it against the belief semantics, so that a hard liveness former and the  $\lambda = \infty$  tilt are interchangeable obligations in one worked certificate rather than two spellings the reader must trust to agree.
3. **The composite on a shipped witness.** The spec catalogue’s hitting-shape witness (goal reached by deadline with probability at least  $\frac{3}{4}$ ) is a hard liveness claim beside a safety wall. Restate it composite-form — wall on the bad set, hinge on the goal mass at two finite rates — and produce the certificate with half-space walls, FARKAS, and ReLU rows only; then check that the  $\lambda = \infty$  run of the same pipeline recovers the original hard claim. Success condition: the composite certifies with no row species beyond the shipped roster, and the two finite- $\lambda$  certificates price the same shortfall the table above prices.

## PART IV

# Appendices

---

# The Lean Cone

---

Throughout this book, claims that lean on a mechanized result cite the result by its Lean name, set in typewriter face. This appendix collects those citations into one place: for every cited declaration, it reproduces the declaration’s statement from the `lean/` trees of `maitria-coalition/qtsl`, with its file and line. We call this set the *Lean cone* of the book: the set of mechanized results the book’s prose binds to, closed under nothing — what you see is what the text actually cites, re-derived from the sources on every build, so the listing cannot drift from the tree it describes.

A word on how to read the entries. Statements are reproduced at signature level: a theorem up to its proof, a definition up to its body, a structure up to its fields (an ellipsis marks the elision), and long source lines re-wrapped to the measure of the page. Each entry is shown in the namespace where it lives, and surrounding context (open and variable declarations, notation) lives in the cited file, so an entry here reads as a faithful excerpt rather than a self-contained module. To verify an entry, clone the repository and run `lake build` inside the relevant Lake package under `lean/` (for example `lean/qtsl` or `lean/su-functor`); the toolchain is pinned by each package’s `lean-toolchain` file, and a build that succeeds has kernel-checked every statement listed here, since none of the trees admits sorry.

Two boundaries of the cone, so that its edges are as legible as its contents. First, the Rule Catalogue displays a Lean or HOL4 statement beside each rule already, extracted from the mechanized rule sources and hash-bound at emission; this appendix lists the catalogue’s citations that resolve into the `lean/` trees of this repository and defers to the catalogue’s own apparatus for its rule-source and HOL4 layers. Second, module citations (a dotted name denoting a whole file or directory of results) are listed per chapter without statements; the module’s file is the reference.

The listing follows the book’s chapter order, and each declaration appears once, under the first chapter that cites it; later chapters that cite the same result say so in a closing line rather than repeating the statement.

## A.1 The Semantic Universe

Citations from Chapter 9.

|                                                                                                                     |                                                                    |
|---------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| <code>Qtsl.DiamondSite.ChronologyKit</code>                                                                         | <code>lean/qtsl-aux/QtslAux/DiamondSite.lean</code> , line 79      |
| <code>structure ChronologyKit (X : Type*) [PartialOrder X] where ...</code>                                         |                                                                    |
| <code>Qtsl.DiamondSite.minkowskiKit</code>                                                                          | <code>lean/qtsl-aux/QtslAux/DiamondTopology.lean</code> , line 283 |
| <code>def minkowskiKit : ChronologyKit (Mink E) where ...</code>                                                    |                                                                    |
| <code>Qtsl.DiamondSite.diamondTopology</code>                                                                       | <code>lean/qtsl-aux/QtslAux/DiamondTopology.lean</code> , line 114 |
| <code>def diamondTopology (K : ChronologyKit X) :<br/>GrothendieckTopology (Diamond X) where ...</code>             |                                                                    |
| <code>Qtsl.DiamondSite.mem_of_not_thick</code>                                                                      | <code>lean/qtsl-aux/QtslAux/DiamondTopology.lean</code> , line 142 |
| <code>theorem mem_of_not_thick {D : Diamond X} (h : ¬ Thick K D)<br/>(S : Sieve D) : S ∈ diamondTopology K D</code> |                                                                    |
| <code>Qtsl.DiamondSite.mem_diamondTopology_degenerate_iff</code>                                                    | <code>lean/qtsl-aux/QtslAux/DiamondTopology.lean</code> , line 158 |

```

theorem mem_diamondTopology_degenerate_iff
  (hK :  $\forall a b : X, K.\text{chron } a b \leftrightarrow a \leq b$ ) {D : Diamond X} {S : Sieve D} :
  S  $\in$  diamondTopology K D  $\leftrightarrow$  S =  $\tau$ 

Qtsl.DiamondSite.int_lt_no_chronologyKit
  lean/qtsl-aux/QtslAux/DiamondTopology.lean, line 383

theorem int_lt_no_chronologyKit :
   $\neg \exists K : \text{ChronologyKit } \mathbb{Z}, \forall a b : \mathbb{Z}, K.\text{chron } a b \leftrightarrow a < b$ 

SuFunctor.Site.TTT.anchorQuot
  lean/su-functor/SuFunctor/Site/TemporalTypeTheory.lean, line 156

def anchorQuot : Window  $\Rightarrow$  IntervalBase where ...

SuFunctor.Site.TTT.anchorTrajIso
  lean/su-functor/SuFunctor/Site/TemporalTypeTheory.lean, line 323

def anchorTrajIso : anchorQuot.op  $\Rightarrow$  ssTraj X  $\cong$  windowTraj X

SuFunctor.Site.ttObject
  lean/su-functor/SuFunctor/Site/BehaviorSpec.lean, line 246

def ttObject (U :  $\iota \rightarrow \text{Set } T$ ) {A : WireSpan El} (B : WBehavior U (states A)) :
  SiteSpecTotal U El

SuFunctor.Site.satAt_specOfBehavior_iff
  lean/su-functor/SuFunctor/Site/BehaviorSpec.lean, line 208

theorem satAt_specOfBehavior_iff (U :  $\iota \rightarrow \text{Set } T$ ) (B : WBehavior U X) (j :  $\iota$ )
  (v : Joint (Traj (U j) X)) :
  SatAt v (specOfBehavior U B j)  $\leftrightarrow$  ( $\iota v.\text{wt.support} : \text{Set } (\text{Traj } (U j) X)$ )  $\subseteq$  B.cls j

SuFunctor.Site.wSpecMap_specOfBehavior
  lean/su-functor/SuFunctor/Site/BehaviorSpec.lean, line 256

theorem wSpecMap_specOfBehavior (U :  $\iota \rightarrow \text{Set } T$ ) {A B : WireSpan El}
  ( $\phi : A \rightarrow B$ ) (Bh : WBehavior U (states A)) :
  wSpecMap U  $\phi$  (specOfBehaviorW U Bh) =
  specOfBehaviorW U (Bh.pull (res  $\phi$ ))

SuFunctor.Site.TTT.mem_interiorTopology_iff_derived
  lean/su-functor/SuFunctor/Site/CausalConcordance.lean, line 110

theorem mem_interiorTopology_iff_derived {W : Window} (S : Sieve W) :
  S  $\in$  interiorTopology W  $\leftrightarrow$ 
  S.functorPushforward windowToDiamond  $\in$ 
  diamondTopology (linearOrderKit  $\mathbb{R}$ ) W.toDiamond

SuFunctor.Site.TTT.wavy_mem_iff_derived
  lean/su-functor/SuFunctor/Site/CausalConcordance.lean, line 133

theorem wavy_mem_iff_derived {W : Window} (S : Sieve W) :
  S.functorPushforward anchorQuot  $\in$  wavyTopology (W.len)  $\leftrightarrow$ 
  S.functorPushforward windowToDiamond  $\in$ 
  diamondTopology (linearOrderKit  $\mathbb{R}$ ) W.toDiamond

Qtsl.QuantumStates.recordShadowQ_separates_bell_mixed
  lean/qtsl-aux/QtslAux/QuantumStates.lean, line 939

theorem recordShadowQ_separates_bell_mixed :
  recordShadowQ zzRecord zzRecord_mem_top
  (stateClassMk qubitNet Dstar (bell, bell_isState))  $\neq$ 
  recordShadowQ zzRecord zzRecord_mem_top
  (stateClassMk qubitNet Dstar (mixed, mixed_isState))

Qtsl.DiamondSite.minkowski_diamond_spacelike_pair
  lean/qtsl-aux/QtslAux/DiamondTopology.lean, line 245

theorem minkowski_diamond_spacelike_pair {x0 : E} (hx : x0  $\neq$  0)
  (hx1 :  $\|x_0\| \leq 1$ ) :
   $\exists p q : \text{Mink } E,$ 
  (( $\langle 0, 0 \rangle : \text{Mink } E$ )  $\leq p \wedge p \leq \langle 2, 0 \rangle$ )  $\wedge$ 
  (( $\langle 0, 0 \rangle : \text{Mink } E$ )  $\leq q \wedge q \leq \langle 2, 0 \rangle$ )  $\wedge$ 
   $\neg p \leq q \wedge \neg q \leq p$ 

```

```

SuFunctor.Universe.FellClosededs.compactSpace      lean/su-functor/SuFunctor/Universe/FellTopology.lean, line 268
| instance compactSpace : CompactSpace (FellClosededs  $\alpha$ )

SuFunctor.Universe.fell_eq_vietoris                lean/su-functor/SuFunctor/Universe/FellTopology.lean, line 181
| theorem fell_eq_vietoris [CompactSpace  $\alpha$ ] [T2Space  $\alpha$ ] :
|   SuFunctor.Universe.fell  $\alpha$  = TopologicalSpace.vietoris  $\alpha$ 

SuFunctor.Universe.fell.continuous_restrictOpen    lean/su-functor/SuFunctor/Universe/FellRestrict.lean, line 81
| theorem continuous_restrictOpen (hU : IsOpen U) :
|   Continuous (restrictOpen U)

SuFunctor.Universe.fell.continuous_image_of_isProperMap lean/su-functor/SuFunctor/Universe/FellRestrict.lean,
line 130
| theorem continuous_image_of_isProperMap (hf : IsProperMap f) :
|   Continuous (Set.image f : Set  $\alpha \rightarrow$  Set  $\beta$ )

SuFunctor.Universe.FellClosededs.projLimHomeomorph lean/su-functor/SuFunctor/Universe/FellDescent.lean, line 315
| def projLimHomeomorph (U :  $\iota \rightarrow$  Set  $\alpha$ ) (hUo :  $\forall i$ , IsOpen (U i))
|   (hdir : Directed ( $\cdot \subseteq \cdot$ ) U) (hcov :  $\bigcup i$ , U i = univ) :
|   FellClosededs  $\alpha \approx_{\tau}$  ProjLim U where ...

SuFunctor.Universe.FellClosededs.IsLocalLocus.isClosed lean/su-functor/SuFunctor/Universe/FellRecordLocus.lean,
line 99
| theorem IsLocalLocus.isClosed (hloc : IsLocalLocus U  $R_0$  R)
|   (hUo :  $\forall i$ , IsOpen (U i)) (hRcl :  $\forall i$ , IsClosed (R i)) :
|   IsClosed  $R_0$ 

SuFunctor.Universe.FellClosededs.IsLocalLocus.isCompact lean/su-functor/SuFunctor/Universe/FellRecordLocus.lean,
line 113
| theorem IsLocalLocus.isCompact (hloc : IsLocalLocus U  $R_0$  R)
|   (hUo :  $\forall i$ , IsOpen (U i)) (hRcl :  $\forall i$ , IsClosed (R i)) :
|   IsCompact  $R_0$ 

SuFunctor.Universe.FellClosededs.IsLocalLocusEq.glue_mem lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.
lean, line 125
| theorem IsLocalLocusEq.glue_mem {U :  $\iota \rightarrow$  Set  $\alpha$ } {R0 : Set (FellClosededs  $\alpha$ )}
|   {R :  $\forall i$ , Set (FellClosededs  $\iota$ (U i))} (hloc : IsLocalLocusEq U  $R_0$  R)
|   (hUo :  $\forall i$ , IsOpen (U i)) (hcov :  $\bigcup i$ , U i = univ)
|   {p : Equalizer U} (hp :  $\forall i$ , p.1 i  $\in$  R i) :
|   glueEqualizer U hUo hcov p  $\in$  R0

SuFunctor.Universe.FellClosededs.speedLocus        lean/su-functor/SuFunctor/Universe/FellRecordLocus.lean, line 234
| def speedLocus : Set (FellClosededs  $\beta$ )

SuFunctor.Universe.FellClosededs.isClosed_speedLocus lean/su-functor/SuFunctor/Universe/FellRecordLocus.lean,
line 259
| theorem isClosed_speedLocus (ht : Continuous  $\tau$ ) (h $\sigma$  : Continuous  $\sigma$ )
|   (c :  $\mathbb{R}_{\geq 0}$ ) : IsClosed (speedLocus  $\tau$   $\sigma$  c)

SuFunctor.Universe.FellClosededs.isSpeedGraph_iff_forall_clip lean/su-functor/SuFunctor/Universe/
FellRecordLocus.lean, line 308

```

```

theorem isSpeedGraph_iff_forall_clip (hdir : Directed (· ≤ ·) U)
  (hcov : ⋃ i, U i = univ) (F : FellClosed β) :
  IsSpeedGraph τ σ c (F : Set β)
  ↔ ∀ i, FellClosed.clip (U i) F
    ∈ speedLocus (τ ∘ (↑)) (σ ∘ (↑)) c

```

SuFunctor.Universe.FellClosed.s**peedLocusHomeomorph** lean/su-functor/SuFunctor/Universe/FellRecordLocus.lean, line 346

```

noncomputable def speedLocusHomeomorph (hUo : ∀ i, IsOpen (U i))
  (hdir : Directed (· ≤ ·) U) (hcov : ⋃ i, U i = univ) :
  ↑(speedLocus τ σ c)
  ≈τ ↑(FellClosed.levelLocus U
    fun _ => speedLocus (τ ∘ (↑)) (σ ∘ (↑)) c)

```

SuFunctor.Universe.FellClosed.**bot\_mem\_speedLocus** lean/su-functor/SuFunctor/Universe/FellRecordLocus.lean, line 242

```

theorem bot_mem_speedLocus : (⊥ : FellClosed β) ∈ speedLocus τ σ c

```

SuFunctor.Universe.FellClosed.**isLocallyClosed\_speedLocus** lean/su-functor/SuFunctor/Universe/FellRecordLocus.lean, line 294

```

theorem isLocallyClosed_speedLocus (hτ : Continuous τ)
  (hσ : Continuous σ) (c : ℝ≥0) :
  IsLocallyClosed (speedLocus τ σ c)

```

SuFunctor.Universe.FellClosed.**isSpeedGraph\_iff\_exists\_lipschitzOnWith** lean/su-functor/SuFunctor/Universe/FellRecordLocus.lean, line 440

```

theorem isSpeedGraph_iff_exists_lipschitzOnWith [CompleteSpace E]
  [Nonempty E] {F : FellClosed (T × E)} :
  IsSpeedGraph Prod.fst Prod.snd c (F : Set (T × E))
  ↔ ∃ D : Set T, IsClosed D ∧ ∃ f : T → E, LipschitzOnWith c f D ∧
    (F : Set (T × E)) = (fun t => (t, f t)) '' D

```

SuFunctor.Universe.FellClosed.**topology\_onePoint\_eq\_vietoris** lean/su-functor/SuFunctor/Universe/FellOnePoint.lean, line 319

```

theorem topology_onePoint_eq_vietoris [WeaklyLocallyCompactSpace α] [T2Space α] :
  (FellClosed.topology : TopologicalSpace (FellClosed (OnePoint α)))
  = .induced (↑) (TopologicalSpace.vietoris (OnePoint α))

```

SuFunctor.Universe.FellClosed.**nablaGen** lean/su-functor/SuFunctor/Universe/FellCapacity.lean, line 75

```

def nablaGen (B : ι → Set Y) (hits misses : Finset ι) : Set (FellClosed Y)

```

SuFunctor.Universe.FellClosed.**isTopologicalBasis\_nablaGen** lean/su-functor/SuFunctor/Universe/FellCapacity.lean, line 136

```

theorem isTopologicalBasis_nablaGen [LocallyCompactSpace Y] [T2Space Y]
  {B : ι → Set Y} (hB : IsTopologicalBasis (Set.range B))
  (hBcl : ∀ i, IsCompact (closure (B i))) :
  IsTopologicalBasis
  (Set.range fun p : Finset ι × Finset ι => nablaGen B p.1 p.2)

```

SuFunctor.Universe.FellClosed.**measure\_ext\_of\_nablaGen** lean/su-functor/SuFunctor/Universe/FellCapacity.lean, line 278

```

theorem measure_ext_of_nablaGen [LocallyCompactSpace Y] [T2Space Y] [Countable ι]
  {B : ι → Set Y} (hB : IsTopologicalBasis (Set.range B))
  (hBcl : ∀ i, IsCompact (closure (B i)))
  [MeasurableSpace (FellClosed Y)] [BorelSpace (FellClosed Y)]
  {μ ν : Measure (FellClosed Y)} [IsFiniteMeasure μ]

```

```

(h : ∀ hits misses : Finset ι,
  μ (nablaGen B hits misses) = ν (nablaGen B hits misses)) :
  μ = ν

```

SuFunctor.Universe.FellClosededs.**exists\_nablaGen\_measure\_ne** lean/su-functor/SuFunctor/Universe/FellCapacity.lean, line 302

```

theorem exists_nablaGen_measure_ne [LocallyCompactSpace Y] [T2Space Y] [Countable ι]
  {B : ι → Set Y} (hB : IsTopologicalBasis (Set.range B))
  (hBcl : ∀ i, IsCompact (closure (B i)))
  [MeasurableSpace (FellClosededs Y)] [BorelSpace (FellClosededs Y)]
  {μ ν : Measure (FellClosededs Y)} [IsFiniteMeasure μ]
  (hμν : μ ≠ ν) :
  ∃ hits misses : Finset ι,
    μ (nablaGen B hits misses) ≠ ν (nablaGen B hits misses)

```

SuFunctor.Universe.FellClosededs.**scottContinuous\_ext\_of\_nablaGen** lean/su-functor/SuFunctor/Universe/FellCapacity.lean, line 326

```

theorem scottContinuous_ext_of_nablaGen [LocallyCompactSpace Y] [T2Space Y]
  {B : ι → Set Y} (hB : IsTopologicalBasis (Set.range B))
  (hBcl : ∀ i, IsCompact (closure (B i)))
  {β : Type*} [PartialOrder β] {f g : Opens (FellClosededs Y) → β}
  (hf : ScottContinuous f) (hg : ScottContinuous g)
  (h : ∀ U : Opens (FellClosededs Y),
    (∃ s : Finset (Finset ι × Finset ι),
      (U : Set (FellClosededs Y)) = ⋃ p ∈ s, nablaGen B p.1 p.2) →
    f U = g U) :
  f = g

```

SuFunctor.Universe.FellClosededs.**not\_isOpen\_disjoint\_univ** lean/su-functor/SuFunctor/Universe/FellCapacity.lean, line 375

```

theorem not_isOpen_disjoint_univ [T1Space Y] (hY : ¬CompactSpace Y) :
  ¬IsOpen {F : FellClosededs Y | Disjoint (F : Set Y) univ}

```

SuFunctor.Universe.FellClosededs.**exists\_scottContinuous\_eq\_on\_lower\_ne** lean/su-functor/SuFunctor/Universe/FellCapacity.lean, line 401

```

theorem exists_scottContinuous_eq_on_lower_ne [T1Space Y] [Nonempty Y] :
  ∃ f g : Opens (FellClosededs Y) → ℝ≥0∞,
    ScottContinuous f ∧ ScottContinuous g ∧
    (∀ U : Opens (FellClosededs Y),
      IsOpen[TopologicalSpace.generateFrom
        ((fun V => {F : FellClosededs Y | ((F : Set Y) ∩ V).Nonempty})
        '' {V : Set Y | IsOpen V})]
      (U : Set (FellClosededs Y)) → f U = g U) ∧
    f ≠ g

```

Pointfree.NablaData.Valuation.**ext\_of\_nabla** lean/pointfree/Pointfree/Frame/HyperspacePresentation.lean, line 259

```

theorem ext_of_nabla
  (hμS : μ.IsScottContinuous) (hνS : ν.IsScottContinuous)
  (hfin : ∀ p, μ (nabla geom p) ≠ ∞)
  (h : ∀ p, μ (nabla geom p) = ν (nabla geom p)) : μ = ν

```

Pointfree.NablaData.Valuation.**exists\_nabla\_ne** lean/pointfree/Pointfree/Frame/HyperspacePresentation.lean, line 271

```

theorem exists_nabla_ne
  (hμS : μ.IsScottContinuous) (hνS : ν.IsScottContinuous)
  (hfin : ∀ p, μ (nabla geom p) ≠ ∞)
  (hne : μ ≠ ν) : ∃ p, μ (nabla geom p) ≠ ν (nabla geom p)

```



Pointfree.Valuation.**ext\_of\_generators** lean/pointfree/Pointfree/Valuation/Determination.lean, line 131

```
theorem ext_of_generators
  (hμS : μ.IsScottContinuous) (hνS : ν.IsScottContinuous)
  (hGinf : ∀ i j, ∃ k, G i ⊆ G j = G k)
  (hGgen : ∀ x : L, ∃ S : Set κ, x = ⋃ i ∈ S, G i)
  (hfin : ∀ i, μ (G i) ≠ ∞)
  (h : ∀ i, μ (G i) = ν (G i)) : μ = ν
```

Pointfree.Valuation.**ext\_of\_isTopologicalBasis** lean/pointfree/Pointfree/Valuation/SpatialDetermination.lean, line 60

```
theorem ext_of_isTopologicalBasis
  (hB : IsTopologicalBasis (Set.range fun i => (N i : Set X)))
  (hNinf : ∀ i j, ∃ k, N i ⊆ N j = N k)
  {μ ν : Valuation (Opens X)}
  (hμS : μ.IsScottContinuous) (hνS : ν.IsScottContinuous)
  (hfin : ∀ i, μ (N i) ≠ ∞)
  (h : ∀ i, μ (N i) = ν (N i)) : μ = ν
```

Pointfree.SubProb.**promoteHomeomorph** lean/pointfree/Pointfree/Valuation/Topology.lean, line 230

```
noncomputable def promoteHomeomorph : SubProb L ≈τ Prob (L × Prop)
```

Pointfree.SubProb.**induced\_promote\_eq** lean/pointfree/Pointfree/Valuation/Topology.lean, line 171

```
theorem induced_promote_eq :
  TopologicalSpace.induced (promote : SubProb L → Prob (L × Prop))
    (Prob.instTopologicalSpace (L × Prop))
  = SubProb.instTopologicalSpace L
```

SuFunctor.Universe.General.**SubProbabilityMeasure** lean/su-functor/SuFunctor/Universe/General/Carrier.lean, line 51

```
def SubProbabilityMeasure : Type _
```

Pointfree.SubProbabilityMeasure.**continuous\_toSubProb** lean/pointfree/Pointfree/Valuation/BridgeContinuity.lean, line 180

```
theorem SubProbabilityMeasure.continuous_toSubProb :
  Continuous (SubProbabilityMeasure.toSubProb (X := X))
```

Pointfree.**daggerChartMeasure** lean/pointfree/Pointfree/Measure/DaggerChart.lean, line 390

```
noncomputable def daggerChartMeasure :
  SubProbabilityMeasure X ≈τ ProbabilityMeasure (X ⊗ Unit) where ...
```

Pointfree.**compactSpace\_of\_daggerChartMeasure** lean/pointfree/Pointfree/Measure/DaggerChart.lean, line 415

```
theorem compactSpace_of_daggerChartMeasure [T2Space X] [BorelSpace X]
  [CompactSpace X] : CompactSpace (SubProbabilityMeasure X)
```

Pointfree.**metrizableSpace\_subProbabilityMeasure** lean/pointfree/Pointfree/Measure/DaggerChart.lean, line 425

```
theorem metrizableSpace_subProbabilityMeasure [MetrizableSpace X]
  [SeparableSpace X] [BorelSpace X] :
  MetrizableSpace (SubProbabilityMeasure X)
```

SuFunctor.Universe.**belCarrier** lean/su-functor/SuFunctor/Universe/Carrier.lean, line 63

```
def belCarrier : Set (V → ℝ)
```

SuFunctor.Universe.**BelState** lean/su-functor/SuFunctor/Universe/Carrier.lean, line 119

```
abbrev BelState : Type _
```

SuFunctor.Universe.**convex\_belCarrier** lean/su-functor/SuFunctor/Universe/Carrier.lean, line 72  
 | theorem convex\_belCarrier : Convex  $\mathbb{R}$  (belCarrier V)

SuFunctor.Universe.**isCompact\_belCarrier** lean/su-functor/SuFunctor/Universe/Carrier.lean, line 107  
 | theorem isCompact\_belCarrier : IsCompact (belCarrier V)

SuFunctor.Universe.**probCarrier** lean/su-functor/SuFunctor/Universe/ProbCarrier.lean, line 45  
 | def probCarrier : Set (V  $\rightarrow$   $\mathbb{R}$ )

SuFunctor.Universe.**ProbState** lean/su-functor/SuFunctor/Universe/ProbCarrier.lean, line 85  
 | abbrev ProbState : Type \_

SuFunctor.Universe.General.**sat\_sup\_iff** lean/su-functor/SuFunctor/Universe/General/Faces.lean, line 72  
 | theorem sat\_sup\_iff { $\sigma \ \phi \ \psi$  : StatePenalty X} :  
 | Sat  $\sigma$  ( $\phi \sqcup \psi$ )  $\leftrightarrow$  Sat  $\sigma \ \phi \wedge$  Sat  $\sigma \ \psi$

SuFunctor.Universe.General.**entails\_inf\_iff** lean/su-functor/SuFunctor/Universe/General/Faces.lean, line 101  
 | theorem entails\_inf\_iff { $\phi \ \psi \ \rho$  : StatePenalty X} :  
 |  $\rho \leq \phi \sqcap \psi \leftrightarrow \rho \leq \phi \wedge \rho \leq \psi$

SuFunctor.Universe.General.**exists\_sat\_inf\_not\_sat** lean/su-functor/SuFunctor/Universe/General/Faces.lean, line 112  
 | theorem exists\_sat\_inf\_not\_sat [Nonempty X] :  
 |  $\exists \sigma \ \phi \ \psi : \text{StatePenalty } X, \text{ Sat } \sigma (\phi \sqcap \psi) \wedge \neg \text{ Sat } \sigma \ \phi \wedge \neg \text{ Sat } \sigma \ \psi$

SuFunctor.Universe.**iSup\_boxRows\_eq\_self** lean/su-functor/SuFunctor/Universe/Generation.lean, line 146  
 | theorem iSup\_boxRows\_eq\_self {f : ProbPenalty V}  
 | (hf : LowerSemicontinuous f) :  
 |  $\sqcup (q : \mathbb{Q}) (l : V \rightarrow \mathbb{Q}) (u : V \rightarrow \mathbb{Q}) (\_ : 0 < q)$   
 | ( $\_ : \forall v, \text{ boxRow } q \ l \ u \ v \leq f \ v$ ), boxRow q l u = f

SuFunctor.Universe.**iSup\_tents\_eq\_self** lean/su-functor/SuFunctor/Universe/Tents.lean, line 76  
 | theorem iSup\_tents\_eq\_self { $\sigma$  : Penalty V} (h $\sigma$  : LowerSemicontinuous  $\sigma$ ) :  
 |  $\sqcup (\mu_0 : \text{BelState } V) (L : \mathbb{R}) (M : \mathbb{R})$   
 | ( $\_ : \forall v, \text{ tent } \mu_0 \ L \ M \ v \leq \sigma \ v$ ), tent  $\mu_0 \ L \ M = \sigma$

SuFunctor.Universe.**ProbPenalty** lean/su-functor/SuFunctor/Universe/ProbCarrier.lean, line 157  
 | abbrev ProbPenalty : Type \_

SuFunctor.Universe.General.**sat\_sup\_iff** lean/su-functor/SuFunctor/Universe/General/Faces.lean, line 72  
 | theorem sat\_sup\_iff { $\sigma \ \phi \ \psi$  : StatePenalty X} :  
 | Sat  $\sigma$  ( $\phi \sqcup \psi$ )  $\leftrightarrow$  Sat  $\sigma \ \phi \wedge$  Sat  $\sigma \ \psi$

SuFunctor.Universe.General.**entails\_inf\_iff** lean/su-functor/SuFunctor/Universe/General/Faces.lean, line 101  
 | theorem entails\_inf\_iff { $\phi \ \psi \ \rho$  : StatePenalty X} :  
 |  $\rho \leq \phi \sqcap \psi \leftrightarrow \rho \leq \phi \wedge \rho \leq \psi$

SuFunctor.Universe.General.**exists\_sat\_inf\_not\_sat** lean/su-functor/SuFunctor/Universe/General/Faces.lean, line 112  
 | theorem exists\_sat\_inf\_not\_sat [Nonempty X] :  
 |  $\exists \sigma \ \phi \ \psi : \text{StatePenalty } X, \text{ Sat } \sigma (\phi \sqcap \psi) \wedge \neg \text{ Sat } \sigma \ \phi \wedge \neg \text{ Sat } \sigma \ \psi$

SuFunctor.Universe.General.**lowerSemicontinuous\_wall\_iff** lean/su-functor/SuFunctor/Universe/General/Faces.lean, line 222

```
theorem lowerSemicontinuous_wall_iff {C : Set (SubProbabilityMeasure X)} :
  LowerSemicontinuous (wall C) ↔ IsClosed C
```

SuFunctor.Universe.General.**iSup\_gambleRows\_eq\_self** lean/su-functor/SuFunctor/Universe/General/GenerationGeneral.lean, line 248

```
theorem iSup_gambleRows_eq_self {ι : Y → FiniteMeasure X}
  (hι : Topology.IsInducing ι) {f : Y → ℝ≥0∞} (hf : LowerSemicontinuous f) :
  ⊔ (q : ℚ) (F : Finset ((X →b ℝ≥0) × ℚ × ℚ)) ( _ : 0 < q)
  ( _ : ∀ v, gambleRow ι (ENNReal.ofReal q) F v ≤ f v),
  gambleRow ι (ENNReal.ofReal q) F = f
```

SuFunctor.Universe.General.**isInducing\_testProfile** lean/su-functor/SuFunctor/Universe/General/GenerationGeneral.lean, line 176

```
theorem isInducing_testProfile {ι : Y → FiniteMeasure X}
  (hι : Topology.IsInducing ι) : Topology.IsInducing (testProfile ι)
```

SuFunctor.Universe.General.**exists\_countable\_gambleRows\_eq\_self** lean/su-functor/SuFunctor/Universe/General/GenerationGeneral.lean, line 305

```
theorem exists_countable_gambleRows_eq_self [SecondCountableTopology Y]
  {ι : Y → FiniteMeasure X} (hι : Topology.IsInducing ι) {f : Y → ℝ≥0∞}
  (hf : LowerSemicontinuous f) :
  ∃ R : Set (ℚ × Finset ((X →b ℝ≥0) × ℚ × ℚ)), R.Countable
  ∧ (∀ p ∈ R, ∀ v, gambleRow ι (ENNReal.ofReal p.1) p.2 v ≤ f v)
  ∧ ⊔ p ∈ R, gambleRow ι (ENNReal.ofReal p.1) p.2 = f
```

SuFunctor.Universe.General.**secondCountableTopology\_probabilityMeasure** lean/su-functor/SuFunctor/Universe/General/GenerationGeneral.lean, line 433

```
theorem secondCountableTopology_probabilityMeasure
  [CompactSpace X] [TopologicalSpace.MetrizableSpace X] [BorelSpace X] :
  SecondCountableTopology (ProbabilityMeasure X)
```

SuFunctor.Plain.Joint.**toMeas\_surjective** lean/su-functor/SuFunctor/Plain/SeamDensity.lean, line 333

```
theorem Joint.toMeas_surjective :
  Function.Surjective (Joint.toMeas : Joint X → MeasJoint X)
```

Qtsl.Imprecise.**sum\_dominates\_sup** lean/qtsl-aux/QtslAux/Imprecise/MarginalCompat.lean, line 103

```
theorem sum_dominates_sup (m : ∀ i, FinPMF X → FinPMF (Xi i))
  (f : ∀ i, FinPMF (Xi i) → ENNRat) (v : FinPMF X) :
  supPenalty m f v ≤ compositePenalty m f v
```

Qtsl.Imprecise.**sublevel\_marginal\_of\_dominates** lean/qtsl-aux/QtslAux/Imprecise/MarginalCompat.lean, line 136

```
theorem sublevel_marginal_of_dominates (m : ∀ i, FinPMF X → FinPMF (Xi i))
  (f : ∀ i, FinPMF (Xi i) → ENNRat) {F : FinPMF X → ENNRat} {v : FinPMF X}
  (hdom : supPenalty m f v ≤ F v) {t : ENNRat}
  (h : F v ≤ t) (i : ι) : f i (m i v) ≤ t
```

SuFunctor.Universe.General.**compileEquiv** lean/su-functor/SuFunctor/Universe/RadialUniverse.lean, line 99

```
noncomputable def compileEquiv [Nonempty X] :
  {f : Penalty X // LowerSemicontinuous f} ≈ RadialUniverse X where ...
```

SuFunctor.Adequacy.**rwall\_ne\_top** lean/su-functor/SuFunctor/Universe/RadialUniverse.lean, line 349

```
theorem rwall_ne_top (c : SubPMF V → Prop) :
  rwall c ≠ (T : Plain.Pen V)
```

SuFunctor.Adequacy.**contradictory\_composite\_denotes\_bot** lean/su-functor/SuFunctor/Universe/RadialUniverse.lean, line 589

```
theorem contradictory_composite_denotes_bot :
  rwall xyEqC ⊔ rwall xFalseC ⊔ rwall yTrueC
  = rayPoint (SubPMF.zero (Bool × Bool))
```

SuFunctor.Universe.General.**compile\_ne\_singleton** lean/su-functor/SuFunctor/Universe/RadialUniverse.lean, line 172

```
theorem compile_ne_singleton (f : Penalty X) {v : SubProbabilityMeasure X}
  (hv : v ≠ 0) : compile f ≠ ({v} : Set (SubProbabilityMeasure X))
```

SuFunctor.StateSpace.**sortedCone** lean/su-functor/SuFunctor/StateSpace/FamCompletion.lean, line 165

```
def sortedCone (n : ℕ) : OmegaSA
```

SuFunctor.StateSpace.**mem\_sortedCone\_iff\_monotone** lean/su-functor/SuFunctor/StateSpace/FamCompletion.lean, line 132

```
theorem mem_sortedCone_iff_monotone {n : ℕ} (x : Fin n → ℝ) :
  x ∈ BasicSet (sortedConds n) ↔ Monotone x
```

SuFunctor.BagFormer.**bagObj** lean/su-functor/SuFunctor/BagFormer.lean, line 99

```
def bagObj (A : WireSpan El) : WireSpan El
```

SuFunctor.StateSpace.**bagIsColimit** lean/su-functor/SuFunctor/StateSpace/FamCompletion.lean, line 188

```
def bagIsColimit :
  IsColimit (famComponentsCofan bagObj)
```

Pointfree.Prob.**lowerSemicontinuous\_eval** lean/pointfree/Pointfree/Valuation/Topology.lean, line 79

```
theorem lowerSemicontinuous_eval (a : M) :
  LowerSemicontinuous fun μ : Prob M => μ a
```

Pointfree.SubProb.**upperSemicontinuous\_mass** lean/pointfree/Pointfree/Valuation/Topology.lean, line 122

```
theorem upperSemicontinuous_mass :
  UpperSemicontinuous fun μ : SubProb L => μ τ
```

Pointfree.SubProb.**isClosed\_liveness\_wall** lean/pointfree/Pointfree/Valuation/Topology.lean, line 132

```
theorem isClosed_liveness_wall (p : ℝ≥0∞) :
  IsClosed {μ : SubProb L | p ≤ μ τ}
```

SuFunctor.**satCredal\_lower\_rayC\_geC** lean/su-functor/SuFunctor/Adequacy/Envelope.lean, line 390

```
theorem satCredal_lower_rayC_geC {μ : SubPMF V} (hμ : μ ≠ SubPMF.zero V)
  (A : Set V) (q : ℚ) :
  SatCredal .lower (rayC μ) (geC A q) ↔ q ≤ μ.prob A
```

SuFunctor.**maxSlice\_rayC\_empty\_iff** lean/su-functor/SuFunctor/Adequacy/Envelope.lean, line 216

```
theorem maxSlice_rayC_empty_iff {μ : SubPMF V} :
  (∀ v, ¬ maxSlice (rayC μ) v) ↔ μ = SubPMF.zero V
```

SuFunctor.**not\_satCredal\_upper\_rayC\_geC** lean/su-functor/SuFunctor/Adequacy/Envelope.lean, line 381

```
theorem not_satCredal_upper_rayC_geC (μ : SubPMF V) {A : Set V} {q : ℚ}
  (hq : 0 < q) : ¬ SatCredal .upper (rayC μ) (geC A q)
```

SuFunctor.Universe.SpecDnf.**top\_notMem\_modelClass** lean/su-functor/SuFunctor/Universe/SpecDnf.lean, line 216

```
theorem top_notMem_modelClass (D : SpecDnf K) :
  (τ : K → ℝ≥0∞) ∉ D.modelClass
```

SuFunctor.Universe.General.**isClosed\_compile\_iff** lean/su-functor/SuFunctor/Universe/General/Closedness.lean, line 308

```
theorem isClosed_compile_iff [Nonempty X] {f : Penalty X} :
  IsClosed (compile f) ↔ LowerSemicontinuous f
```

SuFunctor.Universe.General.**exists\_lsc\_compile\_eq\_iff** lean/su-functor/SuFunctor/Universe/General/Closedness.lean, line 294

```
theorem exists_lsc_compile_eq_iff [Nonempty X]
  {R : Set (SubProbabilityMeasure X)} :
  (∃ f : Penalty X, LowerSemicontinuous f ∧ compile f = R)
  ↔ R.Nonempty ∧ IsClosed R ∧ IsRadial R
```

SuFunctor.Universe.General.**compile\_rayPenalty** lean/su-functor/SuFunctor/Universe/General/Closedness.lean, line 260

```
theorem compile_rayPenalty [Nonempty X] {R : Set (SubProbabilityMeasure X)}
  (hcl : IsClosed R) (hrad : IsRadial R) (hne : R.Nonempty) :
  compile (rayPenalty R) = R
```

SuFunctor.Universe.General.**compile\_injective** lean/su-functor/SuFunctor/Universe/General/Dictionary.lean, line 238

```
theorem compile_injective [Nonempty X] :
  Function.Injective (compile (X := X))
```

SuFunctor.Universe.General.**rayPenalty\_compile** lean/su-functor/SuFunctor/Universe/General/Dictionary.lean, line 231

```
theorem rayPenalty_compile [Nonempty X] (f : Penalty X) :
  rayPenalty (compile f) = f
```

SuFunctor.Universe.General.**upperSemicontinuous\_rayGauge** lean/su-functor/SuFunctor/Universe/General/Closedness.lean, line 233

```
theorem upperSemicontinuous_rayGauge {R : Set (SubProbabilityMeasure X)}
  (hcl : IsClosed R) (hrad : IsRadial R) (hne : R.Nonempty) :
  UpperSemicontinuous (rayGauge R)
```

SuFunctor.Universe.General.**le\_gradeN\_of\_forall\_lt** lean/su-functor/SuFunctor/Universe/General/Carrier.lean, line 326

```
theorem le_gradeN_of_forall_lt {a : ℝ≥0∞} {c : ℝ≥0} (hc : c ≤ 1)
  (h : ∀ y, y < a → c ≤ gradeN y) : c ≤ gradeN a
```

SuFunctor.Universe.**compile** lean/su-functor/SuFunctor/Universe/VietorisDictionary.lean, line 168

```
def compile (f : ProbPenalty V) : Set (BelState V)
```

SuFunctor.Universe.EpistemicStates.**condTarget\_mix** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 935

```
theorem condTarget_mix (k : B → ProbState A) {μ v : ProbState (A × B)}
  {s t : ℝ} (hs : 0 ≤ s) (ht : 0 ≤ t) (hst : s + t = 1) :
  condTarget k (ProbState.mix s t hs ht hst μ v)
  = ProbState.mix s t hs ht hst (condTarget k μ)
  (condTarget k v)
```

SuFunctor.Universe.EpistemicStates.**klq\_mix\_mix\_le** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 429

```

theorem klq_mix_mix_le {μ ν ρ σ : ProbState V} {s t : ℝ}
  (hs : 0 < s) (ht : 0 < t) (hst : s + t = 1) :
  klq (ProbState.mix s t hs.le ht.le hst μ ν)
    (ProbState.mix s t hs.le ht.le hst ρ σ)
    ≤ ENNReal.ofReal s * klq μ ρ + ENNReal.ofReal t * klq ν σ

```

SuFunctor.Universe.EpistemicStates.**lowerSemicontinuous\_condPenalty** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 1056

```

theorem lowerSemicontinuous_condPenalty {β : ℝ≥0∞} (hβ : β ≠ τ)
  (k : B → ProbState A) :
  LowerSemicontinuous (condPenalty β k)

```

SuFunctor.Universe.EpistemicStates.**klq\_condTarget\_eq\_zero\_iff** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 974

```

theorem klq_condTarget_eq_zero_iff (k : B → ProbState A)
  (μ : ProbState (A × B)) :
  klq μ (condTarget k μ) = 0
  ↔ ∀ b, 0 < (sndMarg μ : B → ℝ) b
    → ∀ a, (μ : A × B → ℝ) (a, b) / (sndMarg μ : B → ℝ) b
      = (k b : A → ℝ) a

```

SuFunctor.Universe.EpistemicStates.**mutualInfoQ\_eq\_zero\_iff** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 844

```

theorem mutualInfoQ_eq_zero_iff (μ : ProbState (A × B)) :
  mutualInfoQ μ = 0 ↔ IsProductLaw μ

```

SuFunctor.Universe.EpistemicStates.**continuous\_mutualInfoQ** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 882

```

theorem continuous_mutualInfoQ :
  Continuous (mutualInfoQ (A := A) (B := B))

```

SuFunctor.Universe.EpistemicStates.Witness.**not\_penaltyConvex\_indepPenalty** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 1262

```

theorem not_penaltyConvex_indepPenalty {β : ℝ≥0∞} (hβ : β ≠ 0) :
  ¬ PenaltyConvex (indepPenalty (A := Fin 2) (B := Fin 2) β)

```

SuFunctor.Universe.EpistemicStates.**indepBody\_credalBody\_not\_convex** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 1435

```

theorem indepBody_credalBody_not_convex :
  IsCredalBody (indepBody (Fin 2) (Fin 2))
  ∧ ¬ IsConvexCredalBody (indepBody (Fin 2) (Fin 2))

```

SuFunctor.Universe.EpistemicStates.**compile\_marginalWallSum** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 1145

```

theorem compile_marginalWallSum {MX : Set (ProbState A)}
  {MY : Set (ProbState B)} (hX : MX.Nonempty) (hY : MY.Nonempty) :
  compile (fun μ : ProbState (A × B) =>
    probWall MX (fstMarg μ) + probWall MY (sndMarg μ))
    = {x : BelState (A × B) | ∃ μ ∈ frechetClass MX MY, ∃ γ : ℝ,
      0 ≤ γ ∧ γ ≤ 1 ∧ (x : A × B → ℝ) = γ • (μ : A × B → ℝ)}

```

SuFunctor.Universe.EpistemicStates.**marginalSum\_eq\_zero\_iff** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 1078

```

theorem marginalSum_eq_zero_iff (g : ProbState A → ℝ≥0∞)
  (h : ProbState B → ℝ≥0∞) (μ : ProbState (A × B)) :
  g (fstMarg μ) + h (sndMarg μ) = 0
  ↔ g (fstMarg μ) = 0 ∧ h (sndMarg μ) = 0

```

SuFunctor.Universe.EpistemicStates.Witness.**frechet\_does\_not\_imply\_independence** lean/su-functor/SuFunctor/Universe/EpistemicStates.lean, line 1321

```
theorem frechet_does_not_imply_independence :
  (probWall {uniform2} (fstMarg chi)
    + probWall {uniform2} (sndMarg chi) = 0)
   $\wedge$   $\neg$  IsProductLaw chi  $\wedge$  mutualInfoQ chi  $\neq$  0
```

SuFunctor.Universe.CredalPoset.**house\_includes\_all** lean/su-functor/SuFunctor/Universe/CredalPoset.lean, line 47

```
theorem house_includes_all {V : Type*} [Fintype V] [Nonempty V]
  (S : Set (BelState V)) :
  (IsConvexCredalBody S  $\rightarrow$  IsCredalBody S)  $\wedge$ 
  (IsRichardsonBody S  $\rightarrow$  IsCredalBody S)  $\wedge$ 
  (IsUltra S  $\rightarrow$  IsCredalBody S)  $\wedge$ 
  (IsMVS S  $\rightarrow$  IsCredalBody S)  $\wedge$ 
  (IsHL S  $\rightarrow$  IsCredalBody S)  $\wedge$ 
  (IsCozman S  $\rightarrow$  IsCredalBody S)
```

SuFunctor.Universe.General.CredalPoset.**scaleSegment\_eq\_compile\_dirWall** lean/su-functor/SuFunctor/Universe/General/CredalPoset.lean, line 446

```
theorem scaleSegment_eq_compile_dirWall {K : Set (ProbabilityMeasure X)}
  (hne : K.Nonempty) : scaleSegment K = compile (dirWall K)
```

SuFunctor.Universe.CredalPoset.**exists\_penaltyConvex\_not\_closed** lean/su-functor/SuFunctor/Universe/CredalPoset/Adjudication.lean, line 172

```
theorem exists_penaltyConvex_not_closed :
   $\exists$  f : ProbPenalty (Fin 2),
    PenaltyConvex f  $\wedge$   $\neg$  IsClosed (compile f)
```

SuFunctor.Universe.CredalPoset.**exists\_penaltyConvex\_compile\_not\_richardson** lean/su-functor/SuFunctor/Universe/CredalPoset/Adjudication.lean, line 201

```
theorem exists_penaltyConvex_compile_not_richardson :
   $\exists$  f : ProbPenalty (Fin 2),
    PenaltyConvex f  $\wedge$   $\neg$  IsRichardsonBody (compile f)
```

SuFunctor.Universe.CredalPoset.IsRichardsonBody.**isConvexCredalBody** lean/su-functor/SuFunctor/Universe/CredalPoset/Inclusions.lean, line 176

```
theorem IsRichardsonBody.isConvexCredalBody [Nonempty V] {S : Set (BelState V)}
  (h : IsRichardsonBody S) : IsConvexCredalBody S
```

SuFunctor.Universe.General.CredalPoset.**GradeConvex** lean/su-functor/SuFunctor/Universe/General/CredalPoset.lean, line 165

```
def GradeConvex (f : Penalty X) : Prop
```

SuFunctor.Universe.General.CredalPoset.**convexBody\_compile\_of\_gradeConvex** lean/su-functor/SuFunctor/Universe/General/CredalPoset.lean, line 252

```
theorem convexBody_compile_of_gradeConvex [Nonempty X] {f : Penalty X}
  (hgc : GradeConvex f) : ConvexBody (compile f)
```

SuFunctor.Universe.CredalPoset.IsUltra.**isConvexCredalBody** lean/su-functor/SuFunctor/Universe/CredalPoset/Defs.lean, line 196

```
theorem IsUltra.isConvexCredalBody {S : Set (BelState V)}
  (h : IsUltra S) : IsConvexCredalBody S
```

SuFunctor.Universe.CredalPoset.IsPrevisionBody.**isUltra** lean/su-functor/SuFunctor/Universe/CredalPoset/Previsions.lean, line 50



```
theorem IsPrevisionBody.isUltra {S : Set (BelState V)}
  (hP : IsPrevisionBody S) : IsUltra S
```

SuFunctor.Universe.CredalPoset.**isPrevisionBody\_iff\_isUltra** lean/su-functor/SuFunctor/Universe/CredalPoset/Previsions.lean, line 202

```
theorem isPrevisionBody_iff_isUltra {S : Set (BelState V)} :
  IsPrevisionBody S  $\leftrightarrow$  IsUltra S
```

SuFunctor.Universe.General.CredalPoset.**isClosed\_ptwiseDownHull** lean/su-functor/SuFunctor/Universe/General/CredalPoset.lean, line 588

```
theorem isClosed_ptwiseDownHull
  (horder : IsClosed {p : FiniteMeasure X  $\times$  FiniteMeasure X |
    (p.1 : Measure X)  $\leq$  (p.2 : Measure X)})
  (F : Finset (SubProbabilityMeasure X)) :
  IsClosed (ptwiseDownHull F)
```

SuFunctor.Universe.CredalPoset.IsMVS.**isUltra** lean/su-functor/SuFunctor/Universe/CredalPoset/Inclusions.lean, line 270

```
theorem IsMVS.isUltra {S : Set (BelState V)} (h : IsMVS S) : IsUltra S
```

SuFunctor.**maxSlice\_rayC** lean/su-functor/SuFunctor/Adequacy/Envelope.lean, line 176

```
theorem maxSlice_rayC {μ : SubPMF V} (hμ : μ  $\neq$  SubPMF.zero V)
  (ν : SubPMF V) : maxSlice (rayC μ) ν  $\leftrightarrow$  singletonC μ ν
```

SuFunctor.**satCredal\_lower\_rayC** lean/su-functor/SuFunctor/Adequacy/Envelope.lean, line 327

```
theorem satCredal_lower_rayC {μ : SubPMF V} (hμ : μ  $\neq$  SubPMF.zero V)
  (c : SubPMF V  $\rightarrow$  Prop) : SatCredal .lower (rayC μ) c  $\leftrightarrow$  c μ
```

SuFunctor.**DownClosed** lean/su-functor/SuFunctor/Adequacy/Envelope.lean, line 305

```
def DownClosed (c : SubPMF V  $\rightarrow$  Prop) : Prop
```

SuFunctor.**rayC\_eq\_singletonC\_iff** lean/su-functor/SuFunctor/Adequacy/Envelope.lean, line 251

```
theorem rayC_eq_singletonC_iff {μ : SubPMF V} :
  ( $\forall$  ν, rayC μ ν  $\leftrightarrow$  singletonC μ ν)  $\leftrightarrow$  μ = SubPMF.zero V
```

SuFunctor.**maxSlice\_unionRays\_subset** lean/su-functor/SuFunctor/Adequacy/EnvelopeUnion.lean, line 138

```
theorem maxSlice_unionRays_subset {μs : ι  $\rightarrow$  SubPMF V} {ν : SubPMF V}
  (h : maxSlice (unionRays μs) ν) : unionSingletons μs ν
```

SuFunctor.**maxSlice\_unionRays** lean/su-functor/SuFunctor/Adequacy/EnvelopeUnion.lean, line 183

```
theorem maxSlice_unionRays {μs : ι  $\rightarrow$  SubPMF V} {m :  $\mathbb{Q}}$ 
  (hm :  $\forall$  i, (μs i).mass = m) (hm0 : m  $\neq$  0) (ν : SubPMF V) :
  maxSlice (unionRays μs) ν  $\leftrightarrow$  unionSingletons μs ν
```

Qtsl.Imprecise.**natExtG\_le\_lowerEnvG** lean/qtsl-aux/QtslAux/Imprecise/NaturalExtensionGeneral.lean, line 372

```
theorem natExtG_le_lowerEnvG {M' : Set (ProbabilityMeasure X)}
  (hM' : M'.Nonempty) (hdom :  $\forall$  i, c i  $\leq$  lowerEnvG M' (h i))
  (g : BddGamble X) : natExtG h c g  $\leq$  lowerEnvG M' g
```

Qtsl.Imprecise.**avoidsSureLossG\_of\_dominating** lean/qtsl-aux/QtslAux/Imprecise/NaturalExtensionGeneral.lean, line 381

```
theorem avoidsSureLossG_of_dominating {M' : Set (ProbabilityMeasure X)}
  (hM' : M'.Nonempty) (hdom :  $\forall$  i, c i  $\leq$  lowerEnvG M' (h i)) :
  AvoidsSureLossG h c
```

Pointfree.**Prevision** lean/pointfree/Pointfree/Prevision/Basic.lean, line 286

```
structure Prevision (L : Type*) [Order.Frame L] where ...
```

SuFunctor.Universe.**singletons\_not\_definable** lean/su-functor/SuFunctor/Universe/Points.lean, line 165

```
theorem singletons_not_definable {σ : ProbPenalty V}
  (hσ : LowerSemicontinuous σ) (hproper : ∃ μ, σ μ ≠ τ)
  {n : ℕ} (u : Fin n → ProbPenalty V) (ℳ : Fin n → Set (ProbPenalty V))
  (hfin : ∀ i, (ℳ i).Finite) :
  ¬ ∀ τ : ProbPenalty V, LowerSemicontinuous τ →
    ((∃ i, CoupledSat τ (u i) (ℳ i)) ↔ τ = σ)
```

SuFunctor.Plain.**idealPoint\_le\_iff** lean/su-functor/SuFunctor/Plain/Battery.lean, line 117

```
theorem idealPoint_le_iff (σ : Pen (states A)) (ℳ : SpecClass A) :
  ({σ} : SpecClass A) ⊆ ℳ ↔ σ ∈ ℳ
```

SuFunctor.Universe.**crescentClass** lean/su-functor/SuFunctor/Universe/VietorisStratum.lean, line 315

```
def crescentClass (u : ProbPenalty V) (ℳ : Set (ProbPenalty V)) :
  Set (NonemptyCompacts (BelState V))
```

SuFunctor.Universe.General.**isLocallyClosed\_crescentClass** lean/su-functor/SuFunctor/Universe/General/CrescentStratum.lean, line 246

```
theorem isLocallyClosed_crescentClass [Nonempty X] {u : Penalty X}
  {ℳ : Set (Penalty X)} (hu : LowerSemicontinuous u)
  (hℳ : ℳ.Finite) (hlsc : ∀ v ∈ ℳ, LowerSemicontinuous v) :
  IsLocallyClosed (crescentClass u ℳ)
```

SuFunctor.Universe.SpecDnf.**compileNC\_mem\_crescentClass\_iff** lean/su-functor/SuFunctor/Universe/SpecDnf.lean, line 512

```
theorem SpecDnf.compileNC_mem_crescentClass_iff [Nonempty V]
  {σ : ProbPenalty V} (hσ : LowerSemicontinuous σ)
  (D : SpecDnf (ProbState V)) :
  compileNC σ hσ ∈ D.crescentClass ↔ σ ∈ D.modelClass
```

SuFunctor.Universe.General.**compile\_subset\_compile\_iff** lean/su-functor/SuFunctor/Universe/General/Dictionary.lean, line 122

```
theorem compile_subset_compile_iff {f h : Penalty X} :
  compile f ⊆ compile h ↔ h ≤ f
```

SuFunctor.Universe.**mem\_modelClass\_reachAvoid\_iff** lean/su-functor/SuFunctor/Universe/ReachAvoid.lean, line 94

```
theorem mem_modelClass_reachAvoid_iff {u ℓ σ : K → ℝ≥0∞} :
  σ ∈ (SpecAtom.reachAvoid u ℓ).modelClass ↔ u ≤ σ ∧ Dip σ ℓ
```

SuFunctor.Universe.**coupledSat\_crispRA\_iff** lean/su-functor/SuFunctor/Universe/ReachAvoid.lean, line 494

```
theorem coupledSat_crispRA_iff {σ : K → ℝ≥0∞} {S T : Set K} :
  σ ∈ (SpecAtom.reachAvoid (wallOn S) (wallOn Tᶜ)).modelClass
  ↔ (∀ x, x ∉ S → σ x = T) ∧ ∃ x ∈ T, σ x < T
```

SuFunctor.Universe.**exists\_genClosure\_reachAvoid\_not\_crispGenerated** lean/su-functor/SuFunctor/Universe/ReachAvoid.lean, line 563

```
theorem exists_genClosure_reachAvoid_not_crispGenerated :
  ∃ C : Set (Unit → ℝ≥0∞),
    GenClosure (reachAvoidClasses Unit) C
    ∧ ¬ GenClosure (crispReachAvoidClasses Unit) C
```

SuFunctor.Universe.genClosure\_reachAvoid\_iff\_exists\_specDnf lean/su-functor/SuFunctor/Universe/ReachAvoid.lean, line 259

```
theorem genClosure_reachAvoid_iff_exists_specDnf {C : Set (K → ℝ≥0∞)} :
  GenClosure (reachAvoidClasses K) C
  ↔ ∃ D : SpecDnf K, C = D.modelClass
```

SuFunctor.Universe.genClosure\_reachAvoidCres\_iff\_exists\_specDnf lean/su-functor/SuFunctor/Universe/ReachAvoid.lean, line 370

```
theorem genClosure_reachAvoidCres_iff_exists_specDnf
  {C : Set (NonemptyCompacts (BelState V))} :
  GenClosure (reachAvoidCrescents V) C
  ↔ ∃ D : SpecDnf (ProbState V), C = D.crescentClass
```

SuFunctor.Universe.isConstructibleUnion\_of\_genClosure\_reachAvoidCrescentsLsc lean/su-functor/SuFunctor/Universe/ReachAvoid.lean, line 456

```
theorem isConstructibleUnion_of_genClosure_reachAvoidCrescentsLsc
  {C : Set (NonemptyCompacts (BelState V))}
  (h : GenClosure (reachAvoidCrescentsLsc V) C) :
  IsConstructibleUnion C
```

SuFunctor.Universe.modelClass\_eq\_iInter\_reachAvoid lean/su-functor/SuFunctor/Universe/ReachAvoid.lean, line 122

```
theorem modelClass_eq_iInter_reachAvoid (a : SpecAtom K) :
  a.modelClass
  = ⋂ ℓ ∈ a.marks, (SpecAtom.reachAvoid a.floor ℓ).modelClass
```

SuFunctor.Universe.crescentClass\_eq\_iInter\_reachAvoid lean/su-functor/SuFunctor/Universe/ReachAvoid.lean, line 296

```
theorem crescentClass_eq_iInter_reachAvoid (a : SpecAtom (ProbState V)) :
  crescentClass a.floor a.marks
  = ⋂ ℓ ∈ a.marks, crescentClass a.floor {ℓ, τ}
```

SuFunctor.Universe.exists\_modelClass\_ne\_iUnion\_reachAvoid lean/su-functor/SuFunctor/Universe/ReachAvoid.lean, line 141

```
theorem exists_modelClass_ne_iUnion_reachAvoid :
  ∃ a : SpecAtom Unit,
  a.modelClass
  ≠ ⋃ ℓ ∈ a.marks, (SpecAtom.reachAvoid a.floor ℓ).modelClass
```

SuFunctor.Universe.dip\_sup\_iff lean/su-functor/SuFunctor/Universe/CoupledJudgment.lean, line 226

```
theorem dip_sup_iff {σ v₁ v₂ : K → ℝ≥0∞} :
  Dip σ (v₁ ⊔ v₂) ↔ Dip σ v₁ v Dip σ v₂
```

SuFunctor.Universe.IsConstructibleUnion lean/su-functor/SuFunctor/Universe/LocallyClosedLayer.lean, line 406

```
def IsConstructibleUnion (s : Set X) : Prop
```

SuFunctor.Universe.General.crescentClass\_inter\_crescentClass lean/su-functor/SuFunctor/Universe/General/CrescentStratum.lean, line 286

```
theorem crescentClass_inter_crescentClass {u₁ u₂ : Penalty X}
  {V₁ V₂ : Set (Penalty X)} :
  crescentClass u₁ V₁ ∩ crescentClass u₂ V₂
  = crescentClass (u₁ ⊔ u₂) (V₁ ∩ V₂)
```

SuFunctor.Universe.inter\_dnfClass\_append lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 204

```

theorem inter_dnfClass_append {n m : ℕ}
  (γ : Set (NonemptyCompacts (BelState V)))
  (u : Fin n → ProbPenalty V) (γ' : Fin n → Set (ProbPenalty V))
  (u' : Fin m → ProbPenalty V) (γ'' : Fin m → Set (ProbPenalty V)) :
  γ n dnfClass (Fin.append u u') (Fin.append γ' γ'')
    = (γ n dnfClass u γ') u (γ n dnfClass u' γ'')

```

SuFunctor.Universe.General.**compile\_pullPen**    lean/su-functor/SuFunctor/Universe/General/TransportGeneral.lean, line 176

```

theorem compile_pullPen [Nonempty X] (hθ : Continuous θ) (f : Penalty Y) :
  compile (pullPen hθ f) = push θ -1 compile f

```

SuFunctor.Universe.General.**transportClass\_union**    lean/su-functor/SuFunctor/Universe/General/TransportGeneral.lean, line 278

```

theorem transportClass_union (θ : X → Y)
  (C1 C2 : Set (FellClosed (SubProbabilityMeasure Y))) :
  transportClass θ (C1 u C2)
    = transportClass θ C1 u transportClass θ C2

```

SuFunctor.Universe.**ctxOrE\_valid\_iff**    lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 99

```

theorem ctxOrE_valid_iff :
  (∀ γ ψ : Set α, γ n A ⊆ ψ → γ n B ⊆ ψ → γ n J ⊆ ψ) ↔ J ⊆ A u B

```

SuFunctor.Universe.**orI\_ctxOrE\_valid\_iff**    lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 107

```

theorem orI_ctxOrE_valid_iff :
  (A ⊆ J ∧ B ⊆ J ∧
   ∀ γ ψ : Set α, γ n A ⊆ ψ → γ n B ⊆ ψ → γ n J ⊆ ψ)
    ↔ J = A u B

```

SuFunctor.Universe.**exists\_ctxOrE\_countermodel\_of\_ssubset**    lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 123

```

theorem exists_ctxOrE_countermodel_of_ssubset (h : A u B < J) :
  ∃ γ ψ : Set α, γ n A ⊆ ψ ∧ γ n B ⊆ ψ ∧ γ n J ⊆ ψ

```

SuFunctor.Universe.**exists\_union\_crescentClass\_not\_isLocallyClosed**    lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 403

```

theorem exists_union_crescentClass_not_isLocallyClosed :
  ∃ u v : ProbPenalty Bool,
    LowerSemicontinuous u ∧ LowerSemicontinuous v ∧
    ¬ IsLocallyClosed
      (crescentClass u ∅
       u crescentClass (θ : ProbPenalty Bool) {v})

```

SuFunctor.Universe.**dnfClass\_append**    lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 146

```

theorem dnfClass_append {n m : ℕ} (u : Fin n → ProbPenalty V)
  (γ' : Fin n → Set (ProbPenalty V)) (u' : Fin m → ProbPenalty V)
  (γ'' : Fin m → Set (ProbPenalty V)) :
  dnfClass (Fin.append u u') (Fin.append γ' γ'')
    = dnfClass u γ' u dnfClass u' γ''

```

SuFunctor.Universe.**isConstructibleUnion\_dnfClass\_append**    lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 226

```

theorem isConstructibleUnion_dnfClass_append {n m : ℕ}
  {u : Fin n → ProbPenalty V} {γ' : Fin n → Set (ProbPenalty V)}
  {u' : Fin m → ProbPenalty V} {γ'' : Fin m → Set (ProbPenalty V)}
  (hu : ∀ i, LowerSemicontinuous (u i)) (hγ' : ∀ i, (γ' i).Finite)

```

```

(hlsc : ∀ i, ∀ v ∈ ℳ i, LowerSemicontinuous v)
(hu' : ∀ i, LowerSemicontinuous (u' i)) (hℳ' : ∀ i, (ℳ' i).Finite)
(hlsc' : ∀ i, ∀ v ∈ ℳ' i, LowerSemicontinuous v) :
IsConstructibleUnion
  (dnfClass (Fin.append u u') (Fin.append ℳ ℳ'))

```

SuFunctor.Universe.**ctxOrE\_dnfClass** lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 216

```

theorem ctxOrE_dnfClass {n m : ℕ} {u : Fin n → ProbPenalty V}
  {ℳ : Fin n → Set (ProbPenalty V)} {u' : Fin m → ProbPenalty V}
  {ℳ' : Fin m → Set (ProbPenalty V)}
  {ψ : Set (NonemptyCompacts (BelState V))}
  (h₁ : γ n dnfClass u ℳ ⊆ ψ) (h₂ : γ n dnfClass u' ℳ' ⊆ ψ) :
  γ n dnfClass (Fin.append u u') (Fin.append ℳ ℳ') ⊆ ψ

```

SuFunctor.Universe.**isClosed\_pairHull** lean/su-functor/SuFunctor/Universe/HullClosure.lean, line 270

```

theorem isClosed_pairHull {C D : Set (ProbabilityMeasure X)}
  (hC : IsClosed C) (hD : IsClosed D) : IsClosed (pairHull C D)

```

SuFunctor.Universe.**isClosed\_binaryHull** lean/su-functor/SuFunctor/Universe/HullClosure.lean, line 395

```

theorem isClosed_binaryHull {C D : Set (ProbabilityMeasure X)}
  (hC : IsClosed C) (hD : IsClosed D) : IsClosed (binaryHull C D)

```

SuFunctor.Universe.**sublevel\_qCoproduct\_eq** lean/su-functor/SuFunctor/Universe/HullClosure.lean, line 686

```

theorem sublevel_qCoproduct_eq {σ τ' : ProbabilityMeasure X → ℝ≥0∞}
  (hσ : LowerSemicontinuous σ) (hτ' : LowerSemicontinuous τ') (c : ℝ≥0∞) :
  {μ | qCoproduct σ τ' μ ≤ c}
  = binaryHull {ν | σ ν ≤ c} {ν | τ' ν ≤ c}

```

SuFunctor.Universe.**qCoproduct\_le\_min** lean/su-functor/SuFunctor/Universe/HullClosure.lean, line 663

```

theorem qCoproduct_le_min (σ τ' : ProbabilityMeasure X → ℝ≥0∞)
  (μ : ProbabilityMeasure X) : qCoproduct σ τ' μ ≤ min (σ μ) (τ' μ)

```

SuFunctor.Universe.**le\_qCoproduct\_of\_forall\_mixConvex** lean/su-functor/SuFunctor/Universe/HullClosure.lean, line 670

```

theorem le_qCoproduct_of_forall_mixConvex {σ τ' h : ProbabilityMeasure X → ℝ≥0∞}
  (hqC : ∀ c, MixConvex {ν | h ν ≤ c}) (h₁ : ∀ ν, h ν ≤ σ ν)
  (h₂ : ∀ ν, h ν ≤ τ' ν) (μ : ProbabilityMeasure X) :
  h μ ≤ qCoproduct σ τ' μ

```

SuFunctor.Universe.**boxClass\_union\_ssubset** lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 275

```

theorem boxClass_union_ssubset {x y : α} (hxy : x ≠ y) :
  boxClass ({x} : Set α) ∪ boxClass {y} ⊂ boxClass ({x, y} : Set α)

```

SuFunctor.Universe.**exists\_ctxOrE\_countermodel\_body\_reading\_carrier** lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 311

```

theorem exists_ctxOrE_countermodel_body_reading_carrier :
  ∃ γ ψ : Set (NonemptyCompacts (BelState Bool)),
  γ n boxClass {BelState.dirac true} ⊆ ψ
  ∧ γ n boxClass {BelState.dirac false} ⊆ ψ
  ∧ ¬ γ n boxClass
    ({BelState.dirac true, BelState.dirac false}
     : Set (BelState Bool)) ⊆ ψ

```

SuFunctor.Universe.**mem\_dnfClass\_append\_iff** lean/su-functor/SuFunctor/Universe/DisjunctionElim.lean, line 175

```

theorem mem_dnfClass_append_iff {n m : ℕ} {u : Fin n → ProbPenalty V}
  {V' : Fin n → Set (ProbPenalty V)} {u' : Fin m → ProbPenalty V}
  {V'' : Fin m → Set (ProbPenalty V)} {K : NonemptyCompacts (BelState V)} :
  K ∈ dnfClass (Fin.append u u') (Fin.append V' V'')
    ↔ K ∈ dnfClass u V' v K ∈ dnfClass u' V''

```

SuFunctor.Universe.coupled\_union\_refinement\_iff lean/su-functor/SuFunctor/Universe/LocallyClosedLayer.lean, line 138

```

theorem coupled_union_refinement_iff {P : (K → ℝ≥0∞) → Prop} {u : K → ℝ≥0∞}
  {V' : Set (K → ℝ≥0∞)} {u' : Fin m → K → ℝ≥0∞}
  {V'' : Fin m → Set (K → ℝ≥0∞)}
  (hPc : ∀ c, SelValid V' c → P (selSup u c)) :
  (∀ σ, P σ → CoupledSat σ u V' → ∃ j, CoupledSat σ (u' j) (V'' j))
    ↔ ∀ c, SelValid V' c → Blocked u V' u' c

```

SuFunctor.Universe.exists\_naive\_cover\_incomplete lean/su-functor/SuFunctor/Universe/LocallyClosedLayer.lean, line 235

```

theorem exists_naive_cover_incomplete :
  ∃ u v : ProbPenalty (Fin 2),
    LowerSemicontinuous u ∧ LowerSemicontinuous v
    ∧ (∀ σ, CoupledSat σ u {u ∪ v} ↔ CoupledSat σ u {v})
    ∧ ¬ u ∪ v ≤ v

```

SuFunctor.Universe.exists\_essential\_floor\_raise lean/su-functor/SuFunctor/Universe/LocallyClosedLayer.lean, line 294

```

theorem exists_essential_floor_raise :
  ∃ u₁ w : ProbPenalty (Fin 2),
    LowerSemicontinuous u₁ ∧ LowerSemicontinuous w
    ∧ (∀ σ : ProbPenalty (Fin 2), CoupledSat σ 0 {(τ : ProbPenalty (Fin 2))} →
      CoupledSat σ u₁ {(τ : ProbPenalty (Fin 2))} v CoupledSat σ 0 {w})
    ∧ ¬ (∀ σ : ProbPenalty (Fin 2), LowerSemicontinuous σ →
      CoupledSat σ 0 {(τ : ProbPenalty (Fin 2))} →
      CoupledSat σ u₁ {(τ : ProbPenalty (Fin 2))})
    ∧ ¬ (∀ σ : ProbPenalty (Fin 2), LowerSemicontinuous σ →
      CoupledSat σ 0 {(τ : ProbPenalty (Fin 2))} →
      CoupledSat σ 0 {w})
    ∧ ¬ u₁ ≤ (0 : ProbPenalty (Fin 2))

```

SuFunctor.Universe.le\_infProj\_iff lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 79

```

theorem le_infProj_iff {p : Z → W} {σ : Z → ℝ≥0∞} {φ : W → ℝ≥0∞} :
  φ ≤ infProj p σ ↔ φ ∘ p ≤ σ

```

SuFunctor.Universe.le\_hide\_iff lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 195

```

theorem le_hide_iff {p : Z → W} {σ : Z → ℝ≥0∞} {φ : W → ℝ≥0∞}
  (hφ : LowerSemicontinuous φ) :
  φ ≤ hide p σ ↔ φ ∘ p ≤ σ

```

SuFunctor.Universe.hide\_sup\_le lean/su-functor/SuFunctor/Universe/HideFrobenius.lean, line 156

```

theorem hide_sup_le (p : Z → W) (σ : Z → ℝ≥0∞) {g : W → ℝ≥0∞}
  (hg : LowerSemicontinuous g) :
  hide p σ ∪ g ≤ hide p (σ ∪ g ∘ p)

```

SuFunctor.Universe.hide\_add\_le lean/su-functor/SuFunctor/Universe/HideFrobenius.lean, line 168

```

theorem hide_add_le (p : Z → W) (σ : Z → ℝ≥0∞) {g : W → ℝ≥0∞}
  (hg : LowerSemicontinuous g) :
  hide p σ + g ≤ hide p (σ + g ∘ p)

```

SuFunctor.Universe.exists\_lowerSemicontinuous\_hide\_add\_ne lean/su-functor/SuFunctor/Universe/HideFrobenius.lean, line 309

```
theorem exists_lowerSemicontinuous_hide_add_ne :
  ∃ (σ : ProbabilityMeasure ℕ → ℝ≥0∞) (g : ℝ≥0∞ → ℝ≥0∞),
    LowerSemicontinuous σ ∧ LowerSemicontinuous g ∧
    hide liveMass (σ + g ∘ liveMass) ≠ hide liveMass σ + g
```

SuFunctor.Universe.lscEnv\_charFn lean/su-functor/SuFunctor/Universe/HideFrobenius.lean, line 111

```
theorem lscEnv_charFn (C : Set Z) : lscEnv (charFn C) = charFn (closure C)
```

SuFunctor.Universe.exists\_judgment\_separating\_hide\_add lean/su-functor/SuFunctor/Universe/HideFrobenius.lean, line 347

```
theorem exists_judgment_separating_hide_add :
  ∃ (σ : ProbabilityMeasure ℕ → ℝ≥0∞) (g φ : ℝ≥0∞ → ℝ≥0∞),
    LowerSemicontinuous σ ∧ LowerSemicontinuous g ∧
    LowerSemicontinuous φ ∧
    φ ≤ hide liveMass (σ + g ∘ liveMass) ∧
    ¬ φ ≤ hide liveMass σ + g
```

SuFunctor.Universe.hide\_sup\_comp lean/su-functor/SuFunctor/Universe/HideFrobenius.lean, line 189

```
theorem hide_sup_comp [T2Space W] {p : Z → W} (hp : Continuous p)
  {σ : Z → ℝ≥0∞} (hσ : LowerSemicontinuous σ)
  (hcpt : ∀ c : ℝ≥0∞, c ≠ ⊤ → IsCompact {z | σ z ≤ c})
  {g : W → ℝ≥0∞} (hg : LowerSemicontinuous g) :
  hide p (σ ∪ g ∘ p) = hide p σ ∪ g
```

SuFunctor.Universe.hide\_add\_comp lean/su-functor/SuFunctor/Universe/HideFrobenius.lean, line 209

```
theorem hide_add_comp [T2Space W] {p : Z → W} (hp : Continuous p)
  {σ : Z → ℝ≥0∞} (hσ : LowerSemicontinuous σ)
  (hcpt : ∀ c : ℝ≥0∞, c ≠ ⊤ → IsCompact {z | σ z ≤ c})
  {g : W → ℝ≥0∞} (hg : LowerSemicontinuous g) :
  hide p (σ + g ∘ p) = hide p σ + g
```

Qtsl.Imprecise.hurwicz\_ge\_iff\_exists\_chord lean/qtsl-aux/QtslAux/Imprecise/HurwiczCrescents.lean, line 67

```
theorem hurwicz_ge_iff_exists_chord (α : ℚ) (hα0 : 0 ≤ α) (hα1 : α ≤ 1)
  (M : ι → FinPMF X) (f : X → ℚ) (l0 : ℚ) :
  l0 ≤ hurwicz α M f
  ↔ ∃ a b : ℚ, l0 ≤ α * a + (1 - α) * b
    ∧ a ≤ upperE M f ∧ b ≤ lowerE M f
```

Qtsl.Imprecise.hurwiczSuperlevel\_eq\_iUnion\_crescents lean/qtsl-aux/QtslAux/Imprecise/HurwiczCrescents.lean, line 81

```
theorem hurwiczSuperlevel_eq_iUnion_crescents (α : ℚ) (hα0 : 0 ≤ α)
  (hα1 : α ≤ 1) (M : ι → FinPMF X) (l0 : ℚ) :
  {f : X → ℚ | l0 ≤ hurwicz α M f}
  = ⋃ p : {q : ℚ × ℚ // l0 ≤ α * q.1 + (1 - α) * q.2},
    ({f : X → ℚ | p.val.1 ≤ upperE M f}
     ∩ {f : X → ℚ | p.val.2 ≤ lowerE M f})
```

Qtsl.Imprecise.finite\_chords\_strict lean/qtsl-aux/QtslAux/Imprecise/HurwiczCrescents.lean, line 112

```
theorem finite_chords_strict {α l0 : ℚ} (hα0 : 0 < α) (hα1 : α < 1)
  (S : Finset (ℚ × ℚ)) (hS : ∀ p ∈ S, l0 ≤ α * p.1 + (1 - α) * p.2) :
  ∃ (M : Bool → FinPMF Bool) (f : Bool → ℚ),
    l0 ≤ hurwicz α M f ∧
    ∀ p ∈ S, ¬(p.2 ≤ lowerE M f ∧ p.1 ≤ upperE M f)
```



Qtsl.Imprecise.**le\_hurwicz\_of\_chord** lean/qtsl-aux/QtslAux/Imprecise/HurwiczCrescents.lean, line 54

```
theorem le_hurwicz_of_chord {α lo a b : ℚ} (hα0 : 0 ≤ α) (hα1 : α ≤ 1)
  {M : ι → FinPMF X} {f : X → ℚ}
  (hchord : lo ≤ α * a + (1 - α) * b)
  (ha : a ≤ upperE M f) (hb : b ≤ lowerE M f) :
  lo ≤ hurwicz α M f
```

Qtsl.Imprecise.**penUpperE** lean/qtsl-aux/QtslAux/Imprecise/PenaltyEnvelope.lean, line 100

```
def penUpperE (M : ι → FinPMF X) (c : ι → ℚ) (f : X → ℚ) : ℚ
```

Qtsl.Imprecise.**penLowerE** lean/qtsl-aux/QtslAux/Imprecise/PenaltyEnvelope.lean, line 105

```
def penLowerE (M : ι → FinPMF X) (c : ι → ℚ) (f : X → ℚ) : ℚ
```

Qtsl.Imprecise.**penUpperE\_zero\_cost** lean/qtsl-aux/QtslAux/Imprecise/PenaltyEnvelope.lean, line 138

```
theorem penUpperE_zero_cost (M : ι → FinPMF X) (f : X → ℚ) :
  penUpperE M 0 f = upperE M f
```

Qtsl.Imprecise.**penUpperE\_eq\_sup\_levelProfile** lean/qtsl-aux/QtslAux/Imprecise/PenaltyEnvelope.lean, line 332

```
theorem penUpperE_eq_sup_levelProfile (M : ι → FinPMF X) (c : ι → ℚ)
  (f : X → ℚ) :
  penUpperE M c f
  = univ.sup' univ_nonempty (fun i₀ => levelProfile M c f i₀ - c i₀)
```

Qtsl.Imprecise.**penUpperE\_compFam** lean/qtsl-aux/QtslAux/Imprecise/PenaltyKernelCompose.lean, line 71

```
theorem penUpperE_compFam (M : ι → FinPMF X) (c : ι → ℚ)
  (K : (x : X) → κ x → FinPMF Y) (d : (x : X) → κ x → ℚ)
  (f : X × Y → ℚ) :
  penUpperE (compFam M K) (penCompCost M c d) f
  = penUpperE M c fun x => penUpperE (K x) (d x) fun y => f (x, y)
```

Qtsl.Imprecise.**penUpperE\_mix\_le** lean/qtsl-aux/QtslAux/Imprecise/PenaltyEnvelope.lean, line 219

```
theorem penUpperE_mix_le (M : ι → FinPMF X) (c : ι → ℚ) (f g : X → ℚ)
  {t : ℚ} (h₀ : 0 ≤ t) (h₁ : t ≤ 1) :
  penUpperE M c (fun x => t * f x + (1 - t) * g x)
  ≤ t * penUpperE M c f + (1 - t) * penUpperE M c g
```

Qtsl.Imprecise.Mutants.**penUpperE\_not\_posHomogeneous** lean/qtsl-aux/QtslAux/Imprecise/PenaltyEnvelope.lean, line 362

```
theorem penUpperE_not_posHomogeneous :
  penUpperE (fun _ : Unit => FinPMF.dirac (default : Unit)) (fun _ => 1)
    (fun x => (2 : ℚ) * (fun _ : Unit => (0 : ℚ)) x)
  ≠ 2 * penUpperE (fun _ : Unit => FinPMF.dirac (default : Unit))
    (fun _ => 1) (fun _ : Unit => (0 : ℚ))
```

Qtsl.Imprecise.**penUpperE\_dirac\_family** lean/qtsl-aux/QtslAux/Imprecise/PenaltyEnvelope.lean, line 292

```
theorem penUpperE_dirac_family (w : M → ℚ) (g : M → ℚ) :
  penUpperE (fun m => FinPMF.dirac m) w g
  = univ.sup' univ_nonempty (fun m => g m - w m)
```

SuFunctor.Plain.**MeasJoint** lean/su-functor/SuFunctor/Plain/MeasJoint.lean, line 110

```
def MeasJoint (X : Type u) [MeasurableSpace X] : Type u
```

SuFunctor.Plain.Joint.**toMeas\_push** lean/su-functor/SuFunctor/Plain/GeneralSeam.lean, line 131

```
theorem toMeas_push (h : X → Y) (hm : Measurable h) (μ : Joint X) :
  toMeas (μ.push h) = (toMeas μ).push h
```

SuFunctor.Site.**penExtM** lean/su- functor/SuFunctor/Site/GeneralWindow.lean, line 86

```
abbrev penExtM {V W : Set T} (hVW : V  $\subseteq$  W) {X : Type u} [MeasurableSpace X] :  
  MeasPen (Traj V X)  $\rightarrow$  MeasPen (Traj W X)
```

SuFunctor.Site.**penExtM\_comp\_toMeas** lean/su- functor/SuFunctor/Site/GeneralWindow.lean, line 265

```
theorem penExtM_comp_toMeas {V W : Set T} (hVW : V  $\subseteq$  W) {X : Type u}  
  [MeasurableSpace X] (F : MeasPen (Traj V X)) :  
  (penExtM hVW F)  $\circ$  Joint.toMeas = penExt hVW (F  $\circ$  Joint.toMeas)
```

SuFunctor.Site.IsPersistentM.**seamRestrict** lean/su- functor/SuFunctor/Site/GeneralWindow.lean, line 273

```
theorem IsPersistentM.seamRestrict {U :  $\iota \rightarrow$  Set T} {X : Type u}  
  [MeasurableSpace X] { $\Phi$  :  $\forall i$ , Set (MeasPen (Traj (U i) X))}  
  (h $\Phi$  : IsPersistentM U  $\Phi$ ) :  
  IsPersistent U (fun i => (fun F => F  $\circ$  Joint.toMeas) ''  $\Phi$  i)
```

SuFunctor.Plain.Joint.**toMeas\_injective** lean/su- functor/SuFunctor/Plain/GeneralSeam.lean, line 119

```
theorem toMeas_injective [MeasurableSingletonClass X] :  
  Function.Injective (toMeas : Joint X  $\rightarrow$  MeasJoint X)
```

SuFunctor.Site.**penExt** lean/su- functor/SuFunctor/Site/WindowPenalty.lean, line 76

```
abbrev penExt {V W : Set T} (hVW : V  $\subseteq$  W) {X : Type u} :  
  Pen (Traj V X)  $\rightarrow$  Pen (Traj W X)
```

SuFunctor.Plain.Joint.**dense\_ratSeam** lean/su- functor/SuFunctor/Plain/SeamDensity.lean, line 299

```
theorem Joint.dense_ratSeam  
  (hd : DenseRange (Joint.toMeas : Joint X  $\rightarrow$  MeasJoint X)) :  
  Dense (Joint.toMeas '' {v : Joint X | v.IsRatWeighted})
```

SuFunctor.Plain.Joint.**denseRange\_toMeas\_nat** lean/su- functor/SuFunctor/Plain/SeamDensity.lean, line 443

```
theorem Joint.denseRange_toMeas_nat :  
  DenseRange (Joint.toMeas : Joint  $\mathbb{N}$   $\rightarrow$  MeasJoint  $\mathbb{N}$ )
```

SuFunctor.Plain.MeasPen.**le\_of\_seam\_le** lean/su- functor/SuFunctor/Plain/SeamDensity.lean, line 149

```
theorem MeasPen.le_of_seam_le  
  (hd : DenseRange (Joint.toMeas : Joint X  $\rightarrow$  MeasJoint X))  
  {F G : MeasPen X} (hF : LowerSemicontinuous F)  
  (hG : UpperSemicontinuous G)  
  (h :  $\forall v : \text{Joint } X, F \ v.\text{toMeas} \leq G \ v.\text{toMeas}$ ) : F  $\leq$  G
```

SuFunctor.Plain.MeasPen.**iSup\_seam\_eq** lean/su- functor/SuFunctor/Plain/SeamDensity.lean, line 176

```
theorem MeasPen.iSup_seam_eq  
  (hd : DenseRange (Joint.toMeas : Joint X  $\rightarrow$  MeasJoint X))  
  {F : MeasPen X} (hF : LowerSemicontinuous F) :  
   $\sqcup v : \text{Joint } X, F \ v.\text{toMeas} = \sqcup \mu : \text{MeasJoint } X, F \ \mu$ 
```

SuFunctor.Plain.MeasPen.**eq\_of\_seam\_eq** lean/su- functor/SuFunctor/Plain/SeamDensity.lean, line 166

```
theorem MeasPen.eq_of_seam_eq  
  (hd : DenseRange (Joint.toMeas : Joint X  $\rightarrow$  MeasJoint X))  
  {F G : MeasPen X} (hF : Continuous F) (hG : Continuous G)  
  (h :  $\forall v : \text{Joint } X, F \ v.\text{toMeas} = G \ v.\text{toMeas}$ ) : F = G
```

SuFunctor.Plain.**dense\_seam\_inf\_gap** lean/su- functor/SuFunctor/Plain/SeamDensity.lean, line 635

```

theorem dense_seam_inf_gap :
  DenseRange (Joint.toMeas : Joint  $\mathbb{N} \rightarrow$  MeasJoint  $\mathbb{N}$ )  $\wedge$ 
   $\exists F : \text{MeasPen } \mathbb{N}, \text{LowerSemicontinuous } F \wedge$ 
   $(\forall v : \text{Joint } \mathbb{N}, 1 \leq F v.\text{toMeas}) \wedge$ 
   $(\sqcap v : \text{Joint } \mathbb{N}, F v.\text{toMeas}) \neq \sqcap \mu : \text{MeasJoint } \mathbb{N}, F \mu$ 

```

SuFunctor.Site.IsMatching lean/su-functor/SuFunctor/Site/WindowPenalty.lean, line 241

```

def IsMatching (U :  $\iota \rightarrow$  Set T) {X : Type u}
  ( $\Phi : \forall i, \text{Set } (\text{Pen } (\text{Traj } (U i) X))$ ) : Prop

```

SuFunctor.Site.IsPersistent lean/su-functor/SuFunctor/Site/WindowPenalty.lean, line 234

```

def IsPersistent (U :  $\iota \rightarrow$  Set T) {X : Type u}
  ( $\Phi : \forall i, \text{Set } (\text{Pen } (\text{Traj } (U i) X))$ ) : Prop

```

SuFunctor.Site.IsPersistent.image2\_boxplusW lean/su-functor/SuFunctor/Site/WindowPenalty.lean, line 269

```

theorem IsPersistent.image2_boxplusW {U :  $\iota \rightarrow$  Set T}
  {A B C : WireSpan EL} {M : Pat A B} {N : Pat B C}
  { $\Phi : \forall i, \text{Set } (\text{Pen } (\text{Traj } (U i) (\text{Pat.Tot } M)))$ }
  { $\Psi : \forall i, \text{Set } (\text{Pen } (\text{Traj } (U i) (\text{Pat.Tot } N)))$ }
  (h $\Phi : \text{IsPersistent } U \Phi$ ) (h $\Psi : \text{IsPersistent } U \Psi$ ) :
  IsPersistent U (fun i => Set.image2 boxplusW ( $\Phi i$ ) ( $\Psi i$ ))

```

SuFunctor.Site.classDescent lean/su-functor/SuFunctor/Site/ClassDescent.lean, line 243

```

def classDescent (hUo :  $\forall i, \text{IsOpen } (U i)$ ) (hdir : Directed ( $\cdot \subseteq \cdot$ ) U)
  (hcov :  $\bigcup i, U i = \text{univ}$ ) :
  Set (FellClosed  $\alpha$ )  $\approx_o$  Set (ProjLim U) where ...

```

SuFunctor.Site.isConstructibleUnion\_classDescent\_iff lean/su-functor/SuFunctor/Site/ClassDescent.lean, line 268

```

theorem isConstructibleUnion_classDescent_iff (hUo :  $\forall i, \text{IsOpen } (U i)$ )
  (hdir : Directed ( $\cdot \subseteq \cdot$ ) U) (hcov :  $\bigcup i, U i = \text{univ}$ )
  { $\Phi : \text{Set } (\text{FellClosed } \alpha)$ } :
  IsConstructibleUnion (classDescent hUo hdir hcov  $\Phi$ )
   $\leftrightarrow$  IsConstructibleUnion  $\Phi$ 

```

SuFunctor.Site.IsLocallyClosed.classExt lean/su-functor/SuFunctor/Site/ClassDescent.lean, line 167

```

theorem IsLocallyClosed.classExt (hV : IsOpen V) (hVW :  $V \subseteq W$ )
  { $\Phi : \text{Set } (\text{FellClosed } \mathfrak{V})$ } (h $\Phi : \text{IsLocallyClosed } \Phi$ ) :
  IsLocallyClosed (classExt hVW  $\Phi$ )

```

SuFunctor.Site.IsConstructibleUnion.classExt lean/su-functor/SuFunctor/Site/ClassDescent.lean, line 174

```

theorem IsConstructibleUnion.classExt (hV : IsOpen V) (hVW :  $V \subseteq W$ )
  { $\Phi : \text{Set } (\text{FellClosed } \mathfrak{V})$ } (h $\Phi : \text{IsConstructibleUnion } \Phi$ ) :
  IsConstructibleUnion (Site.classExt hVW  $\Phi$ )

```

SuFunctor.Site.bijective\_toProjLim lean/su-functor/SuFunctor/Site/ClassDescent.lean, line 281

```

theorem bijective_toProjLim (hUo :  $\forall i, \text{IsOpen } (U i)$ )
  (hdir : Directed ( $\cdot \subseteq \cdot$ ) U) (hcov :  $\bigcup i, U i = \text{univ}$ ) :
  Function.Bijective (toProjLim U)

```

SuFunctor.Site.clip\_mem\_classExt\_iff lean/su-functor/SuFunctor/Site/ClassDescent.lean, line 159

```

theorem clip_mem_classExt_iff (hVW :  $V \subseteq W$ ) { $\Phi : \text{Set } (\text{FellClosed } \mathfrak{V})$ }
  (F : FellClosed  $\alpha$ ) :
  clip W F  $\in$  classExt hVW  $\Phi \leftrightarrow$  clip V F  $\in \Phi$ 

```

SuFunctor.SingleSiteSemanticUniverse lean/su-functor/SuFunctor/SingleSiteSemanticUniverse.lean, line 489

|                                                                                                                                                                                                                                                                                                                                                                                                          |                                                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| abbrev SingleSiteSemanticUniverse (El : Type u)                                                                                                                                                                                                                                                                                                                                                          |                                                           |
| SuFunctor.QtslSem. <b>SpecJudgLayer</b>                                                                                                                                                                                                                                                                                                                                                                  | lean/su-functor/SuFunctor/QtslSem/SpecJudg.lean, line 108 |
| abbrev SpecJudgLayer (El : Type u)                                                                                                                                                                                                                                                                                                                                                                       |                                                           |
| SuFunctor.QtslSem. <b>cellOfSubset</b>                                                                                                                                                                                                                                                                                                                                                                   | lean/su-functor/SuFunctor/QtslSem/SpecJudg.lean, line 128 |
| def cellOfSubset {El : Type u} {a b : SpecJudgLayer El}<br>{M : Pat a.un b.un} { $\mathcal{U} \mathcal{V}$ : Cpl M} (h : $\mathcal{U} \subseteq \mathcal{V}$ ) :<br>Cell (C := SpecJudgLayer El)<br>((M, $\mathcal{U}$ ) : JudgLayer.JLoose (specJudgOrder El) a b) (M, $\mathcal{V}$ )<br>( $\mathbb{1}$ a) ( $\mathbb{1}$ b)                                                                           |                                                           |
| SuFunctor.Site. <b>wComp_mono</b>                                                                                                                                                                                                                                                                                                                                                                        | lean/su-functor/SuFunctor/Site/Battery.lean, line 127     |
| theorem wComp_mono {M : Pat A B} {N : Pat B C} { $\mathcal{U} \mathcal{U}'$ : WCpl U M}<br>{ $\mathcal{V} \mathcal{V}'$ : WCpl U N} (h <sub>1</sub> : $\forall i, \mathcal{U}.1 i \subseteq \mathcal{U}'.1 i$ )<br>(h <sub>2</sub> : $\forall i, \mathcal{V}.1 i \subseteq \mathcal{V}'.1 i$ ) :<br>$\forall i, (wComp U \mathcal{U} \mathcal{V}).1 i \subseteq (wComp U \mathcal{U}' \mathcal{V}').1 i$ |                                                           |
| SuFunctor.Site. <b>wUnit_transparent</b>                                                                                                                                                                                                                                                                                                                                                                 | lean/su-functor/SuFunctor/Site/Battery.lean, line 87      |
| theorem wUnit_transparent {M : Pat A B} ( $\mathcal{U}$ : WCpl U M) :<br>wCellMap U (PseudoDoubleCategoryStruct.lLeftUnitor (C := WireSpan El) M)<br>(wComp U (wUnit U A) $\mathcal{U}$ ) = $\mathcal{U}$                                                                                                                                                                                                |                                                           |
| CategoryTheory. <b>PseudoDoubleCategoryStruct</b>                                                                                                                                                                                                                                                                                                                                                        | lean/double-cat/DoubleCat/Basic.lean, line 101            |
| class PseudoDoubleCategoryStruct (C : Type u) [Category.{v} C] where ...                                                                                                                                                                                                                                                                                                                                 |                                                           |
| CategoryTheory. <b>PseudoDoubleCategory</b>                                                                                                                                                                                                                                                                                                                                                              | lean/double-cat/DoubleCat/Basic.lean, line 322            |
| class PseudoDoubleCategory (C : Type u) [Category.{v} C]<br>[PseudoDoubleCategoryStruct C] : Prop where ...                                                                                                                                                                                                                                                                                              |                                                           |
| CategoryTheory.PseudoDoubleCategoryStruct. <b>Cell.cast</b>                                                                                                                                                                                                                                                                                                                                              | lean/double-cat/DoubleCat/Basic.lean, line 178            |
| def Cell.cast {a b c d : C} {m : Loose a b} {n : Loose c d} {f f' : a → c}<br>{g g' : b → d} (hf : f = f') (hg : g = g') (α : Cell m n f g) :<br>Cell m n f' g'                                                                                                                                                                                                                                          |                                                           |
| CategoryTheory. <b>LaxDoubleFunctor</b>                                                                                                                                                                                                                                                                                                                                                                  | lean/double-cat/DoubleCat/Functor/Lax.lean, line 48       |
| structure LaxDoubleFunctor extends C ⇒ D where ...                                                                                                                                                                                                                                                                                                                                                       |                                                           |
| SuFunctor.Plain.MeasJoint. <b>push_id</b>                                                                                                                                                                                                                                                                                                                                                                | lean/su-functor/SuFunctor/Plain/MeasJoint.lean, line 156  |
| @[simp] theorem push_id (μ : MeasJoint X) : μ.push id = μ                                                                                                                                                                                                                                                                                                                                                |                                                           |
| SuFunctor.Plain.MeasJoint. <b>push_push</b>                                                                                                                                                                                                                                                                                                                                                              | lean/su-functor/SuFunctor/Plain/MeasJoint.lean, line 163  |
| theorem push_push (h <sub>1</sub> : X → Y) (h <sub>2</sub> : Y → Z) (hm <sub>1</sub> : Measurable h <sub>1</sub> )<br>(hm <sub>2</sub> : Measurable h <sub>2</sub> ) (μ : MeasJoint X) :<br>(μ.push h <sub>1</sub> ).push h <sub>2</sub> = μ.push (h <sub>2</sub> ∘ h <sub>1</sub> )                                                                                                                     |                                                           |
| SuFunctor.Plain.MeasJoint. <b>mass_push</b>                                                                                                                                                                                                                                                                                                                                                              | lean/su-functor/SuFunctor/Plain/MeasJoint.lean, line 170  |
| theorem mass_push (h : X → Y) (hm : Measurable h) (μ : MeasJoint X) :<br>(μ.push h).1.mass = μ.1.mass                                                                                                                                                                                                                                                                                                    |                                                           |
| SuFunctor.Plain. <b>continuous_push</b>                                                                                                                                                                                                                                                                                                                                                                  | lean/su-functor/SuFunctor/Plain/MeasJoint.lean, line 336  |
| theorem continuous_push {h : X → Y} (h_cont : Continuous h) :<br>Continuous (fun μ : MeasJoint X => μ.push h)                                                                                                                                                                                                                                                                                            |                                                           |

SuFunctor.Plain.MeasJoint.**push\_toMeasure** lean/su-functor/SuFunctor/Plain/MeasJoint.lean, line 153

```
@[simp] theorem push_toMeasure (h : X → Y) (μ : MeasJoint X) :
  ((μ.push h).1 : Measure Y) = (μ.1 : Measure X).map h
```

SuFunctor.Universe.**exists\_lowerSemicontinuous\_not\_lowerSemicontinuous\_infProj** lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 558

```
theorem exists_lowerSemicontinuous_not_lowerSemicontinuous_infProj :
  ∃ σ : ProbabilityMeasure ℕ → ℝ≥0∞, LowerSemicontinuous σ ∧
    ¬ LowerSemicontinuous (infProj liveMass σ)
```

SuFunctor.Universe.**le\_lscEnv\_iff** lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 173

```
theorem le_lscEnv_iff {g φ : Z → ℝ≥0∞} (hφ : LowerSemicontinuous φ) :
  φ ≤ lscEnv g ↔ φ ≤ g
```

SuFunctor.Universe.**isCompact\_sublevel\_of\_isTightMeasureSet** lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 317

```
theorem isCompact_sublevel_of_isTightMeasureSet
  {σ : ProbabilityMeasure ℳ → ℝ≥0∞} (hσ : LowerSemicontinuous σ)
  {c : ℝ≥0∞}
  (ht : IsTightMeasureSet
    {((μ : ProbabilityMeasure ℳ) : Measure ℳ) | μ ∈ {ν | σ ν ≤ c}}) :
  IsCompact {ν | σ ν ≤ c}
```

SuFunctor.Universe.**sublevel\_infProj\_eq** lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 213

```
theorem sublevel_infProj_eq [T1Space W] {p : Z → W} (hp : Continuous p)
  {σ : Z → ℝ≥0∞} (hσ : LowerSemicontinuous σ)
  (hcpt : ∀ c : ℝ≥0∞, c ≠ ⊤ → IsCompact {z | σ z ≤ c})
  {c : ℝ≥0∞} (hc : c ≠ ⊤) :
  {w | infProj p σ w ≤ c} = p '' {z | σ z ≤ c}
```

SuFunctor.Universe.**exists\_infProj\_attained** lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 270

```
theorem exists_infProj_attained [T1Space W] {p : Z → W} (hp : Continuous p)
  {σ : Z → ℝ≥0∞} (hσ : LowerSemicontinuous σ)
  (hcpt : ∀ c : ℝ≥0∞, c ≠ ⊤ → IsCompact {z | σ z ≤ c})
  {v : W} (hv : infProj p σ v ≠ ⊤) :
  ∃ z, p z = v ∧ σ z = infProj p σ v
```

SuFunctor.Universe.**isCompact\_sublevel\_infProj** lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 283

```
theorem isCompact_sublevel_infProj [T1Space W] {p : Z → W} (hp : Continuous p)
  {σ : Z → ℝ≥0∞} (hσ : LowerSemicontinuous σ)
  (hcpt : ∀ c : ℝ≥0∞, c ≠ ⊤ → IsCompact {z | σ z ≤ c})
  {c : ℝ≥0∞} (hc : c ≠ ⊤) :
  IsCompact {w | infProj p σ w ≤ c}
```

SuFunctor.Universe.**lowerSemicontinuous\_infProj** lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 292

```
theorem lowerSemicontinuous_infProj [T2Space W] {p : Z → W} (hp : Continuous p)
  {σ : Z → ℝ≥0∞} (hσ : LowerSemicontinuous σ)
  (hcpt : ∀ c : ℝ≥0∞, c ≠ ⊤ → IsCompact {z | σ z ≤ c}) :
  LowerSemicontinuous (infProj p σ)
```

SuFunctor.Universe.**sublevel\_infProj\_margFst\_eq** lean/su-functor/SuFunctor/Universe/Marginalization.lean, line 347

```
theorem sublevel_infProj_margFst_eq
  {σ : ProbabilityMeasure (ℳ × ℳ) → ℝ≥0∞} (hσ : LowerSemicontinuous σ)
  (ht : ∀ c : ℝ≥0∞, c ≠ ⊤ → IsTightMeasureSet
    {((ρ : ProbabilityMeasure (ℳ × ℳ)) : Measure (ℳ × ℳ)) | ρ ∈ {ν | σ ν ≤ c}})
  {c : ℝ≥0∞} (hc : c ≠ ⊤) :
  {w | infProj margFst σ w ≤ c} = margFst '' {ρ | σ ρ ≤ c}
```

|                                                                                                                                                                                                                                                   |                                                                     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|
| <b>SuFunctor.Plain.Pen.pull</b>                                                                                                                                                                                                                   | lean/su-functor/SuFunctor/Plain/Penalty.lean, line 100              |
| def pull (h : X → Y) (f : Pen Y) : Pen X                                                                                                                                                                                                          |                                                                     |
| <b>SuFunctor.Site.siteSpecFunctor</b>                                                                                                                                                                                                             | lean/su-functor/SuFunctor/Site/SiteFunctor.lean, line 128           |
| def siteSpecFunctor (U : $\mathfrak{u} \rightarrow \text{Set } T$ ) (El : Type u) :<br>SpanValued.{u + 1, u, u, u + 1, u} (WireSpan El) where ...                                                                                                 |                                                                     |
| <b>SuFunctor.Plain.measSpecObj</b>                                                                                                                                                                                                                | lean/su-functor/SuFunctor/Plain/GeneralFunctor.lean, line 56        |
| def measSpecObj : MeasCat.{u} <sup>op</sup> $\Rightarrow$ Type u where ...                                                                                                                                                                        |                                                                     |
| <b>SuFunctor.Site.SiteSpecTotal</b>                                                                                                                                                                                                               | lean/su-functor/SuFunctor/Site/SiteFunctor.lean, line 279           |
| abbrev SiteSpecTotal (U : $\mathfrak{u} \rightarrow \text{Set } T$ ) (El : Type u)                                                                                                                                                                |                                                                     |
| <b>SuFunctor.Site.siteSpecTotal_pseudoDoubleCategory</b>                                                                                                                                                                                          | lean/su-functor/SuFunctor/Site/SiteFunctor.lean, line 285           |
| theorem siteSpecTotal_pseudoDoubleCategory (U : $\mathfrak{u} \rightarrow \text{Set } T$ ) (El : Type u) :<br>$\exists \_ : \text{PseudoDoubleCategory (SiteSpecTotal U El)}, \text{True}$                                                        |                                                                     |
| <b>SuFunctor.Site.siteSpecProj_isPseudo</b>                                                                                                                                                                                                       | lean/su-functor/SuFunctor/Site/SiteFunctor.lean, line 295           |
| theorem siteSpecProj_isPseudo (U : $\mathfrak{u} \rightarrow \text{Set } T$ ) (El : Type u) :<br>(siteSpecProj U El).IsPseudo                                                                                                                     |                                                                     |
| <b>SuFunctor.Site.siteSpecProj_discreteOpfibration</b>                                                                                                                                                                                            | lean/su-functor/SuFunctor/Site/SiteFunctor.lean, line 304           |
| theorem siteSpecProj_discreteOpfibration (U : $\mathfrak{u} \rightarrow \text{Set } T$ ) (El : Type u) :<br>IsDiscreteDoubleOpfibration (SpanValued.proj (siteSpecFunctor U El))                                                                  |                                                                     |
| <b>SuFunctor.SingleSiteSemanticUniverse</b>                                                                                                                                                                                                       | lean/su-functor/SuFunctor/SingleSiteSemanticUniverse.lean, line 489 |
| abbrev SingleSiteSemanticUniverse (El : Type u)                                                                                                                                                                                                   |                                                                     |
| <b>SuFunctor.dnfSpecFunctor</b>                                                                                                                                                                                                                   | lean/su-functor/SuFunctor/SingleSiteSemanticUniverse.lean, line 365 |
| def dnfSpecFunctor (El : Type u) :<br>SpanValued.{u + 1, u, u, u + 1, u} (WireSpan El) where ...                                                                                                                                                  |                                                                     |
| <b>SuFunctor.singleSiteUniverseProj</b>                                                                                                                                                                                                           | lean/su-functor/SuFunctor/SingleSiteSemanticUniverse.lean, line 500 |
| def singleSiteUniverseProj (El : Type u) :<br>LaxDoubleFunctor (SingleSiteSemanticUniverse El) (WireSpan El)                                                                                                                                      |                                                                     |
| <b>SuFunctor.singleSiteUniverseProj_isPseudo</b>                                                                                                                                                                                                  | lean/su-functor/SuFunctor/SingleSiteSemanticUniverse.lean, line 505 |
| theorem singleSiteUniverseProj_isPseudo (El : Type u) :<br>(singleSiteUniverseProj El).IsPseudo                                                                                                                                                   |                                                                     |
| <b>SuFunctor.singleSiteUniverseProj_discreteOpfibration</b>                                                                                                                                                                                       | lean/su-functor/SuFunctor/SingleSiteSemanticUniverse.lean, line 514 |
| theorem singleSiteUniverseProj_discreteOpfibration (El : Type u) :<br>IsDiscreteDoubleOpfibration (singleSiteUniverseProj El)                                                                                                                     |                                                                     |
| <b>SuFunctor.specMap_modelClass_eq</b>                                                                                                                                                                                                            | lean/su-functor/SuFunctor/SingleSiteSemanticUniverse.lean, line 533 |
| theorem specMap_modelClass_eq ( $\phi : A \rightarrow B$ )<br>(h $\phi$ : Function.Surjective (res $\phi$ )) (D : DnfSpec A) :<br>specMap $\phi$ D.modelClass<br>= (dnfPull (res $\phi$ ) D).modelClass $\cap$ Set.range (Pen.pull (res $\phi$ )) |                                                                     |
| <b>SuFunctor.specMap_modelClass_subset</b>                                                                                                                                                                                                        | lean/su-functor/SuFunctor/SingleSiteSemanticUniverse.lean, line 540 |

```

theorem specMap_modelClass_subset (φ : A → B)
  (hφ : Function.Surjective (res φ)) (D : DnfSpec A) :
  specMap φ D.modelClass ⊆ (dnfPull (res φ) D).modelClass

```

SuFunctor.Universe.SpecDnf.isConstructibleUnion\_probRead lean/su-functor/SuFunctor/SingleSiteSemanticUniverse.lean, line 594

```

theorem Universe.SpecDnf.isConstructibleUnion_probRead
  (D : SpecDnf (Joint V))
  (hfloor : ∀ a ∈ D.probRead.atoms, LowerSemicontinuous a.floor)
  (hmarks : ∀ a ∈ D.probRead.atoms, ∀ v ∈ a.marks, LowerSemicontinuous v) :
  IsConstructibleUnion D.probRead.crescentClass

```

SuFunctor.Site.singleton\_boxplusW lean/su-functor/SuFunctor/Site/Battery.lean, line 103

```

theorem singleton_boxplusW {W : Set T} {M : Pat A B} {N : Pat B C}
  (u1 : Pen (Traj W (Pat.Tot M))) (u2 : Pen (Traj W (Pat.Tot N))) :
  Set.image2 boxplusW ({u1} : Set (Pen (Traj W (Pat.Tot M))))
    ({u2} : Set (Pen (Traj W (Pat.Tot N)))) =
  {boxplusW u1 u2}

```

SuFunctor.Site.transport\_singletonW lean/su-functor/SuFunctor/Site/Battery.lean, line 113

```

theorem transport_singletonW {W : Set T} (φ : A → B)
  (σ : Pen (Traj W (states A))) :
  Pen.pull (trajMap (res φ)) '' ({σ} : Set (Pen (Traj W (states A)))) =
  {Pen.pull (trajMap (res φ)) σ}

```

SuFunctor.Site.wIdealPoint\_le\_iff lean/su-functor/SuFunctor/Site/Battery.lean, line 121

```

theorem wIdealPoint_le_iff {W : Set T} (σ : Pen (Traj W (states A)))
  (S : Set (Pen (Traj W (states A)))) :
  ({σ} : Set (Pen (Traj W (states A)))) ⊆ S ↔ σ ∈ S

```

SuFunctor.Universe.exists\_coupledSat\_add\_fails lean/su-functor/SuFunctor/Universe/SpecDnf.lean, line 420

```

theorem exists_coupledSat_add_fails :
  ∃ (σ1 σ2 u1 u2 : Unit → ℝ≥0) (V1 V2 : Set (Unit → ℝ≥0)),
    T ∈ V1 ∧ T ∈ V2 ∧
    CoupledSat σ1 u1 V1 ∧ CoupledSat σ2 u2 V2 ∧
    ¬ CoupledSat (σ1 + σ2) (u1 + u2) (V1 ∪ V2)

```

SuFunctor.Universe.add\_comp\_safety\_le lean/su-functor/SuFunctor/Universe/SpecDnf.lean, line 405

```

theorem add_comp_safety_le (g1 : K → K1) (g2 : K → K2)
  {u1 σ1 : K1 → ℝ≥0} {u2 σ2 : K2 → ℝ≥0}
  (h1 : u1 ≤ σ1) (h2 : u2 ≤ σ2) :
  (u1 ∘ g1) + (u2 ∘ g2) ≤ (σ1 ∘ g1) + (σ2 ∘ g2)

```

CategoryTheory.ImplDouble.cell\_subsingleton lean/double-cat/DoubleCat/ImplDouble.lean, line 68

```

theorem cell_subsingleton {a b c d : ImplDouble C} (m : Loose a b)
  (n : Loose c d) (f : a → c) (g : b → d) :
  Subsingleton (Cell m n f g)

```

SuFunctor.Site.mem\_wComp\_iff lean/su-functor/SuFunctor/Site/Battery.lean, line 69

```

theorem mem_wComp_iff {M : Pat A B} {N : Pat B C} {W : WCpl U M}
  {V : WCpl U N} {i : ι}
  {w : Pen (Traj (U i) (Pat.Tot (Pat.comp M N)))) :
  w ∈ (wComp U W V).1 i ↔
    ∃ u1 ∈ W.1 i, ∃ u2 ∈ V.1 i, boxplusW u1 u2 = w

```

SuFunctor.Site.wComp\_nonempty lean/su-functor/SuFunctor/Site/Battery.lean, line 78



```

theorem wComp_nonempty {M : Pat A B} {N : Pat B C} {W : WCpl U M}
  {V : WCpl U N} (h1 : V i, (W.l i).Nonempty)
  (h2 : V i, (V.l i).Nonempty) :
  V i, ((wComp U W V).l i).Nonempty

```

SuFunctor.Site.**taxedBoxplusW\_breaks\_unit\_transparency** lean/su-functor/SuFunctor/Site/Battery.lean, line 152

```

theorem taxedBoxplusW_breaks_unit_transparency {W : Set T} (M : Pat A B)
  {c : ℝ≥0∞} (hc : c ≠ 0) :
  ¬ V u : Pen (Traj W (Pat.Tot M)),
  Pen.pull (trajMap (PatCell.tot
    (PseudoDoubleCategoryStruct.lLeftUnitor (C := WireSpan El) M)))
  (taxedBoxplusW c (unitPenW A W) u) = u

```

SuFunctor.Site.**funnelCompW\_breaks\_unit\_coherence** lean/su-functor/SuFunctor/Site/Battery.lean, line 176

```

theorem funnelCompW_breaks_unit_coherence {W : Set T} (M : Pat A B) :
  ¬ V W : Set (Pen (Traj W (Pat.Tot M))),
  Pen.pull (trajMap (PatCell.tot
    (PseudoDoubleCategoryStruct.lLeftUnitor (C := WireSpan El) M))) ''
  funnelCompW ({unitPenW A W} :
    Set (Pen (Traj W (Pat.Tot (Pat.id A))))) W = W

```

QtSl.Imprecise.**sublevel\_compositeG\_subset\_glueSetG** lean/qtSl-aux/QtSlAux/Imprecise/ContainmentGeneral.lean, line 111

```

theorem sublevel_compositeG_subset_glueSetG (m : V i, J → Mi i)
  (f : V i, Mi i → ENNRat) (F : ENNRat) :
  {v | compositePenaltyG m f v ≤ F} ≤ glueSetG m f F

```

QtSl.Imprecise.**certificate\_transfer** lean/qtSl-aux/QtSlAux/Imprecise/ContainmentGeneral.lean, line 121

```

theorem certificate_transfer (m : V i, J → Mi i) (f : V i, Mi i → ENNRat)
  (F : ENNRat) {P : J → Prop} (h : V v ∈ glueSetG m f F, P v) :
  V v, compositePenaltyG m f v ≤ F → P v

```

QtSl.Imprecise.**sublevel\_pmComposite\_subset\_glue** lean/qtSl-aux/QtSlAux/Imprecise/ContainmentGeneral.lean, line 307

```

theorem sublevel_pmComposite_subset_glue {proj : V i, X → Xi i}
  (hproj : V i, Continuous (proj i))
  (f : V i, ProbabilityMeasure (Xi i) → ENNRat) (F : ENNRat) :
  {v | pmCompositePenalty proj (fun i => (hproj i).measurable) f v ≤ F}
  ≤ glueSetG (pmMarginal proj fun i => (hproj i).measurable) f F

```

QtSl.Imprecise.**isClosed\_pmGlueSet** lean/qtSl-aux/QtSlAux/Imprecise/ContainmentGeneral.lean, line 300

```

theorem isClosed_pmGlueSet {proj : V i, X → Xi i}
  (hproj : V i, Continuous (proj i))
  {f : V i, ProbabilityMeasure (Xi i) → ENNRat}
  (hf : V i, LowerSemicontinuous (f i)) (F : ENNRat) :
  IsClosed (glueSetG (pmMarginal proj fun i => (hproj i).measurable) f F)

```

QtSl.Imprecise.**forall\_sublevel\_eq\_iff** lean/qtSl-aux/QtSlAux/Imprecise/ContainmentGeneral.lean, line 162

```

theorem forall_sublevel_eq_iff {g h : J → ENNRat} :
  (V F, {v | g v ≤ F} = {v | h v ≤ F}) ↔ g = h

```

QtSl.Imprecise.ENNRat.**add\_eq\_max\_iff** lean/qtSl-aux/QtSlAux/Imprecise/ContainmentGeneral.lean, line 81

```

theorem ENNRat.add_eq_max_iff (a b : ENNRat) :
  a + b = max a b ↔ min a b = 0 v max a b = T

```

QtSl.Imprecise.**binaryComposite\_ne\_glue** lean/qtSl-aux/QtSlAux/Imprecise/ContainmentGeneral.lean, line 191

```

theorem binaryComposite_ne_glue :
  (fun v : Unit => (fun _ : Unit => (1 : ENNRat)) v
    + (fun _ : Unit => (1 : ENNRat)) v)
  ≠ fun v : Unit => max ((fun _ : Unit => (1 : ENNRat)) v)
    ((fun _ : Unit => (1 : ENNRat)) v)

```

Qtsl.Imprecise.sublevel\_ssubset\_glue\_witness lean/qtsl-aux/QtslAux/Imprecise/ContainmentGeneral.lean, line 206

```

theorem sublevel_ssubset_glue_witness :
  {v : Unit | compositePenaltyG (fun _ : Bool => id)
    (fun _ : Bool => fun _ : Unit => (1 : ENNRat)) v ≤ 1}
  ⊆ glueSetG (fun _ : Bool => id)
    (fun _ : Bool => fun _ : Unit => (1 : ENNRat)) 1

```

Qtsl.Imprecise.signed\_relaxation\_fails lean/qtsl-aux/QtslAux/Imprecise/MarginalCompat.lean, line 179

```

theorem signed_relaxation_fails :
  ∃ f : Fin 2 → FinPMF (Fin 2) → ℚ,
    ¬ ∀ (i : Fin 2) (v : FinPMF (Fin 2)),
      f i v ≤ compositePenaltyQ (Xi := fun _ => Fin 2)
        (fun _ v => v) f v

```

Qtsl.Imprecise.subunit\_weight\_fails lean/qtsl-aux/QtslAux/Imprecise/MarginalCompat.lean, line 189

```

theorem subunit_weight_fails :
  ∃ (c : Fin 1 → ℚ) (f : Fin 1 → FinPMF (Fin 2) → ℚ),
    (∀ i v, 0 ≤ f i v) ∧ (∀ i, 0 ≤ c i) ∧
    ¬ ∀ (i : Fin 1) (v : FinPMF (Fin 2)),
      f i v ≤ compositePenaltyWQ (Xi := fun _ => Fin 2) c
        (fun _ v => v) f v

```

SuFunctor.Plain.jointEquivBel lean/su-functor/SuFunctor/Plain/Seam.lean, line 100

```

def jointEquivBel : Joint V ≈ Universe.BelState V where ...

```

SuFunctor.Plain.jointEquivProb lean/su-functor/SuFunctor/Plain/Seam.lean, line 147

```

def jointEquivProb :
  {μ : Joint V // μ.wt.sum (fun _ v => v) = 1} ≈ Universe.ProbState V where ...

```

SuFunctor.Plain.penEquiv lean/su-functor/SuFunctor/Plain/Seam.lean, line 200

```

def penEquiv : Pen V ≈0 (Universe.BelState V → ℝ≥0∞) where ...

```

SuFunctor.Plain.Joint.toMeas lean/su-functor/SuFunctor/Plain/GeneralSeam.lean, line 76

```

def toMeas (μ : Joint X) : MeasJoint X

```

SuFunctor.Plain.Joint.pull\_toMeas lean/su-functor/SuFunctor/Plain/GeneralSeam.lean, line 150

```

theorem pull_toMeas (h : X → Y) (hm : Measurable h) (F : MeasPen Y) :
  (MeasPen.pull h F) ∘ toMeas = Pen.pull h (F ∘ toMeas)

```

SuFunctor.Plain.Joint.image\_pull\_toMeas lean/su-functor/SuFunctor/Plain/GeneralSeam.lean, line 159

```

theorem image_pull_toMeas (h : X → Y) (hm : Measurable h)
  (ℳ : Set (MeasPen Y)) :
  (fun F => F ∘ toMeas) '' (MeasPen.pull h '' ℳ) =
    Pen.pull h '' ((fun F => F ∘ toMeas) '' ℳ)

```

SuFunctor.Universe.General.coupledSat\_pullPen\_iff lean/su-functor/SuFunctor/Universe/General/TransportGeneral.lean, line 239

```

theorem coupledSat_pullPen_iff (hθ : Continuous θ)
  (hsurj : Function.Surjective (probPush hθ))
  {σ u : Penalty Y} {V : Set (Penalty Y)} :
  CoupledSat (pullPen hθ σ) (pullPen hθ u) (pullPen hθ '' V)
  ↔ CoupledSat σ u V

```

SuFunctor.Universe.General.**transportClass\_crescentClass**      lean/su-functor/SuFunctor/Universe/General/TransportGeneral.lean, line 418

```

theorem transportClass_crescentClass [Nonempty X] (hθ : Continuous θ)
  (hsurj : Function.Surjective
    (push θ : SubProbabilityMeasure X → SubProbabilityMeasure Y))
  (hmap : IsClosedMap
    (push θ : SubProbabilityMeasure X → SubProbabilityMeasure Y))
  (u : Penalty Y) (V : Set (Penalty Y)) :
  transportClass θ (crescentClass u V)
  = crescentClass (pullPen hθ u) (pullPen hθ '' V)
  n {L | Saturated θ (L : Set (SubProbabilityMeasure X))}

```

SuFunctor.Universe.General.**exists\_transport\_not\_subset\_pulled\_crescent**      lean/su-functor/SuFunctor/Universe/General/TransportGeneral.lean, line 665

```

theorem exists_transport_not_subset_pulled_crescent :
  ∃ (θ : Unit → Bool) (hθ : Continuous θ) (u : Penalty Bool)
  (V : Set (Penalty Bool)),
  LowerSemicontinuous u ∧ (∀ v ∈ V, LowerSemicontinuous v) ∧ V.Finite ∧
  ¬ (transportClass θ (crescentClass u V)
    ⊆ crescentClass (pullPen hθ u) (pullPen hθ '' V))

```

SuFunctor.Universe.General.**compile\_add\_radProd**      lean/su-functor/SuFunctor/Universe/General/TransportGeneral.lean, line 478

```

theorem compile_add_radProd [Nonempty X] (f g : Penalty X) :
  compile (f + g) = radProd (compile f) (compile g)

```

SuFunctor.Universe.General.**probPush\_surjective\_of\_section**      lean/su-functor/SuFunctor/Universe/General/TransportGeneral.lean, line 208

```

theorem probPush_surjective_of_section (hθ : Continuous θ) {s : Y → X}
  (hs_mble : Measurable s) (hs : Function.RightInverse s θ) :
  Function.Surjective (probPush hθ)

```

SuFunctor.Universe.**cross\_site\_union\_refinement\_iff**      lean/su-functor/SuFunctor/Universe/DemonicClock.lean, line 262

```

theorem cross_site_union_refinement_iff {P : (K → ℝ≥0∞) → Prop}
  {u w : K → ℝ≥0∞} {V : Set (K → ℝ≥0∞)} {u' : Fin m → K → ℝ≥0∞}
  {V' : Fin m → Set (K → ℝ≥0∞)}
  (hPc : ∀ c, SelValid V' c → P (selSup u c ∪ w)) :
  (∀ σ, P σ → CoupledSat σ (u ∪ w) V → ∃ j, CoupledSat σ (u' j) (V' j))
  ↔ ∀ c, SelValid V' c → Blocked (u ∪ w) V u' c

```

SuFunctor.Universe.**lowerSemicontinuous\_selSup**      lean/su-functor/SuFunctor/Universe/DemonicClock.lean, line 279

```

theorem lowerSemicontinuous_selSup {u : K → ℝ≥0∞}
  {c : Fin m → Option (K → ℝ≥0∞)} (hu : LowerSemicontinuous u)
  (hc : ∀ j w, c j = some w → LowerSemicontinuous w) :
  LowerSemicontinuous (selSup u c)

```

SuFunctor.Universe.General.**probWall**      lean/su-functor/SuFunctor/Universe/General/DemonicGeneral.lean, line 49

```

noncomputable def probWall (B : Set (ProbabilityMeasure X)) : Penalty X

```

SuFunctor.Universe.**crescentClass\_inter\_wall**      lean/su-functor/SuFunctor/Universe/DemonicClock.lean, line 323

```
theorem crescentClass_inter_wall (u w : ProbPenalty V)
  (V' : Set (ProbPenalty V)) :
  crescentClass u V' n crescentClass w 0 = crescentClass (u ∪ w) V'
```

SuFunctor.Universe.**cross\_site\_union\_refinement\_iff\_probWall** lean/su-functor/SuFunctor/Universe/DemonicClock.lean, line 332

```
theorem cross_site_union_refinement_iff_probWall {B : Set (ProbState V)}
  (hB : IsClosed B) {u : ProbPenalty V} {V' : Set (ProbPenalty V)}
  {u' : Fin m → ProbPenalty V} {V' : Fin m → Set (ProbPenalty V)}
  (hu : LowerSemicontinuous u)
  (hV' : ∀ j, ∀ v ∈ V' j, LowerSemicontinuous v) :
  (∀ σ, LowerSemicontinuous σ → CoupledSat σ (u ∪ probWall B) V' →
    ∃ j, CoupledSat σ (u' j) (V' j))
  ↔ ∀ c, SelValid V' c → Blocked (u ∪ probWall B) V' u' c
```

SuFunctor.Universe.**exists\_no\_functional\_transport** lean/su-functor/SuFunctor/Universe/DemonicClock.lean, line 186

```
theorem exists_no_functional_transport :
  ∃ r : Bool → Bool → Prop, ¬ ∃ f : Bool → Bool, ∀ x y, r x y ↔ f x = y
```

SuFunctor.Universe.**emLift\_comp** lean/su-functor/SuFunctor/Universe/DemonicClock.lean, line 143

```
theorem emLift_comp (r : X → Y → Prop) (s : Y → Z → Prop)
  (A : Set X) (C : Set Z) :
  emLift (relComp r s) A C ↔ ∃ B, emLift r A B ∧ emLift s B C
```

SuFunctor.Spacetime.**spacetimeSpecFunctor\_fibre\_eq** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 141

```
theorem spacetimeSpecFunctor_fibre_eq (A : SpacetimeBase Clk Rec) :
  (spacetimeSpecFunctor Clk Rec).toFunctor.obj A
  = SpecDnf (Joint (A.wires → ClockedEl Clk Rec))
```

SuFunctor.Spacetime.**profileLiftHom** lean/su-functor/SuFunctor/Spacetime/Base.lean, line 364

```
def profileLiftHom (S : Type u) (φ : S → B.wires) :
  profileLift (Rec := Rec) S φ → B
```

SuFunctor.Spacetime.**profile\_unique\_of\_hom** lean/su-functor/SuFunctor/Spacetime/Base.lean, line 373

```
theorem profile_unique_of_hom {S : Type u} {κ : S → Clk}
  {φ : S → B.wires} (h : ∀ w, B.profile (φ w) = κ w) :
  κ = fun w => B.profile (φ w)
```

SuFunctor.Spacetime.**mixedInterface** lean/su-functor/SuFunctor/Spacetime/Base.lean, line 382

```
def mixedInterface (c c' : Clk) : SpacetimeBase Clk Rec
```

SuFunctor.Spacetime.**profiledEmbed** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 94

```
def profiledEmbed (Clk : Type u) (Rec : Clk → Type u) :
  LaxDoubleFunctor (SpacetimeBase Clk Rec)
  (WireSpan (ClockedEl Clk Rec)) where ...
```

SuFunctor.Spacetime.**SemanticUniverse** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 159

```
abbrev SemanticUniverse (Clk : Type u) (Rec : Clk → Type u)
```

SuFunctor.Spacetime.**universeProj\_discreteOpfibration** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 185

```
theorem universeProj_discreteOpfibration
  (Clk : Type u) (Rec : Clk → Type u) :
  IsDiscreteDoubleOpfibration (universeProj Clk Rec)
```

SuFunctor.Spacetime.**constHomEquiv** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 216

```
def constHomEquiv (S T : WireSpan (Rec c)) :
  (S → T) ≈ (constObj (Rec := Rec) c S.wires → constObj c T.wires) where ...
```

SuFunctor.Universe.**fibInf** lean/su-functor/SuFunctor/Universe/CertificateLayer.lean, line 79

```
def fibInf (g : K1 → K2) (τ : K1 → ℝ≥0∞) : K2 → ℝ≥0∞
```

SuFunctor.Universe.**coupledSat\_comap\_iff\_fibInf** lean/su-functor/SuFunctor/Universe/CertificateLayer.lean, line 128

```
theorem coupledSat_comap_iff_fibInf {g : K1 → K2} {τ : K1 → ℝ≥0∞}
  {u : K2 → ℝ≥0∞} {γ' : Set (K2 → ℝ≥0∞)} :
  CoupledSat τ (u ∘ g) ((· ∘ g) '' γ') ↔ CoupledSat (fibInf g τ) u γ'
```

SuFunctor.Universe.SpecDnf.**comap\_modelClass\_mono** lean/su-functor/SuFunctor/Universe/CertificateLayer.lean, line 181

```
theorem SpecDnf.comap_modelClass_mono {g : K1 → K2} {D E : SpecDnf K2}
  (h : D.modelClass ⊆ E.modelClass) :
  (D.comap g).modelClass ⊆ (E.comap g).modelClass
```

SuFunctor.Universe.**exists\_comp\_notMem\_modelClass\_comap** lean/su-functor/SuFunctor/Universe/CertificateLayer.lean, line 213

```
theorem exists_comp_notMem_modelClass_comap :
  ∃ (g : Unit → Bool) (a : SpecAtom Bool) (σ : Bool → ℝ≥0∞),
  σ ∈ a.modelClass ∧ σ ∘ g ∉ (a.comap g).modelClass
```

SuFunctor.Universe.**coupledSat\_comp\_add\_of\_dips** lean/su-functor/SuFunctor/Universe/CertificateLayer.lean, line 258

```
theorem coupledSat_comp_add_of_dips (g1 : K → K1) (g2 : K → K2)
  {σ1 u1 : K1 → ℝ≥0∞} {σ2 u2 : K2 → ℝ≥0∞}
  {γ1 : Set (K1 → ℝ≥0∞)} {γ2 : Set (K2 → ℝ≥0∞)}
  (h1 : CoupledSat σ1 u1 γ1) (h2 : CoupledSat σ2 u2 γ2)
  (hcon : ∀ w ∈ ((· ∘ g1) '' γ1 ∪ ((· ∘ g2) '' γ2),
    Dip ((σ1 ∘ g1) + (σ2 ∘ g2)) w) :
  CoupledSat ((σ1 ∘ g1) + (σ2 ∘ g2)) ((u1 ∘ g1) + (u2 ∘ g2))
  (((· ∘ g1) '' γ1 ∪ ((· ∘ g2) '' γ2))
```

SuFunctor.Universe.**exists\_boxplus\_not\_semantically\_monotone** lean/su-functor/SuFunctor/Universe/CertificateLayer.lean, line 285

```
theorem exists_boxplus_not_semantically_monotone :
  ∃ (a e e1 e2 : SpecAtom Unit) (τ : Unit → ℝ≥0∞),
  e.modelClass = e1.modelClass ∪ e2.modelClass ∧
  CoupledSat τ (a.floor + e.floor) (a.marks ∪ e.marks) ∧
  ¬ CoupledSat τ (a.floor + e1.floor) (a.marks ∪ e1.marks) ∧
  ¬ CoupledSat τ (a.floor + e2.floor) (a.marks ∪ e2.marks)
```

SuFunctor.Universe.SuFunctor.**dnfBoxplus\_mono** lean/su-functor/SuFunctor/Universe/CertificateLayer.lean, line 405

```
theorem dnfBoxplus_mono {A B C : WireSpan El} {M : Pat A B} {N : Pat B C}
  {D1 E1 : DnfCpl M} {D2 E2 : DnfCpl N}
  (h1 : D1.atoms ⊆ E1.atoms) (h2 : D2.atoms ⊆ E2.atoms) :
  (dnfBoxplus D1 D2).atoms ⊆ (dnfBoxplus E1 E2).atoms
```

SuFunctor.Spacetime.**spacetimeSpecFunctor** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 134

```
noncomputable def spacetimeSpecFunctor (Clk : Type u) (Rec : Clk → Type u) :
  SpanValued.{u + 1, u, u, u + 1, u} (SpacetimeBase Clk Rec)
```

SuFunctor.Spacetime.**semanticUniverse\_pseudoDoubleCategory** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 164

```
theorem semanticUniverse_pseudoDoubleCategory
  (Clk : Type u) (Rec : Clk → Type u) :
  ∃ _ : PseudoDoubleCategory (SemanticUniverse Clk Rec), True
```

SuFunctor.Spacetime.**universeProj\_isPseudo** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 177

```
theorem universeProj_isPseudo (Clk : Type u) (Rec : Clk → Type u) :
  (universeProj Clk Rec).IsPseudo
```

SuFunctor.Spacetime.**profiledEmbed\_faithful\_tight** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 122

```
theorem profiledEmbed_faithful_tight {A B : SpacetimeBase Clk Rec}
  {φ ψ : A → B}
  (h : (profiledEmbed Clk Rec).map φ = (profiledEmbed Clk Rec).map ψ) :
  φ = ψ
```

SuFunctor.Spacetime.**constProfile** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 298

```
def constProfile :
  LaxDoubleFunctor (WireSpan (Rec c)) (SpacetimeBase Clk Rec) where ...
```

SuFunctor.Spacetime.**constProfile\_isPseudo** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 340

```
theorem constProfile_isPseudo :
  (constProfile (Rec := Rec) c).IsPseudo where
  laxComp_isInvertible { _ T _ } M N
```

SuFunctor.Spacetime.**siteRead** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 273

```
noncomputable def siteRead (c : Clk) {S : Type u}
  (D : SpecDnf (Joint (S → ClockedEl Clk Rec))) :
  SpecDnf (Joint (S → Rec c))
```

SuFunctor.Spacetime.SuFunctor.Universe.SpecDnf.**mem\_modelClass\_withWall** lean/su-functor/SuFunctor/Spacetime/Universe.lean, line 408

```
theorem SpecDnf.mem_modelClass_withWall {D : SpecDnf K}
  {w σ : K → ℝ≥0∞} :
  σ ∈ (D.withWall w).modelClass ↔ w ≤ σ ∧ σ ∈ D.modelClass
```

SuFunctor.Adequacy.**rwall\_singletonC** lean/su-functor/SuFunctor/Universe/RadialUniverse.lean, line 303

```
theorem rwall_singletonC (μ : SubPMF V) :
  rwall (singletonC μ) = rayPoint μ
```

SuFunctor.Adequacy.**idealPoint** lean/su-functor/SuFunctor/Adequacy/SemanticUniverseWiring.lean, line 213

```
noncomputable def idealPoint (μ : SubPMF V) : Plain.Pen V
```

SuFunctor.SubPMF.**toJoint** lean/su-functor/SuFunctor/Adequacy/SemanticUniverseWiring.lean, line 99

```
noncomputable def toJoint (μ : SubPMF V) : Plain.Joint V where ...
```

SuFunctor.Adequacy.**diagramDnf** lean/su-functor/SuFunctor/Adequacy/SemanticUniverseWiring.lean, line 385

```
noncomputable def diagramDnf (L : List (SubPMF V → Prop)) :
  SpecDnf (Plain.Joint V) where ...
```

SuFunctor.Adequacy.**coupledSat\_idealPoint\_cwall\_iff** lean/su-functor/SuFunctor/Adequacy/SemanticUniverseWiring.lean, line 295

```

theorem coupledSat_idealPoint_cwall_iff {c : SubPMF V → Prop}
  {μ : SubPMF V} :
  CoupledSat (idealPoint μ) (cwall c) {(τ : Plain.Pen V)} ↔ c μ

SuFunctor.Adequacy.idealPoint_mem_diagramDnf_iff lean/su-functor/SuFunctor/Adequacy/SemanticUniverseWiring.
lean, line 394

theorem idealPoint_mem_diagramDnf_iff (L : List (SubPMF V → Prop))
  (μ : SubPMF V) :
  idealPoint μ ∈ (diagramDnf L).modelClass ↔ looseConj L μ

SuFunctor.Plain.semanticallyValid_iff_subset lean/su-functor/SuFunctor/Plain/CheckerFace.lean, line 215

theorem semanticallyValid_iff_subset (H : Hypernet) (seq : Sequent) :
  seq.semanticallyValid H ↔ admCyl H seq ⊆ postEvt H seq

SuFunctor.Plain.upperProb_eq_zero_iff lean/su-functor/SuFunctor/Plain/CheckerFace.lean, line 177

theorem upperProb_eq_zero_iff (B E : Set X) :
  upperProb B E = 0 ↔ E ∩ B = ∅

SuFunctor.Plain.semanticallyValid_iff_face lean/su-functor/SuFunctor/Plain/CheckerFace.lean, line 224

theorem semanticallyValid_iff_face (H : Hypernet) (seq : Sequent) :
  seq.semanticallyValid H ↔
  upperProb (admCyl H seq) (postEvt H seq)c = 0

SuFunctor.Plain.suppIn_empty lean/su-functor/SuFunctor/Plain/CheckerFace.lean, line 78

theorem suppIn_empty : suppIn (∅ : Set X) = {Joint.zero X}

SuFunctor.Plain.face_ex_falso lean/su-functor/SuFunctor/Plain/CheckerFace.lean, line 243

theorem face_ex_falso (H : Hypernet) (seq : Sequent)
  (h : admCyl H seq = ∅) :
  upperProb (admCyl H seq) (postEvt H seq)c = 0

SuFunctor.hnetInclusion lean/su-functor/SuFunctor/HypernetDouble.lean, line 559

def hnetInclusion : LaxDoubleFunctor (HNet ty) (SingleSiteSemanticUniverse ElVal)

SuFunctor.Universe.General.sat_wall_iff lean/su-functor/SuFunctor/Universe/General/CheckerAnchor.lean, line 82

theorem sat_wall_iff {σ : StatePenalty X}
  {C : Set (SubProbabilityMeasure X)} :
  Sat σ (wall C) ↔ ∀ v, σ v ≠ 1 → v ∈ C

SuFunctor.Universe.General.sat_wall_wall_iff lean/su-functor/SuFunctor/Universe/General/CheckerAnchor.lean, line
92

theorem sat_wall_wall_iff {S C : Set (SubProbabilityMeasure X)} :
  Sat (wall S) (wall C) ↔ S ⊆ C

SuFunctor.Universe.General.sat_iff_sublevel_subset lean/su-functor/SuFunctor/Universe/General/CheckerAnchor.
lean, line 111

theorem sat_iff_sublevel_subset {σ φ : StatePenalty X} :
  Sat σ φ ↔ ∀ s : ℝ≥0∞,
  {v : SubProbabilityMeasure X | σ v ≤ s}
  ⊆ {v : SubProbabilityMeasure X | φ v ≤ s}

PPLC.OpenHypernet lean/qtsl/Qtsl/HypernetCospan.lean, line 222

structure OpenHypernet where ...

```



PPLC.OpenHypernet.**glue\_admissible\_iff** lean/qtsl/Qtsl/HypernetCospan.lean, line 444

```
theorem glue_admissible_iff (σ : (glueNet hA).Valuation) :
  σ.Admissible ↔
    (∀ a ∈ H.net.applications, a.SemOk σ) ∧
    (∀ a ∈ G.net.applications, a.SemOk σ)
```

PPLC.OpenHypernet.**unitOpen** lean/qtsl/Qtsl/HypernetCospan.lean, line 465

```
def unitOpen (X : List Wire) (wt₀ : Wire → Option TensorType)
  (hX : X.Nodup) : OpenHypernet where ...
```

PPLC.OpenHypernet.**glue\_assoc** lean/qtsl/Qtsl/HypernetCospan.lean, line 558

```
theorem glue_assoc {H G K : OpenHypernet}
  (hHG : Aligned H G) (hGK : Aligned G K)
  (hL : Aligned (glue hHG) K) (hR : Aligned H (glue hGK)) :
  glue hL = glue hR
```

SuFunctor.**theta\_rename** lean/su-functor/SuFunctor/HypernetInclusion.lean, line 912

```
theorem theta_rename {m : AlignedLoose X Y} {m' : AlignedLoose X' Y'}
  (c : AlignedCell ty m m') (T : List PrimApp)
  (hTC : ∀ a ∈ T, a.TypeConsistent (tableOn ty m.apex))
  (hw : ∀ a ∈ T, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ m.apex) :
  dnfPull (renameTot ty c) (austereDnf ty m T)
    = austereDnf ty m' (T.map (PrimApp.rename c.map))
```

PPLC.ProofObject.**sound** lean/qtsl/Qtsl/Soundness.lean, line 8330

```
theorem ProofObject.sound {H : Hypernet} {seq : Sequent}
  (pf : ProofObject H seq) : seq.semanticallyValid H
```

PPLC.**check\_sound** lean/qtsl/Qtsl/Soundness.lean, line 8439

```
theorem check_sound {H : Hypernet} {seq : Sequent} (pf : ProofObject H seq)
  (_hcheck : check pf = true) : seq.semanticallyValid H
```

Modules cited: QtslAux.DiamondSite, QtslAux.DiamondTopology, SuFunctor.Site.TemporalTypeTheory, SuFunctor.Site.BehaviorSpec, SuFunctor.Site.CausalConcordance, SuFunctor.StateSpace, SuFunctor.StateSpace.FamCat, QtslAux.QuantumCovariance, QtslAux.OpenQuantum, QtslAux.QuantumRecords, QtslAux.QuantumDescent, QtslAux.QuantumStates, QtslAux.QuantumAssembly, QtslAux.QuantumDeterminacy, QtslAux.QuantumOuterStorey, SuFunctor.Universe.FellTopology, SuFunctor.Universe.FellRestrict, SuFunctor.Universe.FellSeamWitness, SuFunctor.Universe.FellDescent, SuFunctor.Universe.FellRecordLocus, SuFunctor.Universe.FellRecordWitness, SuFunctor.Universe.FellOnePoint, SuFunctor.Universe.FellCapacity, Pointfree.Frame.Presentation, Pointfree.Frame.HyperspacePresentation, Pointfree.Valuation.Basic, Pointfree.Valuation.Topology, Pointfree.Valuation.EscapeWitness, Pointfree.Valuation.Bridge, Pointfree.Measure.DaggerChart, SuFunctor.Universe.General, SuFunctor.Universe.Judgment, SuFunctor.Universe.Closure, SuFunctor.Universe.Generati, SuFunctor.Universe.Tents, QtslAux.Imprecise.MarginalCompat, QtslAux.Imprecise.MarginalCompatGeneral, SuFunctor.Universe.RadialUniverse, SuFunctor.StateSpace.FamCompletion, SuFunctor.Universe.VietorisStratum, SuFunctor.Universe.ImageChar, QtslAux.Imprecise.PenaltyCredalIso, SuFunctor.Universe.EpistemicStates, SuFunctor.Universe.General.CredalPoset, SuFunctor.Universe.General.Richardson, SuFunctor.Universe.CredalPoset, SuFunctor.Universe.HausdorffKL, SuFunctor.Adequacy.Envelope, SuFunctor.Adequacy.EnvelopeUnion, QtslAux.Imprecise.Credal, QtslAux.Imprecise.NaturalExtensionGeneral, QtslAux.Imprecise.NaturalExtension, QtslAux.LeakAware, SuFunctor.Universe.General.CrescentStratum, SuFunctor.Universe.HullClosure, SuFunctor.Site.GeneralWindow, SuFunctor.Plain.SeamDensity, Pointfree.MeasureAlgebra.Basic, Pointfree.Prevision.B, SuFunctor.QtslSem.SpecJudg, DoubleCat.Basic, DoubleCat.Functor.Lax, DoubleCat.DepSpan, SuFunctor.Universe.Marginal, DoubleCat.WireSpan, SuFunctor.Plain.Penalty, SuFunctor.Plain.SpecFunctor, SuFunctor.Plain.Battery, SuFunctor.Site.WindowPenalty, SuFunctor.Site.SiteFunctor, SuFunctor.Site.Battery, SuFunctor.Site.MatchingFork, SuFunctor.Plain.GeneralFunctor, DoubleCat.Grothendieck, DoubleCat.Opfibration, DoubleCat.JudgLayer, QtslAux.Imprecise.Containment, QtslAux.Imprecise.ContainmentGeneral, QtslAux.Imprecise.CouplingGlue, QtslAux.Imprecise.FeasibilityGlue, SuFunctor.Plain.Seam, SuFunctor.Plain.GeneralSeam, SuFunctor.Universe.Transport

SuFunctor.Universe.General.TransportGeneral, SuFunctor.Universe.DemonicClock, SuFunctor.Universe.General.DemonicClock, DoubleCat.CrossSite, SuFunctor.Spacetime.Base, SuFunctor.Universe.CertificateLayer, SuFunctor.Spacetime.Universe, SuFunctor.Target, SuFunctor.Adequacy.Inclusions, SuFunctor.Adequacy.Dtmc, SuFunctor.Adequacy.Pgm, SuFunctor.Adequacy.Mdp, SuFunctor.Adequacy.MdpDemonic, SuFunctor.Adequacy.DelayLine, SuFunctor.Adequacy.HeatTorus, SuFunctor.Adequacy.MarginalGlue, SuFunctor.Adequacy.HeatTorusInit, SuFunctor.Adequacy.SolutionSetMesh, SuFunctor.Plain.CheckerFace, SuFunctor.HypernetDouble, SuFunctor.Universe.General.CheckerAnchor, Qtsl.HypernetCospan, SuFunctor.HypernetInclusion, Qtsl.Soundness.

## A.2 Hypernets: Primitive Functions

Citations from Chapter 10.

PPLC.OpenHypernet.glue\_unit\_left

lean/qtsl/Qtsl/HypernetCospan.lean, line 517

```
theorem glue_unit_left (H : OpenHypernet)
  (wt₀ : Wire → Option TensorType)
  (hsub : ∀ w τ, wt₀ w = some τ → H.net.wireType w = some τ)
  (hA : Aligned (unitOpen H.inputs wt₀ H.inputs_nodup) H) :
  glue hA = H
```

PPLC.OpenHypernet.glue\_unit\_right

lean/qtsl/Qtsl/HypernetCospan.lean, line 532

```
theorem glue_unit_right (H : OpenHypernet)
  (wt₀ : Wire → Option TensorType)
  (hsub : ∀ w τ, wt₀ w = some τ → H.net.wireType w = some τ)
  (hA : Aligned H (unitOpen H.outputs wt₀ H.outputs_nodup)) :
  glue hA = H
```

SuFunctor.theta\_unit

lean/su-functor/SuFunctor/HypernetInclusion.lean, line 842

```
theorem theta_unit (X : Finset Wire) :
  dnfPull (idTot ty X) (dnfUnit (ctxIface X))
  = austereDnf ty (AlignedLoose.unit X) []
```

SuFunctor.theta\_comp

lean/su-functor/SuFunctor/HypernetInclusion.lean, line 864

```
theorem theta_comp (m : AlignedLoose X Y) (n : AlignedLoose Y Z)
  (T₁ T₂ : List PrimApp)
  (hTC₁ : ∀ a ∈ T₁, a.TypeConsistent (tableOn ty m.apex))
  (hw₁ : ∀ a ∈ T₁, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ m.apex)
  (hTC₂ : ∀ a ∈ T₂, a.TypeConsistent (tableOn ty n.apex))
  (hw₂ : ∀ a ∈ T₂, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ n.apex) :
  dnfPull (splitTot ty m n)
    (dnfBoxplus (austereDnf ty m T₁) (austereDnf ty n T₂))
  = austereDnf ty (m.seq n) (T₁ ++ T₂)
```

SuFunctor.vacDnf\_dnfPull

lean/su-functor/SuFunctor/HypernetInclusion.lean, line 811

```
theorem vacDnf_dnfPull {A B : Type} (h : A → B) :
  dnfPull h (vacDnf (Plain.Joint B)) = vacDnf (Plain.Joint A)
```

SuFunctor.hnetInclusion\_pushGLoose

lean/su-functor/SuFunctor/HypernetDouble.lean, line 574

```
theorem hnetInclusion_pushGLoose {p q : HNet ty}
  (M : SpanValued.GLoose p q) {c d : AlignedCtx} {n : Loose c d}
  {f' : p.1 → c} {g' : q.1 → d} (γ : Cell M.base n f' g') :
  SpanValued.bmapLoose (stateBase ty) (hnetTheta ty)
    (SpanValued.pushGLoose M γ) ≈
  SpanValued.pushGLoose
    (SpanValued.bmapLoose (stateBase ty) (hnetTheta ty) M)
    ((stateBase ty).mapCell γ)
```

SuFunctor.**austereDnf\_eq\_iff**

lean/su-functor/SuFunctor/HypernetInclusion.lean, line 934

```
theorem austereDnf_eq_iff (m : AlignedLoose X Y) (T T' : List PrimApp) :
  austereDnf ty m T = austereDnf ty m T'
  ↔ AdmRegion ty m T = AdmRegion ty m T'
```

PPLC.**substituteRaw\_eq\_glue**

lean/qtsl/Qtsl/SubstitutionBridge.lean, line 1960

```
theorem substituteRaw_eq_glue :
  substituteRaw outer  $\varphi$  E inner =
    { wireTypes := bridgeTable outer  $\varphi$  E inner
      apps := (substituteOpen outer  $\varphi$  E inner
        hwf0 hwfI hpo hti hto).net.applications
      ins := (substituteOpen outer  $\varphi$  E inner
        hwf0 hwfI hpo hti hto).inputs
      outs := (substituteOpen outer  $\varphi$  E inner
        hwf0 hwfI hpo hti hto).outputs }
```

PPLC.**substitute\_eq\_glue\_renumber**

lean/qtsl/Qtsl/SubstitutionBridge.lean, line 1985

```
theorem substitute_eq_glue_renumber :
  substitute outer  $\varphi$  E inner =
    (HypernetP.mk (bridgeTable outer  $\varphi$  E inner)
      ((substituteOpen outer  $\varphi$  E inner
        hwf0 hwfI hpo hti hto).net.applications)
      ((substituteOpen outer  $\varphi$  E inner hwf0 hwfI hpo hti hto).inputs)
      ((substituteOpen outer  $\varphi$  E inner
        hwf0 hwfI hpo hti hto).outputs)).renumber
```

PPLC.**substituteOpen\_admissible\_iff**

lean/qtsl/Qtsl/SubstitutionBridge.lean, line 2001

```
theorem substituteOpen_admissible_iff
  ( $\sigma$  : (substituteOpen outer  $\varphi$  E inner
    hwf0 hwfI hpo hti hto).net.Valuation) :
   $\sigma$ .Admissible ↔
    (( $\forall$  a  $\in$  inner.apps, a.SemOk  $\sigma$ )  $\wedge$ 
      ( $\forall$  a  $\in$  (substSeam outer  $\varphi$  E inner).repairs, a.SemOk  $\sigma$ ))  $\wedge$ 
    ( $\forall$  a  $\in$  outer.apps.map
      (PrimApp.remapWires (substOuterWire outer  $\varphi$  E inner)),
      a.SemOk  $\sigma$ )
```

Modules cited: Qtsl.SubstitutionBridge, Qtsl.PackingGap.

Also cited in this chapter, listed under an earlier one: Qtsl.HypernetCospan, SuFunctor.HypernetDouble, SuFunctor.HypernetInclusion, glue\_admissible\_iff, glue\_assoc, hnetInclusion, theta\_rename.

## A.3 Hypernets: Markov Kernels

Citations from Chapter 11.

SuFunctor.Stochastic.**sum\_noiseDenotFun**

lean/su-functor/SuFunctor/Stochastic/NoiseSemantics.lean, line 117

```
theorem sum_noiseDenotFun (g : (Fin N  $\rightarrow$   $\mathbb{R}$ )  $\rightarrow$  V) (hg : Measurable g) :
   $\sum v$ , noiseDenotFun g v = 1
```

SuFunctor.Stochastic.**noiseDenot\_comp**

lean/su-functor/SuFunctor/Stochastic/NoiseSemantics.lean, line 162

```
theorem noiseDenot_comp (g : (Fin N  $\rightarrow$   $\mathbb{R}$ )  $\rightarrow$  V) ( $\pi$  : V  $\rightarrow$  W)
  (hg : Measurable g) (hng : Measurable ( $\pi \circ g$ )) :
  noiseDenot ( $\pi \circ g$ ) hng = ProbState.push  $\pi$  (noiseDenot g hg)
```

SuFunctor.Stochastic.**noiseDenot\_seqComp**

lean/su-functor/SuFunctor/Stochastic/NoiseKleisli.lean, line 232

```

theorem noiseDenot_seqComp (g : (Fin N1 → ℝ) → V) (k : V → (Fin N2 → ℝ) → W)
  (hg : Measurable g) (hk : ∀ v, Measurable (k v)) :
  noiseDenot (seqComp g k) (measurable_seqComp hg hk)
    = (noiseDenot g hg).bind fun v => noiseDenot (k v) (hk v)

```

SuFunctor.Stochastic.**copy\_ne\_kleisli** lean/su-functor/SuFunctor/Stochastic/NoiseKleisli.lean, line 332

```

theorem copy_ne_kleisli :
  noiseDenot copyPair measurable_copyPair
    ≠ (noiseDenot bitStage measurable_bitStage).bind
      (fun v => noiseDenot (pairStage v) (measurable_pairStage v))

```

SuFunctor.Stochastic.**ThinningForm** lean/su-functor/SuFunctor/Stochastic/JumpMeasure.lean, line 297

```

structure ThinningForm (V : Type u) [Fintype V] : Type u where ...

```

SuFunctor.Stochastic.JumpTerm.**elaborate\_mass\_partition** lean/su-functor/SuFunctor/Stochastic/JumpMeasure.lean, line 541

```

theorem elaborate_mass_partition (J : JumpTerm V) (x : V) :
  ((elaborate J).rate : ℝ) *
    ((∑ y, ((elaborate J).kernel x : Step V → ℝ) (Step.next y))
      + ((elaborate J).kernel x : Step V → ℝ) Step.stay
      + ((elaborate J).kernel x : Step V → ℝ) Step.cemetery)
    = (J.intensity : ℝ)

```

SuFunctor.Stochastic.JumpTerm.**outflow\_le\_intensity** lean/su-functor/SuFunctor/Stochastic/JumpMeasure.lean, line 557

```

theorem outflow_le_intensity (J : JumpTerm V) (x : V) :
  ((elaborate J).rate : ℝ) *
    ((∑ y, ((elaborate J).kernel x : Step V → ℝ) (Step.next y))
      + ((elaborate J).kernel x : Step V → ℝ) Step.cemetery)
    ≤ (J.intensity : ℝ)

```

SuFunctor.Stochastic.JumpTerm.**generator\_elaborate** lean/su-functor/SuFunctor/Stochastic/JumpMeasure.lean, line 381

```

theorem generator_elaborate (J : JumpTerm V) :
  (elaborate J).generator = J.generator

```

SuFunctor.Stochastic.ThinningForm.**generator\_dilate** lean/su-functor/SuFunctor/Stochastic/JumpMeasure.lean, line 592

```

theorem generator_dilate (c : ℚ≥0) (hc : 1 ≤ c) (T : ThinningForm V) :
  (T.dilate c hc).generator = T.generator

```

SuFunctor.Stochastic.ThinningForm.**generator\_absorbDiagonal** lean/su-functor/SuFunctor/Stochastic/JumpMeasure.lean, line 681

```

theorem generator_absorbDiagonal (T : ThinningForm V) :
  T.absorbDiagonal.generator = T.generator

```

SuFunctor.Stochastic.ThinningForm.JumpTerm.**elaborate\_superpose** lean/su-functor/SuFunctor/Stochastic/JumpAlgebra.lean, line 534

```

theorem elaborate_superpose (J1 J2 : JumpTerm V) :
  (superpose J1 J2).elaborate
    = (J1.elaborate).superpose (J2.elaborate)

```

SuFunctor.Stochastic.ThinningForm.JumpTerm.**generator\_elaborate\_interleave** lean/su-functor/SuFunctor/Stochastic/JumpAlgebra.lean, line 561

```

theorem generator_elaborate_interleave (J1 : JumpTerm V1)
  (J2 : JumpTerm V2) :
  (interleave J1 J2).elaborate.generator
    = ((J1.elaborate).interleave (J2.elaborate)).generator

```

SuFunctor.Stochastic.**generator\_eq\_stepMix** lean/su-functor/SuFunctor/Stochastic/JumpDiffusionThinning.lean, line 187

```

theorem generator_eq_stepMix (P : JdProfile n m) (V : State n → ℝ)
  (gradV : State n → State n) (hessV : State n → Fin n → Fin n → ℝ)
  (z : State n) :
  Qtsl.JumpDiffusion.generator P V gradV hessV z
    = (∑ i, gradV z i * P.f z i)
      + (1 / 2) * (∑ i, ∑ j, P.gram z i j * hessV z i j)
      + P.kill z * stepMixGenerator (killMixture n) V z
      + (P.legs.map fun lm =>
          lm.1 z * stepMixGenerator (deficitToCemetery lm.2) V z).sum

```

Modules cited: SuFunctor.Stochastic, SuFunctor.Stochastic.JumpAlgebra.

## A.4 Systems: Mathematical Models

Citations from Chapter 12.

SuFunctor.Adequacy.**is0deSol\_iff\_integralCurve** lean/su-functor/SuFunctor/Adequacy/Ode.lean, line 209

```

theorem is0deSol_iff_integralCurve {f : E → E} {x0 : E} {y : ℝ → E}
  (hf : Continuous f) :
  Is0deSol f x0 y ↔
    y 0 = x0 ∧ IsIntegralCurveOn y (fun _ => f) (Ici 0)

```

SuFunctor.Adequacy.**is0deSol\_iff\_integral** lean/su-functor/SuFunctor/Adequacy/Ode.lean, line 180

```

theorem is0deSol_iff_integral {f : E → E} {x0 : E} {y : ℝ → E}
  (hf : Continuous f) :
  Is0deSol f x0 y ↔ ContinuousOn y (Ici 0) ∧
    ∀ t ∈ Ici (0 : ℝ), y t = x0 + ∫ s in (0 : ℝ)..t, f (y s)

```

SuFunctor.Adequacy.**ode\_adequacy** lean/su-functor/SuFunctor/Adequacy/Ode.lean, line 319

```

theorem ode_adequacy [CompleteSpace E] {f : E → E} (hf : Continuous f)
  (x0 : E) {a : Crd (ℝ → E)} :
  ∃ α : Cell (looseConj (odeAtoms f x0) : Loose a a)
    (suppC (integralCurves f x0)) (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.**ode\_box\_adequacy** lean/su-functor/SuFunctor/Adequacy/Ode.lean, line 414

```

theorem ode_box_adequacy [CompleteSpace E] {f : E → E} (hf : Continuous f)
  (B : Set E) {a : Crd (ℝ → E)} :
  ∃ α : Cell (looseConj (odeBoxAtoms f B) : Loose a a)
    (suppC (⋃ x0 ∈ B, integralCurves f x0)) (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.**odeSols\_eq0n\_of\_lipschitz0nWith** lean/su-functor/SuFunctor/Adequacy/Ode.lean, line 234

```

theorem odeSols_eq0n_of_lipschitz0nWith {f : E → E} {K : ℝ≥0} {s : Set E}
  (hL : Lipschitz0nWith K f s) {x0 : E} {y1 y2 : ℝ → E}
  (h1 : Is0deSol f x0 y1) (h2 : Is0deSol f x0 y2)
  (hs1 : ∀ t ∈ Ici (0 : ℝ), y1 t ∈ s)
  (hs2 : ∀ t ∈ Ici (0 : ℝ), y2 t ∈ s) :
  Eq0n y1 y2 (Ici 0)

```

SuFunctor.Adequacy.**su\_ode\_adequacy**

lean/su-functor/SuFunctor/Adequacy/Ode.lean, line 453

```

theorem su_ode_adequacy [CompleteSpace E] {f : E → E} (hf : Continuous f)
  (x₀ : E) (μ : SubPMF (ℝ → E)) :
  idealPoint μ ∈ (odeDnf f x₀).modelClass ↔
  suppC (integralCurves f x₀) μ

```

SuFunctor.Adequacy.**heat\_mesh\_glues**

lean/su-functor/SuFunctor/Adequacy/HeatTorus.lean, line 695

```

theorem heat_mesh_glues (hT : IsOpen T)
  (h1 : IsHeatSol D M.I1 T c1) (h2 : IsHeatSol D M.I2 T c2)
  (h3 : IsHeatSol D M.I3 T c3)
  (hs12 : ∀ x ∈ M.I1 ∩ M.I2, ∀ t ∈ T, c1 x t = c2 x t)
  (hs23 : ∀ x ∈ M.I2 ∩ M.I3, ∀ t ∈ T, c2 x t = c3 x t)
  (hw : WrapWall M T c1 c3) :
  ∃ u : ℝ → ℝ → E, IsHeatTorusSol D M.L T u ∧
    (∀ x ∈ M.I1, ∀ t ∈ T, u x t = c1 x t) ∧
    (∀ x ∈ M.I2, ∀ t ∈ T, u x t = c2 x t) ∧
    (∀ x ∈ M.I3, ∀ t ∈ T, u x t = c3 x t)

```

SuFunctor.Adequacy.**heat\_adequacy**

lean/su-functor/SuFunctor/Adequacy/HeatTorus.lean, line 1122

```

theorem heat_adequacy (M : RingMesh) (D : ℝ) {T : Set ℝ}
  (hT : IsOpen T) {a : Crd (HeatRec E)} :
  ∃ α : Cell (looseConj (heatAtoms M D T) : Loose a a)
    (suppC (heatClassicalRecs M D T)) (1 a) (1 a),
  Cell.IsInvertible α

```

SuFunctor.Adequacy.**chainAtoms\_dirac\_lin**

lean/su-functor/SuFunctor/Adequacy/HeatTorus.lean, line 919

```

theorem chainAtoms_dirac_lin (e : E) :
  looseConj (heatChainAtoms M D T) (SubPMF.dirac (linRec e))

```

SuFunctor.Adequacy.**not\_wrap\_lin**

lean/su-functor/SuFunctor/Adequacy/HeatTorus.lean, line 932

```

theorem not_wrap_lin {e : E} (he : e ≠ 0) {t₀ : ℝ} (ht₀ : t₀ ∈ T) :
  ¬ suppC (wrapC M T) (SubPMF.dirac (linRec e))

```

SuFunctor.Adequacy.**heat\_chain\_glues**

lean/su-functor/SuFunctor/Adequacy/HeatTorus.lean, line 484

```

theorem heat_chain_glues {M : RingMesh} {D : ℝ} {T : Set ℝ}
  (hT : IsOpen T) {c1 c2 c3 : ℝ → ℝ → E}
  (h1 : IsHeatSol D M.I1 T c1) (h2 : IsHeatSol D M.I2 T c2)
  (h3 : IsHeatSol D M.I3 T c3)
  (hs12 : ∀ x ∈ M.I1 ∩ M.I2, ∀ t ∈ T, c1 x t = c2 x t)
  (hs23 : ∀ x ∈ M.I2 ∩ M.I3, ∀ t ∈ T, c2 x t = c3 x t) :
  ∃ w : ℝ → ℝ → E, IsHeatSol D M.chart T w ∧
    (∀ x ∈ M.I1, ∀ t ∈ T, w x t = c1 x t) ∧
    (∀ x ∈ M.I2, ∀ t ∈ T, w x t = c2 x t) ∧
    (∀ x ∈ M.I3, ∀ t ∈ T, w x t = c3 x t)

```

SuFunctor.**vorobev\_no\_joint**

lean/su-functor/SuFunctor/Adequacy/MarginalGlue.lean, line 246

```

theorem vorobev_no_joint (μ : SubPMF (Bool × Bool × Bool)) :
  ¬ (μ.push proj12 = pairCorr ∧ μ.push proj23 = pairCorr ∧
    μ.push proj13 = pairIndep)

```

SuFunctor.**credalCompile\_vorobevGlue**

lean/su-functor/SuFunctor/Adequacy/MarginalGlue.lean, line 415

```

theorem credalCompile_vorobevGlue :
  credalCompile vorobevGlue = rayC (SubPMF.zero (Bool × Bool × Bool))

```

SuFunctor.Adequacy.**heat\_ring\_glues\_vorobev\_ring\_not**lean/su-functor/SuFunctor/Adequacy/HeatTorus.lean, line 1038

```

theorem heat_ring_glues_vorobev_ring_not :
  (∃ μ : SubPMF (HeatRec E), looseConj (heatAtoms M D T) μ ∧
    μ ≠ SubPMF.zero (HeatRec E)) ∧
    ¬ ∃ μ : SubPMF (Bool × Bool × Bool), vorobevGlue μ

```

SuFunctor.Adequacy.**heat\_init\_adequacy** lean/su-functor/SuFunctor/Adequacy/HeatTorusInit.lean, line 291

```

theorem heat_init_adequacy (M : RingMesh) (D : ℝ) {T : Set ℝ}
  (hT : IsOpen T) (g : ℝ → E) {a : Crd (HeatRec E)} :
  ∃ α : Cell (looseConj (heatInitAtoms M D T g) : Loose a a)
    (suppC (heatInitClassicalRecs M D T g)) (1 a) (1 a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.**heatInitAtoms\_consistent\_of\_feed** lean/su-functor/SuFunctor/Adequacy/HeatTorusInit.lean, line 323

```

theorem heatInitAtoms_consistent_of_feed {M : RingMesh} {D : ℝ}
  {T : Set ℝ} (hT : IsOpen T) {g : ℝ → E}
  (hfeed : PortFeed M D T g) :
  ∃ μ : SubPMF (HeatRec E), looseConj (heatInitAtoms M D T g) μ ∧
    μ ≠ SubPMF.zero (HeatRec E)

```

SuFunctor.Adequacy.**initFace\_wrap\_compat** lean/su-functor/SuFunctor/Adequacy/HeatTorusInit.lean, line 178

```

theorem initFace_wrap_compat {M : RingMesh} {D : ℝ} {T : Set ℝ}
  {g : ℝ → E} {p : HeatRec E} (h : IsInitMeshSol M D T g p)
  (hne : (ℳ[T] (0 : ℝ)).NeBot) {x : ℝ} (hx3 : x ∈ M.I3)
  (hx1 : x - M.L ∈ M.I1) : g x = g (x - M.L)

```

SuFunctor.Adequacy.IsHeatTorusSolInit.**trace\_wrap\_compat** lean/su-functor/SuFunctor/Adequacy/HeatTorusInit.lean, line 208

```

theorem IsHeatTorusSolInit.trace_wrap_compat {M : RingMesh} {D : ℝ}
  {T : Set ℝ} {u : ℝ → ℝ → E} {g : ℝ → E}
  (h : IsHeatTorusSolInit M D T u g) (hne : (ℳ[T] (0 : ℝ)).NeBot)
  {x : ℝ} (hx : x ∈ M.chart) (hxL : x + M.L ∈ M.chart) :
  g (x + M.L) = g x

```

SuFunctor.Adequacy.**portFeed\_const** lean/su-functor/SuFunctor/Adequacy/HeatTorusInit.lean, line 335

```

theorem portFeed_const (M : RingMesh) (D : ℝ) (T : Set ℝ) (k : E) :
  PortFeed M D T (fun _ => k)

```

SuFunctor.Adequacy.**portFeed\_mode** lean/su-functor/SuFunctor/Adequacy/HeatTorusInit.lean, line 436

```

theorem portFeed_mode (M : RingMesh) (D : ℝ) (T : Set ℝ) (e : E)
  {ω : ℝ} (k : ℤ) (hω : ω * M.L = k * (2 * Real.pi)) :
  PortFeed M D T (modeDatum ω e)

```

SuFunctor.Adequacy.**modeDatum\_ne\_const** lean/su-functor/SuFunctor/Adequacy/HeatTorusInit.lean, line 447

```

theorem modeDatum_ne_const {e : E} (he : e ≠ 0) {ω : ℝ} (hω : ω ≠ 0) :
  modeDatum ω e (ω-1 * (Real.pi / 2)) ≠ modeDatum ω e 0

```

SuFunctor.Adequacy.**initFace\_neBot** lean/su-functor/SuFunctor/Adequacy/HeatTorusInit.lean, line 99

```

theorem initFace_neBot {τ : ℝ} (hτ : 0 < τ) :
  (ℳ[Ioo (0 : ℝ) τ] (0 : ℝ)).NeBot

```

SuFunctor.Universe.FellClosed.s.**equalizerHomeomorph** lean/su-functor/SuFunctor/Universe/FellDescentEqualizer.lean, line 287

```

def equalizerHomeomorph [R1Space α] (U : ι → Set α) (hUo : ∀ i, IsOpen (U i))
  (hcov : ⋃ i, U i = univ) :
  FellClosed.s α ≈τ Equalizer U where ...

```



SuFunctor.Adequacy.**meshDescent** lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.lean, line 235

```
noncomputable def meshDescent (M : RingMesh) (T : Set ℝ) :
  FellClosedS (MeshPt M T E) ≈t
  FellClosedS.Equalizer (arcRegion M T E)
```

SuFunctor.Adequacy.**setStratum\_holonomy** lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.lean, line 688

```
theorem setStratum_holonomy (D : ℝ) {t₀ : ℝ} (ht₀ : t₀ ∈ T) :
  ∃ p : FellClosedS.Equalizer (arcRegion M T ℝ),
    (∀ i, IsCellTube M D T i (p.1 i : Set ℝ (arcRegion M T ℝ i))) ∧
    TubeWrapWall M T (p.1 0 : Set ℝ (arcRegion M T ℝ 0))
    (p.1 2 : Set ℝ (arcRegion M T ℝ 2)) ∧
    IsChartTube M D T (meshGlue M T p : Set (MeshPt M T ℝ)) ∧
    ¬ IsTorusTube M D T (meshGlue M T p : Set (MeshPt M T ℝ))
```

SuFunctor.Adequacy.**orbitTube\_not\_torusTube** lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.lean, line 640

```
theorem orbitTube_not_torusTube (D : ℝ) {t₀ : ℝ} (ht₀ : t₀ ∈ T) :
  ¬ IsTorusTube M D T (orbitTube M T)
```

SuFunctor.Adequacy.**not\_isLocalLocusEq\_torusTube** lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.lean, line 723

```
theorem not_isLocalLocusEq_torusTube (D : ℝ) {t₀ : ℝ} (ht₀ : t₀ ∈ T) :
  ¬ FellClosedS.IsLocalLocusEq (arcRegion M T ℝ)
    (torusTubeLocus M D T) (cellTubeLocus M D T)
```

SuFunctor.Adequacy.**tubeAtoms\_not\_classical** lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.lean, line 863

```
theorem tubeAtoms_not_classical {t₀ : ℝ} (ht₀ : t₀ ∈ T) (D : ℝ) :
  ¬ ∀ μ : SubPMF (FellClosedS.Equalizer (arcRegion M T ℝ)),
    looseConj (tubeAtoms M D T) μ ↔ suppC (torusTubeEqs M D T) μ
```

SuFunctor.Adequacy.**solset\_forward\_cell** lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.lean, line 818

```
theorem solset_forward_cell
  {a : Crd (FellClosedS.Equalizer (arcRegion M T E))} :
  ∃ _ : Cell (suppC (torusTubeEqs M D T) : Loose a a)
    (looseConj (tubeAtoms M D T)) (1 a) (1 a), True
```

SuFunctor.Adequacy.**isTorusTube\_univ** lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.lean, line 372

```
theorem isTorusTube_univ :
  IsTorusTube M D T (univ : Set (MeshPt M T E))
```

SuFunctor.Adequacy.**labelled\_ensembles\_glue** lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.lean, line 385

```
theorem labelled_ensembles_glue {ι' : Type*} (hT : IsOpen T)
  {c1 c2 c3 : ι' → ℝ → ℝ → E}
  (h1 : ∀ j, IsHeatSol D M.I1 T (c1 j))
  (h2 : ∀ j, IsHeatSol D M.I2 T (c2 j))
  (h3 : ∀ j, IsHeatSol D M.I3 T (c3 j))
  (hs12 : ∀ j, ∀ x ∈ M.I1 ∩ M.I2, ∀ t ∈ T, c1 j x t = c2 j x t)
  (hs23 : ∀ j, ∀ x ∈ M.I2 ∩ M.I3, ∀ t ∈ T, c2 j x t = c3 j x t)
  (hw : ∀ j, WrapWall M T (c1 j) (c3 j)) :
  ∃ u : ι' → ℝ → ℝ → E, ∀ j,
    IsHeatTorusSol D M.L T (u j) ∧
    (∀ x ∈ M.I1, ∀ t ∈ T, u j x t = c1 j x t) ∧
    (∀ x ∈ M.I2, ∀ t ∈ T, u j x t = c2 j x t) ∧
    (∀ x ∈ M.I3, ∀ t ∈ T, u j x t = c3 j x t)
```

SuFunctor.Adequacy.**orbit\_enclosure\_forces\_zero** lean/su-functor/SuFunctor/Adequacy/SolutionSetMesh.lean, line 991

```

theorem orbit_enclosure_forces_zero (D : ℝ) {t₀ : ℝ} (ht₀ : t₀ ∈ T) :
  ∀ μ : SubPMF (HeatRec ℝ),
    looseConj (heatAtoms M D T (E := ℝ)
      ++ [suppC (enclose1C M T (orbitSet M)),
        suppC (enclose2C M T (orbitSet M)),
        suppC (enclose3C M T (orbitSet M))]) μ →
    μ = SubPMF.zero (HeatRec ℝ)

```

SuFunctor.Adequacy.LocalRows.**mesh\_adequacy**

lean/su- functor/SuFunctor/Adequacy/Pde.lean, line 400

```

theorem mesh_adequacy (R : LocalRows E) (M : RingMesh) {T : Set ℝ}
  (hT : IsOpen T) {a : Crd (HeatRec E)} :
  ∃ α : Cell (looseConj (R.meshAtoms M T) : Loose a a)
    (suppC (R.classicalRecs M T)) (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.LocalRows.PdeAt.**congr**

lean/su- functor/SuFunctor/Adequacy/Pde.lean, line 644

```

theorem PdeAt.congr {u v : ℝ → ℝ → ι → ℝ} {x t : ℝ}
  {Sx St : Set ℝ} (hSx : IsOpen Sx) (hSt : IsOpen St)
  (hx : x ∈ Sx) (ht : t ∈ St)
  (hagree : ∀ y ∈ Sx, ∀ s ∈ St, u y s = v y s)
  (h : PdeAt P v x t) : PdeAt P u x t

```

SuFunctor.Adequacy.LocalRows.PdeAt.**comp\_sub\_const**

lean/su- functor/SuFunctor/Adequacy/Pde.lean, line 683

```

theorem PdeAt.comp_sub_const {w : ℝ → ℝ → ι → ℝ} {x t c : ℝ}
  (h : PdeAt P w (x - c) t) :
  PdeAt P (fun y s => w (y - c) s) x t

```

SuFunctor.Adequacy.LocalRows.**pde\_adequacy**

lean/su- functor/SuFunctor/Adequacy/Pde.lean, line 768

```

theorem pde_adequacy (P : PdeMechanism ι k) (M : RingMesh) {T : Set ℝ}
  (hT : IsOpen T) {a : Crd (HeatRec (ι → ℝ))} :
  ∃ α : Cell (looseConj (P.rows.meshAtoms M T) : Loose a a)
    (suppC (P.rows.classicalRecs M T)) (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.LocalRows.**heat\_adequacy\_of\_mesh**

lean/su- functor/SuFunctor/Adequacy/Pde.lean, line 510

```

theorem heat_adequacy_of_mesh (M : RingMesh) (D : ℝ) {T : Set ℝ}
  (hT : IsOpen T) {a : Crd (HeatRec E)} :
  ∃ α : Cell (looseConj (heatAtoms M D T) : Loose a a)
    (suppC (heatClassicalRecs M D T)) (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.LocalRows.**heatAt\_iff\_pdeAt**

lean/su- functor/SuFunctor/Adequacy/Pde.lean, line 835

```

theorem heatAt_iff_pdeAt {D : ℝ} {u : ℝ → ℝ → ℝ} {x t : ℝ} :
  HeatAt D u x t ↔ PdeAt (heatMechanism D) (fun y s => u y s) x t

```

SuFunctor.Adequacy.LocalRows.**burgers\_adequacy**

lean/su- functor/SuFunctor/Adequacy/Pde.lean, line 918

```

theorem burgers_adequacy (v : ℝ) (M : RingMesh) {T : Set ℝ}
  (hT : IsOpen T) {a : Crd (HeatRec (PUnit → ℝ))} :
  ∃ α : Cell
    (looseConj ((burgersMechanism v).rows.meshAtoms M T) : Loose a a)
    (suppC ((burgersMechanism v).rows.classicalRecs M T))
    (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.LocalRows.**kpp\_adequacy**

lean/su- functor/SuFunctor/Adequacy/Pde.lean, line 928

```

theorem kpp_adequacy (M : RingMesh) {T : Set ℝ}
  (hT : IsOpen T) {a : Crd (HeatRec (PUnit → ℝ))} :
  ∃ α : Cell
    (looseConj (kppMechanism.rows.meshAtoms M T) : Loose a a)
    (suppC (kppMechanism.rows.classicalRecs M T)) (⊥ a) (⊥ a),
  Cell.IsInvertible α

```

SuFunctor.Adequacy.LocalRows.**coupled\_adequacy** lean/su-functor/SuFunctor/Adequacy/Pde.lean, line 937

```

theorem coupled_adequacy (M : RingMesh) {T : Set ℝ}
  (hT : IsOpen T) {a : Crd (HeatRec (Fin 2 → ℝ))} :
  ∃ α : Cell
    (looseConj (coupledMechanism.rows.meshAtoms M T) : Loose a a)
    (suppC (coupledMechanism.rows.classicalRecs M T)) (⊥ a) (⊥ a),
  Cell.IsInvertible α

```

SuFunctor.Adequacy.LocalRows.**PdeRung** lean/su-functor/SuFunctor/Adequacy/Pde.lean, line 1010

```

structure PdeRung {k : ℕ} (P : PdeMechanism ι k) where ...

```

SuFunctor.Adequacy.LocalRows.**rung\_adequacy** lean/su-functor/SuFunctor/Adequacy/Pde.lean, line 1045

```

theorem rung_adequacy {k : ℕ} {P : PdeMechanism ι k} (B : PdeRung P)
  (M : RingMesh) {T : Set ℝ} (hT : IsOpen T)
  {a : Crd (HeatRec (ι → ℝ))} :
  ∃ α : Cell (looseConj (B.rows.meshAtoms M T) : Loose a a)
    (suppC (B.rows.classicalRecs M T)) (⊥ a) (⊥ a),
  Cell.IsInvertible α

```

SuFunctor.Adequacy.LocalRows.**pde\_rung\_transfer** lean/su-functor/SuFunctor/Adequacy/Pde.lean, line 1070

```

theorem pde_rung_transfer {k : ℕ} {P : PdeMechanism ι k} (B : PdeRung P)
  (M : RingMesh) (T : Set ℝ) {a : Crd (HeatRec (ι → ℝ))} :
  ∃ _ : Cell (looseConj (P.rows.meshAtoms M T) : Loose a a)
    (looseConj (B.rows.meshAtoms M T)) (⊥ a) (⊥ a), True

```

SuFunctor.Adequacy.LocalRows.**pdeAtoms\_consistent\_of\_equilibrium** lean/su-functor/SuFunctor/Adequacy/Pde.lean, line 795

```

theorem pdeAtoms_consistent_of_equilibrium {M : RingMesh} {T : Set ℝ}
  {e : ι → ℝ} (he : P.IsEquilibriumValue e) :
  ∃ μ : SubPMF (HeatRec (ι → ℝ)),
    looseConj (P.rows.meshAtoms M T) μ ∧
    μ ≠ SubPMF.zero (HeatRec (ι → ℝ))

```

SuFunctor.Adequacy.LocalRows.**kpp\_not\_equilibrium\_half** lean/su-functor/SuFunctor/Adequacy/Pde.lean, line 975

```

theorem kpp_not_equilibrium_half :
  ¬ kppMechanism.IsEquilibriumValue (fun _ => (1 / 2 : ℝ))

```

SuFunctor.Adequacy.LocalRows.**su\_pde\_adequacy** lean/su-functor/SuFunctor/Adequacy/Pde.lean, line 1132

```

theorem su_pde_adequacy {k : ℕ} (P : PdeMechanism ι k) (M : RingMesh)
  {T : Set ℝ} (hT : IsOpen T) (μ : SubPMF (HeatRec (ι → ℝ))) :
  idealPoint μ ∈ (P.rows.meshDnf M T).modelClass ↔
  suppC (P.rows.classicalRecs M T) μ

```

SuFunctor.Adequacy.PdeRowSpec.**toMechanism** lean/su-functor/SuFunctor/Adequacy/PdeRows.lean, line 173

```

noncomputable def toMechanism [Inhabited ι] (S : PdeRowSpec ι k) :
  PdeMechanism ι k where ...

```

SuFunctor.Adequacy.**toMv\_congr** lean/su-functor/SuFunctor/Adequacy/PdeRows.lean, line 154

```

theorem toMv_congr {σ : Type} {b1 b2 : ℕ → σ} :
  ∀ rows : List RowTerm,
    (∀ t ∈ rows, ∀ j, j < t.exps.length → b1 j = b2 j) →
      toMv b1 rows = toMv b2 rows
| [], _ => rfl
| t :: ts, h => by
  have hterm : termProd b1 0 t.exps = termProd b2 0 t.exps

```

SuFunctor.Adequacy.PdeRowSpec.**toMechanism\_congr\_inhabited** lean/su-functor/SuFunctor/Adequacy/PdeRows.lean, line 181

```

theorem toMechanism_congr_inhabited (S : PdeRowSpec ι k) (hWF : S.WF)
  (d1 d2 : Inhabited ι) :
  @toMechanism ι k d1 S = @toMechanism ι k d2 S

```

SuFunctor.Adequacy.**heatRowSpec\_toMechanism** lean/su-functor/SuFunctor/Adequacy/PdeRows.lean, line 220

```

theorem heatRowSpec_toMechanism (c : ℚ) :
  (heatRowSpec c).toMechanism = heatMechanism (c : ℝ)

```

SuFunctor.Adequacy.**heat\_rows\_adequacy** lean/su-functor/SuFunctor/Adequacy/PdeRows.lean, line 239

```

theorem heat_rows_adequacy (c : ℚ) (M : RingMesh) {T : Set ℝ}
  (hT : IsOpen T) {a : Crd (HeatRec (PUnit → ℝ))} :
  ∃ α : Cell
    (looseConj (((heatRowSpec c).toMechanism).rows.meshAtoms M T) : Loose a a)
    (suppC (((heatRowSpec c).toMechanism).rows.classicalRecs M T))
    (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.**su\_pde\_rows\_adequacy** lean/su-functor/SuFunctor/Adequacy/PdeRows.lean, line 310

```

theorem su_pde_rows_adequacy {ι : Type} [Fintype ι] [Inhabited ι] {k : ℕ}
  (S : PdeRowSpec ι k) (M : RingMesh) {T : Set ℝ} (hT : IsOpen T)
  (μ : SubPMF (HeatRec (ι → ℝ))) :
  idealPoint μ ∈ ((S.toMechanism).rows.meshDnf M T).modelClass ↔
  suppC ((S.toMechanism).rows.classicalRecs M T) μ

```

SuFunctor.Adequacy.**composite\_iff\_dde** lean/su-functor/SuFunctor/Adequacy/DelayLine.lean, line 442

```

theorem composite_iff_dde {f : E → E → E} {c τ : ℝ} {φ : ℝ → E}
  (hc : 0 < c) (hτ : 0 < τ) (hφ : ContinuousOn φ (Icc (-τ) 0))
  {y z : ℝ → E} {u : ℝ → ℝ → E} :
  IsCompositeSol f c τ φ y z u ↔
  ∃ Y, IsDdeSol f τ φ Y ∧ RecMatches c τ Y y z u

```

SuFunctor.Adequacy.**delayline\_adequacy** lean/su-functor/SuFunctor/Adequacy/DelayLine.lean, line 592

```

theorem delayline_adequacy (f : E → E → E) {c τ : ℝ} (hc : 0 < c)
  (hτ : 0 < τ) {φ : ℝ → E} (hφ : ContinuousOn φ (Icc (-τ) 0))
  {a : Crd (DelayRec E)} :
  ∃ α : Cell (looseConj (delayAtoms f c τ φ) : Loose a a)
    (suppC (classicalRecs f c τ φ)) (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.IsCompositeSol.**outflow\_delay** lean/su-functor/SuFunctor/Adequacy/DelayLine.lean, line 295

```

theorem outflow_delay (hc : 0 < c) (hτ : 0 < τ)
  (h : IsCompositeSol f c τ φ y z u) :
  ∀ t ∈ Ici (0 : ℝ), z t = concat φ y (t - τ)

```

SuFunctor.Adequacy.IsCompositeSol.**field\_slice\_is\_history** lean/su-functor/SuFunctor/Adequacy/DelayLine.lean, line 310

```

theorem field_slice_is_history (hc : 0 < c)
  (h : IsCompositeSol f c τ φ y z u) {t : ℝ} (ht : t ∈ Ici (0 : ℝ)) :
  ∀ s ∈ Icc (t - τ) t, u (c * (t - s)) t = concat φ y s

```

Qtsl.JumpDiffusion.MpSolution lean/qtsl-aux/QtslAux/JumpDiffusionMartingaleProblem.lean, line 191

```

structure MpSolution (P : JdProfile n m) (D : Set (State n))
  (Ω : Type*) [MeasureSpace Ω] where ...

```

Qtsl.JumpDiffusion.MpSolution.supermartingale\_of\_gen\_nonpos lean/qtsl-aux/QtslAux/JumpDiffusionMartingaleProblem.lean, line 234

```

theorem supermartingale_of_gen_nonpos
  [IsProbabilityMeasure (volume : Measure Ω)]
  (mp : MpSolution P D Ω) (hC2 : IsC2Fn V gradV hessV)
  (hgint : ∀ (ω : Ω) (t : ℝ), IntervalIntegrable
    (fun s => generator P V gradV hessV (mp.X (Real.toNNReal s) ω))
    volume 0 t)
  (hVint : ∀ t : ℝ≥0, Integrable (fun ω => V (mp.X t ω))
    (volume : Measure Ω))
  (hrow : ∀ z ∈ D, generator P V gradV hessV z ≤ 0) :
  Supermartingale (fun (t : ℝ≥0) (ω : Ω) => V (mp.X t ω)) mp.ℱ
    (volume : Measure Ω)

```

Qtsl.SdeMartingaleProblem.MpSolution.toSdeDynkinVille lean/qtsl-aux/QtslAux/SdeMartingaleProblem.lean, line 409

```

noncomputable def toSdeDynkinVille
  [IsProbabilityMeasure (volume : Measure Ω)]
  (mp : MpSolution P D Ω) (τ : ℕ → ℝ≥0) (ht : Monotone τ)
  (hC2 : IsC2Fn V gradV hessV)
  (hgint : ∀ (ω : Ω) (t : ℝ), IntervalIntegrable
    (fun s => generator P V gradV hessV (mp.X (Real.toNNReal s) ω))
    volume 0 t)
  (hVint : ∀ t : ℝ≥0, Integrable (fun ω => V (mp.X t ω))
    (volume : Measure Ω)) :
  SdeDynkinVille P D V gradV hessV Ω where ...

```

Qtsl.JumpDiffusion.genPoly\_eval\_eq\_generator lean/qtsl-aux/QtslAux/JumpDiffusion.lean, line 269

```

theorem genPoly_eval_eq_generator (P : PolyProfile n m)
  (V : MvPolynomial (Fin n) ℚ) (z : State n) :
  (aeval z (genPoly P V) : ℝ)
  = generator P.toJd
    (fun w => (aeval w V : ℝ))
    (fun w i => (aeval w (pderiv i V) : ℝ))
    (fun w i j => (aeval w (pderiv i (pderiv j V)) : ℝ)) z

```

Qtsl.SdeMartingaleProblem.isC2Fn\_poly lean/qtsl-aux/QtslAux/SdeMartingaleProblem.lean, line 285

```

theorem isC2Fn_poly (V : MvPolynomial (Fin n) ℚ) :
  IsC2Fn (fun z : State n => (aeval z V : ℝ))
    (fun z i => (aeval z (pderiv i V) : ℝ))
    (fun z i j => (aeval z (pderiv i (pderiv j V)) : ℝ)) where
  hasFDerivAt_V z

```

Qtsl.JumpDiffusion.mp\_avoid\_row\_of\_genPoly lean/qtsl-aux/QtslAux/JumpDiffusionMartingaleProblem.lean, line 326

```

theorem mp_avoid_row_of_genPoly
  [IsProbabilityMeasure (volume : Measure Ω)]
  {P₀ : PolyProfile n m} {D : Set (State n)}
  (V₀ : MvPolynomial (Fin n) ℚ)
  (mp : MpSolution P₀.toJd D Ω)
  (τ : ℕ → ℝ≥0) (ht : Monotone τ)
  (hgint : ∀ (ω : Ω) (t : ℝ), IntervalIntegrable

```

```

    (fun s => (aeval (mp.X (Real.toNNReal s) ω) (genPoly P₀ V₀) : ℝ))
    volume 0 t)
(hVint : ∀ t : ℝ≥0, Integrable
  (fun ω => (aeval (mp.X t ω) V₀ : ℝ)) (volume : Measure Ω))
(hrow : ∀ z ∈ D, (aeval z (genPoly P₀ V₀) : ℝ) ≤ 0)
(hnonneg : ∀ z ∈ D, 0 ≤ (aeval z V₀ : ℝ))
{Bad : Set (State n)} {a : ℝ}
(hfloor : ∀ z ∈ Bad, a ≤ (aeval z V₀ : ℝ)) :
ENNReal.ofReal a * volume {ω | ∃ k, mp.X (τ k) ω ∈ Bad}
≤ ENNReal.ofReal (∫ ω, (aeval (mp.X (τ 0) ω) V₀ : ℝ))

```

QtSl.JumpDiffusion.**mp\_avoid\_init\_of\_genPoly** lean/qtSl-aux/QtSlAux/JumpDiffusionMartingaleProblem.lean, line 370

```

theorem mp_avoid_init_of_genPoly
  [IsProbabilityMeasure (volume : Measure Ω)]
  {P₀ : PolyProfile n m} {D : Set (State n)}
  (V₀ : MvPolynomial (Fin n) ℚ)
  (mp : MpSolution P₀.toJd D Ω)
  (τ : ℕ → ℝ≥0) (ht : Monotone τ)
  (hgint : ∀ (ω : Ω) (t : ℝ), IntervalIntegrable
    (fun s => (aeval (mp.X (Real.toNNReal s) ω) (genPoly P₀ V₀) : ℝ))
    volume 0 t)
  (hVint : ∀ t : ℝ≥0, Integrable
    (fun ω => (aeval (mp.X t ω) V₀ : ℝ)) (volume : Measure Ω))
  (hrow : ∀ z ∈ D, (aeval z (genPoly P₀ V₀) : ℝ) ≤ 0)
  (hnonneg : ∀ z ∈ D, 0 ≤ (aeval z V₀ : ℝ))
  {Bad : Set (State n)} {z₀ : State n}
  (hstart : ∀ ω : Ω, mp.X (τ 0) ω = z₀)
  (hfloor : ∀ z ∈ Bad, 1 ≤ (aeval z V₀ : ℝ)) :
volume {ω | ∃ k, mp.X (τ k) ω ∈ Bad}
≤ ENNReal.ofReal (aeval z₀ V₀ : ℝ)

```

QtSl.JumpDiffusion.**mp\_control** lean/qtSl-aux/QtSlAux/JumpDiffusionMartingaleProblem.lean, line 543

```

theorem mp_control [IsProbabilityMeasure (volume : Measure Ω)]
  (x₀ : State 1) {Bad : Set (State 1)}
  (hfloor : ∀ z ∈ Bad,
    1 ≤ (aeval z (X 0 * X 0 : MvPolynomial (Fin 1) ℚ) : ℝ)) :
volume {ω : Ω | ∃ k : ℕ, x₀ ∈ Bad}
≤ ENNReal.ofReal (aeval x₀ (X 0 * X 0 : MvPolynomial (Fin 1) ℚ) : ℝ)

```

SuFunctor.Adequacy.SpectrumDecl.**plainTrace\_divergent\_none** lean/su-functor/SuFunctor/Adequacy/PdeRows.lean, line 549

```

theorem plainTrace_divergent_none (ds : List DriverDecl) :
  (SpectrumDecl.mk ds .divergent).plainTrace = none

```

SuFunctor.Adequacy.**isRodeSol\_iff\_integralCurve** lean/su-functor/SuFunctor/Adequacy/Rode.lean, line 63

```

theorem isRodeSol_iff_integralCurve [CompleteSpace E] {f : θ → E → E}
  {x₀ : E} {θ : θ} {y : ℝ → E} (hf : Continuous (f θ)) :
  IsRodeSol f x₀ θ y ↔
    y 0 = x₀ ∧ IsIntegralCurveOn y (fun _ => f θ) (Ici 0)

```

SuFunctor.Adequacy.**rode\_draw\_adequacy** lean/su-functor/SuFunctor/Adequacy/Rode.lean, line 71

```

theorem rode_draw_adequacy [CompleteSpace E] {f : θ → E → E} {θ : θ}
  (hf : Continuous (f θ)) (x₀ : E) {a : Crd (ℝ → E)} :
  ∃ α : Cell (looseConj (odeAtoms (f θ) x₀) : Loose a a)
    (suppC (integralCurves (f θ) x₀)) (1 a) (1 a),
  Cell.IsInvertible α

```

SuFunctor.Adequacy.**rode\_adequacy** lean/su-functor/SuFunctor/Adequacy/Rode.lean, line 160

```

theorem rode_adequacy [CompleteSpace E] {f :  $\theta \rightarrow E \rightarrow E$ } {P : Set  $\theta$ }
  (hf :  $\forall \theta \in P$ , Continuous (f  $\theta$ )) ( $x_0 : E$ ) {a : Crd ( $\mathbb{R} \rightarrow E$ )} :
   $\exists \alpha : \text{Cell (looseConj (rodeAtoms f P } x_0)) : \text{Loose a a)}$ 
    (suppC ( $\bigcup \theta \in P$ , integralCurves (f  $\theta$ )  $x_0$ )) ( $\mathbb{1}$  a) ( $\mathbb{1}$  a),
    Cell.IsInvertible  $\alpha$ 

```

SuFunctor.Adequacy.**rode\_box\_adequacy**

lean/su-functor/SuFunctor/Adequacy/Rode.lean, line 241

```

theorem rode_box_adequacy [CompleteSpace E] {f :  $\theta \rightarrow E \rightarrow E$ } {P : Set  $\theta$ }
  (hf :  $\forall \theta \in P$ , Continuous (f  $\theta$ )) (B : Set E) {a : Crd ( $\mathbb{R} \rightarrow E$ )} :
   $\exists \alpha : \text{Cell (looseConj (rodeBoxAtoms f P B)) : \text{Loose a a)}$ 
    (suppC ( $\bigcup \theta \in P$ ,  $\bigcup x_0 \in B$ , integralCurves (f  $\theta$ )  $x_0$ ))
    ( $\mathbb{1}$  a) ( $\mathbb{1}$  a),
    Cell.IsInvertible  $\alpha$ 

```

SuFunctor.Adequacy.**rodeSols\_eqOn\_of\_lipschitzOnWith**

lean/su-functor/SuFunctor/Adequacy/Rode.lean, line 81

```

theorem rodeSols_eqOn_of_lipschitzOnWith {f :  $\theta \rightarrow E \rightarrow E$ } {K :  $\mathbb{R}_{\geq 0}$ }
  {s : Set E} {P : Set  $\theta$ } (hL :  $\forall \theta \in P$ , LipschitzOnWith K (f  $\theta$ ) s)
  { $x_0 : E$ } { $\theta : \theta$ } (h $\theta : \theta \in P$ ) { $y_1 y_2 : \mathbb{R} \rightarrow E$ }
  (h $_1 : \text{IsRodeSol f } x_0 \theta y_1$ ) (h $_2 : \text{IsRodeSol f } x_0 \theta y_2$ )
  (hs $_1 : \forall t \in \text{Ici } (\theta : \mathbb{R}), y_1 t \in s$ )
  (hs $_2 : \forall t \in \text{Ici } (\theta : \mathbb{R}), y_2 t \in s$ ) :
  EqOn  $y_1 y_2$  (Ici  $\theta$ )

```

SuFunctor.Adequacy.**su\_rode\_adequacy**

lean/su-functor/SuFunctor/Adequacy/Rode.lean, line 281

```

theorem su_rode_adequacy [CompleteSpace E] {f :  $\theta \rightarrow E \rightarrow E$ } {P : Set  $\theta$ }
  (hf :  $\forall \theta \in P$ , Continuous (f  $\theta$ )) ( $x_0 : E$ ) ( $\mu : \text{SubPMF } (\mathbb{R} \rightarrow E)$ ) :
  idealPoint  $\mu \in (\text{rodeDnf f P } x_0).\text{modelClass} \leftrightarrow$ 
  suppC ( $\bigcup \theta \in P$ , integralCurves (f  $\theta$ )  $x_0$ )  $\mu$ 

```

QtSl.JumpDiffusion.**JdProfile**

lean/qtSl-aux/QtSlAux/JumpDiffusion.lean, line 121

```

structure JdProfile (n m :  $\mathbb{N}$ ) where ...

```

QtSl.JumpDiffusion.**generator\_one\_dirac**

lean/qtSl-aux/QtSlAux/JumpDiffusion.lean, line 178

```

theorem generator_one_dirac (P : JdProfile n m) (lam : State n  $\rightarrow \mathbb{R}$ )
  (g : State n  $\rightarrow$  State n) (hkill :  $\forall z$ , P.kill z = 0)
  (hlegs : P.legs = [(lam, [(1 :  $\mathbb{R}$ ), g])])
  (V : State n  $\rightarrow \mathbb{R}$ ) (gradV : State n  $\rightarrow$  State n)
  (hessV : State n  $\rightarrow$  Fin n  $\rightarrow$  Fin n  $\rightarrow \mathbb{R}$ ) (z : State n) :
  generator P V gradV hessV z
    = SdeGenerator.generator (P.f, P. $\sigma$ , lam, g) V gradV hessV z

```

QtSl.JumpDiffusion.**toJd\_generator\_one\_dirac**

lean/qtSl-aux/QtSlAux/JumpDiffusion.lean, line 293

```

theorem toJd_generator_one_dirac (P : PolyProfile n m)
  (lam : MvPolynomial (Fin n)  $\mathbb{Q}$ ) (g : Fin n  $\rightarrow$  MvPolynomial (Fin n)  $\mathbb{Q}$ )
  (hkill : P.kill = 0) (hlegs : P.legs = [(lam, [(1 :  $\mathbb{Q}$ ), g])])
  (V : State n  $\rightarrow \mathbb{R}$ ) (gradV : State n  $\rightarrow$  State n)
  (hessV : State n  $\rightarrow$  Fin n  $\rightarrow$  Fin n  $\rightarrow \mathbb{R}$ ) (z : State n) :
  generator P.toJd V gradV hessV z
    = SdeGenerator.generator (P.toSde lam g) V gradV hessV z

```

QtSl.JumpDiffusion.**transferSum\_sub\_self**

lean/qtSl-aux/QtSlAux/JumpDiffusion.lean, line 101

```

theorem transferSum_sub_self ( $\mu : \text{MixtureR } n$ ) (V : State n  $\rightarrow \mathbb{R}$ )
  (z : State n) :
  transferSum  $\mu$  V z - V z
    = ( $\mu.\text{map fun wg} \Rightarrow \text{wg.1} * (V (\text{wg.2 } z) - V z)$ ).sum
      - (1 - ( $\mu.\text{map fun wg} \Rightarrow \text{wg.1}$ )).sum) * V z

```



Qtsl.JumpDiffusion.two\_leg\_control

lean/qtsl-aux/QtslAux/JumpDiffusion.lean, line 329

```

theorem two_leg_control (z : State 1) :
  4 * generator twoLeg.toJd (fun w => (w 0) ^ 2)
    (fun w _ => 2 * w 0) (fun _ _ => 2) z
    = 4 + (z 0) ^ 2 - 3 * (z 0) ^ 4

```

Qtsl.JumpDiffusion.JdDynkinVille

lean/qtsl-aux/QtslAux/JumpDiffusionMartingaleProblem.lean, line 82

```

structure JdDynkinVille (P : JdProfile n m) (D : Set (State n))
  (V : State n → ℝ) (gradV : State n → State n)
  (hessV : State n → Fin n → Fin n → ℝ)
  (Ω : Type*) [MeasureSpace Ω] where ...

```

Qtsl.JumpDiffusion.generator\_toJd

lean/qtsl-aux/QtslAux/JumpDiffusionMartingaleProblem.lean, line 431

```

theorem generator_toJd (P : SdeProfile n m) (V : State n → ℝ)
  (gradV : State n → State n) (hessV : State n → Fin n → Fin n → ℝ)
  (z : State n) :
  generator P.toJd V gradV hessV z
    = SdeGenerator.generator P V gradV hessV z

```

Qtsl.JumpDiffusion.\_root\_.Qtsl.SdeMartingaleProblem.MpSolution.toJd

lean/qtsl-aux/QtslAux/

JumpDiffusionMartingaleProblem.lean, line 445

```

noncomputable def _root_.Qtsl.SdeMartingaleProblem.MpSolution.toJd
  {P : SdeProfile n m} {D : Set (State n)} {Ω : Type*} [MeasureSpace Ω]
  (mp : SdeMartingaleProblem.MpSolution P D Ω) :
  MpSolution P.toJd D Ω where ...

```

SuFunctor.Adequacy.satAtoms\_iff

lean/su-functor/SuFunctor/Adequacy/Dtmc.lean, line 598

```

theorem satAtoms_iff (hR : R.WF) (μ : SubPMF (Traj S k)) :
  looseConj (atoms R k) μ ↔ μ = rowsLaw R hR k

```

SuFunctor.Adequacy.dtmc\_adequacy

lean/su-functor/SuFunctor/Adequacy/Dtmc.lean, line 622

```

theorem dtmc_adequacy (M : FinDTMC S) (k : ℕ) {a : Crd (Traj S k)} :
  ∃ α : Cell (looseConj (atoms (compile M) k) : Loose a a)
    (singletonC (M.pathLaw k)) (1 a) (1 a),
  Cell.IsInvertible α

```

SuFunctor.Adequacy.FinDTMC.pathLaw\_mass

lean/su-functor/SuFunctor/Adequacy/Dtmc.lean, line 259

```

theorem FinDTMC.pathLaw_mass (M : FinDTMC S) (k : ℕ) :
  (M.pathLaw k).mass = 1

```

SuFunctor.Adequacy.FinDTMC.trajPMF\_eq

lean/su-functor/SuFunctor/Adequacy/Dtmc.lean, line 743

```

theorem FinDTMC.trajPMF_eq :
  ∀ (k : ℕ) (ω : Traj S k),
  (M.trajPMF k) ω = q2e ((M.pathLaw k).wt ω)

```

SuFunctor.Adequacy.ctmc\_jump\_adequacy

lean/su-functor/SuFunctor/Adequacy/CtmcJumpChain.lean, line 371

```

theorem ctmc_jump_adequacy (M : FinCTMC S) (k : ℕ) {a : Crd (Traj S k)} :
  ∃ α : Cell
    (looseConj (atoms (jumpRows (compileCtmc M)) k) : Loose a a)
    (singletonC (M.jumpLaw k)) (1 a) (1 a),
  Cell.IsInvertible α

```

SuFunctor.Adequacy.compile\_jump\_comm

lean/su-functor/SuFunctor/Adequacy/CtmcJumpChain.lean, line 352

```

theorem compile_jump_comm (M : FinCTMC S) :
  jumpRows (compileCtmc M) = compile M.jumpChain

```

SuFunctor.Adequacy.**jumpRows\_wf** lean/su-functor/SuFunctor/Adequacy/CtmcJumpChain.lean, line 338

```
theorem jumpRows_wf {R : CtmcRows S} (hR : R.WF) : (jumpRows R).WF where
  outcome_nonneg p
```

SuFunctor.Adequacy.FinCTMC.**jumpChain\_step\_mul\_exit** lean/su-functor/SuFunctor/Adequacy/CtmcJumpChain.lean, line 210

```
theorem FinCTMC.jumpChain_step_mul_exit (M : FinCTMC S) (s s' : S) :
  M.jumpChain.step s s' * M.exitRate s = M.offRate s s'
```

SuFunctor.Adequacy.**compileCtmc\_diag\_blind** lean/su-functor/SuFunctor/Adequacy/CtmcJumpChain.lean, line 385

```
theorem compileCtmc_diag_blind (M M' : FinCTMC S)
  (hoff : ∀ s s', s ≠ s' → M.rate s s' = M'.rate s s')
  (hinit : M.init = M'.init) : compileCtmc M = compileCtmc M'
```

SuFunctor.Adequacy.FinCTMC.**jumpLaw\_diag\_blind** lean/su-functor/SuFunctor/Adequacy/CtmcJumpChain.lean, line 407

```
theorem FinCTMC.jumpLaw_diag_blind (M M' : FinCTMC S)
  (hoff : ∀ s s', s ≠ s' → M.rate s s' = M'.rate s s')
  (hinit : M.init = M'.init) (k : ℕ) :
  M.jumpLaw k = M'.jumpLaw k
```

SuFunctor.Adequacy.FinCTMC.**jumpTrajPMF\_eq** lean/su-functor/SuFunctor/Adequacy/CtmcJumpChain.lean, line 440

```
theorem FinCTMC.jumpTrajPMF_eq (M : FinCTMC S) (k : ℕ) (ω : Traj S k) :
  M.jumpTrajPMF k ω = q2e ((M.jumpLaw k).wt ω)
```

SuFunctor.Adequacy.**mdp\_adequacy** lean/su-functor/SuFunctor/Adequacy/Mdp.lean, line 284

```
theorem mdp_adequacy (M : FinMDP S A) (π : Policy S A) (k : ℕ)
  {a : Crd (Traj S k)} :
  ∃ α : Cell
    (looseConj (atoms (closeRows (compileMdp M) (compilePolicy π)) k) :
      Loose a a)
    (singletonC (M.pathLaw π k)) (⊥ a) (⊥ a),
  Cell.IsInvertible α
```

SuFunctor.Adequacy.**compile\_close\_comm** lean/su-functor/SuFunctor/Adequacy/Mdp.lean, line 266

```
theorem compile_close_comm (M : FinMDP S A) (π : Policy S A) :
  closeRows (compileMdp M) (compilePolicy π) =
  compile (M.closedLoop π)
```

SuFunctor.Adequacy.FinMDP.**closedLoop\_stepPMF** lean/su-functor/SuFunctor/Adequacy/Mdp.lean, line 334

```
theorem FinMDP.closedLoop_stepPMF (M : FinMDP S A) (π : Policy S A)
  (s : S) :
  (M.closedLoop π).stepPMF s =
  (π.actPMF s).bind (fun a => M.stepPMF s a)
```

SuFunctor.**unionSingletons** lean/su-functor/SuFunctor/Adequacy/EnvelopeUnion.lean, line 104

```
def unionSingletons (μs : ι → SubPMF V) : SubPMF V → Prop
```

SuFunctor.Adequacy.FinMDP.**demonicCredal** lean/su-functor/SuFunctor/Adequacy/MdpDemonic.lean, line 81

```
def FinMDP.demonicCredal (M : FinMDP S A) (k : ℕ) :
  SubPMF (Traj S k) → Prop
```

SuFunctor.Adequacy.**mdp\_demonic\_adequacy** lean/su-functor/SuFunctor/Adequacy/MdpDemonic.lean, line 157

```

theorem mdp_demonic_adequacy (M : FinMDP S A) (k : ℕ)
  {a : Crd (Traj S k)} :
  ∃ α : Cell (demonicC (compileMdp M) k : Loose a a)
    (M.demonicLaws k) (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.**demonicC** lean/su-functor/SuFunctor/Adequacy/MdpDemonic.lean, line 92

```

def demonicC (R : MdpRows S A) (k : ℕ) : SubPMF (Traj S k) → Prop

```

SuFunctor.Adequacy.**demonicC\_iff** lean/su-functor/SuFunctor/Adequacy/MdpDemonic.lean, line 128

```

theorem demonicC_iff (M : FinMDP S A) (k : ℕ) (μ : SubPMF (Traj S k)) :
  demonicC (compileMdp M) k μ ↔ M.demonicLaws k μ

```

SuFunctor.Adequacy.**demonic\_maxSlice** lean/su-functor/SuFunctor/Adequacy/MdpDemonic.lean, line 184

```

theorem demonic_maxSlice (M : FinMDP S A) (k : ℕ)
  (v : SubPMF (Traj S k)) :
  maxSlice (M.demonicCredal k) v ↔ M.demonicLaws k v

```

SuFunctor.Adequacy.**satCredal\_lower\_demonicCredal** lean/su-functor/SuFunctor/Adequacy/MdpDemonic.lean, line 205

```

theorem satCredal_lower_demonicCredal (M : FinMDP S A) (k : ℕ)
  (c : SubPMF (Traj S k) → Prop) :
  SatCredal .lower (M.demonicCredal k) c ↔
  ∀ π : Policy S A, c (M.pathLaw π k)

```

SuFunctor.Adequacy.**satCredal\_upper\_demonicCredal** lean/su-functor/SuFunctor/Adequacy/MdpDemonic.lean, line 214

```

theorem satCredal_upper_demonicCredal (M : FinMDP S A) (k : ℕ)
  {c : SubPMF (Traj S k) → Prop} (hc : DownClosed c) :
  SatCredal .upper (M.demonicCredal k) c ↔
  ∀ π : Policy S A, c (M.pathLaw π k)

```

SuFunctor.Adequacy.**not\_satCredal\_upper\_demonicCredal\_geC** lean/su-functor/SuFunctor/Adequacy/MdpDemonic.lean, line 265

```

theorem not_satCredal_upper_demonicCredal_geC (M : FinMDP S A) (k : ℕ)
  [Nonempty A] {E : Set (Traj S k)} {q : ℚ} (hq : 0 < q) :
  ¬ SatCredal .upper (M.demonicCredal k) (geC E q)

```

SuFunctor.Adequacy.**isEmpty\_policy\_iff** lean/su-functor/SuFunctor/Adequacy/MdpDemonic.lean, line 290

```

theorem isEmpty_policy_iff (M : FinMDP S A) :
  IsEmpty (Policy S A) ↔ IsEmpty A

```

SuFunctor.**not\_maxSlice\_unionRays\_halfPair** lean/su-functor/SuFunctor/Adequacy/EnvelopeUnion.lean, line 316

```

theorem not_maxSlice_unionRays_halfPair (μ : SubPMF V) :
  ¬ maxSlice (unionRays (halfPair μ)) (halfPair μ false)

```

SuFunctor.Adequacy.**ma\_adequacy** lean/su-functor/SuFunctor/Adequacy/MarkovAutomaton.lean, line 839

```

theorem ma_adequacy (M : FinMA S I A) (σ : Policy I A) (r : I → ℕ)
  (hr : ZenoFree (M.imStep σ) r) (k : ℕ) {a : Crd (Traj S k)} :
  ∃ α : Cell
    (looseConj (atoms (jumpRows
      (closeMaRows (compileMa M) (compilePolicy σ) (absFuel r))) k) :
      Loose a a)
    (singletonC (M.pathLaw σ r hr k)) (⊥ a) (⊥ a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.**compile\_closeMa\_comm** lean/su-functor/SuFunctor/Adequacy/MarkovAutomaton.lean, line 819

```

theorem compile_closeMa_comm (M : FinMA S I A) (σ : Policy I A)
  (r : I → ℕ) (hr : ZenoFree (M.imStep σ) r) :
  closeMaRows (compileMa M) (compilePolicy σ) (absFuel r)
    = compileCtmc (M.close σ r hr)

```

SuFunctioner.Adequacy.**absorbIter\_mass** lean/su-functor/SuFunctioner/Adequacy/MarkovAutomaton.lean, line 246

```

theorem absorbIter_mass (f : I → S ⊗ I → Q) (r : I → ℕ)
  (hr : ZenoFree f r) (hrow : ∀ i, ∑ y, f i y = 1) :
  ∀ (n : ℕ) (i : I), r i < n →
    ∑ m, absorbIter f n (Sum.inr i) m = 1
| 0, _, hn => absurd hn (Nat.not_lt_zero _)
| n + 1, i, hn => by
  simp_rw [absorbIter_succ_inr]
  rw [Finset.sum_comm]
  simp_rw [← Finset.mul_sum]
  have hterm : ∀ y, f i y * (∑ m, absorbIter f n y m) = f i y * 1

```

SuFunctioner.Adequacy.**absorbIter\_unique** lean/su-functor/SuFunctioner/Adequacy/MarkovAutomaton.lean, line 276

```

theorem absorbIter_unique (f : I → S ⊗ I → Q) (r : I → ℕ)
  (hr : ZenoFree f r) (g : S ⊗ I → S → Q)
  (hg_inl : ∀ m' m, g (Sum.inl m') m = hitMass (Sum.inl m' : S ⊗ I) m)
  (hg_step : ∀ i m, g (Sum.inr i) m = ∑ y, f i y * g y m) :
  ∀ (n : ℕ) (i : I) (m : S), r i < n →
    g (Sum.inr i) m = absorbIter f n (Sum.inr i) m
| 0, _, _, hn => absurd hn (Nat.not_lt_zero _)
| n + 1, i, m, hn => by
  rw [hg_step, absorbIter_succ_inr]
  refine Finset.sum_congr rfl fun y _ => ?_
  by_cases hz : f i y = 0
  · rw [hz, zero_mul, zero_mul]
  · congr 1
    cases y with
    | inl m' => rw [hg_inl, absorbIter_inl]
    | inr j =>
      exact absorbIter_unique f r hr g hg_inl hg_step n j m
        (lt_of_lt_of_le (hr i j hz) (Nat.lt_succ_iff.mp hn))

def absFuel (r : I → ℕ) : ℕ

```

SuFunctioner.Adequacy.**absorbIter\_stab** lean/su-functor/SuFunctioner/Adequacy/MarkovAutomaton.lean, line 223

```

theorem absorbIter_stab (f : I → S ⊗ I → Q) (r : I → ℕ)
  (hr : ZenoFree f r) (i : I) (m : S) {n n' : ℕ}
  (h : r i < n) (h' : r i < n') :
  absorbIter f n (Sum.inr i) m = absorbIter f n' (Sum.inr i) m

```

SuFunctioner.Adequacy.FinMA.**close\_offRate\_factor** lean/su-functor/SuFunctioner/Adequacy/MarkovAutomaton.lean, line 540

```

theorem FinMA.close_offRate_factor (M : FinMA S I A) (σ : Policy I A)
  (r : I → ℕ) (hr : ZenoFree (M.imStep σ) r) {s s' : S}
  (hne : s ≠ s') :
  (M.close σ r hr).offRate s s'
    = M.exitRate s * ∑ x, M.preChain.jumpChain.step (Sum.inl s) x
      * M.absorb σ r x s'

```

SuFunctioner.Adequacy.FinMA.**close\_exitRate\_thinning** lean/su-functor/SuFunctioner/Adequacy/MarkovAutomaton.lean, line 562

```

theorem FinMA.close_exitRate_thinning (M : FinMA S I A)
  (σ : Policy I A) (r : I → ℕ) (hr : ZenoFree (M.imStep σ) r)
  (s : S) :

```

```
(M.close σ r hr).exitRate s
= M.exitRate s - ∑ x, M.offRate s x * M.absorb σ r x s
```

SuFunctor.Adequacy.FinMA.**close\_exitRate\_le** lean/su-functor/SuFunctor/Adequacy/MarkovAutomaton.lean, line 594

```
theorem FinMA.close_exitRate_le (M : FinMA S I A) (σ : Policy I A)
(r : I → ℕ) (hr : ZenoFree (M.imStep σ) r) (s : S) :
(M.close σ r hr).exitRate s ≤ M.exitRate s
```

SuFunctor.Adequacy.**compileMa\_diag\_blind** lean/su-functor/SuFunctor/Adequacy/MarkovAutomaton.lean, line 713

```
theorem compileMa_diag_blind (M M' : FinMA S I A)
(hoff : ∀ s x, Sum.inl s ≠ x → M.rate s x = M'.rate s x)
(hbr : M.branch = M'.branch) (hinit : M.init = M'.init) :
compileMa M = compileMa M'
```

SuFunctor.Adequacy.FinMA.**trajPMF\_eq** lean/su-functor/SuFunctor/Adequacy/MarkovAutomaton.lean, line 869

```
theorem FinMA.trajPMF_eq (M : FinMA S I A) (σ : Policy I A)
(r : I → ℕ) (hr : ZenoFree (M.imStep σ) r) (k : ℕ)
(ω : Traj S k) :
M.trajPMF σ r hr k ω = q2e ((M.pathLaw σ r hr k).wt ω)
```

SuFunctor.Adequacy.**ma\_demonic\_adequacy** lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 299

```
theorem ma_demonic_adequacy (M : FinMA S I A) {r : I → ℕ}
(hb : M.BranchZenoFree r) (k : ℕ) {a : Crd (Traj S k)} :
∃ α : Cell (maDemonicC (compileMa M) (absFuel r) k : Loose a a)
(M.demonicLaws r hb k) (1 a) (1 a),
Cell.IsInvertible α
```

SuFunctor.Adequacy.**maDemonicC** lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 238

```
def maDemonicC (R : MaRows S I A) (n k : ℕ) :
SubPMF (Traj S k) → Prop
```

SuFunctor.Adequacy.**maDemonicC\_iff** lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 262

```
theorem maDemonicC_iff (M : FinMA S I A) {r : I → ℕ}
(hb : M.BranchZenoFree r) (k : ℕ) (μ : SubPMF (Traj S k)) :
maDemonicC (compileMa M) (absFuel r) k μ ↔
M.demonicLaws r hb k μ
```

SuFunctor.Adequacy.FinMA.**BranchZenoFree** lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 122

```
def FinMA.BranchZenoFree (M : FinMA S I A) (r : I → ℕ) : Prop
```

SuFunctor.Adequacy.**zenoFree\_of\_branch** lean/su-functor/SuFunctor/Adequacy/MarkovAutomaton.lean, line 411

```
theorem zenoFree_of_branch (M : FinMA S I A) (r : I → ℕ)
(hb : ∀ i a j, M.branch i a (Sum.inr j) ≠ 0 → r j < r i)
(σ : Policy I A) : ZenoFree (M.imStep σ) r
```

SuFunctor.Adequacy.FinMA.**pathLaw\_rank\_irrel** lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 169

```
theorem FinMA.pathLaw_rank_irrel (M : FinMA S I A) (σ : Policy I A)
{r r' : I → ℕ} (hr : ZenoFree (M.imStep σ) r)
(hr' : ZenoFree (M.imStep σ) r') (k : ℕ) :
M.pathLaw σ r hr k = M.pathLaw σ r' hr' k
```

SuFunctor.Adequacy.FinMA.**demonicLaws\_rank\_irrel** lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 207

```
theorem FinMA.demonicLaws_rank_irrel (M : FinMA S I A) {r r' : I → ℕ}
(hb : M.BranchZenoFree r) (hb' : M.BranchZenoFree r') (k : ℕ) :
M.demonicLaws r hb k = M.demonicLaws r' hb' k
```

SuFunctor.Adequacy.ma\_demonic\_maxSlice lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 326

```
theorem ma_demonic_maxSlice (M : FinMA S I A) {r : I → ℕ}
  (hb : M.BranchZenoFree r) (k : ℕ) (v : SubPMF (Traj S k)) :
  maxSlice (M.demonicCredal r hb k) v ↔ M.demonicLaws r hb k v
```

SuFunctor.Adequacy.satCredal\_lower\_maDemonicCredal lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 349

```
theorem satCredal_lower_maDemonicCredal (M : FinMA S I A) {r : I → ℕ}
  (hb : M.BranchZenoFree r) (k : ℕ) (c : SubPMF (Traj S k) → Prop) :
  SatCredal .lower (M.demonicCredal r hb k) c ↔
  ∀ σ : Policy I A, c (M.pathLaw σ r (hb.zenoFree σ) k)
```

SuFunctor.Adequacy.satCredal\_upper\_maDemonicCredal lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 359

```
theorem satCredal_upper_maDemonicCredal (M : FinMA S I A) {r : I → ℕ}
  (hb : M.BranchZenoFree r) (k : ℕ) {c : SubPMF (Traj S k) → Prop}
  (hc : DownClosed c) :
  SatCredal .upper (M.demonicCredal r hb k) c ↔
  ∀ σ : Policy I A, c (M.pathLaw σ r (hb.zenoFree σ) k)
```

SuFunctor.Adequacy.Policy.nonempty\_of\_isEmpty lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 406

```
theorem Policy.nonempty_of_isEmpty {X : Type} [IsEmpty X] :
  Nonempty (Policy X A)
```

SuFunctor.Adequacy.Policy.isEmpty\_iff lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 418

```
theorem Policy.isEmpty_iff {X : Type} [Nonempty X] [DecidableEq A] :
  IsEmpty (Policy X A) ↔ IsEmpty A
```

SuFunctor.Adequacy.not\_satCredal\_upper\_maDemonicCredal\_geC lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 438

```
theorem not_satCredal_upper_maDemonicCredal_geC (M : FinMA S I A)
  {r : I → ℕ} (hb : M.BranchZenoFree r) (k : ℕ)
  [hp : Nonempty (Policy I A)] {E : Set (Traj S k)} {q : ℚ}
  (hq : 0 < q) :
  ¬ SatCredal .upper (M.demonicCredal r hb k) (geC E q)
```

SuFunctor.Adequacy.FinMA.nonempty\_markovian lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 474

```
theorem FinMA.nonempty_markovian (M : FinMA S I A) [Nonempty I]
  [Nonempty A] {r : I → ℕ} (hb : M.BranchZenoFree r) : Nonempty S
```

SuFunctor.Adequacy.FinMA.CertScheduler lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 504

```
structure FinMA.CertScheduler (M : FinMA S I A) where ...
```

SuFunctor.Adequacy.FinMA.demonicLawsCert lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 515

```
def FinMA.demonicLawsCert (M : FinMA S I A) (k : ℕ) :
  SubPMF (Traj S k) → Prop
```

SuFunctor.Adequacy.FinMA.demonicLawsCert\_iff lean/su-functor/SuFunctor/Adequacy/MaDemonic.lean, line 531

```
theorem FinMA.demonicLawsCert_iff (M : FinMA S I A) {r : I → ℕ}
  (hb : M.BranchZenoFree r) (k : ℕ) (v : SubPMF (Traj S k)) :
  M.demonicLawsCert k v ↔ M.demonicLaws r hb k v
```

SuFunctor.Adequacy.Bag.sortTuple\_apply\_eq\_inf\_sup lean/su-functor/SuFunctor/Adequacy/Bag.lean, line 128

```

theorem sortTuple_apply_eq_inf_sup (x : Fin n → B) (k : Fin n) :
  sortTuple x k =
    ((univ : Finset (Fin n)).powersetCard (k.val + 1)).attach.inf'
      ((powersetCard_succ_nonempty k).attach)
      (fun S => S.l.sup' (powersetCard_mem_nonempty S.2) x)

```

SuFunctor.Adequacy.Bag.measurable\_symSec

lean/su-functor/SuFunctor/Adequacy/Bag.lean, line 377

```

theorem measurable_symSec
  (hle : MeasurableSet {p : B × B | p.1 ≤ p.2}) :
  Measurable (symSec : Sym B n → (Fin n → B))

```

SuFunctor.Adequacy.Bag.monotone\_eq\_of\_toSym\_eq

lean/su-functor/SuFunctor/Adequacy/Bag.lean, line 335

```

theorem monotone_eq_of_toSym_eq {x y : Fin n → B}
  (hx : Monotone x) (hy : Monotone y) (h : toSym x = toSym y) :
  x = y

```

SuFunctor.Adequacy.Bag.quotKernel\_comp\_eq

lean/su-functor/SuFunctor/Adequacy/Bag.lean, line 411

```

theorem quotKernel_comp_eq (L : Kernel α T) (hs : Measurable s)
  (hL : ∀ x, L (s (q x)) = L x) (x : α) :
  quotKernel L hs (q x) = L x

```

SuFunctor.Adequacy.Bag.iter\_intertwine

lean/su-functor/SuFunctor/Adequacy/Bag.lean, line 451

```

theorem iter_intertwine (hq : Measurable q) (hs : Measurable s)
  (K : Kernel α α)
  (hK : ∀ x, (K.map q) (s (q x)) = (K.map q) x) (k : ℕ) :
  Kernel.deterministic q hq ∘k iterK K k =
    iterK (quotKernel (K.map q) hs) k ∘k Kernel.deterministic q hq

```

SuFunctor.Adequacy.Bag.comap\_section\_congr

lean/su-functor/SuFunctor/Adequacy/Bag.lean, line 419

```

theorem comap_section_congr (L : Kernel α T)
  {s' : α → α} (hs : Measurable s) (hs' : Measurable s')
  (hsec' : ∀ m, q (s' m) = m)
  (hL : ∀ x, L (s (q x)) = L x) :
  L.comap s' hs' = L.comap s hs

```

SuFunctor.Adequacy.Bag.bag\_transport

lean/su-functor/SuFunctor/Adequacy/Bag.lean, line 598

```

theorem bag_transport [LinearOrder B] [MeasurableSpace B]
  (hle : MeasurableSet {p : B × B | p.1 ≤ p.2})
  (K : Kernel (BagPoints B) (BagPoints B))
  (hK : ∀ (n : ℕ) (x : Fin n → B) (σ : Equiv.Perm (Fin n)),
    (K.map toSymSigma) {n, x ∘ σ} = (K.map toSymSigma) {n, x})
  (k : ℕ) (p : BagPoints B) :
  (iterK K k p).map toSymSigma =
    iterK (quotKernel (K.map toSymSigma) (measurable_symSecSigma hle)) k
      (toSymSigma p)

```

SuFunctor.Adequacy.Bag.invariance\_iff\_perm

lean/su-functor/SuFunctor/Adequacy/Bag.lean, line 572

```

theorem invariance_iff_perm [LinearOrder B] [MeasurableSpace B]
  {T : Type*} [MeasurableSpace T]
  (L : Kernel (BagPoints B) T) :
  (∀ p : BagPoints B, L (symSecSigma (toSymSigma p)) = L p) ↔
    (∀ (n : ℕ) (x : Fin n → B) (σ : Equiv.Perm (Fin n)),
      L {n, x ∘ σ} = L {n, x})

```

SuFunctor.Adequacy.Bag.measurableSet\_le\_lex

lean/su-functor/SuFunctor/Adequacy/Bag.lean, line 627



```

theorem measurableSet_le_lex (d : ℕ) :
  MeasurableSet {q : (Fin d → ℝ) × (Fin d → ℝ) |
    toLex q.1 ≤ toLex q.2}

```

SuFunctor.Adequacy.FinCISPN.**not\_branchZenoFree\_of\_disabled** lean/su-functor/SuFunctor/Adequacy/  
StochasticPetriColouredDemonic.lean, line 121

```

theorem not_branchZenoFree_of_disabled {i : N.Vanishing} {a : N.IAct}
  (h : ¬ N.IEnabled a i.1) (r : N.Vanishing → ℕ) :
  ¬ N.toFinMA.BranchZenoFree r

```

SuFunctor.Adequacy.FinCISPN.**RespScheduler** lean/su-functor/SuFunctor/Adequacy/StochasticPetriColouredDemonic.  
lean, line 139

```

def RespScheduler : Type

```

SuFunctor.Adequacy.SpnDemonicControls.**arena\_plural** lean/su-functor/SuFunctor/Adequacy/StochasticPetriDemonic.  
lean, line 532

```

theorem arena_plural :
  (net1.uniformResp : net1.RespScheduler).1.choice i10 1
  ≠ discardResp.1.choice i10 1

```

SuFunctor.Adequacy.FinCISPN.**RespectingRows** lean/su-functor/SuFunctor/Adequacy/StochasticPetriColouredDemonic.  
lean, line 213

```

structure RespectingRows where ...

```

SuFunctor.Adequacy.FinISPN.**ispn\_demonic\_adequacy** lean/su-functor/SuFunctor/Adequacy/StochasticPetriDemonic.  
lean, line 295

```

theorem ispn_demonic_adequacy {r : N.Vanishing → ℕ}
  (hr : N.CascadeRank r) (k : ℕ) {a : Crd (Traj N.Tangible k)} :
  ∃ α : Cell (N.ispnDemonicC (absFuel r) k : Loose a a)
    (N.demonicTangibleLaws hr k) (1 a) (1 a),
  Cell.IsInvertible α

```

SuFunctor.Adequacy.FinISPN.**ispnDemonicC\_iff** lean/su-functor/SuFunctor/Adequacy/StochasticPetriDemonic.lean, line  
250

```

theorem ispnDemonicC_iff {r : N.Vanishing → ℕ} (hr : N.CascadeRank r)
  (k : ℕ) (μ : SubPMF (Traj N.Tangible k)) :
  N.ispnDemonicC (absFuel r) k μ ↔ N.demonicTangibleLaws hr k μ

```

SuFunctor.Adequacy.FinCISPN.**demonicTangibleLaws\_rank\_irrel** lean/su-functor/SuFunctor/Adequacy/  
StochasticPetriColouredDemonic.lean, line 177

```

theorem demonicTangibleLaws_rank_irrel {r r' : N.Vanishing → ℕ}
  (hr : N.CascadeRank r) (hr' : N.CascadeRank r') (k : ℕ) :
  N.demonicTangibleLaws hr k = N.demonicTangibleLaws hr' k

```

SuFunctor.Adequacy.FinCISPN.**demonicTangibleCredal\_rank\_irrel** lean/su-functor/SuFunctor/Adequacy/  
StochasticPetriColouredDemonic.lean, line 191

```

theorem demonicTangibleCredal_rank_irrel {r r' : N.Vanishing → ℕ}
  (hr : N.CascadeRank r) (hr' : N.CascadeRank r') (k : ℕ) :
  N.demonicTangibleCredal hr k = N.demonicTangibleCredal hr' k

```

SuFunctor.Adequacy.FinISPN.**ispn\_demonic\_maxSlice** lean/su-functor/SuFunctor/Adequacy/StochasticPetriDemonic.  
lean, line 321

```

theorem ispn_demonic_maxSlice {r : N.Vanishing → ℕ}
  (hr : N.CascadeRank r) (k : ℕ) (v : SubPMF (Traj N.Tangible k)) :
  maxSlice (N.demonicTangibleCredal hr k) v ↔
  N.demonicTangibleLaws hr k v

```

SuFunctor.Adequacy.FinISPN.**satCredal\_lower\_ispnDemonicCredal**

lean/su-functor/SuFunctor/Adequacy/

StochasticPetriDemonic.lean, line 346

```

theorem satCredal_lower_ispnDemonicCredal {r : N.Vanishing → ℕ}
  (hr : N.CascadeRank r) (k : ℕ)
  (c : SubPMF (Traj N.Tangible k) → Prop) :
  SatCredal .lower (N.demonicTangibleCredal hr k) c ↔
    ∀ σ : N.RespScheduler,
      c (N.tangiblePathLaw σ.1 r (N.respZenoFree hr σ) k)

```

SuFunctor.Adequacy.FinISPN.**su\_ispn\_demonic\_adequacy** lean/su-functor/SuFunctor/Adequacy/StochasticPetriDemonic.lean, line 412

```

theorem su_ispn_demonic_adequacy {r : N.Vanishing → ℕ}
  (hr : N.CascadeRank r) (k : ℕ) (μ : SubPMF (Traj N.Tangible k)) :
  idealPoint μ ∈ (N.ispnDemonicDnf (absFuel r) k).modelClass
    ↔ N.demonicTangibleLaws hr k μ

```

SuFunctor.Adequacy.FinISPN.**chainFire\_val** lean/su-functor/SuFunctor/Adequacy/StochasticPetriSetFiring.lean, line 196

```

theorem chainFire_val : ∀ (l : List U) (m : BMarking P B),
  N.ChainEnabled l m → ∀ p : P,
  (N.chainFire l m p : ℕ) + N.preSum ↑l p
    = (m p : ℕ) + N.postSum ↑l p
| [], m, _, p => by simp
| u :: l, m, h, p => by
  obtain ⟨hu, hl⟩

```

SuFunctor.Adequacy.FinISPN.**chainFire\_eq\_of\_enabled**

lean/su-functor/SuFunctor/Adequacy/

StochasticPetriSetFiring.lean, line 213

```

theorem chainFire_eq_of_enabled {l1 l2 : List U} {m : BMarking P B}
  (h1 : N.ChainEnabled l1 m) (h2 : N.ChainEnabled l2 m)
  (hml : (↑l1 : Multiset U) = ↑l2) :
  N.chainFire l1 m = N.chainFire l2 m

```

SuFunctor.Adequacy.FinISPN.**chainFire\_eq\_compositeFire**

lean/su-functor/SuFunctor/Adequacy/

StochasticPetriSetFiring.lean, line 262

```

theorem chainFire_eq_compositeFire {l : List U} {m : BMarking P B}
  (h : N.ChainEnabled l m) (hsup : N.Supplied ↑l m) :
  N.chainFire l m = N.compositeFire ↑l m

```

SuFunctor.Adequacy.FinISPN.**JointReservation** lean/su-functor/SuFunctor/Adequacy/StochasticPetriSetFiring.lean, line 296

```

def JointReservation (S : Multiset U) (m : BMarking P B) : Prop

```

SuFunctor.Adequacy.FinISPN.**setNonConflicting\_of\_headroom**

lean/su-functor/SuFunctor/Adequacy/

StochasticPetriSetFiring.lean, line 377

```

theorem setNonConflicting_of_headroom {S : Multiset U} {m : BMarking P B}
  (hJ : N.JointReservation S m)
  (hcap : ∀ p, (m p : ℕ) + N.postSum S p ≤ B) :
  N.SetNonConflicting S m

```

SuFunctor.Adequacy.FinISPN.**rlr3\_composite\_adequacy**

lean/su-functor/SuFunctor/Adequacy/

StochasticPetriSetFiring.lean, line 391

```

theorem rlr3_composite_adequacy {S : Multiset U} {m : BMarking P B}
  (hJ : N.JointReservation S m)
  (hcap : ∀ p, (m p : ℕ) + N.postSum S p ≤ B) :
  ∀ l : List U, (↑l : Multiset U) = S →
    N.ChainEnabled l m ∧ N.chainFire l m = N.compositeFire S m

```

SuFunctor.Adequacy.FinCISPn.**flatten\_triangle\_of\_agree**

lean/su-functor/SuFunctor/Adequacy/

StochasticPetriColouredComposed.lean, line 343

```

theorem flatten_triangle_of_agree (σ : Policy N.Vanishing N.IAct)
  (r : N.Vanishing → ℕ) (hr : ZenoFree (N.toFinMA.imStep σ) r)
  {Mode : Type} [Fintype Mode] (e : Mode ≈ N.Tangible)
  (flat : FinCTMC Mode)
  (hoff : ∀ s s' : N.Tangible, s ≠ s' →
    (flat.relabel e).rate s s'
    = (N.toFinMA.close σ r hr).rate s s')
  (hinit : (flat.relabel e).init = (N.toFinMA.close σ r hr).init)
  (k : ℕ) {a : Crd (Traj N.Tangible k)} :
  ∃ α : Cell
    (looseConj (atoms (jumpRows
      (closeMaRows N.compileCispn (compilePolicy σ) (absFuel r))) k) :
      Loose a a)
    (looseConj (atoms (jumpRows (compileCtmc (flat.relabel e))) k))
    (⊤ a) (⊤ a),
    Cell.IsInvertible α

```

SuFunctor.**SatAngelic**

lean/su-functor/SuFunctor/Adequacy/StochasticPetriAngelic.lean, line 121

```

def SatAngelic (C c : SubPMF V → Prop) : Prop

```

SuFunctor.Adequacy.FinISPn.**satAngelic\_ispnDemonicCredal**

lean/su-functor/SuFunctor/Adequacy/

StochasticPetriAngelic.lean, line 196

```

theorem satAngelic_ispnDemonicCredal {r : N.Vanishing → ℕ}
  (hr : N.CascadeRank r) (k : ℕ)
  (c : SubPMF (Traj N.Tangible k) → Prop) :
  SatAngelic (N.demonicTangibleCredal hr k) c ↔
  ∃ σ : N.RespScheduler,
    c (N.tangiblePathLaw σ.1 r (N.respZenoFree hr σ) k)

```

SuFunctor.Adequacy.FinISPn.**ispn\_demonic\_verdict\_to\_member**

lean/su-functor/SuFunctor/Adequacy/

StochasticPetriAngelic.lean, line 220

```

theorem ispn_demonic_verdict_to_member {r : N.Vanishing → ℕ}
  (hr : N.CascadeRank r) (σ : N.RespScheduler) (k : ℕ)
  {c : SubPMF (Traj N.Tangible k) → Prop}
  (h : SatCredal .lower (N.demonicTangibleCredal hr k) c) :
  c (N.tangiblePathLaw σ.1 r (N.respZenoFree hr σ) k)

```

SuFunctor.Adequacy.FinISPn.**flatten\_angelic\_of\_agree** lean/su-functor/SuFunctor/Adequacy/StochasticPetriAngelic.  
lean, line 264

```

theorem flatten_angelic_of_agree {r : N.Vanishing → ℕ}
  (hrk : N.CascadeRank r) (σ : N.RespScheduler)
  {Mode : Type} [Fintype Mode] (e : Mode ≈ N.Tangible)
  (flat : FinCTMC Mode)
  (hoff : ∀ s s' : N.Tangible, s ≠ s' →
    (flat.relabel e).rate s s'
    = (N.toFinMA.close σ.1 r (N.respZenoFree hrk σ)).rate s s')
  (hinit : (flat.relabel e).init
    = (N.toFinMA.close σ.1 r (N.respZenoFree hrk σ)).init)
  (k : ℕ) {c : SubPMF (Traj N.Tangible k) → Prop}
  (hc : c ((flat.relabel e).jumpLaw k)) :
  SatAngelic (N.demonicTangibleCredal hrk k) c

```

SuFunctor.Adequacy.PetriVanilla.**face\_eq\_reach**

lean/su-functor/SuFunctor/Adequacy/PetriVanilla.lean, line 409

```

theorem face_eq_reach (hd : ∀ t, 0 < d t) :
  face N d m0 = {m | Reach N m0 m}

```

SuFunctor.Adequacy.PetriVanilla.**face\_dwell\_uniform** lean/su-functor/SuFunctor/Adequacy/PetriVanilla.lean, line 418

```
theorem face_dwell_uniform {d' : T → ℚ} (hd : ∀ t, 0 < d t)
  (hd' : ∀ t, 0 < d' t) : face N d m0 = face N d' m0
```

SuFunctor.Adequacy.PetriVanilla.**su\_vanilla\_demonic\_adequacy** lean/su-functor/SuFunctor/Adequacy/PetriVanilla.lean, line 570

```
theorem su_vanilla_demonic_adequacy (k : ℕ)
  (μ : SubPMF (Traj (Marking P) k)) :
  idealPoint μ ∈ (vanillaDemonicDnf N d m0 k).modelClass
  ↔ vanillaDemonicLaws N d m0 k μ
```

SuFunctor.Adequacy.PetriVanilla.**vanillaDemonicC\_iff** lean/su-functor/SuFunctor/Adequacy/PetriVanilla.lean, line 515

```
theorem vanillaDemonicC_iff (k : ℕ) (μ : SubPMF (Traj (Marking P) k)) :
  vanillaDemonicC N d m0 k μ ↔ vanillaDemonicLaws N d m0 k μ
```

SuFunctor.Adequacy.PetriVanilla.**recordVisited\_eq\_face** lean/su-functor/SuFunctor/Adequacy/PetriVanilla.lean, line 622

```
theorem recordVisited_eq_face :
  recordVisited N d m0 = face N d m0
```

SuFunctor.Adequacy.PetriVanilla.**su\_vanilla\_reach\_iff** lean/su-functor/SuFunctor/Adequacy/PetriVanilla.lean, line 650

```
theorem su_vanilla_reach_iff (hd : ∀ t, 0 < d t) (m : Marking P) :
  (∃ (k : ℕ) (μ : SubPMF (Traj (Marking P) k)),
    idealPoint μ ∈ (vanillaDemonicDnf N d m0 k).modelClass
    ∧ ∃ ω, μ.wt ω ≠ 0 ∧ ∃ i, ω i = m)
  ↔ Reach N m0 m
```

SuFunctor.Adequacy.PetriVanilla.**su\_vanilla\_dwell\_uniform** lean/su-functor/SuFunctor/Adequacy/PetriVanilla.lean, line 671

```
theorem su_vanilla_dwell_uniform {d' : T → ℚ} (hd : ∀ t, 0 < d t)
  (hd' : ∀ t, 0 < d' t) :
  recordVisited N d m0 = recordVisited N d' m0
```

SuFunctor.Adequacy.PetriVanilla.**su\_vanilla\_safety** lean/su-functor/SuFunctor/Adequacy/PetriVanilla.lean, line 681

```
theorem su_vanilla_safety (hd : ∀ t, 0 < d t) (Φ : Marking P → Prop) :
  (∀ m ∈ recordVisited N d m0, Φ m) ↔
  ∀ m, Reach N m0 m → Φ m
```

SuFunctor.Adequacy.Pgm.**pgm\_adequacy** lean/su-functor/SuFunctor/Adequacy/Pgm.lean, line 727

```
theorem pgm_adequacy (B : FinBN n Val) {a : Crd (Asn Val)} :
  ∃ α : Cell (looseConj (atoms (compile B))) : Loose a a)
  (singletonC B.jointLaw) (1 a) (1 a),
  Cell.IsInvertible α
```

SuFunctor.Adequacy.Pgm.FinBN.**jointLaw\_mass** lean/su-functor/SuFunctor/Adequacy/Pgm.lean, line 374

```
theorem FinBN.jointLaw_mass (B : FinBN n Val) : (B.jointLaw).mass = 1
```

SuFunctor.Adequacy.Pgm.FinBN.**ancestralPMF\_eq** lean/su-functor/SuFunctor/Adequacy/Pgm.lean, line 921

```
theorem FinBN.ancestralPMF_eq (σ : Asn Val) :
  (B.ancestralPMF) σ = q2e ((B.jointLaw).wt σ)
```

SuFunctor.Adequacy.**noiseDenot\_progTraj** lean/su-functor/SuFunctor/Adequacy/ProbProg.lean, line 200

```

theorem noiseDenot_progTraj (P : ProbProgram S N0 N) :
  ∀ k, noiseDenot (progTraj P k) (measurable_progTraj P k) = pathDist P k
| 0 => by
  calc noiseDenot (progTraj P 0) (measurable_progTraj P 0)
    = noiseDenot (trajZero ∘ P.start)
      (measurable_trajZero.comp P.measurable_start)

```

SuFunctor.Adequacy.**su\_pprog\_adequacy** lean/su-functor/SuFunctor/Adequacy/ProbProg.lean, line 398

```

theorem su_pprog_adequacy {P : ProbProgram S N0 N} {M : FinDTMC S}
  (hPM : Presents P M) (k : N) (μ : SubPMF (Traj S k)) :
  idealPoint μ ∈ (pprogDnf (compile M) k).modelClass
  ↔ μ.probFun = ((noiseDenot (progTraj P k) (measurable_progTraj P k)) :
    ProbState (Traj S k)) : Traj S k → ℝ

```

SuFunctor.Adequacy.QNet.**quantum\_net\_adequacy** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 426

```

theorem quantum_net_adequacy (P : QNetSystem X n ι T κ) {a : Crd κ} :
  ∃ α : Cell (looseConj (QNet.atoms (QNet.compile P))) : Loose a a)
  (singletonC P.shadowLaw) (1 a) (1 a),
  Cell.IsInvertible α

```

SuFunctor.Adequacy.QNetSystem lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 156

```

structure QNetSystem (X : Type*) [PartialOrder X]
  (n : Type*) [Fintype n] [DecidableEq n]
  (ι T κ : Type*) [Fintype κ] where ...

```

SuFunctor.Adequacy.QNetSystem.**shadowPMF\_prob\_eq\_born** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 204

```

theorem shadowPMF_prob_eq_born (α : κ) :
  ((P.shadowPMF.prob α : ℚ) : ℝ) =
  ((P.state * P.frag.records α).trace).re

```

SuFunctor.Adequacy.QNetSystem.**shadowPMF\_eq\_fragment\_shadow** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 213

```

theorem shadowPMF_eq_fragment_shadow (α : κ) :
  ((P.shadowPMF.prob α : ℚ) : ℝ) =
  P.frag.shadow
  (stateClassMk P.frag.net P.frag.ambient
    (P.state, P.state_isState)) α

```

SuFunctor.Adequacy.**shadowPMF\_class\_invariant** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 227

```

theorem shadowPMF_class_invariant (F : LCQFTFragment X n ι T κ)
  {p σ : Matrix n n ℂ} (hp : IsState p) (hσ : IsState σ)
  {q1 q2 : κ → ℚ}
  (h1 : ∀ α, ((q1 α : ℚ) : ℝ) = recordShadow F.records p α)
  (h2 : ∀ α, ((q2 α : ℚ) : ℝ) = recordShadow F.records σ α)
  (hcls : stateClassMk F.net F.ambient (p, hp) =
    stateClassMk F.net F.ambient (σ, hσ)) :
  (QNetSystem.mk F p hp q1 h1).shadowPMF =
  (QNetSystem.mk F σ hσ q2 h2).shadowPMF

```

SuFunctor.Adequacy.QNetSystem.**shadowLaw\_mass** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 198

```

theorem shadowLaw_mass : P.shadowLaw.mass = 1

```

SuFunctor.Adequacy.QNet.**rowsLaw\_mass\_of\_complete** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 313

```

theorem rowsLaw_mass_of_complete (R : QNetRows κ) (hR : R.WF)
  (hC : R.Complete) : (rowsLaw R hR).mass = 1

```

SuFunctor.Adequacy.QNet.**bellQNet\_shadowPMF**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 714  
 | theorem bellQNet\_shadowPMF : bellQNet.shadowPMF = bellShadowPMF

SuFunctor.Adequacy.QNet.**mixedQNet\_shadowPMF**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 718  
 | theorem mixedQNet\_shadowPMF : mixedQNet.shadowPMF = mixedShadowPMF

SuFunctor.Adequacy.QNet.**exists\_incomplete\_mass\_lt\_one**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 726

| theorem exists\_incomplete\_mass\_lt\_one :  
 |     $\exists (P : \text{QNetSystem } (\text{Mink } \mathbb{R}) \text{ (Fin 2} \times \text{Fin 2) (Fin 2) } \mathbb{R}$   
 |        $(\text{Fin 2} \times \text{Fin 2})) (J : \text{Finset (Fin 2} \times \text{Fin 2)}),$   
 |        $(\text{QNet.rowsLaw (QNet.compileSub P J)}$   
 |        $(\text{QNet.compileSub\_wf P J})).\text{mass} < 1$

SuFunctor.Adequacy.QNet.**exists\_nonPSD\_family\_negative\_shadow**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 822

| theorem exists\_nonPSD\_family\_negative\_shadow :  
 |     $\exists (R : \text{Fin 2} \rightarrow \text{Matrix (Fin 1) (Fin 1) } \mathbb{C})$   
 |        $(\rho : \text{Matrix (Fin 1) (Fin 1) } \mathbb{C}),$   
 |        $\text{IsState } \rho \wedge (\sum \alpha, R \alpha = 1) \wedge \exists \alpha, \text{recordShadow } R \rho \alpha < 0$

SuFunctor.Adequacy.QNet.**su\_qnet\_adequacy**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 895

| theorem su\_qnet\_adequacy (P : QNetSystem X n ι T κ) (μ : SubPMF κ) :  
 |    idealPoint (μ.map recState)  
 |    ∈ (qnetSpecW (QNet.compile P)).modelClass  
 |    ↔ μ = P.shadowLaw

SuFunctor.Adequacy.QNet.**quantum\_credal\_adequacy**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 546

| theorem quantum\_credal\_adequacy (F : LCQFTFragment X n ι T κ)  
 |    (S : Set {p : Matrix n n ℂ // IsState p}) {a : Crd κ} :  
 |     $\exists \alpha : \text{Cell (QNet.qnetCredalC F S : Loose a a)}$   
 |        $(\text{QNet.quantumLaws F S}) (\mathbb{1} a) (\mathbb{1} a),$   
 |        $\text{Cell.IsInvertible } \alpha$

SuFunctor.Adequacy.QNet.QNet.**qnetCredalC**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 514

| def qnetCredalC (F : LCQFTFragment X n ι T κ)  
 |    (S : Set {p : Matrix n n ℂ // IsState p}) : SubPMF κ → Prop

SuFunctor.Adequacy.QNet.QNet.**qnetCredalC\_iff**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 522

| theorem qnetCredalC\_iff (F : LCQFTFragment X n ι T κ)  
 |    (S : Set {p : Matrix n n ℂ // IsState p}) (μ : SubPMF κ) :  
 |     $\text{qnetCredalC F S } \mu \leftrightarrow \text{quantumLaws F S } \mu$

SuFunctor.Adequacy.QNet.QNet.**RatShadowMember**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 442

| structure RatShadowMember (F : LCQFTFragment X n ι T κ)  
 |    (S : Set {p : Matrix n n ℂ // IsState p}) where ...

SuFunctor.Adequacy.QNet.**quantum\_maxSlice**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 556

| theorem quantum\_maxSlice (F : LCQFTFragment X n ι T κ)  
 |    (S : Set {p : Matrix n n ℂ // IsState p}) (ν : SubPMF κ) :  
 |     $\text{maxSlice (QNet.quantumCredal F S)} \nu \leftrightarrow \text{QNet.quantumLaws F S } \nu$

SuFunctor.Adequacy.QNet.**quantum\_ray\_singleton\_reconciliation**      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 562

```

theorem quantum_ray_singleton_reconciliation
  (F : LCQFTFragment X n ι T κ)
  (S : Set {p : Matrix n n ℂ // IsState p}) {a : Crd κ} :
  ∃ α : Cell (maxSlice (QNet.quantumCredal F S) : Loose a a)
    (QNet.quantumLaws F S) (1 a) (1 a),
    Cell.IsInvertible α

```

SuFunctor.Adequacy.QNet.**satCredal\_lower\_quantumCredal** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 573

```

theorem satCredal_lower_quantumCredal (F : LCQFTFragment X n ι T κ)
  (S : Set {p : Matrix n n ℂ // IsState p})
  (c : SubPMF κ → Prop) :
  SatCredal .lower (QNet.quantumCredal F S) c ↔
    ∀ w : QNet.RatShadowMember F S, c w.law

```

SuFunctor.Adequacy.QNet.**satCredal\_upper\_quantumCredal** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 582

```

theorem satCredal_upper_quantumCredal (F : LCQFTFragment X n ι T κ)
  (S : Set {p : Matrix n n ℂ // IsState p})
  {c : SubPMF κ → Prop} (hc : DownClosed c) :
  SatCredal .upper (QNet.quantumCredal F S) c ↔
    ∀ w : QNet.RatShadowMember F S, c w.law

```

SuFunctor.Adequacy.QNet.QNet.**quantumLaws\_mix** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 660

```

theorem quantumLaws_mix (hS : MixClosed S)
  (w₁ w₂ : RatShadowMember F S) {t : ℚ} (h₀ : 0 ≤ t) (h₁ : t ≤ 1) :
  quantumLaws F S (FinPMF.mix w₁.pmf w₂.pmf t h₀ h₁).toSubPMF

```

Qtsl.QuantumStates.**recordShadowPMF\_mix** lean/qtsl-aux/QtslAux/QuantumStates.lean, line 678

```

theorem recordShadowPMF_mix [Fintype ι] (R : ι → Matrix n n ℂ)
  (p σ : Matrix n n ℂ) (q₁ q₂ : ι → ℚ)
  (hq₁ : ∀ α, (q₁ α : ℝ) = recordShadow R p α)
  (hq₂ : ∀ α, (q₂ α : ℝ) = recordShadow R σ α)
  (hR : ∀ α, (R α).PosSemidef) (hexh : ∑ α, R α = 1)
  (hp : IsState p) (hσ : IsState σ) {t : ℚ} (h₀ : 0 ≤ t) (h₁ : t ≤ 1) :
  recordShadowPMF R ((t : ℚ) : ℂ) • p + ((1 - t : ℚ) : ℂ) • σ
    (fun α => t * q₁ α + (1 - t) * q₂ α)
    (fun α => recordShadow_mix_rat_witness hq₁ hq₂ α)
    hR hexh (isState_mix hp hσ h₀ h₁) =
  FinPMF.mix (recordShadowPMF R p q₁ hq₁ hR hexh hp)
    (recordShadowPMF R σ q₂ hq₂ hR hexh hσ) t h₀ h₁

```

SuFunctor.Adequacy.QNet.**mixLaw\_not\_mem\_pair** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 775

```

theorem mixLaw_not_mem_pair :
  ¬ QNet.quantumLaws isingQubitFragment bellMixedSet
    (FinPMF.mix bellMember.pmf mixedMember.pmf (1 / 2)
      (by norm_num) (by norm_num)).toSubPMF

```

SuFunctor.Adequacy.QNet.**not\_satCredal\_upper\_quantumCredal\_geC** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 609

```

theorem not_satCredal_upper_quantumCredal_geC
  (F : LCQFTFragment X n ι T κ)
  (S : Set {p : Matrix n n ℂ // IsState p})
  [hne : Nonempty (QNet.RatShadowMember F S)]
  {E : Set κ} {q : ℚ} (hq : 0 < q) :
  ¬ SatCredal .upper (QNet.quantumCredal F S) (geC E q)

```



SuFunctor.Adequacy.QNet.**satCredal\_quantum\_of\_isEmpty** lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 625

```
theorem satCredal_quantum_of_isEmpty (F : LCQFTFragment X n ι T κ)
  (S : Set {p : Matrix n n ℂ // IsState p})
  [h : IsEmpty (QNet.RatShadowMember F S)] (kind : EnvelopeKind)
  (c : SubPMF κ → Prop) :
  SatCredal kind (QNet.quantumCredal F S) c
```

Qtsl.QuantumStates.**existsUnique\_isState\_traceRepr** lean/qtsl-aux/QtslAux/QuantumStates.lean, line 518

```
theorem existsUnique_isState_traceRepr
  (hpos : ∀ v : n → ℂ, 0 ≤ φ (Matrix.vecMulVec v (star v)))
  (hone : φ 1 = 1) :
  ∃! p : Matrix n n ℂ, IsState p ∧ ∀ a, φ a = (p * a).trace
```

Qtsl.QuantumStates.**exists\_shadow\_lowerE\_lt\_upperE** lean/qtsl-aux/QtslAux/QuantumStates.lean, line 953

```
theorem exists_shadow_lowerE_lt_upperE :
  ∃ f : Fin 2 × Fin 2 → ℚ,
    lowerE ![bellShadowPMF, mixedShadowPMF] f <
    upperE ![bellShadowPMF, mixedShadowPMF] f
```

Qtsl.QuantumCoarseGraining.**pushforward\_ofPresentation** lean/qtsl-aux/QtslAux/QuantumCoarseGraining.lean, line 474

```
theorem pushforward_ofPresentation (f : ι → κ) (C : ι → Matrix d d ℂ)
  (ψ : d → ℂ)
  (hnorm : ∑ α, ∑ β, decoherenceFunctional C ψ α β = 1) :
  (QuantumMeasure.ofPresentation C ψ hnorm).pushforward f
  = QuantumMeasure.ofPresentation (coarsenFamily f C) ψ
  (by
    simp only [decoherenceFunctional_coarsenFamily]
    have h := sum_sum_fiberEvent f Finset.univ Finset.univ
    fun a b => decoherenceFunctional C ψ a b
    rw [h, fiberEvent_univ]
    exact hnorm)
```

SuFunctor.Adequacy.**tvDistQ** lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 135

```
def tvDistQ {κ : Type u} [Fintype κ] (μ ν : SubPMF κ) : ℚ
```

SuFunctor.Adequacy.**tvDistQ\_eq\_zero\_iff** lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 166

```
theorem tvDistQ_eq_zero_iff {κ : Type u} [Fintype κ] {μ ν : SubPMF κ} :
  tvDistQ μ ν = 0 ↔ μ = ν
```

SuFunctor.Adequacy.**credalInflate** lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 214

```
def credalInflate {κ : Type u} [Fintype κ] (ε : ℚ)
  (L : SubPMF κ → Prop) : SubPMF κ → Prop
```

SuFunctor.Adequacy.Containment.**mono** lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 282

```
theorem Containment.mono {ι κ : Type u} [Fintype ι] [Fintype κ]
  {LUV : SubPMF ι → Prop} {f : ι → κ} {LIR : SubPMF κ → Prop}
  {ε ε' : ℚ} (h : ε ≤ ε')
  (hc : Containment LUV f LIR ε) : Containment LUV f LIR ε'
```

SuFunctor.Adequacy.Containment.**comp** lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 292

```
theorem Containment.comp {ι κ σ : Type u}
  [Fintype ι] [Fintype κ] [Fintype σ]
  {L₁ : SubPMF ι → Prop} {f : ι → κ} {L₂ : SubPMF κ → Prop} {ε₁ : ℚ}
  {g : κ → σ} {L₃ : SubPMF σ → Prop} {ε₂ : ℚ}
```

```
(h1 : Containment L1 f L2 ε1) (h2 : Containment L2 g L3 ε2) :
  Containment L1 (g ∘ f) L3 (ε1 + ε2)
```

SuFunctor.SubPMF.map\_wt\_apply lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 102

```
theorem map_wt_apply {X Y : Type u} [Fintype X] [DecidableEq Y]
  (f : X → Y) (μ : SubPMF X) (c : Y) :
  (μ.map f).wt c
  = ∑ a ∈ Finset.univ.filter (fun a => f a = c), μ.wt a
```

SuFunctor.Adequacy.tvDistQ\_map\_le lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 193

```
theorem tvDistQ_map_le {X Y : Type u} [Fintype X] [Fintype Y]
  (f : X → Y) (μ v : SubPMF X) :
  tvDistQ (μ.map f) (v.map f) ≤ tvDistQ μ v
```

SuFunctor.Adequacy.recordShadow\_coarsenFamily lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 327

```
theorem recordShadow_coarsenFamily {ι' κ' : Type*} [Fintype ι']
  [Fintype κ'] [DecidableEq ι'] [DecidableEq κ']
  {d : Type*} [Fintype d]
  (f : ι' → κ') (R : ι' → Matrix d d ℂ) (p : Matrix d d ℂ) (c : κ') :
  recordShadow (coarsenFamily f R) p c
  = ∑ a ∈ fiber f c, recordShadow R p a
```

SuFunctor.Adequacy.uv\_ir\_containment lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 763

```
theorem uv_ir_containment :
  Containment (QNet.quantumLaws isingQubitFragment uvSeg) Prod.fst
  (QNet.quantumLaws irFragment irSet) (49 / 625)
```

SuFunctor.Adequacy.uv\_ir\_containment\_lower lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 788

```
theorem uv_ir_containment_lower (ε : ℚ)
  (h : Containment (QNet.quantumLaws isingQubitFragment uvSeg)
    Prod.fst (QNet.quantumLaws irFragment irSet) ε) :
  49 / 625 ≤ ε
```

SuFunctor.Adequacy.not\_uv\_ir\_containment\_zero lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 817

```
theorem not_uv_ir_containment_zero :
  ¬ Containment (QNet.quantumLaws isingQubitFragment uvSeg) Prod.fst
  (QNet.quantumLaws irFragment irSet) 0
```

SuFunctor.Adequacy.onestep\_defect lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 1056

```
theorem onestep_defect (x : Fin 2) :
  tvDistQ (coarseLaw (x + 1)) (irStep (coarseLaw x)) ≤ 49 / 625
```

SuFunctor.Adequacy.horizon\_bound lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 1073

```
theorem horizon_bound (x : Fin 2) (k : ℕ) :
  tvDistQ (coarseLaw ((· + 1)^[k] x)) (irStep^[k] (coarseLaw x))
  ≤ (k : ℚ) * (49 / 625)
```

SuFunctor.Adequacy.lax\_horizon lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 395

```
theorem lax_horizon {A : Type*} {B : Type*}
  (push : A → B) (stepA : A → A) (stepB : B → B)
  (d : B → B → ℚ) (ε : ℚ) (Reach : A → Prop)
  (hself : ∀ x, d x x = 0)
  (htri : ∀ x y z, d x z ≤ d x y + d y z)
  (hnonexp : ∀ x y, d (stepB x) (stepB y) ≤ d x y)
```

```

(hinv : ∀ a, Reach a → Reach (stepA a))
(hdef : ∀ a, Reach a → d (push (stepA a)) (stepB (push a)) ≤ ε) :
∀ a, Reach a → ∀ k : ℕ,
  d (push (stepA^[k] a)) (stepB^[k] (push a)) ≤ (k : ℚ) * ε

```

SuFunctor.Adequacy.**tvDistQ\_stochAct\_le**      lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 368

```

theorem tvDistQ_stochAct_le {κ : Type u} [Fintype κ] (Q : κ → κ → ℚ)
  (hQ0 : ∀ c c', 0 ≤ Q c c') (hQ1 : ∀ c, ∑ c', Q c c' = 1)
  (μ ν : SubPMF κ) :
  tvDistQ (stochAct Q hQ0 hQ1 μ) (stochAct Q hQ0 hQ1 ν)
    ≤ tvDistQ μ ν

```

SuFunctor.Adequacy.**ir\_tot\_containment**      lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 1135

```

theorem ir_tot_containment :
  Containment (QNet.quantumLaws irFragment irSet) totalMerge
    (QNet.quantumLaws totFragment totSet) 0

```

SuFunctor.Adequacy.**uv\_tot\_containment**      lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 1159

```

theorem uv_tot_containment :
  Containment (QNet.quantumLaws isingQubitFragment uvSeg)
    (totalMerge ∘ Prod.fst)
    (QNet.quantumLaws totFragment totSet) (49 / 625 + 0)

```

SuFunctor.Adequacy.**uvSeg\_mixClosed**      lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 570

```

theorem uvSeg_mixClosed : QNet.MixClosed uvSeg

```

SuFunctor.Adequacy.**uvLaws\_mix\_exercised**      lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 620

```

theorem uvLaws_mix_exercised :
  QNet.quantumLaws isingQubitFragment uvSeg
    ((FinPMF.mix upMember.pmf entMember.pmf (1 / 2)
      (by norm_num) (by norm_num)).toSubPMF)

```

SuFunctor.Adequacy.**pushforward\_probRow\_bridge**      lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 1302

```

theorem pushforward_probRow_bridge (c : Fin 2) :
  (uvMeasure.pushforward Prod.fst).probRow c
    = ((entMember.law.map Prod.fst).wt c : ℚ) : ℝ

```

SuFunctor.Adequacy.**eftTower\_offDiagMass\_antitone**      lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 1276

```

theorem eftTower_offDiagMass_antitone :
  Antitone fun n => (eftStage n).offDiagMass

```

SuFunctor.Adequacy.**eftStage\_classical**      lean/su-functor/SuFunctor/Adequacy/QuantumContainment.lean, line 1268

```

theorem eftStage_classical (n : ℕ) :
  ∃ (p : eftTowerX n → ℝ) (hp : ∀ c, 0 ≤ p c) (hs : ∑ c, p c = 1),
    eftStage n = QuantumMeasure.classical p hp hs

```

SuFunctor.Adequacy.**recordShadow\_kchanApply**      lean/su-functor/SuFunctor/Adequacy/QuantumCompression.lean, line 317

```

theorem recordShadow_kchanApply (A : J → Matrix e d ℂ)
  (S : κ' → Matrix e e ℂ) (p : Matrix d d ℂ) (c : κ') :
  recordShadow S (kchanApply A p) c
    = recordShadow (fun c => kchanDual A (S c)) p c

```

SuFunctor.Adequacy.**kchanApply\_ptrKraus\_eq\_ptr**      lean/su-functor/SuFunctor/Adequacy/QuantumCompression.lean, line 450

```
theorem kchanApply_ptrKraus_eq_ptr (p : TwoQubits) :
  kchanApply ptrKraus p = ptr p
```

SuFunctor.Adequacy.uv\_smallIR\_containment      lean/su-functor/SuFunctor/Adequacy/QuantumCompression.lean, line 718

```
theorem uv_smallIR_containment :
  Containment (QNet.quantumLaws isingQubitFragment uvSeg) Prod.fst
    (QNet.quantumLaws singleQubitFragment smallSet) (49 / 625)
```

SuFunctor.Adequacy.QNet.QNetRows      lean/su-functor/SuFunctor/Adequacy/QuantumNet.lean, line 251

```
structure QNetRows (κ : Type*) where ...
```

Modules cited: SuFunctor.Adequacy.QuantumNet, SuFunctor.Adequacy.Ode, SuFunctor.Adequacy.Pde, SuFunctor.Adequacy.PdeRows, QtslAux.SdeGenerator, QtslAux.SdeGeneratorPoly, QtslAux.SdeMartingaleProblem, SuFunctor.Adequacy.Rode, QtslAux.JumpDiffusion, QtslAux.JumpDiffusionMartingaleProblem, SuFunctor.Adequacy.CtmcJ, SuFunctor.Adequacy.MarkovAutomaton, SuFunctor.Adequacy.MaDemonic, SuFunctor.Adequacy.Bag, SuFunctor.TokenGame, SuFunctor.Adequacy.StochasticPetri, SuFunctor.Adequacy.StochasticPetriCascade, SuFunctor.Adequacy.StochasticPetriSetFiring, SuFunctor.Adequacy.StochasticPetriFlatten, SuFunctor.Adequacy.StochasticPetriComposed, QtslAux.KilledTimeIndexed, SuFunctor.Adequacy.StochasticPetriColoured, SuFunctor.Adequacy.StochasticPetriColouredGshsComposed, SuFunctor.Adequacy.StochasticPetriAngelic, SuFunctor.Adequacy.StochasticPetriColoured, SuFunctor.Adequacy.StochasticPetriColouredCascade, SuFunctor.Adequacy.StochasticPetriColouredDemonic, SuFunctor.Adequacy.StochasticPetriKernel, SuFunctor.Adequacy.StochasticPetriColouredKernel, SuFunctor.Adequacy.StochasticPetriCascadeKernel, SuFunctor.Adequacy.StochasticPetriColouredCascadeKernel, SuFunctor.Adequacy.StochasticPetriColouredFlattenKernel, SuFunctor.Adequacy.PetriVanilla, Qtsl.ScmTwin, SuFunctor.Stochastic.NoiseKleisli, SuFunctor.Adequacy.ProbProg, QtslAux.QuantumCoarseGraining, SuFunctor.Adequacy.QuantumContainment.

Also cited in this chapter, listed under an earlier one: QtslAux.QuantumAssembly, QtslAux.QuantumRecords, SuFunctor.Adequacy.DelayLine, SuFunctor.Adequacy.Dtmc, SuFunctor.Adequacy.EnvelopeUnion, SuFunctor.Adequacy.Heat, SuFunctor.Adequacy.HeatTorusInit, SuFunctor.Adequacy.Inclusions, SuFunctor.Adequacy.MarginalGlue, SuFunctor.Adequacy.Mdp, SuFunctor.Adequacy.MdpDemonic, SuFunctor.Adequacy.Pgm, SuFunctor.Adequacy.SolutionSetMesh, copy\_ne\_kleisli, maxSlice\_unionRays, maxSlice\_unionRays\_subset, noiseDenot\_seqComp.

## A.5 Systems: Composition Patterns

Citations from Chapter 13.

SuFunctor.AlignedLoose.seq\_assoc      lean/su-functor/SuFunctor/HypernetComposition.lean, line 219

```
theorem AlignedLoose.seq_assoc (m : AlignedLoose X Y) (n : AlignedLoose Y Z)
  (p : AlignedLoose Z V) : (m.seq n).seq p = m.seq (n.seq p)
```

SuFunctor.austereDnf\_subst      lean/su-functor/SuFunctor/HypernetComposition.lean, line 350

```
theorem austereDnf_subst (m : AlignedLoose X Y) (n : AlignedLoose Y Z)
  {T1 T1' T2 T2' : List PrimApp}
  (h1 : austereDnf ty m T1 = austereDnf ty m T1')
  (h2 : austereDnf ty n T2 = austereDnf ty n T2')
  (hTC1 : ∀ a ∈ T1, a.TypeConsistent (tableOn ty m.apex))
  (hw1 : ∀ a ∈ T1, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ m.apex)
  (hTC2 : ∀ a ∈ T2, a.TypeConsistent (tableOn ty n.apex))
  (hw2 : ∀ a ∈ T2, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ n.apex)
  (hTC1' : ∀ a ∈ T1', a.TypeConsistent (tableOn ty m.apex))
  (hw1' : ∀ a ∈ T1', ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ m.apex)
  (hTC2' : ∀ a ∈ T2', a.TypeConsistent (tableOn ty n.apex))
  (hw2' : ∀ a ∈ T2', ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ n.apex) :
  austereDnf ty (m.seq n) (T1 ++ T2)
    = austereDnf ty (m.seq n) (T1' ++ T2')
```

SuFunctor.admRegion\_append\_empty\_left

lean/su-functor/SuFunctor/HypernetComposition.lean, line 373

```

theorem admRegion_append_empty_left (m : AlignedLoose X Y)
  (n : AlignedLoose Y Z) {T1 T2 : List PrimApp}
  (h : AdmRegion ty m T1 = ∅)
  (hTC1 : ∀ a ∈ T1, a.TypeConsistent (tableOn ty m.apex))
  (hw1 : ∀ a ∈ T1, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ m.apex)
  (hTC2 : ∀ a ∈ T2, a.TypeConsistent (tableOn ty n.apex))
  (hw2 : ∀ a ∈ T2, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ n.apex) :
  AdmRegion ty (m.seq n) (T1 ++ T2) = ∅

```

SuFunctor.share\_contradiction

lean/su-functor/SuFunctor/HypernetComposition.lean, line 465

```

theorem share_contradiction {w : Wire} {τ : TensorType}
  (c1 c2 : TensorValue τ) {i0 : ℕ}
  (hne : c1.entryE i0 ≠ c2.entryE i0) :
  AdmRegion ty
    ((AlignedLoose.unit {w}).seq (AlignedLoose.unit {w}))
    ([PrimApp.const τ c1 w] ++ [PrimApp.const τ c2 w]) = ∅

```

SuFunctor.theta\_rename\_seq

lean/su-functor/SuFunctor/HypernetComposition.lean, line 564

```

theorem theta_rename_seq {m : AlignedLoose X Y} {n : AlignedLoose Y Z}
  {m' : AlignedLoose X' Y'} {n' : AlignedLoose Y' Z'}
  (c1 : AlignedCell ty m m') (c2 : AlignedCell ty n n')
  (hmap : c1.map = c2.map) (T1 T2 : List PrimApp)
  (hTC1 : ∀ a ∈ T1, a.TypeConsistent (tableOn ty m.apex))
  (hw1 : ∀ a ∈ T1, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ m.apex)
  (hTC2 : ∀ a ∈ T2, a.TypeConsistent (tableOn ty n.apex))
  (hw2 : ∀ a ∈ T2, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ n.apex)
  (hTC1' : ∀ a ∈ T1.map (PrimApp.rename c1.map),
    a.TypeConsistent (tableOn ty m'.apex))
  (hw1' : ∀ a ∈ T1.map (PrimApp.rename c1.map),
    ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ m'.apex)
  (hTC2' : ∀ a ∈ T2.map (PrimApp.rename c2.map),
    a.TypeConsistent (tableOn ty n'.apex))
  (hw2' : ∀ a ∈ T2.map (PrimApp.rename c2.map),
    ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ n'.apex) :
  dnfPull (renameTot ty (AlignedCell.seqCell ty c1 c2 hmap))
    (austereDnf ty (m.seq n) (T1 ++ T2))
  = dnfPull (splitTot ty m' n')
    (dnfBoxplus
      (dnfPull (renameTot ty c1) (austereDnf ty m T1))
      (dnfPull (renameTot ty c2) (austereDnf ty n T2)))

```

SuFunctor.admRegion\_append

lean/su-functor/SuFunctor/HypernetInclusion.lean, line 728

```

theorem admRegion_append (m : AlignedLoose X Y) (n : AlignedLoose Y Z)
  (T1 T2 : List PrimApp)
  (hTC1 : ∀ a ∈ T1, a.TypeConsistent (tableOn ty m.apex))
  (hw1 : ∀ a ∈ T1, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ m.apex)
  (hTC2 : ∀ a ∈ T2, a.TypeConsistent (tableOn ty n.apex))
  (hw2 : ∀ a ∈ T2, ∀ w ∈ a.inputWires ++ a.outputWires, w ∈ n.apex) :
  AdmRegion ty (m.seq n) (T1 ++ T2)
  = splitTot ty m n -1
    (Pat.compFst (alignedPat ty m) (alignedPat ty n) -1
      AdmRegion ty m T1
      n Pat.compSnd (alignedPat ty m) (alignedPat ty n) -1
      AdmRegion ty n T2)

```

SuFunctor.share\_feasible

lean/su-functor/SuFunctor/HypernetComposition.lean, line 442

```

theorem share_feasible {w : Wire} {τ : TensorType} (hty : ty w = some τ)
  (c : TensorValue τ) :

```

```
(AdmRegion ty (AlignedLoose.unit {w})
  [PrimApp.const τ c w]).Nonempty
```

SuFunctor.Spacetime.**hybridTime\_square\_subset**      lean/su-functor/SuFunctor/Adequacy/HybridTime.lean, line 489

```
theorem hybridTime_square_subset
  (D : SpecDnf (Joint (W → ClockedEl Clk (SiteRec κ X)))) :
  ((D.withWall (sampleWireWall κ X ports i i' ∅)).withWall
    (possWall (perTickRowsSet κ X ports i i' ∅ Pc))).modelClass ⊆
  ((D.withWall (sampleWireWall κ X ports i i' ∅)).withWall
    (possWall (coarseRowsSet κ X ports i Pc))).modelClass
```

SuFunctor.Spacetime.**tickSeam\_hybridTime\_square\_eq**      lean/su-functor/SuFunctor/Adequacy/HybridTime.lean, line 552

```
theorem tickSeam_hybridTime_square_eq {X : Type u}
  (Pc : SpacetimeRec (tickClkOf.{u} {false}) X → Prop)
  (D : SpecDnf (Joint (ULift.{u} Bool →
    ClockedEl (ULift.{u} Bool) (SiteRec tickClkOf X)))) :
  ((D.withWall (sampleWireWall tickClkOf X tickPorts {false} {true})
    tickSeamHom)).withWall
    (possWall (perTickRowsSet tickClkOf X tickPorts {false} {true})
    tickSeamHom Pc))).modelClass =
  ((D.withWall (sampleWireWall tickClkOf X tickPorts {false} {true})
    tickSeamHom)).withWall
    (possWall (coarseRowsSet tickClkOf X tickPorts {false}
    Pc))).modelClass
```

SuFunctor.Plain.**funnelComp\_breaks\_unit\_coherence**      lean/su-functor/SuFunctor/Plain/Battery.lean, line 177

```
theorem funnelComp_breaks_unit_coherence (M : Pat A B) :
  ¬ ∀ ℳ : Cpl M,
    Pen.pull (PatCell.tot
      (PseudoDoubleCategoryStruct.lLeftUnitor (C := WireSpan El) M)) ''
      funnelComp ({unitPen A} : Cpl (Pat.id A)) ℳ = ℳ
```

SuFunctor.Plain.**unitClass\_transparent**      lean/su-functor/SuFunctor/Plain/Battery.lean, line 86

```
theorem unitClass_transparent (M : Pat A B) (ℳ : Cpl M) :
  Pen.pull (PatCell.tot
    (PseudoDoubleCategoryStruct.lLeftUnitor (C := WireSpan El) M)) ''
    Set.image2 boxplus ({unitPen A} : Cpl (Pat.id A)) ℳ = ℳ
```

SuFunctor.Adequacy.**funnel\_coupledSat\_of\_parts**      lean/su-functor/SuFunctor/Adequacy/Funnel.lean, line 195

```
theorem funnel_coupledSat_of_parts (F : FunnelData K) {σ1 σ2 : K → ℝ≥0∞}
  (h1 : σ1 ∈ F.transit.modelClass) (h2 : σ2 ∈ F.hold.modelClass)
  (hcon : ∀ w ∈ F.composite.marks, Dip (σ1 + σ2) w) :
  σ1 + σ2 ∈ F.composite.modelClass
```

SuFunctor.Adequacy.FunnelParts.**assemble\_isSome\_iff**      lean/su-functor/SuFunctor/Adequacy/Funnel.lean, line 138

```
theorem assemble_isSome_iff {p : FunnelParts K} :
  p.assemble.isSome ↔ p.transitTarget = p.holdInvariant ∧ 0 < p.budget
```

SuFunctor.**vorobev\_record\_glue**      lean/su-functor/SuFunctor/Adequacy/MarginalGlue.lean, line 274

```
theorem vorobev_record_glue (v : Bool × Bool × Bool) :
  (0 < pairCorr.wt (proj12 v) ∧ 0 < pairCorr.wt (proj23 v) ∧
    0 < pairIndep.wt (proj13 v)) ↔
  v = (false, false, false) ∨ v = (true, true, true)
```

Modules cited: SuFunctor.HypernetComposition, SuFunctor.Adequacy.HybridTime, SuFunctor.Adequacy.Funnel, SuFunctor.Adequacy.StochasticPetriSupervise.

Also cited in this chapter, listed under an earlier one: `Qtsl.ScmTwin`, `SuFunctor.Adequacy.MarginalGlue`, `SuFunctor.HypernetInclusion`, `SuFunctor.Plain.Battery`, `SuFunctor.SingleSiteSemanticUniverse`, `coupledSat_comp_add_of_dips`, `credalCompile_vorobevGlue`, `glue_admissible_iff`, `theta_comp`, `vorobev_no_joint`.

## A.6 Specification Language

Citations from Chapter 14.

`SuFunctor.Universe.Sat` lean/su-functor/SuFunctor/Universe/Judgment.lean, line 54  
 | `def Sat (σ φ : Penalty V) : Prop`

`SuFunctor.Universe.wallOn_le_iff` lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 120  
 | `theorem wallOn_le_iff {C : Set K} {σ : K → ℝ≥0∞} :`  
   `wallOn C ≤ σ ↔ permitted σ ≤ C`

`SuFunctor.Universe.dip_top_iff` lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 137  
 | `theorem dip_top_iff {σ : K → ℝ≥0∞} :`  
   `Dip σ (τ : K → ℝ≥0∞) ↔ (permitted σ).Nonempty`

`SuFunctor.SatCredal` lean/su-functor/SuFunctor/Adequacy/Envelope.lean, line 289  
 | `def SatCredal (k : EnvelopeKind) (C c : SubPMF V → Prop) : Prop`

`SuFunctor.Universe.SpecRecord.elabSpec_top_mem` lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 935  
 | `theorem SpecRecord.elabSpec_top_mem {width : α → ℕ} {A : α → Set K}`  
   `{r : SpecRecord α} {D : SpecDnf K}`  
   `(_h : r.elabSpec width A = .ok D) :`  
   `∀ a ∈ D.atoms, (τ : K → ℝ≥0∞) ∈ a.marks`

`SuFunctor.Universe.SpecRecord.elabSpec` lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 902  
 | `noncomputable def SpecRecord.elabSpec (width : α → ℕ) (A : α → Set K)`  
   `(r : SpecRecord α) : Except (ConformanceError α) (SpecDnf K)`

`SuFunctor.Universe.SpecRecord.resolveAry_isOk_iff` lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 742  
 | `theorem SpecRecord.resolveAry_isOk_iff {width : α → ℕ}`  
   `{r : SpecRecord α} :`  
   `isOk (r.resolveAry width) ↔ r.Conformant width`

`SuFunctor.Universe.SpecRecord.resolveAry_error_names_offender` lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 835  
 | `theorem SpecRecord.resolveAry_error_names_offender {width : α → ℕ}`  
   `{r : SpecRecord α} {a : α} {f e : ℕ} {b : Bool}`  
   `(h : r.resolveAry width = .error (.widthMismatch a f e b)) :`  
   `width a = f ∧ f ≠ e ∧`  
   `((b = false ∧ a ∈ r.root.atoms) ∨`  
   `(b = true ∧ ∃ I, r.iface? = some I ∧`  
   `∃ φ, I.assumption? = some φ ∧ a ∈ φ.atoms))`

`SuFunctor.Universe.SpecRecord.elabSpec_isOk_iff` lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 909  
 | `theorem SpecRecord.elabSpec_isOk_iff {width : α → ℕ} {A : α → Set K}`  
   `{r : SpecRecord α} :`  
   `isOk (r.elabSpec width A) ↔ r.Conformant width`

`SuFunctor.Universe.SpecRecord.elabSpec_ok_modelClass` lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 918



|                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| <pre> theorem SpecRecord.elabSpec_ok_modelClass {width : <math>\alpha \rightarrow \mathbb{N}</math>}   {A : <math>\alpha \rightarrow \text{Set } K</math>} {r : SpecRecord <math>\alpha</math>} {D : SpecDnf K}   (h : r.elabSpec width A = .ok D) :     D.modelClass = NnfForm.classSem A r.root.nnf </pre>                                                                                                               |                                                            |
| SuFunctor.Universe.SpecDnf. <b>modelClass_top</b>                                                                                                                                                                                                                                                                                                                                                                          | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 323 |
| <pre> theorem SpecDnf.modelClass_top :   (SpecDnf.top : SpecDnf K).modelClass = safeProper Set.univ </pre>                                                                                                                                                                                                                                                                                                                 |                                                            |
| SuFunctor.Universe.SpecDnf. <b>modelClass_bot</b>                                                                                                                                                                                                                                                                                                                                                                          | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 328 |
| <pre> theorem SpecDnf.modelClass_bot :   (SpecDnf.bot : SpecDnf K).modelClass = <math>\emptyset</math> </pre>                                                                                                                                                                                                                                                                                                              |                                                            |
| SuFunctor.Universe. <b>den_compl_le</b>                                                                                                                                                                                                                                                                                                                                                                                    | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 601 |
| <pre> theorem den_compl_le (f : <math>K \rightarrow \mathbb{R}</math>) (c : <math>\mathbb{Q}</math>) :   {<math>\mu \mid f \mu \leq (c : \mathbb{R})</math>}<sup>c</sup> = {<math>\mu \mid (c : \mathbb{R}) &lt; f \mu</math>} </pre>                                                                                                                                                                                      |                                                            |
| SuFunctor.Universe. <b>safetyAtom</b>                                                                                                                                                                                                                                                                                                                                                                                      | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 176 |
| <pre> noncomputable def safetyAtom (C : Set K) : SpecAtom K where ... </pre>                                                                                                                                                                                                                                                                                                                                               |                                                            |
| SuFunctor.Universe. <b>coupledSat_safetyAtom_iff</b>                                                                                                                                                                                                                                                                                                                                                                       | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 191 |
| <pre> theorem coupledSat_safetyAtom_iff {C : Set K} {<math>\sigma : K \rightarrow \mathbb{R}_{\geq 0\infty}</math>} :   CoupledSat <math>\sigma</math> (wallOn C) {<math>\tau</math>} <math>\leftrightarrow</math>   permitted <math>\sigma \subseteq C \wedge</math> (permitted <math>\sigma</math>).Nonempty </pre>                                                                                                      |                                                            |
| SuFunctor.Universe. <b>safetyAtom_modelClass</b>                                                                                                                                                                                                                                                                                                                                                                           | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 205 |
| <pre> theorem safetyAtom_modelClass (C : Set K) :   (safetyAtom C).modelClass = safeProper C </pre>                                                                                                                                                                                                                                                                                                                        |                                                            |
| SuFunctor.Universe.SpecAtom. <b>modelClass_merge</b>                                                                                                                                                                                                                                                                                                                                                                       | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 243 |
| <pre> theorem SpecAtom.modelClass_merge (a b : SpecAtom K) :   (a.merge b).modelClass = a.modelClass <math>\cap</math> b.modelClass </pre>                                                                                                                                                                                                                                                                                 |                                                            |
| SuFunctor.Universe.SpecDnf. <b>modelClass_conj</b>                                                                                                                                                                                                                                                                                                                                                                         | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 292 |
| <pre> theorem SpecDnf.modelClass_conj (D E : SpecDnf K) :   (D.conj E).modelClass = D.modelClass <math>\cap</math> E.modelClass </pre>                                                                                                                                                                                                                                                                                     |                                                            |
| SuFunctor.Universe.SpecDnf. <b>modelClass_disj</b>                                                                                                                                                                                                                                                                                                                                                                         | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 308 |
| <pre> theorem SpecDnf.modelClass_disj (D E : SpecDnf K) :   (D.disj E).modelClass = D.modelClass <math>\cup</math> E.modelClass </pre>                                                                                                                                                                                                                                                                                     |                                                            |
| SuFunctor.Universe.NnfForm. <b>exists_hedging_gap</b>                                                                                                                                                                                                                                                                                                                                                                      | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 557 |
| <pre> theorem exists_hedging_gap :   <math>\exists</math> (A : Bool <math>\rightarrow</math> Set Bool) (<math>\psi \chi</math> : NnfForm Bool) (<math>\sigma : \text{Bool} \rightarrow \mathbb{R}_{\geq 0\infty}</math>),     <math>\sigma \in \text{safeProper } ((\text{NnfForm.disj } \psi \chi).\text{den } A) \wedge</math>     <math>\sigma \notin \text{classSem } A (\text{NnfForm.disj } \psi \chi)</math> </pre> |                                                            |
| SuFunctor.Universe.NnfForm. <b>elabNnf_modelClass_disjFree</b>                                                                                                                                                                                                                                                                                                                                                             | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 546 |
| <pre> theorem elabNnf_modelClass_disjFree (A : <math>\alpha \rightarrow \text{Set } K</math>) (<math>\psi</math> : NnfForm <math>\alpha</math>)   (h : <math>\psi</math>.DisjFree) :   (elabNnf A <math>\psi</math>).modelClass = safeProper (<math>\psi</math>.den A) </pre>                                                                                                                                              |                                                            |
| SuFunctor.Universe.StateFormula. <b>den_nnf</b>                                                                                                                                                                                                                                                                                                                                                                            | lean/su-functor/SuFunctor/Universe/SpecElab.lean, line 462 |

```
theorem den_nnf (A :  $\alpha \rightarrow \text{Set } K$ ) ( $\phi$  : StateFormula  $\alpha$ ) :
  ( $\phi$ .nnf).den A =  $\phi$ .den A
```

SuFunctor.Universe.**recordCon\_iff\_stages** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 720

```
theorem recordCon_iff_stages {p : NonemptyCompacts X  $\times$  ProbState X} :
  p  $\in$  recordCon  $\leftrightarrow$  stages .dwell p  $\subseteq$  stages .touch p
```

SuFunctor.Universe.**exists\_until0n\_species\_incomparable** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 840

```
theorem exists_until0n_species_incomparable :
   $\exists$  p  $\in$  recordCon (X := Bool),
    (until0n (fun b => cond b 1 0) 1  $\emptyset$  {false} (stages .touch p)  $\wedge$ 
       $\neg$  until0n (fun b => cond b 1 0) 1  $\emptyset$  {false} (stages .dwell p))  $\wedge$ 
    (until0n (fun b => cond b 1 0) 1 {true} {true} (stages .dwell p)  $\wedge$ 
       $\neg$  until0n (fun b => cond b 1 0) 1 {true} {true}
      (stages .touch p))
```

SuFunctor.Universe.**exists\_streett0n\_species\_incomparable** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 888

```
theorem exists_streett0n_species_incomparable :
   $\exists$  p  $\in$  recordCon (X := Bool),
    (streett0n [({true} : Set Bool), ({false} : Set Bool)])
      (stages .touch p)  $\wedge$ 
     $\neg$  streett0n [({true} : Set Bool), ({false} : Set Bool)]
      (stages .dwell p)  $\wedge$ 
    (streett0n [({false} : Set Bool), ( $\emptyset$  : Set Bool)])
      (stages .dwell p)  $\wedge$ 
     $\neg$  streett0n [({false} : Set Bool), ( $\emptyset$  : Set Bool)]
      (stages .touch p))
```

SuFunctor.Universe.**touchWall\_eq\_wall0n** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 744

```
theorem touchWall_eq_wall0n (Bad : Set X) :
  touchWall Bad
  = wall0n {p : NonemptyCompacts X  $\times$  ProbState X |
    always0n Badc (stages .touch p)}
```

SuFunctor.Universe.**dwellWall\_eq\_wall0n** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 761

```
theorem dwellWall_eq_wall0n (Bad : Set X) :
  dwellWall Bad
  = wall0n {p : NonemptyCompacts X  $\times$  ProbState X |
    always0n Badc (stages .dwell p)}
```

SuFunctor.Universe.**mark0n** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 100

```
noncomputable def mark0n (C : Set K) : K  $\rightarrow \mathbb{R}_{\geq 0\infty}$ 
```

SuFunctor.Universe.**mark0n\_univ** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 112

```
@[simp] theorem mark0n_univ : mark0n (Set.univ : Set K) =  $\top$ 
```

SuFunctor.Universe.**coupledSat\_pathAtom\_iff** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 167

```
theorem coupledSat_pathAtom_iff {C G : Set K} { $\sigma$  : K  $\rightarrow \mathbb{R}_{\geq 0\infty}$ } :
  CoupledSat  $\sigma$  (pathAtom C G).floor (pathAtom C G).marks  $\leftrightarrow$ 
  permitted  $\sigma \subseteq C \wedge$  (permitted  $\sigma \cap G$ ).Nonempty
```

SuFunctor.Universe.**stages\_uniformLift** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 732

```
theorem stages_uniformLift (sp : RecordSpecies) (G : NonemptyCompacts X) :
  stages sp (G, uniformLift G) = (G : Set X)
```

SuFunctor.Universe.**always0n** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 266

```
def always0n (Good : Set X) (S : Set X) : Prop
```

SuFunctor.Universe.**isClosed\_recordClass\_always** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 1010

```
theorem isClosed_recordClass_always (sp : RecordSpecies) (τ : X → ℕ)
  (A : α → Set X) (φ : StateFormula α) :
  IsClosed (recordClass sp τ A (.always φ))
```

SuFunctor.Universe.**windowAlways0n** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 270

```
def windowAlways0n (τ : X → ℕ) (n : ℕ) (Good : Set X) (S : Set X) :
  Prop
```

SuFunctor.Universe.**isClosed\_recordClass\_alwaysT** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 1017

```
theorem isClosed_recordClass_alwaysT (sp : RecordSpecies) (τ : X → ℕ)
  (A : α → Set X) (n : ℕ) (φ : StateFormula α) :
  IsClosed (recordClass sp τ A (.alwaysT n φ))
```

SuFunctor.Universe.**windowReach0n** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 276

```
def windowReach0n (τ : X → ℕ) (n : ℕ) (Goal : Set X) (S : Set X) :
  Prop
```

SuFunctor.Universe.**is0Open\_recordClass\_deadline** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 1030

```
theorem is0Open_recordClass_deadline (sp : RecordSpecies) (τ : X → ℕ)
  (A : α → Set X) (n : ℕ) (φ : StateFormula α) (b : Bool) :
  Is0Open (recordClass sp τ A (.deadline n φ b))
```

SuFunctor.Universe.**until0n** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 283

```
def until0n (τ : X → ℕ) (n : ℕ) (Safe Target : Set X) (S : Set X) :
  Prop
```

SuFunctor.Universe.**recordClass\_untilF\_eq** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 1045

```
theorem recordClass_untilF_eq (sp : RecordSpecies) (τ : X → ℕ)
  (A : α → Set X) (n : ℕ) (sf tg : StateFormula α) :
  recordClass sp τ A (.untilF n sf tg)
  = ⋃ t : Fin (n + 1),
    ({p : NonemptyCompacts X × ProbState X |
      (stages sp p n (tg.den A n {x | τ x = (t : ℕ)}))}.Nonempty}
    n {p : NonemptyCompacts X × ProbState X |
      stages sp p ⊆ sf.den A ∪ {x | ¬ τ x < (t : ℕ)}})
```

SuFunctor.Universe.**isConstructibleUnion\_recordClass\_untilF** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 1062

```
theorem isConstructibleUnion_recordClass_untilF (sp : RecordSpecies)
  (τ : X → ℕ) (A : α → Set X) (n : ℕ) (sf tg : StateFormula α) :
  IsConstructibleUnion (recordClass sp τ A (.untilF n sf tg))
```

Qtsl.QStreett.**sound\_QStreettBundle** lean/qtsl-aux/QtslAux/QStreett.lean, line 347

```
theorem sound_QStreettBundle {Rq G : ι → Set S} {W : ι → S → ℝ} {ε C : ι → ℝ}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV₀ : Measurable V₀)
  (hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
  (hW : ∀ i, Measurable (W i))
  (hθ : p + thr ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ U → K y Dbar = 1)
```

```

(hR4 :  $\forall y \in D, y \notin U \rightarrow K y (D \cup U) = 1$ )
(hV0int :  $\forall y \in D, y \notin U \rightarrow \text{Integrable } V_0 (K y)$ )
(hV0drift :  $\forall y \in D, y \notin U \rightarrow \int z, V_0 z \partial K y \leq V_0 y$ )
(hV0floor :  $\forall y \in U, y \in \text{Dbar} \rightarrow 1 \leq V_0 y$ )
(hV0nonneg :  $\forall y \in \text{Dbar}, 0 \leq V_0 y$ )
(h $\epsilon$  :  $\forall i, 0 < \epsilon i$ )
(hW0 :  $\forall i, \forall y \in \text{Dbar}, 0 \leq W i y$ ) (hWC :  $\forall i, \forall y \in \text{Dbar}, W i y \leq C i$ )
(hWint :  $\forall i, \forall y \in D, y \notin U \rightarrow y \notin G i \rightarrow \text{Integrable } (W i) (K y)$ )
(hWdrift :  $\forall i, \forall y \in D, y \notin U \rightarrow y \notin G i \rightarrow \int z, W i z \partial K y \leq W i y$ )
(hWdriftR :  $\forall i, \forall y \in D, y \notin U \rightarrow y \in \text{Rq } i \rightarrow y \notin G i \rightarrow$ 
   $\int z, W i z \partial K y \leq W i y - \epsilon i$ )
(hx :  $x \in D$ ) (hthr :  $V_0 x \leq \text{thr}$ ) :
ENNReal.ofReal p  $\leq \text{pathMeasure } K x$ 
 $\{\omega \mid \forall i, (\exists^f n \text{ in atTop}, \omega n \in \text{Rq } i) \rightarrow \exists^f n \text{ in atTop}, \omega n \in G i\}$ 

```

SuFunctor.Universe.**streett0n** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 292

```
def streett0n (prs : List (Set X  $\times$  Set X)) (S : Set X) : Prop
```

SuFunctor.Universe.**pathNnf\_notPath\_alwaysT** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 677

```

@[simp] theorem pathNnf_notPath_alwaysT (n :  $\mathbb{N}$ ) ( $\phi$  : StateFormula  $\alpha$ ) :
  pathNnf (PathFormula.notPath (.alwaysT n  $\phi$ ))
    = .ok (.deadline n  $\phi$ .neg false)

```

SuFunctor.Universe.**pathNnf\_notPath\_notPath\_deadline** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 683

```

@[simp] theorem pathNnf_notPath_notPath_deadline (n :  $\mathbb{N}$ )
  ( $\phi$  : StateFormula  $\alpha$ ) (b : Bool) :
  pathNnf (PathFormula.notPath (.notPath (.deadline n  $\phi$  b)))
    = .ok (.deadline n  $\phi$  b)

```

SuFunctor.Universe.**pathNnf\_notPath\_always** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 659

```

@[simp] theorem pathNnf_notPath_always ( $\phi$  : StateFormula  $\alpha$ ) :
  pathNnf (PathFormula.notPath (.always  $\phi$ ))
    = .error .negatedAlways

```

SuFunctor.Universe.**pathNnf\_notPath\_untilF** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 664

```

@[simp] theorem pathNnf_notPath_untilF (n :  $\mathbb{N}$ )
  (sf tg : StateFormula  $\alpha$ ) :
  pathNnf (PathFormula.notPath (.untilF n sf tg))
    = .error .negatedUntil

```

SuFunctor.Universe.**pathNnf\_notPath\_streett** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 671

```

@[simp] theorem pathNnf_notPath_streett (prs : List (StreettPair  $\alpha$ )) :
  pathNnf (PathFormula.notPath (.streett prs))
    = .error .negatedStreett

```

SuFunctor.Universe.**not\_always0n\_iff** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 422

```

theorem not_always0n_iff {Good S : Set X} :
   $\neg \text{always0n Good } S \leftrightarrow \exists x \in S, x \in \text{Good}^c$ 

```

SuFunctor.Universe.**satCredal\_lower\_singletonC\_zero** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 143

```

theorem satCredal_lower_singletonC_zero (c : SubPMF V  $\rightarrow$  Prop) :
  SatCredal .lower (singletonC (SubPMF.zero V)) c

```

SuFunctor.Universe.**probThresholdSet** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 168

```
def probThresholdSet (K : Kernel S S) [IsMarkovKernel K] (p : ℝ)
  (E : Set (ℕ → S)) : Set S
```

SuFunctor.Universe.**safetyAtom\_modelClass\_mono** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 92

```
theorem safetyAtom_modelClass_mono {C D : Set X} (h : C ⊆ D) :
  (safetyAtom C).modelClass ⊆ (safetyAtom D).modelClass
```

SuFunctor.Universe.**sprobAtom\_modelClass\_mono** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 210

```
theorem sprobAtom_modelClass_mono {p : ℝ} {E E' : Set (ℕ → S)}
  (hE : E ⊆ E') :
  (safetyAtom (probThresholdSet K p E)).modelClass ⊆
    (safetyAtom (probThresholdSet K p E')).modelClass
```

SuFunctor.Universe.**sprob\_sound\_of\_streettBundle** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 224

```
theorem sprob_sound_of_streettBundle
  {K : Kernel S S} [IsMarkovKernel K]
  {D Dbar U : Set S} {V₀ : S → ℝ} {p thr : ℝ} {x : S}
  {ι : Type*} [Countable ι]
  {Rq G : ι → Set S} {W : ι → S → ℝ} {ε C : ι → ℝ}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar)
  (hU : MeasurableSet U) (hV₀ : Measurable V₀)
  (hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
  (hW : ∀ i, Measurable (W i))
  (hθ : p + thr ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ U → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ U → K y (D ∪ U) = 1)
  (hV₀int : ∀ y ∈ D, y ∉ U → Integrable V₀ (K y))
  (hV₀drift : ∀ y ∈ D, y ∉ U → ∫ z, V₀ z ∂K y ≤ V₀ y)
  (hV₀floor : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V₀ y)
  (hV₀nonneg : ∀ y ∈ Dbar, 0 ≤ V₀ y)
  (hε : ∀ i, 0 < ε i)
  (hW0 : ∀ i, ∀ y ∈ Dbar, 0 ≤ W i y)
  (hWC : ∀ i, ∀ y ∈ Dbar, W i y ≤ C i)
  (hWint : ∀ i, ∀ y ∈ D, y ∉ U → y ∉ G i → Integrable (W i) (K y))
  (hWdrift : ∀ i, ∀ y ∈ D, y ∉ U → y ∉ G i →
    ∫ z, W i z ∂K y ≤ W i y)
  (hWdriftR : ∀ i, ∀ y ∈ D, y ∉ U → y ∈ Rq i → y ∉ G i →
    ∫ z, W i z ∂K y ≤ W i y - ε i)
  (hx : x ∈ D) (hthr : V₀ x ≤ thr)
  {E : Set (ℕ → S)}
  (hE : {ω | ∀ i, (∃f n in atTop, ω n ∈ Rq i) →
    ∃f n in atTop, ω n ∈ G i} ⊆ E) :
  x ∈ probThresholdSet K p E
```

SuFunctor.Universe.**satCredal\_lower\_singletonC\_geC** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 115

```
theorem satCredal_lower_singletonC_geC {μ : SubPMF V}
  (hμ : μ ≠ SubPMF.zero V) (A : Set V) (q : ℚ) :
  SatCredal .lower (singletonC μ) (geC A q) ↔ q ≤ μ.prob A
```

SuFunctor.Universe.**returnGSet** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 284

```
def returnGSet (K : Kernel S S) [IsMarkovKernel K] (r : S → ℝ)
  (b : ℝ) (n : ℕ) : Set S
```

SuFunctor.Universe.**returnGSet\_mono\_reward** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 299

```
theorem returnGSet_mono_reward {r r' : S → ℝ} {b : ℝ} {n : ℕ}
  (hr : ∀ s, r s ≤ r' s)
```

```
(hint : ∀ x, Integrable (finReturn r n) (pathMeasure K x))
(hint' : ∀ x, Integrable (finReturn r' n) (pathMeasure K x)) :
returnGESet K r b n ≤ returnGESet K r' b n
```

SuFunctor.Universe.**returnGEAtom\_modelClass\_mono\_reward** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 334

```
theorem returnGEAtom_modelClass_mono_reward {r r' : S → ℝ} {b : ℝ}
{n : ℕ} (hr : ∀ s, r s ≤ r' s)
(hint : ∀ x, Integrable (finReturn r n) (pathMeasure K x))
(hint' : ∀ x, Integrable (finReturn r' n) (pathMeasure K x)) :
(safetyAtom (returnGESet K r b n)).modelClass ≤
(safetyAtom (returnGESet K r' b n)).modelClass
```

SuFunctor.Universe.**returnGE\_sound\_of\_deadline** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 441

```
theorem returnGE_sound_of_deadline [IsProbabilityMeasure μ]
(hβ : 0 ≤ β) (ha : 0 < a)
(hint : ∀ n, Integrable (L n) μ) (hnonneg : 0 ≤ L)
(hmB : ∀ n, MeasurableSet[ℒ n] {ω | X n ω ∈ B})
(hdrift : ∀ n, ∀n ω ∂μ, X n ω ∉ B →
(μ[L (n + 1) | ℒ n] ω ≤ β * L n ω)
(hbarrier : ∀ n ω, X n ω ∉ B → a ≤ L n ω) (n : ℕ) :
1 - β ^ n * (∫ ω, L 0 ω ∂μ) / a ≤
∫ ω, finReturn (B.indicator fun _ => (1 : ℝ)) (n + 1)
(fun t => X t ω) ∂μ
```

SuFunctor.Universe.**discountedReturn\_le\_of\_drift** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 543

```
theorem discountedReturn_le_of_drift [IsFiniteMeasure μ]
{γ : ℝ} {r : ℕ → Ω → ℝ}
(hβ : 0 ≤ β) (hγ : 0 ≤ γ)
(hint : ∀ n, Integrable (L n) μ) (hnonneg : 0 ≤ L)
(hdrift : ∀ n, ∀n ω ∂μ, (μ[L (n + 1) | ℒ n] ω ≤ β * L n ω)
(hrint : ∀ n, Integrable (r n) μ)
(hr : ∀ n ω, r n ω ≤ L n ω) (n : ℕ) :
∫ ω, discountedPartialReturn γ r n ω ∂μ ≤
(∑ t ∈ Finset.range n, (γ * β) ^ t) * ∫ ω, L 0 ω ∂μ
```

SuFunctor.Universe.**integral\_le\_pow\_mul** lean/su-functor/SuFunctor/Adequacy/SpecBehavioral.lean, line 511

```
theorem integral_le_pow_mul [IsFiniteMeasure μ] (hβ : 0 ≤ β)
(hint : ∀ n, Integrable (L n) μ) (hnonneg : 0 ≤ L)
(hdrift : ∀ n, ∀n ω ∂μ, (μ[L (n + 1) | ℒ n] ω ≤ β * L n ω)
(n : ℕ) :
∫ ω, L n ω ∂μ ≤ β ^ n * ∫ ω, L 0 ω ∂μ
```

SuFunctor.**livenessTilt\_top** lean/su-functor-legacy/SuFunctorLegacy/Threshold.lean, line 275

```
theorem livenessTilt_top (goal : Set V) (p : ℚ) :
livenessTilt goal p τ = livenessWall goal p
```

SuFunctor.Universe.**until0n\_univ\_safe** lean/su-functor/SuFunctor/Adequacy/SpecPathLens.lean, line 392

```
theorem until0n_univ_safe {τ : X → ℕ} {n : ℕ} {Target S : Set X} :
until0n τ n Set.univ Target S ↔ windowReach0n τ n Target S
```

SuFunctor.Universe.**streett0n\_singleton\_univ\_iff** lean/su-functor/SuFunctor/Universe/SpecRewrite.lean, line 599

```
theorem streett0n_singleton_univ_iff {Y : Type*} {G S : Set Y} :
streett0n [(Set.univ, G)] S ↔ (S.Nonempty → (S n G).Nonempty)
```

SuFunctor.Universe.**streett0n\_singleton\_univ\_empty** lean/su-functor/SuFunctor/Universe/SpecRewrite.lean, line 625

```
theorem streettOn_singleton_univ_empty {Y : Type*} {G : Set Y} :
  streettOn [(Set.univ, G)] (∅ : Set Y) ∧ ¬ ((∅ : Set Y) ∩ G).Nonempty
```

Qtsl.TemporalRewrite.**no\_f\_and\_fusion**

lean/qtsl-aux/QtslAux/TemporalRewrite.lean, line 274

```
theorem no_f_and_fusion :
  ∃ V : Nat → Nat → Prop,
    sat V 0 (.and (.F (.atom 0)) (.F (.atom 1))) ∧
    ¬ sat V 0 (.F (.and (.atom 0) (.atom 1)))
```

Qtsl.TemporalRewrite.**no\_g\_or\_fusion**

lean/qtsl-aux/QtslAux/TemporalRewrite.lean, line 285

```
theorem no_g_or_fusion :
  ∃ V : Nat → Nat → Prop,
    sat V 0 (.G (.or (.atom 0) (.atom 1))) ∧
    ¬ sat V 0 (.or (.G (.atom 0)) (.G (.atom 1)))
```

Qtsl.TemporalRewrite.**no\_gf\_and\_fusion**

lean/qtsl-aux/QtslAux/TemporalRewrite.lean, line 298

```
theorem no_gf_and_fusion :
  ∃ V : Nat → Nat → Prop,
    sat V 0 (.and (.G (.F (.atom 0))) (.G (.F (.atom 1)))) ∧
    ¬ sat V 0 (.G (.F (.and (.atom 0) (.atom 1))))
```

Qtsl.TemporalRewrite.**no\_fg\_or\_fusion**

lean/qtsl-aux/QtslAux/TemporalRewrite.lean, line 311

```
theorem no_fg_or_fusion :
  ∃ V : Nat → Nat → Prop,
    sat V 0 (.F (.G (.or (.atom 0) (.atom 1)))) ∧
    ¬ sat V 0 (.or (.F (.G (.atom 0))) (.F (.G (.atom 1))))
```

SuFunctor.Universe.**exists\_denEq\_not\_classEq**

lean/su-functor/SuFunctor/Universe/SpecRewrite.lean, line 206

```
theorem exists_denEq_not_classEq :
  ∃ (A : Fin 3 → Set Bool) (φ ψ : StateFormula (Fin 3)),
    DenEq A φ ψ ∧ ¬ ClassEq A φ ψ
```

SuFunctor.Universe.**classEquiv\_of\_nnf\_eq**

lean/su-functor/SuFunctor/Universe/SpecRewrite.lean, line 139

```
theorem classEquiv_of_nnf_eq {φ ψ : StateFormula α} (h : φ.nnf = ψ.nnf) :
  ClassEquiv.{u} φ ψ
```

Modules cited: SuFunctor.Universe.SpecElab, SuFunctor.Universe.SpecRewrite, SuFunctor.Universe.PairSequent.

Also cited in this chapter, listed under an earlier one: General.iSup\_gambleRows\_eq\_self, iSup\_boxRows\_eq\_self, sat\_sup\_iff, top\_notMem\_modelClass.

## A.7 Rule Catalogue

Citations from Chapter 15.

PPLC.**PxData**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1353

```
inductive PxData : Hypernet → Sequent → Type where ...
```

SuFunctor.StateSpace.FamObj.**famEq**

lean/su-functor/SuFunctor/StateSpace/FamLimits.lean, line 169

```
def famEq : FamObj C
```

PPLC.**sound\_ax**

lean/qtsl/Qtsl/Soundness.lean, line 4165

```
theorem sound_ax {H : Hypernet} {seq : Sequent}
  (h : seq.pre = seq.post) : seq.semanticallyValid H
```



PPLC.**Formula.holds**

lean/qtsl/Qtsl/Catalogue.lean, line 402

```

def Formula.holds {H : Hypernet} (σ : H.Valuation) : Formula → Prop
| .atom b l r => ∃ a a' : QTensor,
  Term.evalQ σ l = some a ∧ Term.evalQ σ r = some a' ∧
  QTensor.rel2 (BowTie.holdsQ b) a a'
| .inRegion t simplex S =>
  simplex = S.simplex ∧
  ∃ q : QTensor, Term.evalQ σ t = some q ∧
  q.shape = [S.simplex.complex.ambDim] ∧
  MaskRegion S.simplex.complex S.simplex.contains S.contains
  (fun j => q.entries.getD j.val 0)
| .and l r => Formula.holds σ l ∧ Formula.holds σ r
| .or l r => Formula.holds σ l ∨ Formula.holds σ r
| .tt => True
| .ff => False

namespace ContractPattern

variable {inputRanks : List Nat} {outputRank : Nat}

def idxShape (pat : ContractPattern inputRanks outputRank) : Shape

```

PPLC.**Sequent.semanticallyValid**

lean/qtsl/Qtsl/Catalogue.lean, line 527

```

def Sequent.semanticallyValid (H : Hypernet) (seq : Sequent) : Prop

```

PPLC.M3.**le\_refl**

lean/qtsl/Qtsl/Admissible.lean, line 112

```

theorem le_refl : ∀ x : M3, le x x = true

```

PPLC.**PWeakenData**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1357

```

inductive PWeakenData : Hypernet → Sequent → Type where ...

```

PPLC.**ProofObject**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1316

```

inductive ProofObject : Hypernet → Sequent → Type where ...

```

PPLC.**sound\_weaken**

lean/qtsl/Qtsl/Soundness.lean, line 4171

```

theorem sound_weaken {H : Hypernet} {seq premise : Sequent} {extra : Formula}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = .and premise.pre extra)
  (hpost : seq.post = premise.post) : seq.semanticallyValid H

```

PPLC.M3.**weaken\_kit**

lean/qtsl/Qtsl/Admissible.lean, line 121

```

theorem weaken_kit : ∀ x y z : M3,
  le x z = true → le (inf x y) z = true

```

PPLC.**sound\_cut**

lean/qtsl/Qtsl/Soundness.lean, line 4180

```

theorem sound_cut {H : Hypernet} {seq leftPremise rightPremise : Sequent}
  {cutFormula : Formula}
  (ihl : leftPremise.semanticallyValid H)
  (ihr : rightPremise.semanticallyValid H)
  (hpre : seq.pre = leftPremise.pre)
  (hpost : seq.post = rightPremise.post)
  (hcutL : leftPremise.post = cutFormula)
  (hcutR : rightPremise.pre = .and leftPremise.pre cutFormula) :
  seq.semanticallyValid H

```

PPLC.**PCutData**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1365

```
inductive PCutData : Hypernet → Sequent → Type where ...
```

PPLC.M3.cut\_kit

lean/qtsl/Qtsl/Admissible.lean, line 125

```
theorem cut_kit : ∀ x m y : M3,
  le x m = true → le (inf x m) y = true → le x y = true
```

PPLC.sound\_andI

lean/qtsl/Qtsl/Soundness.lean, line 4198

```
theorem sound_andI {H : Hypernet} {seq leftPremise rightPremise : Sequent}
  (ihl : leftPremise.semanticallyValid H)
  (ihr : rightPremise.semanticallyValid H)
  (hpreMatch : leftPremise.pre = rightPremise.pre)
  (hpre : seq.pre = leftPremise.pre)
  (hpost : seq.post = .and leftPremise.post rightPremise.post) :
  seq.semanticallyValid H
```

PPLC.PAndIData

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1376

```
inductive PAndIData : Hypernet → Sequent → Type where ...
```

PPLC.M3.andI\_kit

lean/qtsl/Qtsl/Admissible.lean, line 129

```
theorem andI_kit : ∀ x y z : M3,
  le x y = true → le x z = true → le x (inf y z) = true
```

PPLC.sound\_andE\_left

lean/qtsl/Qtsl/Soundness.lean, line 4212

```
theorem sound_andE_left {H : Hypernet} {seq premise : Sequent} {other : Formula}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hpost : premise.post = .and seq.post other) :
  seq.semanticallyValid H
```

PPLC.sound\_andE\_right

lean/qtsl/Qtsl/Soundness.lean, line 4223

```
theorem sound_andE_right {H : Hypernet} {seq premise : Sequent} {other : Formula}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hpost : premise.post = .and other seq.post) :
  seq.semanticallyValid H
```

PPLC.M3.andE\_l\_kit

lean/qtsl/Qtsl/Admissible.lean, line 133

```
theorem andE_l_kit : ∀ x y z : M3,
  le x (inf y z) = true → le x y = true
```

PPLC.M3.andE\_r\_kit

lean/qtsl/Qtsl/Admissible.lean, line 137

```
theorem andE_r_kit : ∀ x y z : M3,
  le x (inf y z) = true → le x z = true
```

PPLC.sound\_orI\_left

lean/qtsl/Qtsl/Soundness.lean, line 4234

```
theorem sound_orI_left {H : Hypernet} {seq premise : Sequent} {other : Formula}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hpost : seq.post = .or premise.post other) :
  seq.semanticallyValid H
```

PPLC.sound\_orI\_right

lean/qtsl/Qtsl/Soundness.lean, line 4244

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|
| <pre> theorem sound_orI_right {H : Hypernet} {seq premise : Sequent} {other : Formula}   (ih : premise.semanticallyValid H)   (hpre : seq.pre = premise.pre)   (hpost : seq.post = .or other premise.post) :   seq.semanticallyValid H </pre>                                                                                                                                                                                                                                                                             |                                          |
| <b>PPLC.M3.orI_l_kit</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | lean/qtsl/Qtsl/Admissible.lean, line 141 |
| <pre> theorem orI_l_kit : ∀ x y z : M3,   le x y = true → le x (sup y z) = true </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                          |
| <b>PPLC.M3.orI_r_kit</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | lean/qtsl/Qtsl/Admissible.lean, line 145 |
| <pre> theorem orI_r_kit : ∀ x y z : M3,   le x z = true → le x (sup y z) = true </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                          |
| <b>PPLC.sound_orE</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | lean/qtsl/Qtsl/Soundness.lean, line 4254 |
| <pre> theorem sound_orE {H : Hypernet} {seq leftPremise rightPremise : Sequent}   (ihl : leftPremise.semanticallyValid H)   (ihr : rightPremise.semanticallyValid H)   (hpostMatch : leftPremise.post = rightPremise.post)   (hpre : seq.pre = .or leftPremise.pre rightPremise.pre)   (hpost : seq.post = leftPremise.post) :   seq.semanticallyValid H </pre>                                                                                                                                                           |                                          |
| <b>PPLC.orECtxStep</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | lean/qtsl/Qtsl/Admissible.lean, line 20  |
| <pre> structure OrECtxStep where ... </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                          |
| <b>PPLC.sound_orECtx</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | lean/qtsl/Qtsl/Admissible.lean, line 33  |
| <pre> theorem sound_orECtx {H : Hypernet} (st : OrECtxStep)   (ihl : st.leftPremise.semanticallyValid H)   (ihr : st.rightPremise.semanticallyValid H) :   st.seq.semanticallyValid H </pre>                                                                                                                                                                                                                                                                                                                              |                                          |
| <b>PPLC.cx_not_derivable</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Admissible.lean, line 657 |
| <pre> theorem cx_not_derivable : IsEmpty (ProofObject cxH cxSeq) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                          |
| <b>PPLC.orECtx_not_elaborable</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Admissible.lean, line 666 |
| <pre> theorem orECtx_not_elaborable :   ¬ (∀ (st : OrECtxStep) (H : Hypernet),     Nonempty (ProofObject H st.leftPremise) →     Nonempty (ProofObject H st.rightPremise) →     Nonempty (ProofObject H st.seq)) </pre>                                                                                                                                                                                                                                                                                                   |                                          |
| <b>PPLC.ProofObject.m3sound</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | lean/qtsl/Qtsl/Admissible.lean, line 406 |
| <pre> theorem ProofObject.m3sound : ∀ {H : Hypernet} {seq : Sequent},   (∀ w, H.wireType w = none) → ProofObject H seq →   M3.le (mval seq.pre) (mval seq.post) = true   _, _, _, .mk _ _ _ (.ax _ _ (.mk _ _ hpp)) =&gt; by   rw [hpp]; exact M3.le_refl _   _, _, hE, .mk _ _ _ (.weaken _ _ (.mk _ _ _ pproof _ hpre hpost)) =&gt; by   rw [hpre, hpost]   exact M3.weaken_kit _ _ _ (pproof.m3sound hE)   _, _, hE, .mk _ _ _ (.cut _ _   (.mk _ _ _ _ lproof _ rproof hpre hpost hcl hcr)) =&gt; by   have hl </pre> |                                          |
| <b>PPLC.cxStep</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | lean/qtsl/Qtsl/Admissible.lean, line 632 |

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                               |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
|                                | def cxStep : OrECtxStep where ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                               |
| PPLC.OrECtxStep.elabHere       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Admissible.lean, line 58       |
|                                | def OrECtxStep.elabHere {H : Hypernet} (st : OrECtxStep)<br>(hpath : st.path = .here)<br>(lpf : ProofObject H st.leftPremise) (rpf : ProofObject H st.rightPremise) :<br>ProofObject H st.seq                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                               |
| PPLC.POrEData                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Core/Catalogue.lean, line 1413 |
|                                | inductive POrEData : Hypernet → Sequent → Type where ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                               |
| PPLC.M3.orE_kit                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Admissible.lean, line 149      |
|                                | theorem orE_kit : $\forall x y z : M3,$<br>$le\ x\ z = true \rightarrow le\ y\ z = true \rightarrow le\ (sup\ x\ y)\ z = true$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                               |
| PPLC.substituteInFormula       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Core/Catalogue.lean, line 581  |
|                                | def substituteInFormula (F : Formula) (w : Wire) (t : Term) (F' : Formula) : Prop                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                               |
| PPLC.sound_eqElim              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Soundness.lean, line 4269      |
|                                | theorem sound_eqElim {H : Hypernet} {seq eqPremise bodyPremise : Sequent}<br>{wire : Wire} {termT termU : Term} {bodyForm : Formula}<br>(ihEq : eqPremise.semanticallyValid H)<br>(ihBody : bodyPremise.semanticallyValid H)<br>(hpre : seq.pre = eqPremise.pre)<br>(hpreMatch : eqPremise.pre = bodyPremise.pre)<br>(heqPost : eqPremise.post = .atom .eq termT termU)<br>(htyEq : $\exists s, termT.bshape\ H = some\ s \wedge termU.bshape\ H = some\ s$ )<br>(hbodySubT : substituteInFormula bodyForm wire termT bodyPremise.post)<br>(hconclSubU : substituteInFormula bodyForm wire termU seq.post) :<br>seq.semanticallyValid H |                                               |
| PPLC.Term.evalQ_subst_congr    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Soundness.lean, line 270       |
|                                | theorem Term.evalQ_subst_congr {H : Hypernet} ( $\sigma : H.Valuation$ )<br>{w : Wire} {u <sub>1</sub> u <sub>2</sub> : Term} (hu : Term.evalQ $\sigma$ u <sub>1</sub> = Term.evalQ $\sigma$ u <sub>2</sub> ) :<br>$\forall (t : Term),$<br>Term.evalQ $\sigma$ (Term.subst w u <sub>1</sub> t) = Term.evalQ $\sigma$ (Term.subst w u <sub>2</sub> t)                                                                                                                                                                                                                                                                                   |                                               |
| PPLC.Formula.holds_subst_congr |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Soundness.lean, line 288       |
|                                | theorem Formula.holds_subst_congr {H : Hypernet} ( $\sigma : H.Valuation$ )<br>{w : Wire} {u <sub>1</sub> u <sub>2</sub> : Term} (hu : Term.evalQ $\sigma$ u <sub>1</sub> = Term.evalQ $\sigma$ u <sub>2</sub> ) :<br>$\forall (F : Formula), (Formula.holds\ \sigma\ (F.subst\ w\ u_1) \leftrightarrow Formula.holds\ \sigma\ (F.subst\ w\ u_2))$                                                                                                                                                                                                                                                                                      |                                               |
| PPLC.Term.evalQ_bshape         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Soundness.lean, line 479       |
|                                | theorem Term.evalQ_bshape {H : Hypernet} ( $\sigma : H.Valuation$ ) :<br>$\forall (t : Term) (s : Shape) (q : QTensor),$<br>$t.bshape\ H = some\ s \rightarrow Term.evalQ\ \sigma\ t = some\ q \rightarrow q.shape = s$                                                                                                                                                                                                                                                                                                                                                                                                                 |                                               |
| PPLC.Hypernet.Valuation        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Catalogue.lean, line 268       |
|                                | def Hypernet.Valuation (H : Hypernet) : Type                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                               |
| PPLC.Term.evalQ                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl/Qtsl/Catalogue.lean, line 352       |

```

def Term.evalQ {H : Hypernet} (σ : H.Valuation) : Term → Option QTensor
| .var w => σ.wireQ w
| .const c => some ([], [c], rfl)
| .add l r => do QTensor.map2 (· + ·) (← Term.evalQ σ l) (← Term.evalQ σ r)
| .mul l r => do QTensor.map2 (· * ·) (← Term.evalQ σ l) (← Term.evalQ σ r)
| .sub l r => do QTensor.map2 (· - ·) (← Term.evalQ σ l) (← Term.evalQ σ r)
| .neg t => (Term.evalQ σ t).map (QTensor.map1 Neg.neg)
| .idx t i => do

  let q ← Term.evalQ σ t
  match q.shape with
  | [e] => if i < e then some ([], [q.entries.getD i 0], rfl) else none
  | _ => none

def BowTie.holdsQ : BowTie → Q → Q → Prop
| .lt => (· < ·)
| .le => (· ≤ ·)
| .eq => (· = ·)
| .ge => fun a b => b ≤ a
| .gt => fun a b => b < a
| .ne => (· ≠ ·)

def finSum (n : ℕ) (f : Fin n → ℝ) : ℝ

```

PPLC.QTensor.**rel2**

lean/qtsl/Qtsl/Catalogue.lean, line 332

```

def rel2 (r : Q → Q → Prop) (a b : QTensor) : Prop

```

PPLC.QTensor.**eq\_of\_rel2**

lean/qtsl/Qtsl/Soundness.lean, line 2431

```

theorem QTensor.eq_of_rel2 {r : Q → Q → Prop} {a b : QTensor}
  (hr : ∀ x y, r x y → x = y) (hs : a.shape = b.shape)
  (h : QTensor.rel2 r a b) : a = b

```

PPLC.Formula.**subst**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 573

```

def Formula.subst (w : Wire) (t : Term) : Formula → Formula
| .atom b l r => .atom b (Term.subst w t l) (Term.subst w t r)
| .inRegion u simplex S => .inRegion (Term.subst w t u) simplex S
| .and l r => .and (Formula.subst w t l) (Formula.subst w t r)
| .or l r => .or (Formula.subst w t l) (Formula.subst w t r)
| .tt => .tt
| .ff => .ff

def substituteInFormula (F : Formula) (w : Wire) (t : Term) (F' : Formula) : Prop

```

PPLC.Term.**bshape**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 517

```

def Term.bshape (H : Hypernet) : Term → Option Shape
| .var w => (H.wireType w).map (·.shape)
| .const _ => some []
| .add l r | .mul l r | .sub l r =>
  match l.bshape H, r.bshape H with
  | some s1, some s2 =>
    if s1 = s2 then some s1
    else if s1 = [] then some s2
    else if s2 = [] then some s1
    else none
| _, _ => none
| .neg t => t.bshape H
| .idx t i =>

  match t.bshape H with

```

|                                                                                                                                                                                                                                                                               |                                               |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| <pre>         some [e] =&gt; if i &lt; e then some [] else none         _ =&gt; none  def Term.finWires (H : Hypernet) (t : Term) : Prop </pre>                                                                                                                               |                                               |
| PPLC.Term. <b>finWires</b>                                                                                                                                                                                                                                                    | lean/qtsl/Qtsl/Core/Catalogue.lean, line 535  |
| <pre> def Term.finWires (H : Hypernet) (t : Term) : Prop </pre>                                                                                                                                                                                                               |                                               |
| PPLC.Term. <b>evalQ_defined_of_bshape_fin</b>                                                                                                                                                                                                                                 | lean/qtsl/Qtsl/Soundness.lean, line 3488      |
| <pre> theorem Term.evalQ_defined_of_bshape_fin {H : Hypernet} (σ : H.Valuation) :   ∀ (t : Term) (s : Shape), t.bshape H = some s → t.finWires H →   ∃ q, Term.evalQ σ t = some q ∧ q.shape = s </pre>                                                                        |                                               |
| PPLC.QTensor. <b>rel2_refl</b>                                                                                                                                                                                                                                                | lean/qtsl/Qtsl/Soundness.lean, line 6773      |
| <pre> theorem QTensor.rel2_refl {r : Q → Q → Prop} (hr : ∀ x, r x x)   (a : QTensor) : QTensor.rel2 r a a </pre>                                                                                                                                                              |                                               |
| PPLC. <b>sound_eqRefl</b>                                                                                                                                                                                                                                                     | lean/qtsl/Qtsl/Soundness.lean, line 6853      |
| <pre> theorem sound_eqRefl {H : Hypernet} {seq : Sequent} {t : Term}   (htyDef : ∃ s, t.bshape H = some s) (hfin : t.finWires H)   (hpost : seq.post = .atom .eq t t) :   seq.semanticallyValid H </pre>                                                                      |                                               |
| PPLC.Hypernet.Valuation. <b>wireQ_isSome</b>                                                                                                                                                                                                                                  | lean/qtsl/Qtsl/Soundness.lean, line 70        |
| <pre> theorem Hypernet.Valuation.wireQ_isSome {H : Hypernet} (σ : H.Valuation)   {w : Wire} {τ : TensorType} (ht : H.wireType w = some τ)   (hfin : τ.dtype.ext = Ext.fin) : ∃ q, σ.wireQ w = some q </pre>                                                                   |                                               |
| PPLC.QTensor. <b>rel2_symm</b>                                                                                                                                                                                                                                                | lean/qtsl/Qtsl/Soundness.lean, line 6789      |
| <pre> theorem QTensor.rel2_symm {r : Q → Q → Prop}   (hr : ∀ x y, r x y → r y x) {a b : QTensor}   (h : QTensor.rel2 r a b) : QTensor.rel2 r b a </pre>                                                                                                                       |                                               |
| PPLC. <b>sound_eqSym</b>                                                                                                                                                                                                                                                      | lean/qtsl/Qtsl/Soundness.lean, line 6863      |
| <pre> theorem sound_eqSym {H : Hypernet} {seq premise : Sequent} {tT tU : Term}   (ih : premise.semanticallyValid H)   (hpre : seq.pre = premise.pre)   (hprePost : premise.post = .atom .eq tT tU)   (hpost : seq.post = .atom .eq tU tT) :   seq.semanticallyValid H </pre> |                                               |
| PPLC. <b>PEqSymData</b>                                                                                                                                                                                                                                                       | lean/qtsl/Qtsl/Core/Catalogue.lean, line 1443 |
| <pre> inductive PEqSymData : Hypernet → Sequent → Type where ... </pre>                                                                                                                                                                                                       |                                               |
| PPLC.QTensor. <b>rel2_conj</b>                                                                                                                                                                                                                                                | lean/qtsl/Qtsl/Soundness.lean, line 6834      |
| <pre> theorem QTensor.rel2_conj {r s t : Q → Q → Prop}   (hrst : ∀ x y, r x y → s x y → t x y) {a b : QTensor}   (hr : QTensor.rel2 r a b) (hs : QTensor.rel2 s a b) :   QTensor.rel2 t a b </pre>                                                                            |                                               |
| PPLC. <b>sound_antisym</b>                                                                                                                                                                                                                                                    | lean/qtsl/Qtsl/Soundness.lean, line 6888      |
| <pre> theorem sound_antisym {H : Hypernet} {seq premiseLe premiseGe : Sequent}   {tT tU : Term}   (ihLe : premiseLe.semanticallyValid H)   (ihGe : premiseGe.semanticallyValid H) </pre>                                                                                      |                                               |

```

(hpreLe : seq.pre = premiseLe.pre)
(hpreGe : seq.pre = premiseGe.pre)
(hpreLePost : premiseLe.post = .atom .le tT tU)
(hpreGePost : premiseGe.post = .atom .ge tT tU)
(hpost : seq.post = .atom .eq tT tU) :
seq.semanticallyValid H

```

**PPLC.baryCoeffs1D** lean/qtsl/Qtsl/Core/Catalogue.lean, line 584

```

def baryCoeffs1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool) :
Option (List (Fin cx.nVertices × Rat × Rat))

```

**PPLC.primAffine** lean/qtsl/Qtsl/Core/Catalogue.lean, line 599

```

def primAffine (s c : Rat) : Rat × Rat

```

**PPLC.f4VertexAtom** lean/qtsl/Qtsl/Core/Catalogue.lean, line 612

```

def f4VertexAtom (inS : Bool) (off slp : Rat) (t : Term) : Formula

```

**PPLC.barycentricHalfspaceBuild** lean/qtsl/Qtsl/Core/Catalogue.lean, line 616

```

def barycentricHalfspaceBuild (t : Term) (simplex : Simplex)
(S : ExtensionSet) : Option Formula

```

**PPLC.barycentricHalfspaceForm** lean/qtsl/Qtsl/Core/Catalogue.lean, line 625

```

def barycentricHalfspaceForm (t : Term) (simplex : Simplex)
(S : ExtensionSet) (F : Formula) : Prop

```

**PPLC.barycentricHalfspaceBuild\_spec** lean/qtsl/Qtsl/Soundness.lean, line 634

```

theorem barycentricHalfspaceBuild_spec {t : Term} {simplex : Simplex}
{S : ExtensionSet} {F : Formula}
(h : barycentricHalfspaceBuild t simplex S = some F) :
S.simplex = simplex ∧ simplex.complex.ambDim = 1 ∧
∃ (a b : Fin simplex.complex.nVertices) (V₀ V₁ : ℚ),
(List.finRange simplex.complex.nVertices).filter
(fun v => simplex.contains v) = [a, b] ∧
(∀ j, simplex.complex.coord a j = V₀) ∧
(∀ j, simplex.complex.coord b j = V₁) ∧
V₀ ≠ V₁ ∧
F = .and
(f4VertexAtom (S.containsNat a.val)
(V₁ / (V₁ - V₀)) (-1 / (V₁ - V₀)) t)
(f4VertexAtom (S.containsNat b.val)
(-V₀ / (V₁ - V₀)) (1 / (V₁ - V₀)) t)

```

**PPLC.sound\_convElim** lean/qtsl/Qtsl/Soundness.lean, line 4672

```

theorem sound_convElim {H : Hypernet} {seq premise : Sequent}
{term : Term} {simplex : Simplex} {S : ExtensionSet}
(ih : premise.semanticallyValid H)
(hpre : seq.pre = premise.pre)
(hpremiseForm : premise.post = .inRegion term simplex S)
(hhalf : barycentricHalfspaceForm term simplex S seq.post) :
seq.semanticallyValid H

```

**PPLC.f4VertexAtom\_holds\_iff** lean/qtsl/Qtsl/Soundness.lean, line 1393

```

theorem f4VertexAtom_holds_iff {H : Hypernet} {σ : H.Valuation} {t : Term}
{q : QTensor} (hq : Term.evalQ σ t = some q) (hsq : SingletonShape q.shape)
(inS : Bool) (off slp : ℚ) :
Formula.holds σ (f4VertexAtom inS off slp t) ↔

```



```

      (if inS then off + slp * q.scalar ≤ 0
       else 0 ≤ off + slp * q.scalar)

```

PPLC.maskRegion\_pair\_elim

lean/qtsl/Qtsl/Soundness.lean, line 2526

```

theorem maskRegion_pair_elim {C : SimpC} {mask ext : Fin C.nVertices → Bool}
  {x : Fin C.ambDim → ℚ} {a b : Fin C.nVertices} {V₀ V₁ : ℚ}
  (hfil : (List.finRange C.nVertices).filter (fun v => mask v) = [a, b])
  (hVa : ∀ j, C.coord a j = V₀) (hVb : ∀ j, C.coord b j = V₁)
  (hne : V₀ ≠ V₁) (j0 : Fin C.ambDim)
  (h : MaskRegion C mask ext x) :
  (ext a = true → (V₁ - x j0) / (V₁ - V₀) ≤ 0) ∧
  (ext a = false → 0 ≤ (V₁ - x j0) / (V₁ - V₀)) ∧
  (ext b = true → (x j0 - V₀) / (V₁ - V₀) ≤ 0) ∧
  (ext b = false → 0 ≤ (x j0 - V₀) / (V₁ - V₀))

```

PPLC.sound\_convIntro

lean/qtsl/Qtsl/Soundness.lean, line 4593

```

theorem sound_convIntro {H : Hypernet} {seq premise : Sequent}
  {term : Term} {simplex : Simplex} {S : ExtensionSet}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (htermShape : term.bshape H = some [simplex.complex.ambDim])
  (hhalf : barycentricHalfspaceForm term simplex S premise.post)
  (hpost : seq.post = .inRegion term simplex S) :
  seq.semanticallyValid H

```

PPLC.f4VertexAtom\_holds\_inv

lean/qtsl/Qtsl/Soundness.lean, line 1442

```

theorem f4VertexAtom_holds_inv {H : Hypernet} {σ : H.Valuation} {t : Term}
  {inS : Bool} {off slp : ℚ}
  (h : Formula.holds σ (f4VertexAtom inS off slp t)) :
  ∃ q, Term.evalQ σ t = some q

```

PPLC.maskRegion\_pair\_intro

lean/qtsl/Qtsl/Soundness.lean, line 2575

```

theorem maskRegion_pair_intro {C : SimpC} {mask ext : Fin C.nVertices → Bool}
  {x : Fin C.ambDim → ℚ} {a b : Fin C.nVertices} {V₀ V₁ : ℚ}
  (hfil : (List.finRange C.nVertices).filter (fun v => mask v) = [a, b])
  (hVa : ∀ j, C.coord a j = V₀) (hVb : ∀ j, C.coord b j = V₁)
  (hne : V₀ ≠ V₁) (hdim : C.ambDim = 1) (j0 : Fin C.ambDim)
  (hw0t : ext a = true → (V₁ - x j0) / (V₁ - V₀) ≤ 0)
  (hw0f : ext a = false → 0 ≤ (V₁ - x j0) / (V₁ - V₀))
  (hw1t : ext b = true → (x j0 - V₀) / (V₁ - V₀) ≤ 0)
  (hw1f : ext b = false → 0 ≤ (x j0 - V₀) / (V₁ - V₀)) :
  MaskRegion C mask ext x

```

PPLC.sound\_arithAtom\_fold

lean/qtsl/Qtsl/Soundness.lean, line 4297

```

theorem sound_arithAtom_fold {H : Hypernet} {seq premise : Sequent}
  {termL termR : Term} {b : BowTie}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hprem : atomicCompareZero b termL termR premise.post)
  (hconcl : atomicCompareForm b termL termR seq.post) :
  seq.semanticallyValid H

```

PPLC.sound\_arithAtom\_unfold

lean/qtsl/Qtsl/Soundness.lean, line 4318

```

theorem sound_arithAtom_unfold {H : Hypernet} {seq premise : Sequent}
  {termL termR : Term} {b : BowTie}
  (ih : premise.semanticallyValid H)
  (hpre : seq.pre = premise.pre)
  (hprem : atomicCompareForm b termL termR premise.post)

```

```

    (hconcl : atomicCompareZero b termL termR seq.post) :
    seq.semanticallyValid H

```

**PPLC.atomicCompareForm** lean/qtsl/Qtsl/Core/Catalogue.lean, line 553

```

def atomicCompareForm (b : BowTie) (l r : Term) (F : Formula) : Prop

```

**PPLC.atomicCompareZero** lean/qtsl/Qtsl/Core/Catalogue.lean, line 556

```

def atomicCompareZero (b : BowTie) (l r : Term) (F : Formula) : Prop

```

**PPLC.QTensor.map2\_defined\_of\_rel2** lean/qtsl/Qtsl/Soundness.lean, line 3462

```

theorem QTensor.map2_defined_of_rel2 {r : Q → Q → Prop} {a b : QTensor}
  (h : QTensor.rel2 r a b) (f : Q → Q → Q) :
  ∃ c, QTensor.map2 f a b = some c

```

**PPLC.BowTie** lean/qtsl/Qtsl/Core/Catalogue.lean, line 346

```

inductive BowTie where ...

```

**PPLC.QTensor.rel2\_sub\_zero\_iff** lean/qtsl/Qtsl/Soundness.lean, line 2466

```

theorem QTensor.rel2_sub_zero_iff {b : BowTie} {al ar d : QTensor}
  (hd : QTensor.map2 (· - ·) al ar = some d) :
  QTensor.rel2 (BowTie.holdsQ b) d {[], [0], rfl}
  ↔ QTensor.rel2 (BowTie.holdsQ b) al ar

```

**PPLC.Term.evalQ\_sub\_inv** lean/qtsl/Qtsl/Soundness.lean, line 1306

```

theorem Term.evalQ_sub_inv {H : Hypernet} (σ : H.Valuation) {l r : Term}
  {q : QTensor} (h : Term.evalQ σ (.sub l r) = some q) :
  ∃ a b, Term.evalQ σ l = some a ∧ Term.evalQ σ r = some b ∧
  QTensor.map2 (· - ·) a b = some q

```

**PPLC.PArithAtomData** lean/qtsl/Qtsl/Core/Catalogue.lean, line 1471

```

inductive PArithAtomData : Hypernet → Sequent → Type where ...

```

**PPLC.allForwardDiffBernCoeffsSatisfyP** lean/qtsl/Qtsl/Core/Catalogue.lean, line 760

```

def allForwardDiffBernCoeffsSatisfyP
  (simplex : Simplex) (S : ExtensionSet) (degree : Nat)
  (polynomial : Term) (predicate : Formula) : Prop

```

**PPLC.satisfiesPredQ** lean/qtsl/Qtsl/Core/Catalogue.lean, line 701

```

def satisfiesPredQ (P : Formula) (c : Rat) : Bool

```

**PPLC.bernConeCoeffCheck** lean/qtsl/Qtsl/Core/Catalogue.lean, line 727

```

def bernConeCoeffCheck (simplex : Simplex) (S : ExtensionSet) (degree : Nat)
  (polynomial : Term) (predicate : Formula) : Bool

```

**PPLC.sound\_bernCone** lean/qtsl/Qtsl/Soundness.lean, line 5498

```

theorem sound_bernCone {H : Hypernet} {seq regionPremise : Sequent}
  {termX : Term} {simplex : Simplex} {S : ExtensionSet} {degree : Nat}
  {polynomial : Term} {predicate : Formula}
  (ih : regionPremise.semanticallyValid H)
  (hpre : seq.pre = regionPremise.pre)
  (hregionPost : regionPremise.post = .inRegion termX simplex S)
  (hpolyTie : polynomial.freeWires = [] v
    ∃ w, termX = Term.var w ∧ polynomial.freeWires ⊆ [w])
  (hpredShape : (¬ isExtensionSetEmpty S → isConicPredicate predicate)

```

```

     $\wedge$  (isExtensionSetEmpty S  $\rightarrow$  isConvexPredicate predicate))
  (hcoeff : allForwardDiffBernCoeffsSatisfyP simplex S degree polynomial
    predicate)
  (hpostApply : applyPredicateToTerm predicate polynomial seq.post) :
  seq.semanticallyValid H

```

PPLC.**sound\_bernCone\_empty**

lean/qtsl/Qtsl/Soundness.lean, line 5058

```

theorem sound_bernCone_empty {H : Hypernet} {seq regionPremise : Sequent}
  {termX : Term} {simplex : Simplex} {S : ExtensionSet} {degree : Nat}
  {polynomial : Term} {predicate : Formula}
  (ih : regionPremise.semanticallyValid H)
  (hpre : seq.pre = regionPremise.pre)
  (hregionPost : regionPremise.post = .inRegion termX simplex S)
  (hpolyTie : polynomial.freeWires = []  $\vee$ 
     $\exists$  w, termX = Term.var w  $\wedge$  polynomial.freeWires  $\subseteq$  [w])
  (hconvex : isExtensionSetEmpty S  $\rightarrow$  isConvexPredicate predicate)
  (hSempty : isExtensionSetEmpty S)
  (hcoeff : allForwardDiffBernCoeffsSatisfyP simplex S degree polynomial
    predicate)
  (hpostApply : applyPredicateToTerm predicate polynomial seq.post) :
  seq.semanticallyValid H

```

PPLC.**MaskRegion**

lean/qtsl/Qtsl/Catalogue.lean, line 376

```

def MaskRegion (C : SimpC) (mask ext : Fin C.nVertices  $\rightarrow$  Bool)
  (x : Fin C.ambDim  $\rightarrow$   $\mathbb{Q}$ ) : Prop

```

PPLC.Term.**evalQ\_brRead**

lean/qtsl/Qtsl/Soundness.lean, line 1578

```

theorem Term.evalQ_brRead {H : Hypernet} ( $\sigma$  : H.Valuation) (s : Shape) :
   $\forall$  (t : Term),
  ( $\forall$  w  $\in$  t.freeWires,  $\exists$  qw,  $\sigma$ .wireQ w = some qw  $\wedge$ 
    (qw.shape = []  $\vee$  qw.shape = s))  $\rightarrow$ 
  t.idxFree = true  $\rightarrow$ 
   $\exists$  q, Term.evalQ  $\sigma$  t = some q  $\wedge$  (q.shape = []  $\vee$  q.shape = s)  $\wedge$ 
    (q.shape = []  $\rightarrow$   $\forall$  w  $\in$  t.freeWires,  $\forall$  qw,  $\sigma$ .wireQ w = some qw  $\rightarrow$ 
      qw.shape = [])  $\wedge$ 
   $\forall$  i, i < Shape.totalSize s  $\rightarrow$ 
    q.brRead i = QArith.Poly.eval ( $\sigma$ .entryVal i) t.toPoly

```

PPLC.**maskRegion\_rat\_weights**

lean/qtsl/Qtsl/Soundness.lean, line 4798

```

theorem maskRegion_rat_weights {C : SimpC}
  {mask ext : Fin C.nVertices  $\rightarrow$  Bool} {x : Fin C.ambDim  $\rightarrow$   $\mathbb{Q}$ }
  (hmask : MaskRegion C mask ext x)
  (hext0 :  $\forall$  v, ext v = false) :
   $\exists$  qw : Fin ((List.finRange C.nVertices).filter
    (fun v  $\Rightarrow$  mask v)).length  $\rightarrow$   $\mathbb{Q}$ ,
  ( $\forall$  i,  $0 \leq$  qw i)  $\wedge$  ( $\sum$  i, qw i) = 1  $\wedge$ 
   $\forall$  j : Fin C.ambDim,
    ( $\sum$  i, qw i * C.coord ((List.finRange C.nVertices).filter
      (fun v  $\Rightarrow$  mask v)).get i) j) = x j

```

PPLC.ConvexRat.**exists\_nonneg\_rat\_solution**

lean/qtsl/Qtsl/ConvexRat.lean, line 269

```

theorem exists_nonneg_rat_solution (A : Fin k  $\rightarrow$  Fin m  $\rightarrow$   $\mathbb{Q}$ )
  (b : Fin m  $\rightarrow$   $\mathbb{Q}$ ) (c : Fin k  $\rightarrow$   $\mathbb{R}$ ) (hc0 :  $\forall$  i,  $0 \leq$  c i)
  (hcsol :  $\forall$  j, ( $\sum$  i, c i * (A i j :  $\mathbb{R}$ )) = (b j :  $\mathbb{R}$ )) :
   $\exists$  q : Fin k  $\rightarrow$   $\mathbb{Q}$ , ( $\forall$  i,  $0 \leq$  q i)  $\wedge$   $\forall$  j, ( $\sum$  i, q i * A i j) = b j

```

PPLC.**classifyPred**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 667

```

def classifyPred : Formula → Option PredClass
| .and l r => do
  let a ← classifyPred l
  let b ← classifyPred r
  pure (a.conic && b.conic, a.upClosed && b.upClosed)
| f => classifyPredAtom f

def isConicPredicate (P : Formula) : Prop

```

PPLC.satisfiesPredQ\_convex

lean/qtsl/Qtsl/Soundness.lean, line 2871

```

theorem satisfiesPredQ_convex {P : Formula} {cls : PredClass}
(hP : classifyPred P = some cls) :
  ∀ a₁ b₁ l : ℚ, 0 ≤ l → l ≤ 1 →
    satisfiesPredQ P a₁ = true → satisfiesPredQ P b₁ = true →
    satisfiesPredQ P (l * a₁ + (1 - l) * b₁) = true

```

PPLC.QArith.fdBernstein\_sound\_dim1

lean/qtsl/Qtsl/QArithKernel.lean, line 2219

```

theorem fdBernstein_sound_dim1
(verts : List (List ℚ)) (smask : List Bool) (d : Nat) (p : Poly)
(t : FDataTable) (pred : ℚ → Bool) (u₀ : ℚ)
(hmask : smask = [true, false] ∨ smask = [false, true])
(hfd : fdBernstein verts smask d p = some t)
(hgrid : gridAll pred t = true)
(hu : 0 ≤ u₀)
(hzero : pred 0 = true)
(hscale : ∀ a c : ℚ, 0 ≤ c → pred a = true → pred (c * a) = true)
(hconv : ∀ a b l : ℚ, 0 ≤ l → l ≤ 1 → pred a = true → pred b = true →
  pred (l * a + (1 - l) * b) = true) :
pred (Poly.eval (fun j =>
  ((List.range verts.length).map
    fun i => (verts.getD i []).getD j 0 *
      (if smask.getD i false then -u₀ else 1 + u₀)).sum) p) = true

```

PPLC.QArith.fdBernstein\_sound\_const

lean/qtsl/Qtsl/QArithKernel.lean, line 1733

```

theorem fdBernstein_sound_const
(verts : List (List ℚ)) (smask : List Bool) (d : Nat) (c₀ : ℚ)
(t : FDataTable) (pred : ℚ → Bool)
(hfd : fdBernstein verts smask d (Poly.const c₀) = some t)
(hgrid : gridAll pred t = true) :
pred c₀ = true

```

PPLC.satisfiesPredQ\_conic\_zero

lean/qtsl/Qtsl/Soundness.lean, line 3066

```

theorem satisfiesPredQ_conic_zero {P : Formula} {cls : PredClass}
(hP : classifyPred P = some cls) (hc : cls.conic = true) :
  satisfiesPredQ P 0 = true

```

PPLC.satisfiesPredQ\_conic\_scale

lean/qtsl/Qtsl/Soundness.lean, line 3102

```

theorem satisfiesPredQ_conic_scale {P : Formula} {cls : PredClass}
(hP : classifyPred P = some cls) (hc : cls.conic = true) :
  ∀ a c : ℚ, 0 ≤ c → satisfiesPredQ P a = true →
    satisfiesPredQ P (c * a) = true

```

PPLC.QArith.fdBernstein

lean/qtsl/Qtsl/Core/QArith.lean, line 158

```

def fdBernstein (verts : List (List Rat)) (smask : List Bool) (d : Nat)
(p : Poly) : Option FDataTable

```

PPLC.QArith.gridAll

lean/qtsl/Qtsl/Core/QArith.lean, line 190

```
def gridAll (pred : Rat → Bool) (t : FDataTable) : Bool
```

PPLC.Term.idxBounded

lean/qtsl/Qtsl/Core/Catalogue.lean, line 478

```
def Term.idxBounded (n : Nat) : Term → Bool
| .var _ | .const _ => true
| .add l r | .mul l r | .sub l r => l.idxBounded n && r.idxBounded n
| .neg t => t.idxBounded n
| .idx t i => decide (i < n) && t.idxBounded n

def Formula.freeWires : Formula → List Wire
| .atom _ l r => l.freeWires ++ r.freeWires
| .inRegion t _ => t.freeWires
| .and l r | .or l r => l.freeWires ++ r.freeWires
| .tt | .ff => []

def Term.type (H : Hypernet) : Term → Option TensorType
| .var w => H.wireType w
| .const _ => some ([], Dtype.Q)
| .idx t i =>

  match Term.type H t with
  | some τ =>
    match τ.shape with
    | [e] => if i < e then some ([], τ.dtype) else none
    | _ => none
  | none => none
| .add l r | .mul l r | .sub l r =>
  match Term.type H l, Term.type H r with
  | some τ1, some τ2 => if τ1 = τ2 then some τ1 else none
  | _, _ => none
| .neg t => Term.type H t

def Formula.WellTyped (H : Hypernet) : Formula → Prop
| .atom _ l r => ∃ τ, l.type H = some τ ∧ r.type H = some τ
| .inRegion t simplex S =>
  (∃ τ, t.type H = some τ ∧ τ.shape = [simplex.complex.ambDim]) ∧
  S.simplex = simplex
| .and l r => l.WellTyped H ∧ r.WellTyped H
| .or l r => l.WellTyped H ∧ r.WellTyped H
| .tt => True
| .ff => True

def Term.bshape (H : Hypernet) : Term → Option Shape
...
```

PPLC.applyPredicateToTerm

lean/qtsl/Qtsl/Core/Catalogue.lean, line 631

```
def applyPredicateToTerm (P : Formula) (arg : Term) (F : Formula) : Prop
```

PPLC.holds\_substHole\_of\_satisfies

lean/qtsl/Qtsl/Soundness.lean, line 3144

```
theorem holds_substHole_of_satisfies {H : Hypernet} (σ : H.Valuation)
  {P : Formula} {cls : PredClass} (hP : classifyPred P = some cls)
  {arg : Term} {qa : QTensor}
  (ha : Term.evalQ σ arg = some qa) (hsa : SingletonShape qa.shape)
  (hsat : satisfiesPredQ P qa.scalar = true) :
  Formula.holds σ (Formula.subst predicateHole arg P)
```

PPLC.ConeLemma.StarIdentity

lean/qtsl/Qtsl/ConeLemma.lean, line 18

```
def StarIdentity : Prop
```

PPLC.PNormData

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1505

```
inductive PNormData : Hypernet → Sequent → Type where ...
```

PPLC.**sound\_norm**

lean/qtsl/Qtsl/Soundness.lean, line 4494

```
theorem sound_norm {H : Hypernet} {seq : Sequent} {termL termR : Term}
  (hcanon : polynomialCanonicalEq termL termR)
  (htyEq : termL.bshape H = termR.bshape H)
  (htyDef : ∃ s, termL.bshape H = some s)
  (hfinL : termL.finWires H) (hfinR : termR.finWires H)
  (hpost : seq.post = .atom .eq termL termR) :
  seq.semanticallyValid H
```

PPLC.**polynomialCanonicalEq**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 698

```
def polynomialCanonicalEq (l r : Term) : Prop
```

PPLC.Term.**toPoly**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 414

```
def Term.toPoly : Term → QArith.Poly
| .var w => QArith.Poly.var w
| .const c => QArith.Poly.const c
| .add l r => QArith.Poly.add l.toPoly r.toPoly
| .mul l r => QArith.Poly.mul l.toPoly r.toPoly
| .sub l r => QArith.Poly.sub l.toPoly r.toPoly
| .neg t => QArith.Poly.neg t.toPoly

| .idx _ _ => QArith.Poly.const 0

def Term.evalQ? : Term → Option Rat
| .var _ => none
| .const c => some c
| .add l r => do pure ((← l.evalQ?) + (← r.evalQ?))
| .mul l r => do pure ((← l.evalQ?) * (← r.evalQ?))
| .sub l r => do pure ((← l.evalQ?) - (← r.evalQ?))
| .neg t => do pure (-(← t.evalQ?))
| .idx _ _ => none

def BowTie.evalQ : BowTie → Rat → Rat → Bool
| .lt, x, y => decide (x < y)
| .le, x, y => decide (x ≤ y)
| .eq, x, y => decide (x = y)
| .ge, x, y => decide (y ≤ x)
| .gt, x, y => decide (y < x)
| .ne, x, y => decide (¬ x = y)

def Formula.evalClosed : Formula → Option Bool
| .atom b l r => do pure (b.evalQ (← l.evalQ?) (← r.evalQ?))
| .inRegion _ _ _ => none
| .and l r => do pure ((← l.evalClosed) && (← r.evalClosed))
| .or l r => do pure ((← l.evalClosed) || (← r.evalClosed))
| .tt => some true
| .ff => some false

def baryLambdas1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool)
  (t : Term) : Option (List (Fin cx.nVertices × Term))
```

PPLC.Term.**idxFree**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 472

```
def Term.idxFree : Term → Bool
| .var _ | .const _ => true
| .add l r | .mul l r | .sub l r => l.idxFree && r.idxFree
| .neg t => t.idxFree
| .idx _ _ => false
```

```

def Term.idxBounded (n : Nat) : Term → Bool
| .var _ | .const _ => true
| .add l r | .mul l r | .sub l r => l.idxBounded n && r.idxBounded n
| .neg t => t.idxBounded n
| .idx t i => decide (i < n) && t.idxBounded n

def Formula.freeWires : Formula → List Wire
| .atom _ l r => l.freeWires ++ r.freeWires
| .inRegion t _ => t.freeWires
| .and l r | .or l r => l.freeWires ++ r.freeWires
| .tt | .ff => []

def Term.type (H : Hypernet) : Term → Option TensorType
| .var w => H.wireType w
| .const _ => some ([], Dtype.Q)
| .idx t i =>

  match Term.type H t with
  | some τ =>
    match τ.shape with
    | [e] => if i < e then some ([], τ.dtype) else none
    | _ => none
  | none => none
| .add l r | .mul l r | .sub l r =>
  match Term.type H l, Term.type H r with
  | some τ1, some τ2 => if τ1 = τ2 then some τ1 else none
  | _, _ => none
| .neg t => Term.type H t

def Formula.WellTyped (H : Hypernet) : Formula → Prop
| .atom _ l r => ∃ τ, l.type H = some τ ∧ r.type H = some τ
| .inRegion t simplex S =>
  (∃ τ, t.type H = some τ ∧ τ.shape = [simplex.complex.ambDim]) ∧
  S.simplex = simplex
...

```

**PPLC.farkasCombinationValid**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 918

```

def farkasCombinationValid (cc : Term) (premise : Formula)
  (conclusion : Term) : Prop

```

**PPLC.farkasEntries**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 778

```

def farkasEntries : Term → Option (List (Rat × Term))
| .const c => if c = 0 then some [] else none
| .add (.mul (.const l) e) rest =>
  (farkasEntries rest).map ((l, e) :: ·)
| _ => none

def wireCoordKey (w i : Nat) : Nat

```

**PPLC.atomLeZeroPolys**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 769

```

def atomLeZeroPolys : Formula → List QArith.Poly
| .atom .le l r => [QArith.Poly.sub l.toPoly r.toPoly]
| .atom .lt l r => [QArith.Poly.sub l.toPoly r.toPoly]
| .atom .ge l r => [QArith.Poly.sub r.toPoly l.toPoly]
| .atom .gt l r => [QArith.Poly.sub r.toPoly l.toPoly]
| .atom .eq l r => [QArith.Poly.sub l.toPoly r.toPoly,
  QArith.Poly.sub r.toPoly l.toPoly]
| _ => []

```



```

def farkasEntries : Term → Option (List (Rat × Term))
| .const c => if c = 0 then some [] else none
| .add (.mul (.const l) e) rest =>
  (farkasEntries rest).map ((l, e) :: ·)
| _ => none

def wireCoordKey (w i : Nat) : Nat

```

PPLC.**farkasCheckCore**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 899

```

def farkasCheckCore (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

PPLC.**farkasCheck**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 914

```

def farkasCheck (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

PPLC.**sound\_farkas**

lean/qtsl/Qtsl/Soundness.lean, line 4337

```

theorem sound_farkas {H : Hypernet} {seq : Sequent}
{combinationCoeffs conclusion : Term}
(hshape : ∃ s, conclusion.bshape H = some s)
(hfin : conclusion.finWires H)
(hwpos : ∀ w ∈ seq.pre.freeWires ++ conclusion.freeWires,
  ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape)
(hvalid : farkasCombinationValid combinationCoeffs seq.pre conclusion)
(hpost : seq.post = .atom .le conclusion (.const 0)) :
seq.semanticallyValid H

```

PPLC.**atomLeZeroPolysW**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 800

```

def atomLeZeroPolysW : Formula → List QArith.Poly
| .atom op l r =>
  match l.asCoordPolyIdxW? with
  | none => []
  | some pl =>
    match r.asCoordPolyIdxW? with
    | none => []
    | some pr =>
      match op with
      | .le | .lt => [QArith.Poly.sub pl pr]
      | .ge | .gt => [QArith.Poly.sub pr pl]
      | .eq => [QArith.Poly.sub pl pr, QArith.Poly.sub pr pl]
      | .ne => []
  | _ => []

def farkasEntriesW : List (Rat × Term) → Option (List (Rat × QArith.Poly))
| [] => some []
| (l, e) :: rest =>
  match e.asCoordPolyIdxW?, farkasEntriesW rest with
  | some pe, some prest => some ((l, pe) :: prest)
  | _, _ => none

def farkasCheckW (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

PPLC.**atomLeZeroPolysM**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 854

```

def atomLeZeroPolysM : Formula → List QArith.Poly
| .atom op l r =>
  match l.asCoordPolyMixed? with
  | none => []
  | some pl =>
    match r.asCoordPolyMixed? with
    | none => []
    | some pr =>

```

```

      match op with
      | .le | .lt => [QArith.Poly.sub pl pr]
      | .ge | .gt => [QArith.Poly.sub pr pl]
      | .eq => [QArith.Poly.sub pl pr, QArith.Poly.sub pr pl]
      | .ne => []
    | _ => []

def farkasEntriesM : List (Rat × Term) → Option (List (Rat × QArith.Poly))
| [] => some []
| (l, e) :: rest =>
  match e.asCoordPolyMixed?, farkasEntriesM rest with
  | some pe, some prest => some ((l, pe) :: prest)
  | _, _ => none

def farkasCheckM (cc : Term) (premise : Formula) (conclusion : Term) : Bool

```

PPLC.**farkasCheckW** lean/qtsl/Qtsl/Core/Catalogue.lean, line 822

```
def farkasCheckW (cc : Term) (premise : Formula) (conclusion : Term) : Bool
```

PPLC.**farkasCheckM** lean/qtsl/Qtsl/Core/Catalogue.lean, line 876

```
def farkasCheckM (cc : Term) (premise : Formula) (conclusion : Term) : Bool
```

PPLC.**wireCoordKey** lean/qtsl/Qtsl/Core/Catalogue.lean, line 784

```
def wireCoordKey (w i : Nat) : Nat
```

PPLC.**farkas\_fold\_eval\_nonpos** lean/qtsl/Qtsl/Soundness.lean, line 3445

```

theorem farkas_fold_eval_nonpos (p : Nat → ℚ) :
  ∀ (entries : List (ℚ × Term)),
  (∀ e ∈ entries, 0 ≤ e.1 ∧ QArith.Poly.eval p e.2.toPoly ≤ 0) →
  QArith.Poly.eval p
  (entries.foldr
    (fun (le : ℚ × Term) acc =>
      QArith.Poly.add (QArith.Poly.scale le.1 le.2.toPoly) acc) []) ≤ 0

```

PPLC.**farkas\_fold\_eval\_nonpos\_polys** lean/qtsl/Qtsl/Soundness.lean, line 3830

```

theorem farkas_fold_eval_nonpos_polys (p : Nat → ℚ) :
  ∀ (pentries : List (ℚ × QArith.Poly)),
  (∀ e ∈ pentries, 0 ≤ e.1 ∧ QArith.Poly.eval p e.2 ≤ 0) →
  QArith.Poly.eval p
  (pentries.foldr
    (fun (le : ℚ × QArith.Poly) acc =>
      QArith.Poly.add (QArith.Poly.scale le.1 le.2) acc) []) ≤ 0

```

PPLC.**PFarkasData** lean/qtsl/Qtsl/Core/Catalogue.lean, line 1517

```
inductive PFarkasData : Hypernet → Sequent → Type where ...
```

PPLC.**coverInfeasibilityValid** lean/qtsl/Qtsl/Core/Catalogue.lean, line 1151

```

def coverInfeasibilityValid
  (_termX : Term) (simplex : Simplex) (S : ExtensionSet)
  (nBranches : Nat)
  (branchSimplex : Fin nBranches → Simplex)
  (branchS : Fin nBranches → ExtensionSet)
  (_coverCert : Term) : Prop

```

PPLC.**coverExtendPass** lean/qtsl/Qtsl/Core/Catalogue.lean, line 1127

```
def coverExtendPass (ivs : List (Rat × Rat)) (r : Rat) : Rat
```

**PPLC.coverReach**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1130

```

def coverReach : Nat → List (Rat × Rat) → Rat → Rat
| 0, _, r => r
| n + 1, ivs, r => coverReach n ivs (coverExtendPass ivs r)

def coverChainCheck (ivs : List (Rat × Rat)) (lo hi : Rat) : Bool

```

**PPLC.coverChainCheck**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1134

```

def coverChainCheck (ivs : List (Rat × Rat)) (lo hi : Rat) : Bool

```

**PPLC.coverCheck**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1137

```

def coverCheck (simplex : Simplex) (S : ExtensionSet) (nBranches : Nat)
  (branchSimplex : Fin nBranches → Simplex)
  (branchS : Fin nBranches → ExtensionSet) : Bool

```

**PPLC.sound\_subdivide**

lean/qtsl/Qtsl/Soundness.lean, line 7758

```

theorem sound_subdivide {H : Hypernet} {seq regionPremise : Sequent}
  {nBranches : Nat} {branchPremise : Fin nBranches → Sequent}
  {termX : Term} {simplex : Simplex} {S : ExtensionSet}
  {branchSimplex : Fin nBranches → Simplex}
  {branchS : Fin nBranches → ExtensionSet}
  {coverCertificate : Term}
  (ihRegion : regionPremise.semanticallyValid H)
  (ihBranch : ∀ k, (branchPremise k).semanticallyValid H)
  (hpre : seq.pre = regionPremise.pre)
  (hregionPost : regionPremise.post = .inRegion termX simplex S)
  (hbranchPre : ∀ k, (branchPremise k).pre
    = .and regionPremise.pre
      (.inRegion termX (branchSimplex k) (branchS k)))
  (hbranchPost : ∀ k, (branchPremise k).post = seq.post)
  (hcover : coverInfeasibilityValid termX simplex S nBranches
    branchSimplex branchS coverCertificate) :
  seq.semanticallyValid H

```

**PPLC.interval1D\_elim**

lean/qtsl/Qtsl/Soundness.lean, line 7642

```

theorem interval1D_elim {S : ExtensionSet} {lo hi : ℚ}
  (hint : S.interval1D? = some (lo, hi))
  {x : Fin S.simplex.complex.ambDim → ℚ}
  (hmask : MaskRegion S.simplex.complex S.simplex.contains S.contains x)
  (j0 : Fin S.simplex.complex.ambDim) :
  lo ≤ x j0 ∧ x j0 ≤ hi ∧ lo < hi

```

**PPLC.interval1D\_intro**

lean/qtsl/Qtsl/Soundness.lean, line 7676

```

theorem interval1D_intro {S : ExtensionSet} {lo hi : ℚ}
  (hint : S.interval1D? = some (lo, hi))
  {x : Fin S.simplex.complex.ambDim → ℚ}
  (j0 : Fin S.simplex.complex.ambDim)
  (hxlo : lo ≤ x j0) (hxhi : x j0 ≤ hi) :
  MaskRegion S.simplex.complex S.simplex.contains S.contains x

```

**PPLC.ChainInv**

lean/qtsl/Qtsl/Soundness.lean, line 7700

```

def ChainInv (ivs : List (ℚ × ℚ)) (lo r : ℚ) : Prop

```

**PPLC.chainInv\_extendPass**

lean/qtsl/Qtsl/Soundness.lean, line 7704

```

theorem chainInv_extendPass (ivs : List (ℚ × ℚ)) (lo : ℚ) :
  ∀ (sub : List (ℚ × ℚ)), (∀ iv ∈ sub, iv ∈ ivs) →

```

```

    ∀ (r : Q), ChainInv ivs lo r →
    ChainInv ivs lo (coverExtendPass sub r)

```

PPLC.chainInv\_reach

lean/qtsl/Qtsl/Soundness.lean, line 7735

```

theorem chainInv_reach (ivs : List (Q × Q)) (lo : Q) :
  ∀ (n : Nat) (r : Q), ChainInv ivs lo r →
  ChainInv ivs lo (coverReach n ivs r)

```

PPLC.coverChain\_covers

lean/qtsl/Qtsl/Soundness.lean, line 7745

```

theorem coverChain_covers {ivs : List (Q × Q)} {lo hi : Q} (hlt : lo < hi)
  (hcheck : coverChainCheck ivs lo hi = true) :
  ∀ x : Q, lo ≤ x → x ≤ hi → ∃ iv ∈ ivs, iv.1 ≤ x ∧ x ≤ iv.2

```

PPLC.PSubdivideData

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1533

```

inductive PSubdivideData : Hypernet → Sequent → Type where ...

```

PPLC.reluLinked

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1223

```

def reluLinked (H : Hypernet) (preact post : Wire) : Prop

```

PPLC.PrimApp.isReluAt

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1217

```

def PrimApp.isReluAt (preact post : Wire) : PrimApp → Bool
| .relu s i o =>
  decide (i = preact) && decide (o = post) &&
  decide (0 < Shape.totalSize s)
| _ => false

def reluLinked (H : Hypernet) (preact post : Wire) : Prop

```

PPLC.affineWireTerm

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1159

```

def affineWireTerm (offset slope : Rat) (w : Wire) : Term

```

PPLC.lirpaConclusionForm

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1162

```

def lirpaConclusionForm
  (preact post : Wire) (α β γ δ : Rat) (F : Formula) : Prop

```

PPLC.reluLinked\_facts

lean/qtsl/Qtsl/Soundness.lean, line 8057

```

theorem reluLinked_facts {H : Hypernet} {preact post : Wire}
  (hlink : reluLinked H preact post) (σ : H.Valuation)
  (hadm : σ.Admissible) :
  ∃ qp qy : QTensor,
  σ.wireQ preact = some qp ∧ σ.wireQ post = some qy ∧
  qp.shape = qy.shape ∧ 0 < qp.entries.length ∧
  qp.entries.length = qy.entries.length ∧
  ∀ i, i < qp.entries.length →
  (qp.entries.getD i 0 ≤ 0 ∧ qy.entries.getD i 0 = 0) ∨
  (0 ≤ qp.entries.getD i 0 ∧
  qy.entries.getD i 0 = qp.entries.getD i 0)

```

PPLC.sound\_lirpa

lean/qtsl/Qtsl/Soundness.lean, line 8235

```

theorem sound_lirpa {H : Hypernet}
  {seq lowerPremise upperPremise : Sequent}
  {preact post : Wire} {a b α β γ δ : Q}
  (ihl : lowerPremise.semanticallyValid H)
  (ihu : upperPremise.semanticallyValid H)
  (hlink : reluLinked H preact post)
  (hAneg : a < 0) (hBpos : 0 < b)

```

```

(hLower1 :  $\alpha * a + \beta \leq 0$ ) (hLower2 :  $\beta \leq 0$ ) (hLower3 :  $\alpha * b + \beta \leq b$ )
(hUpper1 :  $0 \leq \gamma * a + \delta$ ) (hUpper2 :  $0 \leq \delta$ ) (hUpper3 :  $b \leq \gamma * b + \delta$ )
(hpre : seq.pre = lowerPremise.pre)
(hpreMatch : lowerPremise.pre = upperPremise.pre)
(hlowerPost : lowerPremise.post = .atom .le (.const a) (.var preact))
(hupperPost : upperPremise.post = .atom .le (.var preact) (.const b))
(hpostForm : lirpaConclusionForm preact post  $\alpha \beta \gamma \delta$  seq.post) :
seq.semanticallyValid H

```

PPLC.ReluExpansion.**reluExpH**

lean/qtsl/Qtsl/ReluExpansion.lean, line 156

```
def reluExpH : Hypernet where ...
```

PPLC.ReluExpansion.**relu\_projection**

lean/qtsl/Qtsl/ReluExpansion.lean, line 462

```

theorem relu_projection (p y :  $\mathbb{Q}$ ) :
  ( $\exists \sigma : \text{reluExpH.Vvaluation}, \sigma.\text{Admissible} \wedge$ 
     $\sigma.\text{entryFlat } 0 \ 0 = \text{some } (.fin \ p) \wedge$ 
     $\sigma.\text{entryFlat } 8 \ 0 = \text{some } (.fin \ y)) \leftrightarrow$ 
  ( $(p \leq 0 \wedge y = 0) \vee (0 \leq p \wedge y = p)$ )

```

PPLC.lirpa\_lower\_neg

lean/qtsl/Qtsl/Soundness.lean, line 8014

```

theorem lirpa_lower_neg {a  $\alpha \beta p$  :  $\mathbb{Q}$ } (ha :  $a < 0$ )
  (h1 :  $\alpha * a + \beta \leq 0$ ) (h2 :  $\beta \leq 0$ ) (hap :  $a \leq p$ ) (hp0 :  $p \leq 0$ ) :
   $\alpha * p + \beta \leq 0$ 

```

PPLC.lirpa\_lower\_pos

lean/qtsl/Qtsl/Soundness.lean, line 8025

```

theorem lirpa_lower_pos {b  $\alpha \beta p$  :  $\mathbb{Q}$ } (hb :  $0 < b$ )
  (h2 :  $\beta \leq 0$ ) (h3 :  $\alpha * b + \beta \leq b$ ) (hp0 :  $0 \leq p$ ) (hpb :  $p \leq b$ ) :
   $\alpha * p + \beta \leq p$ 

```

PPLC.lirpa\_upper\_neg

lean/qtsl/Qtsl/Soundness.lean, line 8036

```

theorem lirpa_upper_neg {a  $\gamma \delta p$  :  $\mathbb{Q}$ } (ha :  $a < 0$ )
  (h1 :  $0 \leq \gamma * a + \delta$ ) (h2 :  $0 \leq \delta$ ) (hap :  $a \leq p$ ) (hp0 :  $p \leq 0$ ) :
   $0 \leq \gamma * p + \delta$ 

```

PPLC.lirpa\_upper\_pos

lean/qtsl/Qtsl/Soundness.lean, line 8047

```

theorem lirpa_upper_pos {b  $\gamma \delta p$  :  $\mathbb{Q}$ } (hb :  $0 < b$ )
  (h2 :  $0 \leq \delta$ ) (h3 :  $b \leq \gamma * b + \delta$ ) (hp0 :  $0 \leq p$ ) (hpb :  $p \leq b$ ) :
   $p \leq \gamma * p + \delta$ 

```

PPLC.affineWireTerm\_evalQ

lean/qtsl/Qtsl/Soundness.lean, line 7994

```

theorem affineWireTerm_evalQ {H : Hypernet} ( $\sigma : H.\text{Valuation}$ ) {w : Wire}
  {qw : QTensor} (h :  $\sigma.\text{wireQ } w = \text{some } qw$ ) (off slope :  $\mathbb{Q}$ ) :
   $\exists qa, \text{Term.evalQ } \sigma (\text{affineWireTerm off slope } w) = \text{some } qa \wedge$ 
   $qa.\text{shape} = qw.\text{shape} \wedge$ 
   $qa.\text{entries.length} = qw.\text{entries.length} \wedge$ 
   $\forall i, i < qw.\text{entries.length} \rightarrow$ 
   $qa.\text{entries.getD } i \ 0 = \text{off} + \text{slope} * qw.\text{entries.getD } i \ 0$ 

```

PPLC.sound\_reluPos

lean/qtsl/Qtsl/Soundness.lean, line 8156

```

theorem sound_reluPos {H : Hypernet} {seq premise : Sequent}
  {preact post : Wire}
  (ih : premise.semanticallyValid H)
  (hlink : reluLinked H preact post)
  (hpre : seq.pre = premise.pre)
  (hprePost : premise.post = .atom .ge (.var preact) (.const 0))
  (hpost : seq.post = .atom .eq (.var post) (.var preact)) :
  seq.semanticallyValid H

```

**PPLC.PReLUPosData**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1578

```
inductive PReLUPosData : Hypernet → Sequent → Type where ...
```

**PPLC.QTensor.rel2\_const\_right\_elim**

lean/qtsl/Qtsl/Soundness.lean, line 7914

```
theorem QTensor.rel2_const_right_elim {r :  $\mathbb{Q} \rightarrow \mathbb{Q} \rightarrow \text{Prop}$ } {q : QTensor}
  {c :  $\mathbb{Q}$ } (h : QTensor.rel2 r q ([], [c], rfl)) :
   $\forall x \in q.\text{entries}, r \ x \ c$ 
```

**PPLC.sound\_reluNeg**

lean/qtsl/Qtsl/Soundness.lean, line 8193

```
theorem sound_reluNeg {H : Hypernet} {seq premise : Sequent}
  {preact post : Wire}
  (ih : premise.semanticallyValid H)
  (hlink : reluLinked H preact post)
  (hpre : seq.pre = premise.pre)
  (hprePost : premise.post = .atom .le (.var preact) (.const 0))
  (hpost : seq.post = .atom .eq (.var post) (.const 0)) :
  seq.semanticallyValid H
```

**PPLC.PReLUNegData**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1588

```
inductive PReLUNegData : Hypernet → Sequent → Type where ...
```

**PPLC.PReLUUNData**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1598

```
inductive PReLUUNData : Hypernet → Sequent → Type where ...
```

**PPLC.sound\_reluNN**

lean/qtsl/Qtsl/Soundness.lean, line 8116

```
theorem sound_reluNN {H : Hypernet} {seq : Sequent} {preact post : Wire}
  (hlink : reluLinked H preact post)
  (hpost : seq.post = .atom .ge (.var post) (.const 0)) :
  seq.semanticallyValid H
```

**PPLC.sound\_pConstExact**

lean/qtsl/Qtsl/Soundness.lean, line 7403

```
theorem sound_pConstExact {H : Hypernet} {seq : Sequent} {e1 e2 : Term}
  {b : BowTie}
  (hEval : (Formula.atom b e1 e2).evalClosed = some true)
  (hpost : seq.post = .atom b e1 e2) :
  seq.semanticallyValid H
```

**PPLC.PReLUAboveData**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1605

```
inductive PReLUAboveData : Hypernet → Sequent → Type where ...
```

**PPLC.sound\_reluAbove**

lean/qtsl/Qtsl/Soundness.lean, line 8138

```
theorem sound_reluAbove {H : Hypernet} {seq : Sequent} {preact post : Wire}
  (hlink : reluLinked H preact post)
  (hpost : seq.post = .atom .ge (.var post) (.var preact)) :
  seq.semanticallyValid H
```

**PPLC.sound\_pBound**

lean/qtsl/Qtsl/Soundness.lean, line 7412

```
theorem sound_pBound {H : Hypernet} {seq : Sequent} {components : List Sequent}
  {combination conclusion : Term}
  (hcomps :  $\forall c \in \text{components}, c.\text{semanticallyValid } H$ )
  (hpreC :  $\forall c \in \text{components}, \text{premiseContains } \text{seq.pre } c.\text{pre}$ )
  (hshape :  $\exists s, \text{conclusion.bshape } H = \text{some } s$ )
  (hfin : conclusion.finWires H)
  (hwpos :  $\forall w \in \text{seq.pre.freeWires} ++ \text{conclusion.freeWires},$ 
     $\exists \tau, H.\text{wireType } w = \text{some } \tau \wedge 0 < \text{Shape.totalSize } \tau.\text{shape}$ )
```

```

(hcompPos :  $\forall c \in \text{components}, \forall w \in c.\text{post.freeWires},$ 
   $\exists \tau, H.\text{wireType } w = \text{some } \tau \wedge 0 < \text{Shape.totalSize } \tau.\text{shape}$ )
(hvalid : syzygyCombinationValid combination components seq.pre conclusion)
(hform : syzygyConclusionForm conclusion seq.post) :
seq.semanticallyValid H

```

PPLC.baryLambdas1D

lean/qtsl/Qtsl/Core/Catalogue.lean, line 449

```

def baryLambdas1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool)
  (t : Term) : Option (List (Fin cx.nVertices × Term))

```

PPLC.MaskRegionStrict

lean/qtsl/Qtsl/Catalogue.lean, line 386

```

def MaskRegionStrict (C : SimpC) (mask ext : Fin C.nVertices → Bool)
  (x : Fin C.ambDim →  $\mathbb{Q}$ ) : Prop

```

PPLC.SimpC.CellIntDisjoint

lean/qtsl/Qtsl/Catalogue.lean, line 396

```

def SimpC.CellIntDisjoint (C : SimpC) : Prop

```

PPLC.PCellInIntData

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1612

```

inductive PCellInIntData : Hypernet → Sequent → Type where ...

```

PPLC.sound\_cellInInterior

lean/qtsl/Qtsl/Soundness.lean, line 6531

```

theorem sound_cellInInterior {H : Hypernet} {seq premise : Sequent}
  {schema : SimpC} {input : Wire} {outputs : List Wire}
  {whichI : Fin schema.cells.length}
  (ih : premise.semanticallyValid H)
  (happ : PrimApp.cell schema input outputs ∈ H.applications)
  (hinshape :  $\exists \tau, H.\text{wireType } \text{input} = \text{some } \tau \wedge$ 
     $\tau.\text{shape} = [\text{schema.ambDim}]$ )
  (houts :  $\forall o \in \text{outputs}, \exists \tau, H.\text{wireType } o = \text{some } \tau \wedge$ 
     $\text{Shape.totalSize } \tau.\text{shape} = 1$ )
  (hpre : seq.pre = premise.pre)
  (hint : cellStrictInteriorForm schema (.var input) whichI premise.post)
  (hpost : cellOneHotForm schema outputs whichI seq.post) :
seq.semanticallyValid H

```

PPLC.cellStrictInteriorBuild

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1167

```

def cellStrictInteriorBuild (schema : SimpC) (t : Term)
  (k : Fin schema.cells.length) : Option Formula

```

PPLC.cellOneHotBuild

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1180

```

def cellOneHotBuild (schema : SimpC) (outputs : List Wire)
  (k : Fin schema.cells.length) : Option Formula

```

PPLC.maskRegionStrict\_pair\_intro

lean/qtsl/Qtsl/Soundness.lean, line 6218

```

theorem maskRegionStrict_pair_intro {C : SimpC}
  {mask ext : Fin C.nVertices → Bool}
  {x : Fin C.ambDim →  $\mathbb{Q}$ } {a b : Fin C.nVertices} {V0 V1 :  $\mathbb{Q}$ }
  (hfil : (List.finRange C.nVertices).filter (fun v => mask v) = [a, b])
  (hscope :  $\forall v, \text{ext } v = \text{true} \rightarrow \text{mask } v = \text{true}$ )
  (hVa :  $\forall j, C.\text{coord } a \ j = V_0$ ) (hVb :  $\forall j, C.\text{coord } b \ j = V_1$ )
  (hne :  $V_0 \neq V_1$ ) (hdim : C.ambDim = 1) (j0 : Fin C.ambDim)
  (hw0t :  $\text{ext } a = \text{true} \rightarrow (V_1 - x \ j0) / (V_1 - V_0) < 0$ )
  (hw0f :  $\text{ext } a = \text{false} \rightarrow 0 < (V_1 - x \ j0) / (V_1 - V_0)$ )
  (hw1t :  $\text{ext } b = \text{true} \rightarrow (x \ j0 - V_0) / (V_1 - V_0) < 0$ )
  (hw1f :  $\text{ext } b = \text{false} \rightarrow 0 < (x \ j0 - V_0) / (V_1 - V_0)$ ) :
MaskRegionStrict C mask ext x

```



PPLC.holds\_cellOneHot

lean/qtsl/Qtsl/Soundness.lean, line 6480

```

theorem holds_cellOneHot {H : Hypernet} (σ : H.Valuation) {schema : SimpC}
  {outputs : List Wire} {k : Fin schema.cells.length} {F : Formula}
  (hbuild : cellOneHotForm schema outputs k F)
  (houts : ∀ o ∈ outputs, ∃ τ, H.wireType o = some τ ∧
    Shape.totalSize τ.shape = 1)
  (hsem : ∀ j : Fin outputs.length,
    σ.entryFlat (outputs.get j) 0
      = some (.fin (if j.val = k.val then 1 else 0))) :
    Formula.holds σ F

```

PPLC.cellStrictInteriorBuild\_spec

lean/qtsl/Qtsl/Soundness.lean, line 6091

```

theorem cellStrictInteriorBuild_spec {schema : SimpC} {t : Term}
  {k : Fin schema.cells.length} {F : Formula}
  (h : cellStrictInteriorForm schema t k F) :
  schema.ambDim = 1 ∧
  ∃ (a b : Fin schema.nVertices) (V₀ V₁ : ℚ),
    (List.finRange schema.nVertices).filter
      (fun v => (schema.cells.get k).1 v = [a, b] ∧
        (∀ j, schema.coord a j = V₀) ∧
        (∀ j, schema.coord b j = V₁) ∧
        V₀ ≠ V₁) ∧
    F = .and
      (.atom (if (schema.cells.get k).2 a then .lt else .gt)
        (.add (.const (V₁ / (V₁ - V₀))) (.mul (.const (-1 / (V₁ - V₀))) t))
        (.const 0))
      (.atom (if (schema.cells.get k).2 b then .lt else .gt)
        (.add (.const (-V₀ / (V₁ - V₀))) (.mul (.const (1 / (V₁ - V₀))) t))
        (.const 0))

```

PPLC.PrimApp.SemOk

lean/qtsl/Qtsl/Catalogue.lean, line 445

```

def PrimApp.SemOk {H : Hypernet} (σ : H.Valuation) : PrimApp → Prop
| .const _ty c out =>
  ∀ i : ℕ, σ.entryFlat out i = c.entryE i
| .contract δ r inputs output _inputRanks _outputRank pat =>
  ∀ v₀ ∈ Shape.positions pat.idxShape,
    ∃ eOut eExact,
      σ.entryAt output (pat.outCoords v₀) = some eOut ∧
      contractExact σ inputs pat (pat.outCoords v₀) = some eExact ∧
      δ.Round0kE r eExact eOut
| .cast _δfrom δto r shape input output =>
  ∀ i : Fin (Shape.totalSize shape),
    ∃ eIn eOut, σ.entryFlat input i.val = some eIn ∧
      σ.entryFlat output i.val = some eOut ∧ δto.Round0kE r eIn eOut
| .cell schema input outputs =>
  schema.CellIntDisjoint ∧
  ∃ x : Fin schema.ambDim → ℚ,
    (∀ j : Fin schema.ambDim, σ.entryAt input [j.val] = some (.fin (x j))) ∧
    ∃ k : Fin schema.cells.length,
      MaskRegion schema (schema.cells.get k).1 (schema.cells.get k).2 x ∧
      ∀ j : Fin outputs.length,
        σ.entryFlat (outputs.get j) 0 =
          some (.fin (if j.val = k.val then 1 else 0))
| .reshape _newShape input output =>
  ∀ i : ℕ, σ.entryFlat output i = σ.entryFlat input i
| .broadcast toShape input output =>
  ∃ τ, H.wireType input = some τ ∧
    ∀ c ∈ Shape.positions toShape,
      σ.entryAt output c = σ.entryAt input (Shape.bcastBack τ.shape toShape c)

```

```

| .copy _k input outputs =>
  ∀ o ∈ outputs, ∀ i : ℕ, σ.entryFlat o i = σ.entryFlat input i
| .delete _ => True
| .swap in1 in2 out1 out2 =>
  (∀ i : ℕ, σ.entryFlat out1 i = σ.entryFlat in2 i) ∧
  (∀ i : ℕ, σ.entryFlat out2 i = σ.entryFlat in1 i)
| .stack _k shape inputs output =>

  ∀ i : Fin inputs.length, ∀ j : Fin (Shape.totalSize shape),
    σ.entryFlat output (i.val * Shape.totalSize shape + j.val)
    = σ.entryFlat inputs[i] j.val
...

```

PPLC.PCellOutData

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1628

```

inductive PCellOutData : Hypernet → Sequent → Type where ...

```

PPLC.cellOutZeroBuild

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1207

```

def cellOutZeroBuild (schema : SimpC) (outputs : List Wire)
  (k : Fin schema.cells.length) : Option Formula

```

PPLC.sound\_cellOut

lean/qtsl/Qtsl/Soundness.lean, line 6632

```

theorem sound_cellOut {H : Hypernet} {seq premise : Sequent}
  {schema : SimpC} {input : Wire} {outputs : List Wire}
  {whichJ : Fin schema.cells.length} {wv : Fin schema.nVertices}
  (ih : premise.semanticallyValid H)
  (happ : PrimApp.cell schema input outputs ∈ H.applications)
  (hinshape : ∃ τ, H.wireType input = some τ ∧
    τ.shape = [schema.ambDim])
  (houts : ∀ o ∈ outputs, ∃ τ, H.wireType o = some τ ∧
    Shape.totalSize τ.shape = 1)
  (hpre : seq.pre = premise.pre)
  (hwt : cellOutWitnessForm schema (.var input) whichJ wv premise.post)
  (hpost : cellOutZeroForm schema outputs whichJ seq.post) :
  seq.semanticallyValid H

```

PPLC.cellOutWitnessBuild

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1193

```

def cellOutWitnessBuild (schema : SimpC) (t : Term)
  (k : Fin schema.cells.length) (w : Fin schema.nVertices) :
  Option Formula

```

PPLC.cellOutWitnessBuild\_spec

lean/qtsl/Qtsl/Soundness.lean, line 6145

```

theorem cellOutWitnessBuild_spec {schema : SimpC} {t : Term}
  {k : Fin schema.cells.length} {w : Fin schema.nVertices} {F : Formula}
  (h : cellOutWitnessForm schema t k w F) :
  schema.ambDim = 1 ∧
  (schema.cells.get k).1 w = true ∧
  ∃ (a b : Fin schema.nVertices) (V₀ V₁ : ℚ),
    (List.finRange schema.nVertices).filter
      (fun v => (schema.cells.get k).1 v = [a, b] ∧
        (∀ j, schema.coord a j = V₀) ∧
        (∀ j, schema.coord b j = V₁) ∧
        V₀ ≠ V₁)
    (w = a ∧ F = .atom (if (schema.cells.get k).2 a then .gt else .lt)
      (.add (.const (V₁ / (V₁ - V₀))) (.mul (.const (-1 / (V₁ - V₀))) t))
      (.const 0)) v
    (w = b ∧ F = .atom (if (schema.cells.get k).2 b then .gt else .lt)
      (.add (.const (-V₀ / (V₁ - V₀))) (.mul (.const (1 / (V₁ - V₀))) t))
      (.const 0)))

```

PPLC.**Formula.evalClosed** lean/qtsl/Qtsl/Core/Catalogue.lean, line 441

```
def Formula.evalClosed : Formula → Option Bool
| .atom b l r => do pure (b.evalQ (← l.evalQ?) (← r.evalQ?))
| .inRegion _ _ => none
| .and l r => do pure ((← l.evalClosed) && (← r.evalClosed))
| .or l r => do pure ((← l.evalClosed) || (← r.evalClosed))
| .tt => some true
| .ff => some false

def baryLambdas1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool)
  (t : Term) : Option (List (Fin cx.nVertices × Term))
```

PPLC.**Formula.holds\_of\_evalClosed\_true** lean/qtsl/Qtsl/Soundness.lean, line 2728

```
theorem Formula.holds_of_evalClosed_true {H : Hypernet} (σ : H.Valuation) :
  ∀ (F : Formula), F.evalClosed = some true → F.holds σ
```

PPLC.**Formula.holds\_of\_evalClosed\_true** lean/qtsl/Qtsl/Soundness.lean, line 2728

```
theorem Formula.holds_of_evalClosed_true {H : Hypernet} (σ : H.Valuation) :
  ∀ (F : Formula), F.evalClosed = some true → F.holds σ
```

PPLC.**Formula.evalClosed** lean/qtsl/Qtsl/Core/Catalogue.lean, line 441

```
def Formula.evalClosed : Formula → Option Bool
| .atom b l r => do pure (b.evalQ (← l.evalQ?) (← r.evalQ?))
| .inRegion _ _ => none
| .and l r => do pure ((← l.evalClosed) && (← r.evalClosed))
| .or l r => do pure ((← l.evalClosed) || (← r.evalClosed))
| .tt => some true
| .ff => some false

def baryLambdas1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool)
  (t : Term) : Option (List (Fin cx.nVertices × Term))
```

PPLC.**SyzygyStep** lean/qtsl/Qtsl/Soundness.lean, line 7150

```
structure SyzygyStep (H : Hypernet) where ...
```

PPLC.**sound\_syzygy** lean/qtsl/Qtsl/Soundness.lean, line 7205

```
theorem sound_syzygy {H : Hypernet} (st : SyzygyStep H)
  (hcomps : ∀ c ∈ st.components, c.semanticallyValid H) :
  st.seq.semanticallyValid H
```

PPLC.**syzygyConclusionForm** lean/qtsl/Qtsl/Core/Catalogue.lean, line 559

```
def syzygyConclusionForm (conclusion : Term) (F : Formula) : Prop
```

PPLC.**syzygyCheck** lean/qtsl/Qtsl/Core/Catalogue.lean, line 1026

```
def syzygyCheck (cc : Term) (components : List Sequent) (premise : Formula)
  (conclusion : Term) : Bool
```

PPLC.**PBoundData** lean/qtsl/Qtsl/Core/Catalogue.lean, line 1690

```
inductive PBoundData : Hypernet → Sequent → Type where ...
```

PPLC.**syzygyCombinationValid** lean/qtsl/Qtsl/Core/Catalogue.lean, line 1032

```
def syzygyCombinationValid (cc : Term) (components : List Sequent)
  (premise : Formula) (conclusion : Term) : Prop
```

PPLC.**holds\_of\_syzygyConclusionForm** lean/qtsl/Qtsl/Soundness.lean, line 7165

```

theorem holds_of_syzygyConclusionForm {H : Hypernet} {σ : H.Valuation}
  {conclusion : Term} {F : Formula}
  (hle : Formula.holds σ (.atom .le conclusion (.const 0)))
  (hform : syzygyConclusionForm conclusion F) :
  Formula.holds σ F

```

PPLC.PsdCert.**conclCheck**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1105

```

def conclCheck (n : Nat) (M : Fin n → Fin n → Rat) (ts : List Term)
  (conclusion : Term) : Bool

```

PPLC.Term.**toPoly**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 414

```

def Term.toPoly : Term → QArith.Poly
| .var w => QArith.Poly.var w
| .const c => QArith.Poly.const c
| .add l r => QArith.Poly.add l.toPoly r.toPoly
| .mul l r => QArith.Poly.mul l.toPoly r.toPoly
| .sub l r => QArith.Poly.sub l.toPoly r.toPoly
| .neg t => QArith.Poly.neg t.toPoly

| .idx _ _ => QArith.Poly.const 0

def Term.evalQ? : Term → Option Rat
| .var _ => none
| .const c => some c
| .add l r => do pure ((← l.evalQ?) + (← r.evalQ?))
| .mul l r => do pure ((← l.evalQ?) * (← r.evalQ?))
| .sub l r => do pure ((← l.evalQ?) - (← r.evalQ?))
| .neg t => do pure (-(← t.evalQ?))
| .idx _ _ => none

def BowTie.evalQ : BowTie → Rat → Rat → Bool
| .lt, x, y => decide (x < y)
| .le, x, y => decide (x ≤ y)
| .eq, x, y => decide (x = y)
| .ge, x, y => decide (y ≤ x)
| .gt, x, y => decide (y < x)
| .ne, x, y => decide (¬ x = y)

def Formula.evalClosed : Formula → Option Bool
| .atom b l r => do pure (b.evalQ (← l.evalQ?) (← r.evalQ?))
| .inRegion _ _ _ => none
| .and l r => do pure ((← l.evalClosed) && (← r.evalClosed))
| .or l r => do pure ((← l.evalClosed) || (← r.evalClosed))
| .tt => some true
| .ff => some false

def baryLambdas1D (cx : SimpC) (dContains : Fin cx.nVertices → Bool)
  (t : Term) : Option (List (Fin cx.nVertices × Term))

```

PPLC.PsdCert.**conclCheckCore**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1092

```

def conclCheckCore (n : Nat) (M : Fin n → Fin n → Rat) (ts : List Term)
  (conclusion : Term) : Bool

```

PPLC.PsdCert.**conclCheckW**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1098

```

def conclCheckW (n : Nat) (M : Fin n → Fin n → Rat) (ts : List Term)
  (conclusion : Term) : Bool

```

PPLC.**sound\_psdCert0f**

lean/qtsl/Qtsl/Soundness.lean, line 7466

```

theorem sound_psdCertOf {H : Hypernet} {seq : Sequent} {a : Nat}
  {M : Fin a → Fin a → ℚ} {ts : List Term} {conclusion : Term}
  (hpsd : ∀ x : Fin a → ℚ, 0 ≤ PsdCert.finQSum a fun i =>
    PsdCert.finQSum a fun j => M i j * x i * x j)
  (hconcl : PsdCert.conclCheck a M ts conclusion = true)
  (hshape : ∃ s, conclusion.bshape H = some s)
  (hfin : conclusion.finWires H)
  (hwpos : ∀ w ∈ conclusion.freeWires,
    ∃ τ, H.wireType w = some τ ∧ 0 < Shape.totalSize τ.shape)
  (hform : syzygyConclusionForm conclusion seq.post) :
  seq.semanticallyValid H

```

PPLC.PsdCert.congruence\_eq\_diagonal

lean/qtsl/Qtsl/PsdCert.lean, line 41

```

theorem congruence_eq_diagonal {A Z : Matrix n n ℝ}
  (hA : ∀ i j, A i j = A j i)
  (hZu : ∀ {i j}, j < i → Z i j = 0)
  (hMl : ∀ {i j}, i < j → (A * Z) i j = 0) :
  Zn * (A * Z) = diagonal fun i => Z i i * (A * Z) i i

```

PPLC.PsdCert.posDef\_of\_congruence\_posDef

lean/qtsl/Qtsl/PsdCert.lean, line 77

```

theorem posDef_of_congruence_posDef {A Z : Matrix n n ℝ}
  (hZu : ∀ {i j}, j < i → Z i j = 0)
  (hZd : ∀ i, 0 < Z i i)
  (hBd : (Zn * (A * Z)).PosDef) :
  A.PosDef

```

PPLC.PsdCert.posDef\_of\_triangular\_witness

lean/qtsl/Qtsl/PsdCert.lean, line 98

```

theorem posDef_of_triangular_witness {A Z : Matrix n n ℝ}
  (hA : ∀ i j, A i j = A j i)
  (hZu : ∀ {i j}, j < i → Z i j = 0)
  (hZd : ∀ i, 0 < Z i i)
  (hMl : ∀ {i j}, i < j → (A * Z) i j = 0)
  (hMd : ∀ i, 0 < (A * Z) i i) :
  A.PosDef

```

PPLC.PsdCert.quadForm\_nonneg\_of\_posSemidef

lean/qtsl/Qtsl/PsdCert.lean, line 431

```

theorem quadForm_nonneg_of_posSemidef {a : Nat} {M : Fin a → Fin a → ℚ}
  (h : (matR M).PosSemidef) (x : Fin a → ℚ) :
  0 ≤ finQSum a (fun i => finQSum a (fun j => M i j * x i * x j))

```

PPLC.PsdCert.symb

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1047

```

def symb (n : Nat) (A : Fin n → Fin n → Rat) : Bool

```

PPLC.PsdCert.upperTriPosDiagB

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1051

```

def upperTriPosDiagB (n : Nat) (Z : Fin n → Fin n → Rat) : Bool

```

PPLC.PsdCert.mulEntry

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1044

```

def mulEntry (n : Nat) (A B : Fin n → Fin n → Rat) (i j : Fin n) : Rat

```

PPLC.PsdCert.conclCheck

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1105

```

def conclCheck (n : Nat) (M : Fin n → Fin n → Rat) (ts : List Term)
  (conclusion : Term) : Bool

```

PPLC.PsdCert.negQuadFormPoly

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1085

```

def negQuadFormPoly (n : Nat) (M : Fin n → Fin n → Rat)
  (ps : List QArith.Poly) : QArith.Poly

```

PPLC.PsdCert.posDef\_of\_dominant

lean/qtsl/Qtsl/PsdCert.lean, line 126

```

theorem posDef_of_dominant [DecidableEq n] {B : Matrix n n ℝ}
  (hsym : ∀ i j, B i j = B j i)
  (hdom : ∀ i, (∑ j ∈ Finset.univ.erase i, |B i j|) < B i i) :
  B.PosDef

```

PPLC.PsdCert.posDef\_of\_dd\_witness

lean/qtsl/Qtsl/PsdCert.lean, line 213

```

theorem posDef_of_dd_witness [DecidableEq n] {A Z : Matrix n n ℝ}
  (hA : ∀ i j, A i j = A j i)
  (hZu : ∀ {i j}, j < i → Z i j = 0)
  (hZd : ∀ i, 0 < Z i i)
  (hdom : ∀ i, (∑ j ∈ Finset.univ.erase i, |(ZH * (A * Z)) i j|)
    < (ZH * (A * Z)) i i) :
  A.PosDef

```

PPLC.PsdCert.finQSum

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1042

```

def finQSum (n : Nat) (f : Fin n → Rat) : Rat

```

PPLC.PsdCert.posDef\_of\_dominant

lean/qtsl/Qtsl/PsdCert.lean, line 126

```

theorem posDef_of_dominant [DecidableEq n] {B : Matrix n n ℝ}
  (hsym : ∀ i j, B i j = B j i)
  (hdom : ∀ i, (∑ j ∈ Finset.univ.erase i, |B i j|) < B i i) :
  B.PosDef

```

PPLC.PsdCert.congEntry

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1062

```

def congEntry (n : Nat) (A Z : Fin n → Fin n → Rat) (i j : Fin n) : Rat

```

PPLC.PsdCert.posSemidef\_of\_factor

lean/qtsl/Qtsl/PsdCert.lean, line 233

```

theorem posSemidef_of_factor {m' n' : Type*} [Fintype m'] [Fintype n']
  {A : Matrix n' n' ℝ} {B : Matrix m' m' ℝ} {L : Matrix m' n' ℝ}
  (hB : B.PosSemidef) (hA : A = LH * B * L) : A.PosSemidef

```

PPLC.PsdCert.kernelExt

lean/qtsl/Qtsl/Core/Catalogue.lean, line 1071

```

def kernelExt (m k : Nat) (N : Fin m → Fin k → Rat) (a : Fin m)
  (j : Fin (m + k)) : Rat

```

PPLC.PsdCert.posSemidef\_of\_factor

lean/qtsl/Qtsl/PsdCert.lean, line 233

```

theorem posSemidef_of_factor {m' n' : Type*} [Fintype m'] [Fintype n']
  {A : Matrix n' n' ℝ} {B : Matrix m' m' ℝ} {L : Matrix m' n' ℝ}
  (hB : B.PosSemidef) (hA : A = LH * B * L) : A.PosSemidef

```

PPLC.sound\_reluSplit

lean/qtsl/Qtsl/ReluSplit.lean, line 15

```

theorem sound_reluSplit {H : Hypernet}
  {seq activePremise inactivePremise : Sequent}
  {preact post marker : Wire} {bodyForm : Formula}
  (ihActive : activePremise.semanticallyValid H)
  (ihInactive : inactivePremise.semanticallyValid H)
  (hlink : reluLinked H preact post)
  (hrank0 : ∃ τ, H.wireType post = some τ ∧ τ.shape = [])
  (hActPre : activePremise.pre
    = .and seq.pre (.atom .ge (.var preact) (.const 0)))
  (hInactPre : inactivePremise.pre
    = .and seq.pre (.atom .le (.var preact) (.const 0)))
  (hActPost : substituteInFormula bodyForm marker (.var preact)
    activePremise.post)
  (hInactPost : substituteInFormula bodyForm marker (.const 0))

```

```

    inactivePremise.post)
(hconcl : substituteInFormula bodyForm marker (.var post) seq.post) :
seq.semanticallyValid H

```

PPLC.**reluSplit\_rank0\_necessary**

lean/qtsl/Qtsl/ReluSplit.lean, line 153

```

theorem reluSplit_rank0_necessary :
  ¬ (∀ (H : Hypernet) (seq activePremise inactivePremise : Sequent)
    (preact post marker : Wire) (bodyForm : Formula),
    activePremise.semanticallyValid H →
    inactivePremise.semanticallyValid H →
    reluLinked H preact post →
    activePremise.pre
      = .and seq.pre (.atom .ge (.var preact) (.const 0)) →
    inactivePremise.pre
      = .and seq.pre (.atom .le (.var preact) (.const 0)) →
    substituteInFormula bodyForm marker (.var preact)
      activePremise.post →
    substituteInFormula bodyForm marker (.const 0)
      inactivePremise.post →
    substituteInFormula bodyForm marker (.var post) seq.post →
    seq.semanticallyValid H)

```

PPLC.**ReluSplitScope.HSplit**

lean/qtsl/Qtsl/ReluSplit.lean, line 95

```

def HSplit : Hypernet where ...

```

PPLC.**PromiseStep**

lean/qtsl/Qtsl/Soundness.lean, line 7116

```

structure PromiseStep where ...

```

PPLC.**sound\_promise**

lean/qtsl/Qtsl/Soundness.lean, line 7132

```

theorem sound_promise {H : Hypernet} (st : PromiseStep)
  (ihs : st.supplier.semanticallyValid H)
  (ihc : st.consumer.semanticallyValid H) :
  st.seq.semanticallyValid H

```

PPLC.**Formula.conjGet\_of\_conjReplace**

lean/qtsl/Qtsl/Soundness.lean, line 6994

```

theorem Formula.conjGet_of_conjReplace :
  ∀ {π : ConjPath} {Γ Γ' g : Formula},
  Γ.conjReplace? π g = some Γ' → Γ'.conjGet? π = some g

```

PPLC.**Formula.holds\_of\_conjGet**

lean/qtsl/Qtsl/Soundness.lean, line 6916

```

theorem Formula.holds_of_conjGet {H : Hypernet} {σ : H.Valuation} :
  ∀ {π : ConjPath} {Γ d : Formula},
  Γ.conjGet? π = some d → Γ.holds σ → d.holds σ

```

PPLC.**Formula.conjReplace\_back**

lean/qtsl/Qtsl/Soundness.lean, line 7042

```

theorem Formula.conjReplace_back :
  ∀ {π : ConjPath} {Γ Γ' q g : Formula},
  Γ.conjGet? π = some q → Γ.conjReplace? π g = some Γ' →
  Γ'.conjReplace? π q = some Γ

```

PPLC.**Formula.holds\_conjReplace**

lean/qtsl/Qtsl/Soundness.lean, line 6949

```

theorem Formula.holds_conjReplace {H : Hypernet} {σ : H.Valuation} :
  ∀ {π : ConjPath} {Γ Γ' g : Formula},
  Γ.conjReplace? π g = some Γ' → Γ.holds σ → g.holds σ → Γ'.holds σ

```

PPLC.**formulaWiresWithin**

lean/qtsl/Qtsl/Core/Catalogue.lean, line 953



|                                    |                                                                                                                                                                                                                                                                                                                              |                                               |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
|                                    | def formulaWiresWithin (f : Formula) (iface : List Wire) : Prop                                                                                                                                                                                                                                                              |                                               |
| PPLC.wiresSeparated                |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Core/Catalogue.lean, line 956  |
|                                    | def wiresSeparated (left ctx post : Formula) (iface : List Wire) : Prop                                                                                                                                                                                                                                                      |                                               |
| PPLC.syzygyCheckCore               |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Core/Catalogue.lean, line 964  |
|                                    | def syzygyCheckCore (cc : Term) (components : List Sequent) (premise : Formula)<br>(conclusion : Term) : Bool                                                                                                                                                                                                                |                                               |
| PPLC.syzygyCheckW                  |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Core/Catalogue.lean, line 982  |
|                                    | def syzygyCheckW (cc : Term) (components : List Sequent) (premise : Formula)<br>(conclusion : Term) : Bool                                                                                                                                                                                                                   |                                               |
| PPLC.syzygyCheckM                  |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Core/Catalogue.lean, line 1004 |
|                                    | def syzygyCheckM (cc : Term) (components : List Sequent) (premise : Formula)<br>(conclusion : Term) : Bool                                                                                                                                                                                                                   |                                               |
| PPLC.premiseContains               |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Core/Catalogue.lean, line 950  |
|                                    | def premiseContains (pre sub : Formula) : Prop                                                                                                                                                                                                                                                                               |                                               |
| PPLC.atomLeZeroPolys_sound_entryC  |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Soundness.lean, line 3380      |
|                                    | theorem atomLeZeroPolys_sound_entryC {H : Hypernet} (σ : H.Valuation)<br>{F : Formula} (hF : Formula.holds σ F)<br>(hw : ∀ w ∈ F.freeWires, ∃ τ, H.wireType w = some τ ∧<br>0 < Shape.totalSize τ.shape)<br>(hix : F.atomSidesIdxFree = true) (i : ℕ) :<br>∀ pa ∈ atomLeZeroPolys F, QArith.Poly.eval (σ.entryValC i) pa ≤ 0 |                                               |
| PPLC.wire_discipline_of_bshape_fin |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Soundness.lean, line 2265      |
|                                    | theorem wire_discipline_of_bshape_fin {H : Hypernet} (σ : H.Valuation)<br>{t : Term} {s : Shape} (hb : t.bshape H = some s)<br>(hf : t.finWires H) (hix : t.idxFree = true) :<br>∀ w ∈ t.freeWires, ∃ qw, σ.wireQ w = some qw ∧<br>(qw.shape = [] ∨ qw.shape = s)                                                            |                                               |
| PPLC.Term.evalQ_total_singleton    |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Soundness.lean, line 1009      |
|                                    | theorem Term.evalQ_total_singleton {H : Hypernet} (σ : H.Valuation) :<br>∀ (t : Term),<br>(∀ w ∈ t.freeWires, ∃ qw, σ.wireQ w = some qw ∧ SingletonShape qw.shape) →<br>t.idxFree = true →<br>∃ q, Term.evalQ σ t = some q ∧ SingletonShape q.shape ∧<br>q.scalar = QArith.Poly.eval σ.rho t.toPoly                          |                                               |
| PPLC.SmallGainStep.toSyzygy        |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Composition.lean, line 409     |
|                                    | def toSyzygy (st : SmallGainStep H) : SyzygyStep H where ...                                                                                                                                                                                                                                                                 |                                               |
| PPLC.sound_smallGain               |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Composition.lean, line 435     |
|                                    | theorem sound_smallGain {H : Hypernet} (st : SmallGainStep H)<br>(h1 : st.diss1.semanticallyValid H)<br>(h2 : st.diss2.semanticallyValid H)<br>(hm1 : st.mem1.semanticallyValid H)<br>(hm2 : st.mem2.semanticallyValid H) :<br>st.seq.semanticallyValid H                                                                    |                                               |
| PPLC.SmallGainStep.toSyzygy        |                                                                                                                                                                                                                                                                                                                              | lean/qtsl/Qtsl/Composition.lean, line 409     |

|                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                               |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
|                                        | def toSyzygy (st : SmallGainStep H) : SyzygyStep H where ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                               |
| PPLC.RuleTag                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | lean/qtsl/Qtsl/Core/Catalogue.lean, line 1291 |
|                                        | inductive RuleTag where ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                               |
| PPLC.SmallGainStep                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | lean/qtsl/Qtsl/Composition.lean, line 253     |
|                                        | structure SmallGainStep (H : Hypernet) where ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                               |
| PPLC.SmallGainStep.combTerm            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | lean/qtsl/Qtsl/Composition.lean, line 322     |
|                                        | def combTerm (st : SmallGainStep H) : Term                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                               |
| PPLC.SmallGainStep.combValid           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | lean/qtsl/Qtsl/Composition.lean, line 338     |
|                                        | theorem combValid (st : SmallGainStep H) :<br>syzygyCombinationValid st.combTerm st.components st.seq.pre st.concl                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                               |
| PPLC.SmallGainStep.toSyzygy_components |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | lean/qtsl/Qtsl/Composition.lean, line 430     |
|                                        | theorem toSyzygy_components (st : SmallGainStep H) :<br>st.toSyzygy.components = st.components                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                               |
| PPLC.smallGainN_is_syzygy              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | lean/qtsl/Qtsl/Composition.lean, line 582     |
|                                        | theorem smallGainN_is_syzygy (kk : ℕ) (LV A Sg : Fin kk → Poly)<br>(lam : Fin kk → ℚ) (B : Fin kk → Fin kk → ℚ) (th : ℚ)<br>(hcol : ∀ j, ((List.finRange kk).map (fun i => B i j)).sum = lam j) :<br>(((List.finRange kk).map<br>(fun i => (lam i, Poly.sub (Poly.add (LV i) (A i)) (Sg i)))<br>++ (List.finRange kk).map<br>(fun i => ((1 : ℚ), smallGainNMemPoly kk A Sg lam B th i))).foldr<br>(fun pr acc => Poly.add (Poly.scale pr.1 pr.2) acc) [])<br>= Poly.add<br>((List.finRange kk).foldr<br>(fun i acc => Poly.add (Poly.scale (lam i) (LV i)) acc) [])<br>(Poly.scale th<br>((List.finRange kk).foldr<br>(fun i acc => Poly.add (Poly.scale (lam i) (A i)) acc) [])) |                                               |
| PPLC.SmallGainNStep.toSyzygy           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | lean/qtsl/Qtsl/Composition.lean, line 796     |
|                                        | def toSyzygy (st : SmallGainNStep H kk) : SyzygyStep H where ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                               |
| PPLC.sound_smallGainN                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | lean/qtsl/Qtsl/Composition.lean, line 825     |
|                                        | theorem sound_smallGainN {H : Hypernet} {kk : ℕ} (st : SmallGainNStep H kk)<br>(hdiss : ∀ i, (st.diss i).semanticallyValid H)<br>(hmem : ∀ i, (st.mem i).semanticallyValid H) :<br>st.seq.semanticallyValid H                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                               |
| PPLC.smallGainN_colSum_mutant_refuted  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | lean/qtsl/Qtsl/Composition.lean, line 857     |
|                                        | theorem smallGainN_colSum_mutant_refuted :<br>¬ (∀ (kk : ℕ) (LV A Sg : Fin kk → Poly) (lam : Fin kk → ℚ)<br>(B : Fin kk → Fin kk → ℚ) (th : ℚ),<br>(((List.finRange kk).map<br>(fun i => (lam i, Poly.sub (Poly.add (LV i) (A i)) (Sg i)))<br>++ (List.finRange kk).map<br>(fun i => ((1 : ℚ), smallGainNMemPoly kk A Sg lam B th i))).foldr<br>(fun pr acc => Poly.add (Poly.scale pr.1 pr.2) acc) [])<br>= Poly.add<br>((List.finRange kk).foldr<br>(fun i acc => Poly.add (Poly.scale (lam i) (LV i)) acc) [])                                                                                                                                                                 |                                               |

```

(Poly.scale th
  ((List.finRange kk).foldr
    (fun i acc => Poly.add (Poly.scale (lam i) (A i)) acc) [])))

```

**Qtsl.QtslCompose.Discrete.smallGainN\_drift**    lean/qtsl-aux/QtslAux/QtslCompose/AGComposeDiscrete.lean, line 372

```

theorem smallGainN_drift (hlam : ∀ i, 0 ≤ lam i)
  (hcol : ∀ j, ∑ i, B i j = lam j)
  (hint : ∀ i, ∀ y, y ∉ A → Integrable (V i) (K y))
  (hd : ∀ i, ∀ y, y ∉ A →
    ∫ z, V i z ∂K y ≤ V i y - alpha i y + sigma i y)
  (hm : ∀ i, ∀ y, y ∉ A →
    lam i * sigma i y ≤ (1 - theta) * ∑ j, B i j * alpha j y) :
  ∀ y, y ∉ A → ∫ z, (∑ i, lam i * V i z) ∂K y
    ≤ (∑ i, lam i * V i y) - theta * ∑ j, lam j * alpha j y

```

**Qtsl.QtslCompose.Discrete.smallGain\_drift\_of\_N**    lean/qtsl-aux/QtslAux/QtslCompose/AGComposeDiscrete.lean, line 416

```

theorem smallGain_drift_of_N {V1 V2 alpha1 alpha2 sigma1 sigma2 : S → ℝ}
  {lam theta : ℝ} (hlam : 0 ≤ lam)
  (hint1 : ∀ y, y ∉ A → Integrable V1 (K y))
  (hint2 : ∀ y, y ∉ A → Integrable V2 (K y))
  (hd1 : ∀ y, y ∉ A → ∫ z, V1 z ∂K y ≤ V1 y - alpha1 y + sigma1 y)
  (hd2 : ∀ y, y ∉ A → ∫ z, V2 z ∂K y ≤ V2 y - alpha2 y + sigma2 y)
  (hm1 : ∀ y, y ∉ A → lam * sigma2 y ≤ (1 - theta) * alpha1 y)
  (hm2 : ∀ y, y ∉ A → sigma1 y ≤ lam * ((1 - theta) * alpha2 y)) :
  ∀ y, y ∉ A → ∫ z, (V1 z + lam * V2 z) ∂K y
    ≤ (V1 y + lam * V2 y) - theta * (alpha1 y + lam * alpha2 y)

```

**PPLC.citeFacet**    lean/qtsl/Qtsl/ProductLattice.lean, line 675

```

def citeFacet (s : Sequent) (rider : Formula) : Sequent

```

**PPLC.prodBlock**    lean/qtsl/Qtsl/ProductLattice.lean, line 678

```

def prodBlock (b1 b2 : BlockCert) : BlockCert

```

**PPLC.PProductLatticeData.prodBlocks**    lean/qtsl/Qtsl/ProductLattice.lean, line 688

```

def PProductLatticeData.prodBlocks {H1 H2 : Hypernet}
  (d : PProductLatticeData H1 H2) : List BlockCert

```

**PPLC.CertifiedBlockLattice**    lean/qtsl/Qtsl/ProductLattice.lean, line 658

```

structure CertifiedBlockLattice (H : Hypernet) where ...

```

**PPLC.Hypernet.disjUnion**    lean/qtsl/Qtsl/ProductLattice.lean, line 252

```

def Hypernet.disjUnion (H1 H2 : Hypernet) (hd : H1.WireDisjoint H2) : Hypernet
  where ...

```

**PPLC.sound\_productLattice**    lean/qtsl/Qtsl/ProductLattice.lean, line 712

```

theorem sound_productLattice {H1 H2 : Hypernet}
  (d : PProductLatticeData H1 H2)
  (hL : d.left.Valid) (hR : d.right.Valid) :
  ∀ b ∈ d.prodBlocks, ∀ s ∈ b.facets,
    s.semanticallyValid (H1.disjUnion H2 d.disjoint)

```

**PPLC.Sequent.semanticallyValid\_extend**    lean/qtsl/Qtsl/ProductLattice.lean, line 636

```

theorem Sequent.semanticallyValid_extend {H H' : Hypernet} {seq : Sequent}
  (hext : H.WireExtends H')
  (happs : ∀ app ∈ H.applications, app ∈ H'.applications)
  (hws : seq.wellScoped H) (hval : seq.semanticallyValid H) :
  seq.semanticallyValid H'

```

PPLC.SemanticValidWeaken

lean/qtsl/Qtsl/ProductLattice.lean, line 648

```

theorem Sequent.semanticallyValid_weaken {H : Hypernet}
  {pre extra post : Formula}
  (h : Sequent.semanticallyValid H (pre, post)) :
  Sequent.semanticallyValid H (.and pre extra, post)

```

PPLC.SoundProductLatticeCiteLeft

lean/qtsl/Qtsl/ProductLattice.lean, line 692

```

theorem sound_productLattice_citeLeft {H1 H2 : Hypernet}
  (hd : H1.WireDisjoint H2) {s0 : Sequent} (rider : Formula)
  (hws : s0.wellScoped H1) (hval : s0.semanticallyValid H1) :
  (citeFacet s0 rider).semanticallyValid (H1.disjUnion H2 hd)

```

PPLC.PProductLatticeData.ProdBLOCKS

lean/qtsl/Qtsl/ProductLattice.lean, line 688

```

def PProductLatticeData.prodBLOCKS {H1 H2 : Hypernet}
  (d : PProductLatticeData H1 H2) : List BlockCert

```

PPLC.Hypernet.Valuation.Restrict

lean/qtsl/Qtsl/ProductLattice.lean, line 320

```

def Hypernet.Valuation.restrict {H H' : Hypernet} (σ : H'.Valuation)
  (hext : H.WireExtends H') : H.Valuation

```

PPLC.Formula.HoldsRestrict

lean/qtsl/Qtsl/ProductLattice.lean, line 429

```

theorem Formula.holds_restrict {H H' : Hypernet} (σ : H'.Valuation)
  (hext : H.WireExtends H') :
  ∀ (F : Formula), (∀ w ∈ F.freeWires, (H.wireType w).isSome) →
  (Formula.holds (σ.restrict hext) F ↔ Formula.holds σ F)

```

PPLC.PrimApp.SemOkRestrict

lean/qtsl/Qtsl/ProductLattice.lean, line 485

```

theorem PrimApp.semOk_restrict {H H' : Hypernet} (σ : H'.Valuation)
  (hext : H.WireExtends H') {app : PrimApp}
  (hTC : app.TypeConsistent H.wireType) (h : app.SemOk σ) :
  app.SemOk (σ.restrict hext)

```

PPLC.Hypernet.WireDisjoint

lean/qtsl/Qtsl/ProductLattice.lean, line 17

```

def Hypernet.WireDisjoint (H1 H2 : Hypernet) : Prop

```

PPLC.ProdBLOCKSConfined

lean/qtsl/Qtsl/ProductLattice.lean, line 729

```

theorem prodBlocks_confined {H1 H2 : Hypernet}
  (d : PProductLatticeData H1 H2) :
  ∀ b ∈ d.prodBLOCKS,
  (∀ w ∈ b.region.freeWires,
    ((H1.disjUnion H2 d.disjoint).wireType w).isSome) ∧
  ∀ s ∈ b.facets, s.wellScoped (H1.disjUnion H2 d.disjoint)

```

PPLC.CertifiedBlockLattice.Valid

lean/qtsl/Qtsl/ProductLattice.lean, line 671

```

def CertifiedBlockLattice.Valid {H : Hypernet}
  (L : CertifiedBlockLattice H) : Prop

```

Qtsl.Aliases.SoundQSatFromRegion

lean/qtsl-aux/QtslAux/QtslAliases.lean, line 35

```
theorem sound_QSatFromRegion {X : Type*} {R : Set X} {sat : X → Prop} {x₀ : X}
  (hchild : ∀ x ∈ R, sat x) (heval : x₀ ∈ R) : sat x₀
```

Qtsl.Aliases.**sound\_QSatFromRegion\_sublevel**

lean/qtsl-aux/QtslAux/QtslAliases.lean, line 58

```
theorem sound_QSatFromRegion_sublevel {σ : Type*} {sat : (σ → ℝ) → Prop}
  {P : MvPolynomial σ ℚ} {thr : ℚ} {xq : σ → ℚ}
  (hchild : ∀ x ∈ {x : σ → ℝ |
    MvPolynomial.eval x (MvPolynomial.map (Rat.castHom ℝ) P) ≤ (thr : ℝ)}, sat x)
  (heval : MvPolynomial.eval xq P ≤ thr) :
  sat (fun i => (xq i : ℝ))
```

PPLC.**rootEvalFormula\_true\_iff**

lean/qtsl/Qtsl/Elaborators.lean, line 137

```
theorem rootEvalFormula_true_iff (x0 : Nat → Rat) (V : Poly) (thr : Rat) :
  (rootEvalFormula x0 V thr).evalClosed = some true
  ↔ Poly.eval x0 V ≤ thr
```

Qtsl.Guard.**GuardSat**

lean/qtsl-aux/QtslAux/QGuard.lean, line 57

```
def GuardSat (init R : Set X) (sat : X → Prop) : Prop
```

Qtsl.Guard.**sound\_QGuardIntro**

lean/qtsl-aux/QtslAux/QGuard.lean, line 74

```
theorem sound_QGuardIntro {init R : Set X} {sat : X → Prop}
  (hchild : ∀ x ∈ R, sat x) : GuardSat init R sat
```

Qtsl.Guard.**sound\_QGuardIntro\_sublevel**

lean/qtsl-aux/QtslAux/QGuard.lean, line 171

```
theorem sound_QGuardIntro_sublevel {σ : Type*} {init : Set (σ → ℝ)}
  {sat : (σ → ℝ) → Prop} {P : MvPolynomial σ ℚ} {thr : ℚ}
  (hchild : ∀ x ∈ {x : σ → ℝ |
    MvPolynomial.eval x (MvPolynomial.map (Rat.castHom ℝ) P) ≤ (thr : ℝ)},
    sat x) :
  GuardSat init
  {x : σ → ℝ |
    MvPolynomial.eval x (MvPolynomial.map (Rat.castHom ℝ) P) ≤ (thr : ℝ)}
  sat
```

Qtsl.Guard.**guard\_eval\_sublevel**

lean/qtsl-aux/QtslAux/QGuard.lean, line 185

```
theorem guard_eval_sublevel {σ : Type*} {P : MvPolynomial σ ℚ} {thr : ℚ}
  {xq : σ → ℚ} (heval : MvPolynomial.eval xq P ≤ thr) :
  (fun i => (xq i : ℝ)) ∈ {x : σ → ℝ |
    MvPolynomial.eval x (MvPolynomial.map (Rat.castHom ℝ) P) ≤ (thr : ℝ)}
```

Qtsl.Guard.**sound\_QGuardIntro\_prob**

lean/qtsl-aux/QtslAux/QGuard.lean, line 193

```
theorem sound_QGuardIntro_prob [MeasurableSpace X] {init R : Set X}
  {μ : X → Measure (ℕ → X)} {p : ENNReal} {A : Set (ℕ → X)}
  (hchild : ∀ x ∈ R, p ≤ μ x A) :
  GuardSat init R fun x => p ≤ μ x A
```

Qtsl.Guard.**exists\_blanket\_not\_guardSat**

lean/qtsl-aux/QtslAux/QGuard.lean, line 137

```
theorem exists_blanket_not_guardSat :
  ∃ (init R : Set Bool) (sat : Bool → Prop),
  GuardSatBlanket init R sat ∧ ¬ GuardSat init R sat
```

Qtsl.Guard.**exists\_not\_guardSat\_wrong\_region**

lean/qtsl-aux/QtslAux/QGuard.lean, line 160

```
theorem exists_not_guardSat_wrong_region :
  ∃ (init R R' : Set Bool) (sat : Bool → Prop),
  (∀ x ∈ R, sat x) ∧ ¬ GuardSat init R' sat
```

## Qtsl.Guard.sound\_QGuardElim

lean/qtsl-aux/QtslAux/QGuard.lean, line 81

```
theorem sound_QGuardElim {init R : Set X} {sat : X → Prop}
  (hg : GuardSat init R sat) (heval : init ≤ R) : ∀ x₀ ∈ init, sat x₀
```

## Qtsl.Guard.exists\_guardSat\_not\_unguarded

lean/qtsl-aux/QtslAux/QGuard.lean, line 150

```
theorem exists_guardSat_not_unguarded :
  ∃ (init R : Set Bool) (sat : Bool → Prop),
    GuardSat init R sat ∧ ¬ ∀ x₀ ∈ init, sat x₀
```

## Qtsl.Guard.qsatFromRegion\_derived

lean/qtsl-aux/QtslAux/QGuard.lean, line 88

```
theorem qsatFromRegion_derived {init R : Set X} {sat : X → Prop}
  (hchild : ∀ x ∈ R, sat x) (heval : init ≤ R) : ∀ x₀ ∈ init, sat x₀
```

## Qtsl.LayerD.sound\_GSHS\_safety\_mp

lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 511

```
theorem sound_GSHS_safety_mp [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {U : Set State} (hV : Measurable V) (hU : MeasurableSet U)
  (hmp : ∀ x : State, Nonempty (MPFlow M 0 V x))
  (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, V y ≤ C)
  (hFloor : ∀ y ∈ U, 1 ≤ V y) (x : State) :
  Battery.pathMeasure (skel M) (SkelState.alive x)
    {ω | ∃ n, ω n ∈ skelBad U}
    ≤ ENNReal.ofReal (V x)
```

## Qtsl.LayerD.sound\_GSHS\_safety\_battery\_mp

lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 524

```
theorem sound_GSHS_safety_battery_mp
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {U : Set State} {p thr : ℝ}
  (hV : Measurable V) (hU : MeasurableSet U)
  (hmp : ∀ x : State, Nonempty (MPFlow M 0 V x))
  (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, V y ≤ C)
  (hFloor : ∀ y ∈ U, 1 ≤ V y)
  (h0 : p + thr ≤ 1) (x : State) (hthr : V x ≤ thr) :
  ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
    {ω | ∀ n, ω n ∉ skelBad U}
```

## Qtsl.Battery.sound\_QSafetyBattery\_staged

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 556

```
theorem sound_QSafetyBattery_staged
  {K : Kernel (Md × X) (Md × X)} [IsMarkovKernel K]
  {Dm Dbarm Um : Md → Set X} {p thr : ℚ} {sat : Md × X → Prop}
  (hDm : ∀ m, MeasurableSet (Dm m)) (hDbarm : ∀ m, MeasurableSet (Dbarm m))
  (hUm : ∀ m, MeasurableSet (Um m)) (hVmeas : ∀ m, Measurable (V m))
  (hF : ∀ m z, ¬ sat (m, z) → z ∈ Um m)
  (h0 : p + thr ≤ 1)
  (hDsub : ∀ m, Dm m ⊆ Dbarm m)
  (hR3 : ∀ m z, z ∈ Dm m → z ∉ Um m → K (m, z) {y | y.2 ∈ Dbarm y.1} = 1)
  (hR4 : ∀ m z, z ∈ Dm m → z ∉ Um m →
    K (m, z) {y | y.2 ∈ Dm y.1} ∪ {y | y.2 ∈ Um y.1} = 1)
  (hint : ∀ m z, z ∈ Dm m → z ∉ Um m → Integrable (V m) ((flow m) z))
  (hB1 : ∀ m z, z ∈ Dm m → z ∉ Um m → ∫ ζ, V m ζ ∂(flow m) z ≤ V m z)
  (hB2 : ∀ j, ∀ z ∈ G j, V (dst j) (g j z) ≤ V (src j) z)
  (hB3 : ∀ m z, z ∈ Um m → z ∈ Dbarm m → 1 ≤ V m z)
  (hB4 : ∀ m z, z ∈ Dbarm m → 0 ≤ V m z)
  (hmix : ∀ m z, z ∈ Dm m → z ∉ Um m →
    ∃ (w₀ : ℝ≥0) (w : J → ℝ≥0), w₀ + ∑ j, w j = 1 ∧
      (∀ j, w j ≠ 0 → src j = m ∧ z ∈ G j) ∧
      K (m, z) = (w₀ : ℝ≥0∞) • ((flow m) z).map (Prod.mk m))
```

```

      +  $\sum j, (w\ j : \mathbb{R}_{\geq 0}) \bullet \text{Measure.dirac } (\text{dst } j, g\ j\ z)) :$ 
 $\forall m\ z, z \in D\ m \rightarrow V\ m\ z \leq (\text{thr} : \mathbb{R}) \rightarrow$ 
 $\text{ENNReal.ofReal } (p : \mathbb{R})$ 
 $\leq \text{pathMeasure } K\ (m, z) \{ \omega \mid \forall n, \text{sat } (\omega\ n) \}$ 

```

SuFunctor.Site.**measurableSet\_confinedEvent\_cylinder** lean/su-functor/SuFunctor/Site/FellAcceptance.lean, line 413

```

theorem measurableSet_confinedEvent_cylinder {D : Set X}
  (hD : IsClosed D) :
  MeasurableSet
    (confinedEvent ((Set.univ : Set  $\mathfrak{I}$ W)  $\times^S$  D) : Set (FTraj W X))

```

SuFunctor.Site.**satAtM\_safetySpecF\_iff** lean/su-functor/SuFunctor/Site/FellBehaviorSpec.lean, line 254

```

theorem satAtM_safetySpecF_iff (U :  $\mathfrak{I} \rightarrow \text{Set } T$ )
  (hU :  $\forall i, \text{IsOpen } (U\ i)$ ) {D : Set X} (hD : IsClosed D) (j :  $\mathfrak{I}$ )
  ( $\mu$  : MeasJoint (FTraj (U j) X)) :
  SatAtM  $\mu$  (specOfBehaviorF U (safetyBehaviorF U D) j)  $\leftrightarrow$ 
  ( $\mu.1$  : Measure (FTraj (U j) X))
    (confinedEvent ((Set.univ : Set  $\mathfrak{I}$ (U j))  $\times^S$  D)) $^c$  = 0

```

SuFunctor.Site.**pull\_indPenM** lean/su-functor/SuFunctor/Site/FellBehaviorSpec.lean, line 110

```

theorem pull_indPenM (h : K  $\rightarrow$  L) (hm : Measurable h) {S : Set L}
  (hS : MeasurableSet S) :
  MeasPen.pull h (indPenM S) = indPenM (h $^{-1}$ ' S)

```

SuFunctor.Site.**graphCl\_mem\_confinedEvent\_iff** lean/su-functor/SuFunctor/Site/FellBehaviorSpec.lean, line 340

```

theorem graphCl_mem_confinedEvent_iff {D : Set X} (hD : IsClosed D)
  (f : Traj W X) :
  graphCl f  $\in$  confinedEvent ((Set.univ : Set  $\mathfrak{I}$ W)  $\times^S$  D)  $\leftrightarrow$ 
   $\forall w, f\ w \in D$ 

```

SuFunctor.Site.**mboxplusWF\_indPenM** lean/su-functor/SuFunctor/Site/FellBehaviorSpec.lean, line 280

```

theorem mboxplusWF_indPenM {W : Set T} { $\pi_1$  : Z  $\rightarrow$  U'} { $\pi_2$  : Z  $\rightarrow$  V'}
  (hp $_1$  : IsProperMap  $\pi_1$ ) (hp $_2$  : IsProperMap  $\pi_2$ )
  {S $_1$  : Set (FTraj W U')} {S $_2$  : Set (FTraj W V')}
  (hS $_1$  : MeasurableSet S $_1$ ) (hS $_2$  : MeasurableSet S $_2$ ) :
  mboxplusWF  $\pi_1$   $\pi_2$  (indPenM S $_1$ ) (indPenM S $_2$ )
    = indPenM (fellMap (W := W)  $\pi_1^{-1}$ ' S $_1$   $\cap$ 
      fellMap (W := W)  $\pi_2^{-1}$ ' S $_2$ )

```

Qtsl.Battery.**sound\_QSafetyBattery** lean/qtsl-aux/QtslAux/Battery.lean, line 433

```

theorem sound_QSafetyBattery {sat : S  $\rightarrow$  Prop}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV : Measurable V)
  (hF :  $\forall y, \neg \text{sat } y \rightarrow y \in U$ )
  (h0 :  $p + \text{thr} \leq 1$ )
  (hDsub : D  $\subseteq$  Dbar)
  (hR3 :  $\forall y \in D, y \notin U \rightarrow K\ y\ \text{Dbar} = 1$ )
  (hR4 :  $\forall y \in D, y \notin U \rightarrow K\ y\ (D \cup U) = 1$ )
  (hB12int :  $\forall y \in D, y \notin U \rightarrow \text{Integrable } V\ (K\ y)$ )
  (hB12 :  $\forall y \in D, y \notin U \rightarrow \int z, V\ z\ \partial K\ y \leq V\ y$ )
  (hB3 :  $\forall y \in U, y \in \text{Dbar} \rightarrow 1 \leq V\ y$ )
  (hB4 :  $\forall y \in \text{Dbar}, 0 \leq V\ y$ )
  (hx : x  $\in$  D) (hthr :  $V\ x \leq \text{thr}$ ) :
  ENNReal.ofReal p  $\leq$  pathMeasure K x { $\omega \mid \forall n, \text{sat } (\omega\ n)$ }

```

Qtsl.Battery.**pathMeasure** lean/qtsl-aux/QtslAux/Battery.lean, line 69

```

noncomputable def pathMeasure (K : Kernel S S) [IsMarkovKernel K] (x : S) :
  Measure ( $\mathbb{N} \rightarrow S$ )

```



Qtsl.LayerD.**sound\_GSHS\_safety\_battery**

lean/qtsl-aux/QtslAux/LayerD/Skeleton.lean, line 253

```

theorem sound_GSHS_safety_battery [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {U : Set State} {p thr : ℝ}
  (hV : Measurable V) (hU : MeasurableSet U)
  (hSegD : SegmentDynkin M 0 V) (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, V y ≤ C)
  (hFloor : ∀ y ∈ U, 1 ≤ V y)
  (h0 : p + thr ≤ 1) (x : State) (hthr : V x ≤ thr) :
  ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
  {ω | ∀ n, ω n ∉ skelBad U}

```

Qtsl.Battery.**sound\_QSafetyBattery\_failClosed**

lean/qtsl-aux/QtslAux/Battery.lean, line 311

```

theorem sound_QSafetyBattery_failClosed
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
  (hV : Measurable V)
  (h0 : p + thr ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ U → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ U → K y (D ∪ U) = 1)
  (hB12int : ∀ y ∈ D, y ∉ U → Integrable V (K y))
  (hB12 : ∀ y ∈ D, y ∉ U → ∫ z, V z ∂K y ≤ V y)
  (hB3 : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V y)
  (hB4 : ∀ y ∈ Dbar, 0 ≤ V y)
  (hx : x ∈ D) (hthr : V x ≤ thr) :
  ENNReal.ofReal p ≤ pathMeasure K x {ω | ∀ n, ω n ∈ D \ U}

```

OptionalSamplingRC.**supermartingale\_of\_compensator\_nonpos** lean/optional-sampling/OptionalSamplingRC/Basic.lean, line 1150

```

theorem supermartingale_of_compensator_nonpos
  [IsProbabilityMeasure μ]
  (W : ℝ≥0 → Ω → ℝ) (g : ℝ → Ω → ℝ)
  (hmart : Martingale
    (fun (t : ℝ≥0) (ω : Ω) => W t ω - ∫ s in (0:ℝ)..(t:ℝ), g s ω) ℱ μ)
  (hGadp : Adapted ℱ
    (fun (t : ℝ≥0) (ω : Ω) => ∫ s in (0:ℝ)..(t:ℝ), g s ω))
  (hWint : ∀ t : ℝ≥0, Integrable (W t) μ)
  (hgint : ∀ ω, ∀ t : ℝ, IntervalIntegrable (fun s => g s ω) volume 0 t)
  (hg0 : ∀ ω, ∀ s : ℝ, 0 ≤ s → g s ω ≤ 0) :
  Supermartingale W ℱ μ

```

Qtsl.Battery.**mixture\_drift**

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 488

```

lemma mixture_drift {m : Md} {z : X} {κ : Measure (Md × X)}
  {w₀ : ℝ≥0} {w : J → ℝ≥0}
  (hVmeas : ∀ m', Measurable (V m'))
  (hκ : κ = (w₀ : ℝ≥0∞) • ((flow m) z).map (Prod.mk m)
    + ∑ j, (w j : ℝ≥0∞) • Measure.dirac (dst j, g j z))
  (hsum : w₀ + ∑ j, w j = 1)
  (hguard : ∀ j, w j ≠ 0 → V (dst j) (g j z) ≤ V m z)
  (hflowint : Integrable (fun ζ => V m ζ) ((flow m) z))
  (hflowdrift : ∫ ζ, V m ζ ∂(flow m) z ≤ V m z) :
  ∫ y, V y.1 y.2 ∂κ ≤ V m z

```

Qtsl.Battery.**stronglyAdapted\_stopped**

lean/qtsl-aux/QtslAux/Battery.lean, line 187

```

lemma stronglyAdapted_stopped {f : S → ℝ} {A : Set S} (hA : MeasurableSet A)
  (hf : Measurable f) :
  StronglyAdapted (Filtration.piLE (X := fun _ : ℕ => S)) (stopped f A)

```

Qtsl.Battery.**one\_le\_stopped**

lean/qtsl-aux/QtslAux/Battery.lean, line 133

```
lemma one_le_stopped {f : S → ℝ} {A : Set S} (hf1 : ∀ y ∈ A, 1 ≤ f y) :
  ∀ (n : ℕ) (ω : ℕ → S), (∃ k ≤ n, ω k ∈ A) → 1 ≤ stopped f A n ω
```

Qtsl.Battery.stopped\_congr

lean/qtsl-aux/QtslAux/Battery.lean, line 154

```
lemma stopped_congr {f : S → ℝ} {A : Set S} : ∀ {n : ℕ} {ω ω' : ℕ → S},
  (∀ k ≤ n, ω k = ω' k) → stopped f A n ω = stopped f A n ω'
```

Qtsl.Battery.supermartingale\_stopped

lean/qtsl-aux/QtslAux/Battery.lean, line 263

```
theorem supermartingale_stopped {f : S → ℝ} {A : Set S} {C : ℝ}
  (hA : MeasurableSet A) (hf : Measurable f) (hf0 : ∀ y, 0 ≤ f y) (hfC : ∀ y, f y
    ≤ C)
  (hdrift : ∀ y, y ∉ A → ∫ z, f z ∂K y ≤ f y) (x : S) :
  Supermartingale (stopped f A) (Filtration.piLE (X := fun _ : ℕ => S))
  (pathMeasure K x)
```

Qtsl.Battery.measure\_reach\_le

lean/qtsl-aux/QtslAux/Battery.lean, line 279

```
theorem measure_reach_le {f : S → ℝ} {A : Set S} {C : ℝ}
  (hA : MeasurableSet A) (hf : Measurable f) (hf0 : ∀ y, 0 ≤ f y) (hfC : ∀ y, f y
    ≤ C)
  (hf1 : ∀ y ∈ A, 1 ≤ f y) (hdrift : ∀ y, y ∉ A → ∫ z, f z ∂K y ≤ f y) (x : S) :
  pathMeasure K x {ω | ∃ n, ω n ∈ A} ≤ ENNReal.ofReal (f x)
```

Qtsl.Battery.mixture\_integrable

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 468

```
lemma mixture_integrable {m : Md} {z : X} {κ : Measure (Md × X)}
  {w₀ : ℝ≥0} {w : J → ℝ≥0}
  (hVmeas : ∀ m', Measurable (V m'))
  (hκ : κ = (w₀ : ℝ≥0∞) • ((flow m) z).map (Prod.mk m)
    + ∑ j, (w j : ℝ≥0∞) • Measure.dirac (dst j, g j z))
  (hflowint : Integrable (fun ζ => V m ζ) ((flow m) z)) :
  Integrable (fun y : Md × X => V y.1 y.2) κ
```

Qtsl.LayerD.sound\_GSHS\_deadline\_mp

lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 539

```
theorem sound_GSHS_deadline_mp [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {θ ε : ℝ} {T : ℝ≥0}
  (hV : Measurable V) (hθ : 0 ≤ θ)
  (hmp : ∀ x : State, Nonempty (MPFlow M θ V x))
  (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) (hε : 0 < ε) (hFloor : ∀ y, ε ≤ V y) (x : State) :
  Battery.pathMeasure (tskel M) (TSkelState.alive (θ, x))
    {ω | ∃ n, T < tclock (ω n)}
    ≤ ENNReal.ofReal (Real.exp (-(θ * T)) * V x / ε)
```

Qtsl.LayerD.sound\_GSHS\_deadline\_battery\_mp

lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 552

```
theorem sound_GSHS_deadline_battery_mp
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {V : State → ℝ} {θ ε p thr : ℝ} {T : ℝ≥0}
  (hV : Measurable V) (hθ : 0 ≤ θ)
  (hmp : ∀ x : State, Nonempty (MPFlow M θ V x))
  (hForce : ForcedNonexpansion M V)
  (hNonneg : NonnegV V) (hε : 0 < ε) (hFloor : ∀ y, ε ≤ V y)
  (hθ : p + Real.exp (-(θ * T)) * thr / ε ≤ 1) (x : State)
  (hthr : V x ≤ thr) :
  ENNReal.ofReal p ≤ Battery.pathMeasure (tskel M) (TSkelState.alive (θ, x))
    {ω | ∀ n, tclock (ω n) ≤ T}
```

SuFunctor.Site.satAtM\_acceptanceSpecF\_ae

lean/su-functor/SuFunctor/Site/FellBehaviorSpec.lean, line 180

```

theorem satAtM_acceptanceSpecF_ae (U : ι → Set T)
  (E : ∀ i, Set (FTraj (U i) X)) (j : ι)
  {μ : MeasJoint (FTraj (U j) X)}
  (h : SatAtM μ (acceptanceSpecF U E j)) :
  (μ.1 : Measure (FTraj (U j) X)) (E j)c = 0

```

SuFunctor.Site.**hitEvent\_iUnion**

lean/su-functor/SuFunctor/Site/FellAcceptance.lean, line 123

```

theorem hitEvent_iUnion {ι : Sort*} (A : ι → Set α) :
  hitEvent (⋃ i, A i) = ⋃ i, hitEvent (A i)

```

SuFunctor.Site.**measurableSet\_hitEvent\_univ\_prod\_of\_isCompact**

lean/su-functor/SuFunctor/Site/FellAcceptance.

lean, line 397

```

theorem measurableSet_hitEvent_univ_prod_of_isCompact
  [SigmaCompactSpace ιW] {R : Set X} (hR : IsCompact R) :
  MeasurableSet
    (hitEvent ((Set.univ : Set ιW) ×s R) : Set (FTraj W X))

```

Qtsl.Battery.**sound\_QDeadline**

lean/qtsl-aux/QtslAux/Battery.lean, line 749

```

theorem sound_QDeadline {T : ℕ} {M : ℝ} {sat : S → Prop}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hB : MeasurableSet B)
  (hV : Measurable V)
  (hF : ∀ y ∈ B, sat y)
  (hβ : 0 ≤ β)
  (h0F : β ^ T * thr ≤ 1 - p)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ B → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ B → K y (D ∪ B) = 1)
  (hint : ∀ y ∈ D, y ∉ B → Integrable V (K y))
  (hdecay : ∀ y ∈ D, y ∉ B → ∫ z, V z ∂K y ≤ β * V y)
  (hfloor : ∀ y ∈ Dbar, y ∉ B → 1 ≤ V y)
  (hnonneg : ∀ y ∈ Dbar, 0 ≤ V y)
  (hM : ∀ y ∈ Dbar, V y ≤ M)
  (hx : x ∈ D) (hthr : V x ≤ thr) :
  ENNReal.ofReal p ≤ pathMeasure K x {ω | ∃ k ≤ T, sat (ω k)}

```

Qtsl.Battery.**sound\_QDeadlineBattery**

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 413

```

theorem sound_QDeadlineBattery {n : ℕ} {sat : S → Prop}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hB : MeasurableSet B)
  (hV : Measurable V)
  (hF : ∀ y ∈ B, sat y)
  (hβ : 0 ≤ β) (hβ0 : β ≤ Real.exp (-θ)) (h0 : 0 ≤ θ)
  (hq0 : 0 ≤ q)
  (hq : q ≤ ∑ k ∈ range (n + 1), (θ * T) ^ k / (Nat.factorial k : ℝ))
  (hp : p ≤ 1)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ B → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ B → K y (D ∪ B) = 1)
  (hint : ∀ y ∈ D, y ∉ B → Integrable V (K y))
  (hdecay : ∀ y ∈ D, y ∉ B → ∫ z, V z ∂K y ≤ β * V y)
  (hfloor : ∀ y ∈ Dbar, y ∉ B → 1 ≤ V y)
  (hnonneg : ∀ y ∈ Dbar, 0 ≤ V y)
  (hM : ∀ y ∈ Dbar, V y ≤ M)
  (hx : x ∈ D) (hthr : V x ≤ (1 - p) * q) :
  ENNReal.ofReal p ≤ pathMeasure K x {ω | ∃ k ≤ T, sat (ω k)}

```

Qtsl.Battery.**sound\_QDeadline\_hit**

lean/qtsl-aux/QtslAux/Battery.lean, line 568

```

theorem sound_QDeadline_hit {T : ℕ} {M : ℝ}
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hB : MeasurableSet B)

```

```

(hV : Measurable V)
(hβ : 0 ≤ β)
(h0F : β ^ T * thr ≤ 1 - p)
(hDsub : D ⊆ Dbar)
(hR3 : ∀ y ∈ D, y ∉ B → K y Dbar = 1)
(hR4 : ∀ y ∈ D, y ∉ B → K y (D ∪ B) = 1)
(hint : ∀ y ∈ D, y ∉ B → Integrable V (K y))
(hdecay : ∀ y ∈ D, y ∉ B → ∫ z, V z ∂K y ≤ β * V y)
(hfloor : ∀ y ∈ Dbar, y ∉ B → 1 ≤ V y)
(hnonneg : ∀ y ∈ Dbar, 0 ≤ V y)
(hM : ∀ y ∈ Dbar, V y ≤ M)
(hx : x ∈ D) (hthr : V x ≤ thr) :
ENNReal.ofReal p ≤ pathMeasure K x {ω | ∃ k ≤ T, ω k ∈ B}

```

MeasureTheory.supermartingale\_inv\_pow\_mul

lean/qtsl-aux/QtslAux/Deadline.lean, line 34

```

theorem supermartingale_inv_pow_mul [IsFiniteMeasure μ] {f : ℕ → Ω → ℝ} {β : ℝ}
(hadp : StronglyAdapted ℱ f) (hint : ∀ n, Integrable (f n) μ) (hβ : 0 < β)
(hdrift : ∀ n, μ[f (n + 1) | ℱ n] ≤m[μ] fun ω => β * f n ω) :
Supermartingale (fun n ω => β-1 ^ n * f n ω) ℱ μ

```

Qtsl.Deadline.expMoment\_hitting

lean/qtsl-aux/QtslAux/Deadline.lean, line 128

```

theorem expMoment_hitting (hβ : 0 < β)
(hint : ∀ n, Integrable (V n) μ) (hnonneg : 0 ≤ V)
(hmB : ∀ n, MeasurableSet[ℱ n] {ω | X n ω ∈ B})
(hdrift : ∀ n, ∀m ω ∂μ, X n ω ∉ B → (μ[V (n + 1) | ℱ n]) ω ≤ β * V n ω)
(hbarrier : ∀ n ω, X n ω ∈ B → a ≤ V n ω) :
ENNReal.ofReal a *
  ∑i k, ENNReal.ofReal (β-1 ^ k) * μ {ω | X k ω ∈ B ∧ ∀ j < k, X j ω ∉ B}
≤ ENNReal.ofReal (∫ ω, V 0 ω ∂μ)

```

Qtsl.Battery.ae\_no\_escape

lean/qtsl-aux/QtslAux/Battery.lean, line 502

```

private lemma ae_no_escape (hD : MeasurableSet D) (hB : MeasurableSet B)
(hR4 : ∀ y ∈ D, y ∉ B → K y (D ∪ B) = 1) (hx : x ∈ D) :
pathMeasure K x (⋃ n, {ω : ℕ → S | (∀ k ≤ n, ω k ∉ B) ∧ ω n ∈ D}) = 0

```

Qtsl.Battery.ae\_coord\_zero\_mem

lean/qtsl-aux/QtslAux/Battery.lean, line 484

```

private lemma ae_coord_zero_mem {E : Set S} (hE : MeasurableSet E) (hxE : x ∈ E) :
∀m ω ∂pathMeasure K x, ω 0 ∈ E

```

Qtsl.Battery.compl\_firstExit\_subset

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 70

```

theorem compl_firstExit_subset {S : Type*} (Stop Good : Set S) (T : ℕ) :
{ω : ℕ → S | ∃ n ≤ T, ω n ∈ Good ∧ ∀ j < n, ω j ∉ Stop}c ⊆
{ω : ℕ → S | ∃ n, ω n ∈ Stop \ Good ∧ ∀ j < n, ω j ∉ Stop} ∪
{ω : ℕ → S | ∀ k ≤ T, ω k ∉ Stop}

```

Qtsl.Battery.sound\_QUnionBound

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 50

```

theorem sound_QUnionBound {Ω : Type*} [MeasurableSpace Ω] (μ : Measure Ω)
{E₁ E₂ F : Set Ω} {a b : ℝ≥0∞}
(h₁ : μ E₁ ≤ a) (h₂ : μ E₂ ≤ b) (hsub : F ⊆ E₁ ∪ E₂) :
μ F ≤ a + b

```

Qtsl.Battery.sound\_QUnionBound\_ofReal

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 58

```

theorem sound_QUnionBound_ofReal {Ω : Type*} [MeasurableSpace Ω] (μ : Measure Ω)
{E₁ E₂ F : Set Ω} {a b : ℝ} (ha : 0 ≤ a) (hb : 0 ≤ b)
(h₁ : μ E₁ ≤ ENNReal.ofReal a) (h₂ : μ E₂ ≤ ENNReal.ofReal b)
(hsub : F ⊆ E₁ ∪ E₂) :
μ F ≤ ENNReal.ofReal (a + b)

```

Qtsl.Battery.sound\_QUntilBattery\_firstHit

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 206

```

theorem sound_QUntilBattery_firstHit
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar)
  (hA : MeasurableSet A) (hR : MeasurableSet R)
  (hVa : Measurable Va) (hVd : Measurable Vd)
  (hβ : 0 ≤ β) (hc : β ^ T ≤ c)
  (hDsub : D ⊆ Dbar)
  (hR3 : ∀ y ∈ D, y ∉ A ∪ R → K y Dbar = 1)
  (hR4 : ∀ y ∈ D, y ∉ A ∪ R → K y (D ∪ (A ∪ R)) = 1)
  (hVaint : ∀ y ∈ D, y ∉ A ∪ R → Integrable Va (K y))
  (hVadrift : ∀ y ∈ D, y ∉ A ∪ R → ∫ z, Va z ∂K y ≤ Va y)
  (hVaflor : ∀ y ∈ A, y ∈ Dbar → 1 ≤ Va y)
  (hVanonneg : ∀ y ∈ Dbar, 0 ≤ Va y)
  (hVdint : ∀ y ∈ D, y ∉ A ∪ R → Integrable Vd (K y))
  (hVddecay : ∀ y ∈ D, y ∉ A ∪ R → ∫ z, Vd z ∂K y ≤ β * Vd y)
  (hVdfloor : ∀ y ∈ Dbar, y ∉ A ∪ R → 1 ≤ Vd y)
  (hVdnonneg : ∀ y ∈ Dbar, 0 ≤ Vd y)
  (hVdM : ∀ y ∈ Dbar, Vd y ≤ M)
  (hx : x ∈ D) (hthr : Va x + c * Vd x ≤ 1 - p) :
  ENNReal.ofReal p ≤ pathMeasure K x
  {ω | ∃ n ≤ T, ω n ∈ R ∧ ∀ j < n, ω j ∉ (A ∪ R) ∪ Dc}

```

Qtsl.LayerD.sound\_GSHS\_until\_mp

lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 584

```

theorem sound_GSHS_until_mp [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {Va Vd : State → ℝ} {Us : Set State} {θ ε : ℝ} {T : ℝ≥0}
  (hVa : Measurable Va) (hUs : MeasurableSet Us)
  (hmpa : ∀ x : State, Nonempty (MPFlow M θ Va x))
  (hForcea : ForcedNonexpansion M Va)
  (hNonnega : NonnegV Va) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, Va y ≤ C)
  (hFloora : ∀ y ∈ Us, 1 ≤ Va y)
  (hVd : Measurable Vd) (hθ : 0 ≤ θ)
  (hmpd : ∀ x : State, Nonempty (MPFlow M θ Vd x))
  (hForced : ForcedNonexpansion M Vd)
  (hNonnegd : NonnegV Vd) (hε : 0 < ε) (hFloord : ∀ y, ε ≤ Vd y)
  (x : State) :
  Battery.pathMeasure (tskel M) (TSkelState.alive (θ, x))
    {ω | ∃ n, ω n ∈ tbadIn Us ∨ T < tclock (ω n)}
    ≤ ENNReal.ofReal (Va x)
      + ENNReal.ofReal (Real.exp (-(θ * T)) * Vd x / ε)

```

Qtsl.LayerD.sound\_GSHS\_until\_battery\_mp

lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 644

```

theorem sound_GSHS_until_battery_mp
  [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {Va Vd : State → ℝ} {Us : Set State} {θ ε p thra thrd : ℝ} {T : ℝ≥0}
  (hVa : Measurable Va) (hUs : MeasurableSet Us)
  (hmpa : ∀ x : State, Nonempty (MPFlow M θ Va x))
  (hForcea : ForcedNonexpansion M Va)
  (hNonnega : NonnegV Va) {C : ℝ} (hC0 : 0 ≤ C) (hVC : ∀ y, Va y ≤ C)
  (hFloora : ∀ y ∈ Us, 1 ≤ Va y)
  (hVd : Measurable Vd) (hθ : 0 ≤ θ)
  (hmpd : ∀ x : State, Nonempty (MPFlow M θ Vd x))
  (hForced : ForcedNonexpansion M Vd)
  (hNonnegd : NonnegV Vd) (hε : 0 < ε) (hFloord : ∀ y, ε ≤ Vd y)
  (hθ : p + (thra + Real.exp (-(θ * T)) * thrd / ε) ≤ 1)
  (x : State) (hthra : Va x ≤ thra) (hthrd : Vd x ≤ thrd) :
  ENNReal.ofReal p ≤ Battery.pathMeasure (tskel M) (TSkelState.alive (θ, x))
    {ω | ∀ n, ω n ∉ tbadIn Us ∧ tclock (ω n) ≤ T}

```

SuFunctor.Site.untilEvent

lean/su-functor/SuFunctor/Site/FellAcceptance.lean, line 343

```

def untilEvent (Reach Avoid : ℕ → Set α) : Set (FellClosed α)

```

SuFunctor.Site.measurableSet\_untilEvent

lean/su-functor/SuFunctor/Site/FellAcceptance.lean, line 353

```

theorem measurableSet_untilEvent {Reach Avoid :  $\mathbb{N} \rightarrow \text{Set } \alpha$ }
  (h1 :  $\forall n, \text{MeasurableSet } (\text{hitEvent } (\text{Reach } n))$ )
  (h2 :  $\forall n, \text{MeasurableSet } (\text{missEvent } (\text{Avoid } n))$ ) :
  MeasurableSet (untilEvent Reach Avoid)

```

Qtsl.Battery.sound\_QUntilBattery

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 371

```

theorem sound_QUntilBattery {sat_s sat :  $S \rightarrow \text{Prop}$ }
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar)
  (hA : MeasurableSet A) (hR : MeasurableSet R)
  (hVa : Measurable Va) (hVd : Measurable Vd)
  (hFs :  $\forall y, \neg \text{sat}_s y \rightarrow y \in A$ ) (hFt :  $\forall y \in R, \text{sat } y$ )
  (h $\beta$  :  $0 \leq \beta$ ) (hc :  $\beta \wedge T \leq c$ )
  (hDsub :  $D \subseteq \text{Dbar}$ )
  (hR3 :  $\forall y \in D, y \notin A \cup R \rightarrow K y \text{ Dbar} = 1$ )
  (hR4 :  $\forall y \in D, y \notin A \cup R \rightarrow K y (D \cup (A \cup R)) = 1$ )
  (hVaint :  $\forall y \in D, y \notin A \cup R \rightarrow \text{Integrable } Va (K y)$ )
  (hVadrift :  $\forall y \in D, y \notin A \cup R \rightarrow \int z, Va z \partial K y \leq Va y$ )
  (hVafloor :  $\forall y \in A, y \in \text{Dbar} \rightarrow 1 \leq Va y$ )
  (hVanonneg :  $\forall y \in \text{Dbar}, 0 \leq Va y$ )
  (hVdint :  $\forall y \in D, y \notin A \cup R \rightarrow \text{Integrable } Vd (K y)$ )
  (hVddecay :  $\forall y \in D, y \notin A \cup R \rightarrow \int z, Vd z \partial K y \leq \beta * Vd y$ )
  (hVdfloor :  $\forall y \in \text{Dbar}, y \notin A \cup R \rightarrow 1 \leq Vd y$ )
  (hVdnonneg :  $\forall y \in \text{Dbar}, 0 \leq Vd y$ )
  (hVdM :  $\forall y \in \text{Dbar}, Vd y \leq M$ )
  (hx :  $x \in D$ ) (hthr :  $Va x + c * Vd x \leq 1 - p$ ) :
  ENNReal.ofReal p  $\leq \text{pathMeasure } K x$ 
  { $\omega \mid \exists n \leq T, \text{sat } (\omega n) \wedge \forall j < n, \text{sat}_s (\omega j)$ }

```

Qtsl.Battery.measure\_firstBad\_le

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 110

```

theorem measure_firstBad_le {K : Kernel S S} [IsMarkovKernel K]
  {f :  $S \rightarrow \mathbb{R}$ } {Stop Bad : Set S} {C :  $\mathbb{R}$ }
  (hStop : MeasurableSet Stop) (hf : Measurable f)
  (hf0 :  $\forall y, 0 \leq f y$ ) (hfC :  $\forall y, f y \leq C$ )
  (hf1 :  $\forall y \in \text{Bad}, 1 \leq f y$ )
  (hdrift :  $\forall y, y \notin \text{Stop} \rightarrow \int z, f z \partial K y \leq f y$ ) (x : S) :
  pathMeasure K x { $\omega \mid \exists n, \omega n \in \text{Bad} \wedge \forall j < n, \omega j \notin \text{Stop}$ }
   $\leq \text{ENNReal.ofReal } (f x)$ 

```

Qtsl.Battery.measure\_missTarget\_le

lean/qtsl-aux/QtslAux/QtslBattery.lean, line 143

```

theorem measure_missTarget_le {K : Kernel S S} [IsMarkovKernel K]
  {D Dbar B : Set S} {V :  $S \rightarrow \mathbb{R}$ } { $\beta : \mathbb{R}$ } {x : S} {T :  $\mathbb{N}$ } {M :  $\mathbb{R}$ }
  (hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hB : MeasurableSet B)
  (hV : Measurable V) (h $\beta$  :  $0 \leq \beta$ )
  (hDsub :  $D \subseteq \text{Dbar}$ )
  (hR3 :  $\forall y \in D, y \notin B \rightarrow K y \text{ Dbar} = 1$ )
  (hR4 :  $\forall y \in D, y \notin B \rightarrow K y (D \cup B) = 1$ )
  (hint :  $\forall y \in D, y \notin B \rightarrow \text{Integrable } V (K y)$ )
  (hdecay :  $\forall y \in D, y \notin B \rightarrow \int z, V z \partial K y \leq \beta * V y$ )
  (hfloor :  $\forall y \in \text{Dbar}, y \notin B \rightarrow 1 \leq V y$ )
  (hnonneg :  $\forall y \in \text{Dbar}, 0 \leq V y$ )
  (hM :  $\forall y \in \text{Dbar}, V y \leq M$ )
  (hx :  $x \in D$ ) :
  pathMeasure K x { $\omega \mid \forall k \leq T, \omega k \notin B$ }  $\leq \text{ENNReal.ofReal } (\beta \wedge T * V x)$ 

```

MeasureTheory.Supermartingale.ville\_ineq

lean/qtsl-aux/QtslAux/Ville.lean, line 133

```

theorem Supermartingale.ville_ineq [IsFiniteMeasure  $\mu$ ] (hsup : Supermartingale f  $\mathcal{G}$ 
   $\mu$ )

```

```
(hnonneg : 0 ≤ f) (ε : ℝ≥0) :
ε * μ {ω | ∃ k, (ε : ℝ) ≤ f k ω} ≤ ENNReal.ofReal (∫ ω, f 0 ω dμ)
```

Qtsl.QStreett. **absorptionBound\_of\_kernel**

lean/qtsl-aux/QtslAux/QStreett.lean, line 804

```
theorem absorptionBound_of_kernel {A J Rq : Set S} {V : S → ℝ} {C γ : ℝ} {x : S}
(hRq : MeasurableSet Rq) (hV : Measurable V)
(hV0 : ∀ y, 0 ≤ V y) (hVC : ∀ y, V y ≤ C)
(hVfloor : ∀ y ∈ Rq, 1 ≤ V y)
(hVdrift : ∀ y, y ∉ Rq → ∫ z, V z dK y ≤ V y)
(hVJ : ∀ y ∈ J, V y ≤ 1 - γ) :
AbsorptionBound K x A J Rq γ
```

Qtsl.LayerD. **sound\_GSHS\_streett\_bundle\_mp**

lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 612

```
theorem sound_GSHS_streett_bundle_mp
{ι : Type*} [Countable ι]
{D Dbar U : Set State} {V₀ : State → ℝ} {Rq G : ι → Set State}
{W : ι → State → ℝ} {p thr C₀ : ℝ} {ε C : ι → ℝ} {x : State}
[IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
(hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
(hV₀ : Measurable V₀)
(hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
(hW : ∀ i, Measurable (W i))
(h0 : p + thr ≤ 1) (hDsub : D ⊆ Dbar)
(hLive : NoVanish M (D \ U)) (hHalt : HaltsCaught M (D \ U) (U n Dbar))
(hReset : ResetsInto M (D \ U) ((D ∪ U) n Dbar))
(hmp₀ : ∀ x : State, Nonempty (MPFlow M 0 V₀ x))
(hForce₀ : ForcedNonexpansion M V₀) (hNonneg₀ : NonnegV V₀)
(hC₀0 : 0 ≤ C₀) (hV₀C : ∀ y, V₀ y ≤ C₀)
(hFloor₀ : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V₀ y)
(hε : ∀ i, 0 < ε i)
(hmpW : ∀ i, ∀ x : State, Nonempty (MPFlow M 0 (W i) x))
(hSegDWR : ∀ i, RankingSegmentDynkin M ((D \ U) n Rq i) \ G i) (W i) (ε i))
(hForceW : ∀ i, ForcedNonexpansion M (W i)) (hNonnegW : ∀ i, NonnegV (W i))
(hC0 : ∀ i, 0 ≤ C i) (hWC : ∀ i, ∀ y, W i y ≤ C i)
(hx : x ∈ D) (hthr : V₀ x ≤ thr) :
ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
{ω | ∀ i, (∃ n in atTop, ω n ∈ skelIn (Rq i)) →
  ∃ n in atTop, ω n ∈ skelIn (G i)}
```

Qtsl.LayerD. **sound\_GSHS\_streett\_bundle\_mp\_strict**

lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 1074

```
theorem sound_GSHS_streett_bundle_mp_strict
{ι : Type*} [Countable ι]
{D Dbar U : Set State} {V₀ : State → ℝ} {Rq G : ι → Set State}
{W : ι → State → ℝ} {p thr C₀ : ℝ} {ε C : ι → ℝ} {x : State}
[IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
(hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
(hV₀ : Measurable V₀)
(hRq : ∀ i, MeasurableSet (Rq i)) (hG : ∀ i, MeasurableSet (G i))
(hW : ∀ i, Measurable (W i))
(h0 : p + thr ≤ 1) (hDsub : D ⊆ Dbar)
(hLive : NoVanish M (D \ U)) (hHalt : HaltsCaught M (D \ U) (U n Dbar))
(hReset : ResetsInto M (D \ U) ((D ∪ U) n Dbar))
(hmp₀ : ∀ x : State, Nonempty (MPFlow M 0 V₀ x))
(hForce₀ : ForcedNonexpansion M V₀) (hNonneg₀ : NonnegV V₀)
(hC₀0 : 0 ≤ C₀) (hV₀C : ∀ y, V₀ y ≤ C₀)
(hFloor₀ : ∀ y ∈ U, y ∈ Dbar → 1 ≤ V₀ y)
(hε : ∀ i, 0 < ε i)
(hmpW : ∀ i, ∀ x : State, Nonempty (MPFlow M 0 (W i) x))
(hmpWR : ∀ i, ∀ x ∈ ((D \ U) n Rq i) \ G i,
  Nonempty (MPFlowStrict M 0 (W i) x (ε i)))
```



```

(hForceW : ∀ i, ForcedNonexpansion M (W i)) (hNonnegW : ∀ i, NonnegV (W i))
(hC0 : ∀ i, 0 ≤ C i) (hWC : ∀ i, ∀ y, W i y ≤ C i)
(hx : x ∈ D) (hthr : V 0 x ≤ thr) :
ENNReal.ofReal p ≤ Battery.pathMeasure (skel M) (SkelState.alive x)
{ω | ∀ i, (∃f n in atTop, ω n ∈ skelIn (Rq i)) →
  ∃f n in atTop, ω n ∈ skelIn (G i)}

```

Qtsl.LayerD.**rankingSegmentDynkin\_of\_mp\_strict** lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 1060

```

theorem rankingSegmentDynkin_of_mp_strict (M : SegmentModel State)
  (V : State → ℝ) [IsSFiniteKernel M.jump] (hV : Measurable V)
  {Sc : Set State} {ε : ℝ} (hε : 0 ≤ ε)
  (h : ∀ x ∈ Sc, Nonempty (MPFlowStrict M 0 V x ε)) :
  RankingSegmentDynkin M Sc V ε

```

Qtsl.LayerD.**MPFlowStrict** lean/qtsl-aux/QtslAux/LayerD/MPDischarge.lean, line 1047

```

structure MPFlowStrict (M : SegmentModel State) (θ : ℝ) (V : State → ℝ)
  (x : State) (ε : ℝ) extends MPFlow M θ V x where ...

```

Qtsl.Excursions.**streett\_ae\_frequently\_response** lean/qtsl-aux/QtslAux/Excursions.lean, line 365

```

theorem streett_ae_frequently_response (hε : 0 < ε₀)
  (hint : ∀ n, Integrable (V n) μ) (hnonneg : 0 ≤ V)
  (hmR : ∀ n, MeasurableSet[ℱ n] {ω | X n ω ∈ R})
  (hmG : ∀ n, MeasurableSet[ℱ n] {ω | X n ω ∈ G})
  (hdrift : ∀ n, ∀m ω ∂μ, X n ω ∉ G → (μ[V (n + 1) | ℱ n]) ω ≤ V n ω)
  (hdriftR : ∀ n, ∀m ω ∂μ,
    X n ω ∈ R → X n ω ∉ G → (μ[V (n + 1) | ℱ n]) ω ≤ V n ω - ε₀) :
  ∀m ω ∂μ, (∃f n in atTop, X n ω ∈ R) → ∃f n in atTop, X n ω ∈ G

```

Qtsl.Excursions.**streett\_window\_sum\_le** lean/qtsl-aux/QtslAux/Excursions.lean, line 319

```

theorem streett_window_sum_le (hε : 0 ≤ ε₀)
  (hint : ∀ n, Integrable (V n) μ) (hnonneg : 0 ≤ V)
  (hmR : ∀ n, MeasurableSet[ℱ n] {ω | X n ω ∈ R})
  (hmG : ∀ n, MeasurableSet[ℱ n] {ω | X n ω ∈ G})
  (hdrift : ∀ n, ∀m ω ∂μ, X n ω ∉ G → (μ[V (n + 1) | ℱ n]) ω ≤ V n ω)
  (hdriftR : ∀ n, ∀m ω ∂μ,
    X n ω ∈ R → X n ω ∉ G → (μ[V (n + 1) | ℱ n]) ω ≤ V n ω - ε₀)
  (N : ℕ) :
  ENNReal.ofReal ε₀ * ∑' j : ℕ,
    μ ({ω | ∀ k, N ≤ k → k < N + j + 1 → X k ω ∉ G} ∩ {ω | X (N + j) ω ∈ R})
  ≤ ENNReal.ofReal (∫ ω, V 0 ω ∂μ)

```

Qtsl.Excursions.**ranking\_hitting\_sum\_le** lean/qtsl-aux/QtslAux/Excursions.lean, line 147

```

theorem ranking_hitting_sum_le (hε : 0 ≤ ε₀)
  (hint : ∀ n, Integrable (V n) μ) (hnonneg : 0 ≤ V)
  (hmB : ∀ n, MeasurableSet[ℱ n] {ω | X n ω ∈ B})
  (hdrift : ∀ n, ∀m ω ∂μ, X n ω ∉ B → (μ[V (n + 1) | ℱ n]) ω ≤ V n ω - ε₀) :
  ENNReal.ofReal ε₀ * ∑' k, μ {ω | ∀ j < k + 1, X j ω ∉ B}
  ≤ ENNReal.ofReal (∫ ω, V 0 ω ∂μ)

```

Qtsl.Excursions.**ae\_hits\_of\_ranking** lean/qtsl-aux/QtslAux/Excursions.lean, line 174

```

theorem ae_hits_of_ranking (hε : 0 < ε₀)
  (hint : ∀ n, Integrable (V n) μ) (hnonneg : 0 ≤ V)
  (hmB : ∀ n, MeasurableSet[ℱ n] {ω | X n ω ∈ B})
  (hdrift : ∀ n, ∀m ω ∂μ, X n ω ∉ B → (μ[V (n + 1) | ℱ n]) ω ≤ V n ω - ε₀) :
  ∀m ω ∂μ, ∃ n, X n ω ∈ B

```

MeasureTheory.Supermartingale.**exists\_ae\_tendsto\_of\_nonneg** lean/qtsl-aux/QtslAux/Excursions.lean, line 42

```

theorem Supermartingale.exists_ae_tendsto_of_nonneg [IsFiniteMeasure  $\mu$ ] {f :  $\mathbb{N} \rightarrow \mathbb{R}$ }
  →  $\mathbb{R}$ }
(hf : Supermartingale f  $\mathcal{G}$   $\mu$ ) (hnonneg :  $0 \leq f$ ) :
 $\forall^\mu \omega \partial \mu, \exists c, \text{Tendsto } (\text{fun } n \Rightarrow f\ n\ \omega) \text{ atTop } (\nearrow c)$ 

```

Qtsl.QStreett.streett\_event\_mono

lean/qtsl-aux/QtslAux/QStreett.lean, line 418

```

theorem streett_event_mono {Rq Rqhat G Ghat : Set S} (hR : Rq  $\subseteq$  Rqhat)
(hG : Ghat  $\subseteq$  G) ( $\omega$  :  $\mathbb{N} \rightarrow S$ )
(h : ( $\exists^\text{f} n$  in atTop,  $\omega\ n \in$  Rqhat)  $\rightarrow \exists^\text{f} n$  in atTop,  $\omega\ n \in$  Ghat) :
( $\exists^\text{f} n$  in atTop,  $\omega\ n \in$  Rq)  $\rightarrow \exists^\text{f} n$  in atTop,  $\omega\ n \in$  G

```

Qtsl.QStreett.streett\_pair\_localized

lean/qtsl-aux/QtslAux/QStreett.lean, line 244

```

theorem streett_pair_localized {D Dbar U Rq G : Set S} {W : S  $\rightarrow \mathbb{R}$ } { $\epsilon$  C :  $\mathbb{R}$ } {x : S}
(hD : MeasurableSet D) (hDbar : MeasurableSet Dbar) (hU : MeasurableSet U)
(hRq : MeasurableSet Rq) (hG : MeasurableSet G) (hW : Measurable W)
(h $\epsilon$  :  $0 < \epsilon$ ) (hDsub : D  $\subseteq$  Dbar)
(hR3 :  $\forall y \in D, y \notin U \rightarrow K\ y\ Dbar = 1$ )
(hW0 :  $\forall y \in Dbar, 0 \leq W\ y$ ) (hWC :  $\forall y \in Dbar, W\ y \leq C$ )
(hWint :  $\forall y \in D, y \notin U \rightarrow y \notin G \rightarrow \text{Integrable } W\ (K\ y)$ )
(hWdrift :  $\forall y \in D, y \notin U \rightarrow y \notin G \rightarrow \int z, W\ z\ \partial K\ y \leq W\ y$ )
(hWdriftR :  $\forall y \in D, y \notin U \rightarrow y \in Rq \rightarrow y \notin G \rightarrow \int z, W\ z\ \partial K\ y \leq W\ y - \epsilon$ ) :
 $\forall^\mu \omega \partial \text{pathMeasure } K\ x, (\forall n, \omega\ n \in D \setminus U) \rightarrow$ 
( $\exists^\text{f} n$  in atTop,  $\omega\ n \in$  Rq)  $\rightarrow \exists^\text{f} n$  in atTop,  $\omega\ n \in$  G

```

Qtsl.Excursions.streett\_window\_invariant

lean/qtsl-aux/QtslAux/Excursions.lean, line 201

```

theorem streett_window_invariant
(hint :  $\forall n, \text{Integrable } (V\ n)\ \mu$ ) (hnonneg :  $0 \leq V$ )
(hmR :  $\forall n, \text{MeasurableSet } [\mathcal{G}\ n] \{ \omega \mid X\ n\ \omega \in R \}$ )
(hmG :  $\forall n, \text{MeasurableSet } [\mathcal{G}\ n] \{ \omega \mid X\ n\ \omega \in G \}$ )
(hdrift :  $\forall n, \forall^\mu \omega \partial \mu, X\ n\ \omega \notin G \rightarrow (\mu[V\ (n + 1) \mid \mathcal{G}\ n])\ \omega \leq V\ n\ \omega$ )
(hdriftR :  $\forall n, \forall^\mu \omega \partial \mu,$ 
 $X\ n\ \omega \in R \rightarrow X\ n\ \omega \notin G \rightarrow (\mu[V\ (n + 1) \mid \mathcal{G}\ n])\ \omega \leq V\ n\ \omega - \epsilon_0$ )
(N :  $\mathbb{N}$ ) :  $\forall n, N \leq n \rightarrow$ 
 $\int \omega \text{ in } \{ \omega \mid \forall k, N \leq k \rightarrow k < n \rightarrow X\ k\ \omega \notin G \}, V\ n\ \omega \partial \mu$ 
 $+ \epsilon_0 * \sum k \in \text{Ico } N\ n,$ 
 $\mu.\text{real } (\{ \omega \mid \forall j, N \leq j \rightarrow j < k + 1 \rightarrow X\ j\ \omega \notin G \} \cap \{ \omega \mid X\ k\ \omega \in R \})$ 
 $\leq \int \omega, V\ N\ \omega \partial \mu$ 

```

Qtsl.QStreett.AbsorptionBound

lean/qtsl-aux/QtslAux/QStreett.lean, line 479

```

def AbsorptionBound (K : Kernel S S) [IsMarkovKernel K] (x : S)
(A J Rq : Set S) ( $\gamma$  :  $\mathbb{R}$ ) : Prop

```

Qtsl.QStreett.ae\_not\_frequently\_of\_absorption

lean/qtsl-aux/QtslAux/QStreett.lean, line 490

```

theorem ae_not_frequently_of_absorption {A J Rq : Set S} { $\gamma$  :  $\mathbb{R}$ } {x : S}
(hRq : MeasurableSet Rq) (h $\gamma$  :  $0 < \gamma$ )
(habs : AbsorptionBound K x A J Rq  $\gamma$ ) :
 $\forall^\mu \omega \partial \text{pathMeasure } K\ x,$ 
( $\exists^\text{f} k$  in atTop,  $\omega \in \text{survive } A\ k \wedge \omega\ k \in J$ )  $\rightarrow$ 
 $\neg \exists^\text{f} n$  in atTop,  $\omega\ n \in Rq$ 

```

Qtsl.QStreett.supermartingale\_stopped\_shift

lean/qtsl-aux/QtslAux/QStreett.lean, line 740

```

theorem supermartingale_stopped_shift {f : S  $\rightarrow \mathbb{R}$ } {R : Set S} {C :  $\mathbb{R}$ }
(hR : MeasurableSet R) (hf : Measurable f) (hf0 :  $\forall y, 0 \leq f\ y$ ) (hfC :  $\forall y, f\ y \leq C$ )
(hdrift :  $\forall y, y \notin R \rightarrow \int z, f\ z\ \partial K\ y \leq f\ y$ ) (k :  $\mathbb{N}$ ) (u :  $\prod\_i : \text{Iic } k, S$ ) :
Supermartingale (fun n  $\omega \Rightarrow$  stopped f R n (shift k  $\omega$ )) (piLEShift k)
(Kernel.traj (X := fun _ :  $\mathbb{N} \Rightarrow S$ ) ( $\kappa := \text{mchain } K$ ) k u)

```

Qtsl.QStreett.**measure\_reach\_after\_le** lean/qtsl-aux/QtslAux/QStreett.lean, line 760

```
theorem measure_reach_after_le {f : S → ℝ} {R : Set S} {C : ℝ}
  (hR : MeasurableSet R) (hf : Measurable f) (hf0 : ∀ y, 0 ≤ f y) (hfC : ∀ y, f y
    ≤ C)
  (hf1 : ∀ y ∈ R, 1 ≤ f y) (hdrift : ∀ y, y ∉ R → ∫ z, f z ∂K y ≤ f y)
  (k : ℕ) (u : Π _i : Iic k, S) :
  Kernel.traj (X := fun _ : ℕ => S) (κ := mchain K) k u {ω | ∃ n, k ≤ n ∧ ω n ∈ R}
    ≤ ENNReal.ofReal (f (u (k, mem_Iic.2 le_rfl)))
```

PPLC.Qtsl.Behavior.**AService** lean/qtsl/Qtsl/TimedCore.lean, line 32

```
def Behavior.AService (B : Behavior X) (T : ℝ≥0) (φ : Set X) (x : X) : Prop
```

PPLC.Qtsl.Behavior.**BoxInv** lean/qtsl/Qtsl/TimedCore.lean, line 35

```
def Behavior.BoxInv (B : Behavior X) (φ : Set X) (x : X) : Prop
```

PPLC.Qtsl.Behavior.**FusionClosed** lean/qtsl/Qtsl/TimedCore.lean, line 28

```
def Behavior.FusionClosed (B : Behavior X) : Prop
```

PPLC.Qtsl.Behavior.**FirstVisits** lean/qtsl/Qtsl/TimedCore.lean, line 38

```
def Behavior.FirstVisits (B : Behavior X) (φ : Set X) : Prop
```

PPLC.Qtsl.Behavior.**fsTarget** lean/qtsl/Qtsl/TimedCore.lean, line 41

```
def Behavior.fsTarget (B : Behavior X) {n : ℕ}
  (α : Fin n → Set X) (T : Fin n → ℝ≥0) : Set X
```

PPLC.Qtsl.Behavior.**FirstVisits** lean/qtsl/Qtsl/TimedCore.lean, line 38

```
def Behavior.FirstVisits (B : Behavior X) (φ : Set X) : Prop
```

PPLC.Qtsl.Behavior.**residualCell** lean/qtsl/Qtsl/TimedCore.lean, line 45

```
def Behavior.residualCell (B : Behavior X) {n : ℕ}
  (α : Fin n → Set X) (T : Fin n → ℝ≥0) (c : ℝ≥0) : Set X
```

Qtsl.NNF.**denot\_elimNeg** lean/qtsl-aux/QtslAux/NNFDuality.lean, line 198

```
theorem denot_elimNeg {σ : Type*} (den : α → Set σ) (d : α → α)
  (hd : ∀ a, den (d a) = (den a)c) :
  ∀ φ : NForm α, (φ.elimNeg d).denot den = φ.denot den
```

Qtsl.QStreett.**ranking\_guard\_collapse** lean/qtsl-aux/QtslAux/QStreett.lean, line 464

```
theorem ranking_guard_collapse {W : S → ℝ} {T slab : Set S} {g : S → S}
  (hzero : ∀ y ∈ T, W y = 0) (himg : ∀ y ∈ slab, g y ∈ T) :
  ∀ y ∈ slab, 0 ≤ W y → W (g y) ≤ W y
```

Qtsl.NNF.**compl\_le\_atom** lean/qtsl-aux/QtslAux/NNFDuality.lean, line 36

```
theorem compl_le_atom : {x | f x ≤ c}c = {x | c < f x}
```

Qtsl.NNF.**compl\_lt\_atom** lean/qtsl-aux/QtslAux/NNFDuality.lean, line 41

```
theorem compl_lt_atom : {x | f x < c}c = {x | c ≤ f x}
```

Qtsl.NNF.**compl\_ge\_atom** lean/qtsl-aux/QtslAux/NNFDuality.lean, line 46

```
theorem compl_ge_atom : {x | c ≤ f x}c = {x | f x < c}
```

Qtsl.NNF.**compl\_gt\_atom** lean/qtsl-aux/QtslAux/NNFDuality.lean, line 51

|                                        |                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                  |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
|                                        | theorem compl_gt_atom : $\{x \mid c < f\ x\}^c = \{x \mid f\ x \leq c\}$                                                                                                                                                                                                                                                                                                                                                |                                                  |
| Qtsl.NNF.denot_nnfAux                  |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/NNFDuality.lean, line 111  |
|                                        | theorem denot_nnfAux : $\forall (\varphi : \text{Form } \alpha) (b : \text{Bool}),$<br>$(\varphi.\text{nnfAux } b).\text{denot den} = \text{if } b \text{ then } \varphi.\text{denot den} \text{ else } (\varphi.\text{denot den})^c$                                                                                                                                                                                   |                                                  |
| Qtsl.NNF.Form.nnf                      |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/NNFDuality.lean, line 87   |
|                                        | def Form.nnf ( $\varphi : \text{Form } \alpha$ ) : NForm $\alpha$                                                                                                                                                                                                                                                                                                                                                       |                                                  |
| Qtsl.NNF.denot_nnf                     |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/NNFDuality.lean, line 128  |
|                                        | theorem denot_nnf ( $\varphi : \text{Form } \alpha$ ) : $(\varphi.\text{nnf}).\text{denot den} = \varphi.\text{denot den}$                                                                                                                                                                                                                                                                                              |                                                  |
| Qtsl.NNF.negLits_nnf                   |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/NNFDuality.lean, line 170  |
|                                        | theorem negLits_nnf ( $\varphi : \text{Form } \alpha$ ) : $(\varphi.\text{nnf}).\text{negLits} = \varphi.\text{occs false}$                                                                                                                                                                                                                                                                                             |                                                  |
| Qtsl.NNF.nnf_no_negLit_iff             |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/NNFDuality.lean, line 175  |
|                                        | theorem nnf_no_negLit_iff ( $\varphi : \text{Form } \alpha$ ) ( $P : \alpha \rightarrow \text{Prop}$ ) :<br>$(\forall a \in (\varphi.\text{nnf}).\text{negLits}, \neg P\ a) \leftrightarrow \forall a \in \varphi.\text{occs false}, \neg P\ a$                                                                                                                                                                         |                                                  |
| Qtsl.NNF.negLits_elimNeg               |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/NNFDuality.lean, line 188  |
|                                        | theorem negLits_elimNeg ( $d : \alpha \rightarrow \alpha$ ) : $\forall \varphi : \text{NForm } \alpha, (\varphi.\text{elimNeg } d).\text{negLits} = \emptyset$                                                                                                                                                                                                                                                          |                                                  |
| Qtsl.NNF.negLits_nnfAux                |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/NNFDuality.lean, line 153  |
|                                        | theorem negLits_nnfAux : $\forall (\varphi : \text{Form } \alpha) (b : \text{Bool}),$<br>$(\varphi.\text{nnfAux } b).\text{negLits} = \varphi.\text{occs (!}b\text{)}$                                                                                                                                                                                                                                                  |                                                  |
| Qtsl.Aliases.sugarAvoid_denot          |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/QtslAliases.lean, line 95  |
|                                        | theorem sugarAvoid_denot $\{S : \text{Type}^*\} (B : \text{Set } S)$ :<br>$\{\omega : \mathbb{N} \rightarrow S \mid \forall n, \omega\ n \notin B\} = \{\omega : \mathbb{N} \rightarrow S \mid \forall n, \omega\ n \in B^c\}$                                                                                                                                                                                          |                                                  |
| Qtsl.Aliases.sugarGF_denot             |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/QtslAliases.lean, line 98  |
|                                        | theorem sugarGF_denot $\{S : \text{Type}^*\} (G : \text{Set } S) (\omega : \mathbb{N} \rightarrow S)$ :<br>$((\exists^f n \text{ in atTop}, \omega\ n \in (\text{Set.univ} : \text{Set } S)) \rightarrow \exists^f n \text{ in atTop}, \omega\ n \in G)$<br>$\leftrightarrow \exists^f n \text{ in atTop}, \omega\ n \in G$                                                                                             |                                                  |
| Qtsl.Aliases.sugarFG_denot             |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/QtslAliases.lean, line 105 |
|                                        | theorem sugarFG_denot $\{S : \text{Type}^*\} (B : \text{Set } S) (\omega : \mathbb{N} \rightarrow S)$ :<br>$((\exists^f n \text{ in atTop}, \omega\ n \in B) \rightarrow \exists^f n \text{ in atTop}, \omega\ n \in (\emptyset : \text{Set } S))$<br>$\leftrightarrow \forall^f n \text{ in atTop}, \omega\ n \notin B$                                                                                                |                                                  |
| Qtsl.Aliases.negF_denot_subset         |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/QtslAliases.lean, line 84  |
|                                        | theorem negF_denot_subset $\{S : \text{Type}^*\} (\text{sat} : S \rightarrow \text{Prop}) (T : \mathbb{N})$ :<br>$\{\omega : \mathbb{N} \rightarrow S \mid \forall n, \neg \text{sat } (\omega\ n)\} \subseteq \{\omega : \mathbb{N} \rightarrow S \mid \neg \exists k \leq T, \text{sat } (\omega\ k)\}$                                                                                                               |                                                  |
| Qtsl.Structural.hypMultiplier_sound    |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/Structural.lean, line 75   |
|                                        | theorem hypMultiplier_sound ( $\text{cell childAtom} : \text{Set } X$ ) ( $N \text{ lam } g : X \rightarrow \mathbb{R}$ )<br>$(\text{hbridge} : \text{childAtom} = \{x \mid 0 \leq g\ x\})$<br>$(\text{hlam} : \forall x \in \text{cell}, 0 \leq \text{lam } x)$<br>$(\text{hcomb} : \forall x \in \text{cell}, N\ x + \text{lam } x * g\ x \leq 0)$ :<br>$\forall x \in \text{cell } n \text{ childAtom}, N\ x \leq 0$ |                                                  |
| Qtsl.Structural.fullDomainWeaken_sound |                                                                                                                                                                                                                                                                                                                                                                                                                         | lean/qtsl-aux/QtslAux/Structural.lean, line 88   |

```

theorem fullDomainWeaken_sound (Dm scope : Set X) (P : X → Prop)
  (hbridge : scope ⊆ Dm)
  (hrow : ∀ x ∈ Dm, P x) :
  ∀ x ∈ scope, P x

```

Qtsl.Structural.**fullDomainWeaken\_offScope**

lean/qtsl-aux/QtslAux/Structural.lean, line 94

```

theorem fullDomainWeaken_offScope (Dm Off : Set X) (P : X → Prop)
  (hrow : ∀ x ∈ Dm, P x) :
  ∀ x ∈ Dm \ Off, P x

```

Qtsl.Structural.**regionSat\_mono**

lean/qtsl-aux/QtslAux/Structural.lean, line 62

```

theorem regionSat_mono {R S den : Set X} (h : RegionSat R den) (hsub : S ⊆ R) :
  RegionSat S den

```

Qtsl.Structural.**verifiedCover\_components\_sound**

lean/qtsl-aux/QtslAux/Structural.lean, line 116

```

theorem verifiedCover_components_sound {ι : Type*} (Dm C Off scope : Set X)
  (K : ι → Set X) (P : X → Prop)
  (hbridge : scope = Dm \ Off)
  (hrowC : ∀ x ∈ C, P x)
  (hdecomp : Dm \ C ⊆ ⋃ i, K i)
  (hcomp : ∀ i, K i ⊆ Off) :
  ∀ x ∈ scope, P x

```

Qtsl.Structural.**verifiedCover\_sound**

lean/qtsl-aux/QtslAux/Structural.lean, line 102

```

theorem verifiedCover_sound (Dm C Off scope : Set X) (P : X → Prop)
  (hbridge : scope = Dm \ Off)
  (hrowC : ∀ x ∈ C, P x)
  (hresidual : Dm \ C ⊆ Off) :
  ∀ x ∈ scope, P x

```

Qtsl.Structural.**RegionSat**

lean/qtsl-aux/QtslAux/Structural.lean, line 57

```

def RegionSat (R den : Set X) : Prop

```

Qtsl.Structural.**positiveCombinationAnd\_sound**

lean/qtsl-aux/QtslAux/Structural.lean, line 129

```

theorem positiveCombinationAnd_sound (R1 R2 den1 den2 denAnd : Set X)
  (hbridge : denAnd = den1 ∩ den2)
  (h1 : RegionSat R1 den1) (h2 : RegionSat R2 den2) :
  RegionSat (R1 ∩ R2) denAnd

```

Qtsl.Structural.**positiveCombinationOr\_sound**

lean/qtsl-aux/QtslAux/Structural.lean, line 135

```

theorem positiveCombinationOr_sound (R1 R2 den1 den2 denOr : Set X)
  (hbridge : denOr = den1 ∪ den2)
  (h1 : RegionSat R1 den1) (h2 : RegionSat R2 den2) :
  RegionSat (R1 ∪ R2) denOr

```

Qtsl.Structural.**positiveCombinationInter\_sound**

lean/qtsl-aux/QtslAux/Structural.lean, line 141

```

theorem positiveCombinationInter_sound {ι : Type*} (R den : ι → Set X)
  (denAll : Set X)
  (hbridge : denAll = ⋂ i, den i)
  (h : ∀ i, RegionSat (R i) (den i)) :
  RegionSat (⋂ i, R i) denAll

```

Qtsl.Structural.**positiveCombinationUnion\_sound**

lean/qtsl-aux/QtslAux/Structural.lean, line 148

```

theorem positiveCombinationUnion_sound {ι : Type*} (R den : ι → Set X)
  (denAll : Set X)
  (hbridge : denAll = ⋃ i, den i)
  (h : ∀ i, RegionSat (R i) (den i)) :
  RegionSat (⋃ i, R i) denAll

```

Qtsl.Structural.compl\_over\_approx

lean/qtsl-aux/QtslAux/Structural.lean, line 201

```

theorem compl_over_approx {R den : Set X} (h : RegionSat R den) :
  denc ⊆ Rc

```

Qtsl.Structural.closureExactLt\_sound

lean/qtsl-aux/QtslAux/Structural.lean, line 157

```

theorem closureExactLt_sound (p : X → ℝ) (c : ℝ) (strictAtom : Set X)
  (hbridge : strictAtom = {x | p x < c}) :
  strictAtomc = {x | c ≤ p x}

```

Qtsl.Structural.closureExact\_isClosed\_ge

lean/qtsl-aux/QtslAux/Structural.lean, line 171

```

theorem closureExact_isClosed_ge [TopologicalSpace X] {p : X → ℝ}
  (hp : Continuous p) (c : ℝ) : IsClosed {x | c ≤ p x}

```

Qtsl.Structural.closureExactGt\_sound

lean/qtsl-aux/QtslAux/Structural.lean, line 164

```

theorem closureExactGt_sound (p : X → ℝ) (c : ℝ) (strictAtom : Set X)
  (hbridge : strictAtom = {x | c < p x}) :
  strictAtomc = {x | p x ≤ c}

```

Qtsl.Structural.closureExact\_isClosed\_le

lean/qtsl-aux/QtslAux/Structural.lean, line 175

```

theorem closureExact_isClosed_le [TopologicalSpace X] {p : X → ℝ}
  (hp : Continuous p) (c : ℝ) : IsClosed {x | p x ≤ c}

```

Qtsl.Structural.closureOutward\_sound

lean/qtsl-aux/QtslAux/Structural.lean, line 182

```

theorem closureOutward_sound (p : X → ℝ) (c : ℝ) (cmplAtom : Set X)
  (hbridge : cmplAtom = {x | c ≤ p x}) :
  {x | p x ≤ c}c ⊆ cmplAtom

```

Qtsl.NNF.NForm.elimNeg

lean/qtsl-aux/QtslAux/NNFDuality.lean, line 181

```

def NForm.elimNeg (d : α → α) : NForm α → NForm α
| .pos a => .pos a
| .negLit a => .pos (d a)
| .and φ ψ => .and (φ.elimNeg d) (ψ.elimNeg d)
| .or φ ψ => .or (φ.elimNeg d) (ψ.elimNeg d)

theorem negLits_elimNeg (d : α → α) : ∀ φ : NForm α, (φ.elimNeg d).negLits = ∅

```

Qtsl.Structural.closureOutwardGe\_sound

lean/qtsl-aux/QtslAux/Structural.lean, line 190

```

theorem closureOutwardGe_sound (p : X → ℝ) (c : ℝ) (cmplAtom : Set X)
  (hbridge : cmplAtom = {x | p x ≤ c}) :
  {x | c ≤ p x}c ⊆ cmplAtom

```

Qtsl.Structural.closureInwardEpsLt\_sound

lean/qtsl-aux/QtslAux/Structural.lean, line 209

```

theorem closureInwardEpsLt_sound (p : X → ℝ) (c ε : ℝ) (hε : 0 < ε)
  (roundedAtom : Set X)
  (hbridge : roundedAtom = {x | p x ≤ c - ε}) :
  roundedAtom ⊆ {x | p x < c}

```

Qtsl.Structural.closureInwardEps\_isClosed

lean/qtsl-aux/QtslAux/Structural.lean, line 229

```
theorem closureInwardEps_isClosed [TopologicalSpace X] {p : X → ℝ}
  (hp : Continuous p) (c ε : ℝ) : IsClosed {x | p x ≤ c - ε}
```

Qtsl.Structural.closureInwardEpsGt\_sound

lean/qtsl-aux/QtslAux/Structural.lean, line 219

```
theorem closureInwardEpsGt_sound (p : X → ℝ) (c ε : ℝ) (hε : 0 < ε)
  (roundedAtom : Set X)
  (hbridge : roundedAtom = {x | c + ε ≤ p x}) :
  roundedAtom ⊆ {x | c < p x}
```

Qtsl.Aliases.taylorLeaf\_le\_exp

lean/qtsl-aux/QtslAux/QtslAliases.lean, line 66

```
theorem taylorLeaf_le_exp {q x : ℝ} {n : ℕ} (hx : 0 ≤ x)
  (hq : q ≤ ∑ k ∈ range (n + 1), x ^ k / (Nat.factorial k : ℝ)) :
  q ≤ Real.exp x
```

Qtsl.Aliases.taylorLeaf\_rat\_le\_exp

lean/qtsl-aux/QtslAux/QtslAliases.lean, line 71

```
theorem taylorLeaf_rat_le_exp {qq θ : ℚ} {T n : ℕ} (hθ : 0 ≤ θ)
  (hleaf : qq ≤ ∑ k ∈ range (n + 1), (θ * T) ^ k / (Nat.factorial k : ℚ)) :
  (qq : ℝ) ≤ Real.exp (θ * T)
```

PPLC.taylorElab\_le\_exp

lean/qtsl/Qtsl/Elaborators.lean, line 224

```
theorem taylorElab_le_exp {q θ : Rat} {T n : Nat} (hθ : 0 ≤ θ)
  (hEval : (taylorFormula q θ T n).evalClosed = some true) :
  (q : ℝ) ≤ Real.exp (θ * T)
```

PPLC.taylorElab

lean/qtsl/Qtsl/Elaborators.lean, line 199

```
def taylorElab (H : Hypernet) (seq : Sequent) (q θ : Rat) (T n : Nat)
  (hEval : (taylorFormula q θ T n).evalClosed = some true)
  (hpost : seq.post = taylorFormula q θ T n) :
  PConstExactData H seq
```

PPLC.taylorPartialSum

lean/qtsl/Qtsl/Elaborators.lean, line 180

```
def taylorPartialSum (θ : Rat) (T n : Nat) : Rat
```

PPLC.taylorElab\_sound

lean/qtsl/Qtsl/Elaborators.lean, line 208

```
theorem taylorElab_sound {H : Hypernet} {seq : Sequent} {q θ : Rat} {T n : Nat}
  (hEval : (taylorFormula q θ T n).evalClosed = some true)
  (hpost : seq.post = taylorFormula q θ T n) :
  seq.semanticallyValid H
```

PPLC.taylorFormula\_true\_iff

lean/qtsl/Qtsl/Elaborators.lean, line 190

```
theorem taylorFormula_true_iff (q θ : Rat) (T n : Nat) :
  (taylorFormula q θ T n).evalClosed = some true
  ↔ q ≤ taylorPartialSum θ T n
```

Qtsl.Aliases.rootEval\_eval\_cast

lean/qtsl-aux/QtslAux/QtslAliases.lean, line 39

```
theorem rootEval_eval_cast {σ : Type*} (P : MvPolynomial σ ℚ) (xq : σ → ℚ) :
  ((MvPolynomial.eval xq P : ℚ) : ℝ)
  = MvPolynomial.eval (fun i => (xq i : ℝ)) (MvPolynomial.map (Rat.castHom ℝ) P)
```

Qtsl.Aliases.rootEval\_sublevel\_iff

lean/qtsl-aux/QtslAux/QtslAliases.lean, line 45

```
theorem rootEval_sublevel_iff {σ : Type*} (P : MvPolynomial σ ℚ) (xq : σ → ℚ)
  (thr : ℚ) :
  MvPolynomial.eval xq P ≤ thr
  ↔ MvPolynomial.eval (fun i => (xq i : ℝ))
    (MvPolynomial.map (Rat.castHom ℝ) P) ≤ (thr : ℝ)
```



Qtsl.Aliases.**rootEval\_box\_iff**

lean/qtsl-aux/QtslAux/QtslAliases.lean, line 53

```
theorem rootEval_box_iff {ι : Type*} (aq bq xq : ι → ℚ) :
  (∀ i, aq i ≤ xq i ∧ xq i ≤ bq i)
  ↔ ∀ i, ((aq i : ℝ) ≤ (xq i : ℝ) ∧ (xq i : ℝ) ≤ (bq i : ℝ))
```

PPLC.**polyClosedTerm**

lean/qtsl/Qtsl/Elaborators.lean, line 100

```
def polyClosedTerm (x0 : Nat → Rat) (p : Poly) : Term
```

PPLC.**rootEvalElab**

lean/qtsl/Qtsl/Elaborators.lean, line 147

```
def rootEvalElab (H : Hypernet) (seq : Sequent) (x0 : Nat → Rat) (V : Poly)
  (thr : Rat)
  (hEval : (rootEvalFormula x0 V thr).evalClosed = some true)
  (hpost : seq.post = rootEvalFormula x0 V thr) :
  PConstExactData H seq
```

PPLC.**polyClosedTerm\_evalQ**

lean/qtsl/Qtsl/Elaborators.lean, line 104

```
@[simp] theorem polyClosedTerm_evalQ (x0 : Nat → Rat) :
  ∀ p : Poly, (polyClosedTerm x0 p).evalQ? = some (Poly.eval x0 p)
| [] => by simp [polyClosedTerm, Poly.eval, Term.evalQ?]
| mc :: rest => by
  have ih
```

PPLC.**rootEvalElab\_sound**

lean/qtsl/Qtsl/Elaborators.lean, line 158

```
theorem rootEvalElab_sound {H : Hypernet} {seq : Sequent} {x0 : Nat → Rat}
  {V : Poly} {thr : Rat}
  (hEval : (rootEvalFormula x0 V thr).evalClosed = some true)
  (hpost : seq.post = rootEvalFormula x0 V thr) :
  seq.semanticallyValid H
```

SuFunctor.Universe.**ratBox**

lean/su-functor/SuFunctor/Universe/Generation.lean, line 67

```
def ratBox (l u : V → ℚ) : Set (ProbState V)
```

Qtsl.GridCx.**Box**

lean/qtsl-aux/QtslAux/GridCx.lean, line 49

```
structure Box (d : ℕ) where ...
```

Qtsl.Repar.AffineWitness.**t0\_soundness**

lean/qtsl-aux/QtslAux/Repar.lean, line 196

```
theorem t0_soundness (N Nhat : MvPolynomial (Fin n) ℚ)
  (X D : Set (Fin n → ℝ))
  (hreplay : Nhat = φ.subst N)
  (hcorr : φ.invMapA '' X ⊆ D)
  (hcells : ∀ w ∈ D, aeval w Nhat ≤ 0) :
  ∀ z ∈ X, aeval z N ≤ 0
```

Qtsl.Repar.AffineWitness.**t05\_soundness\_gridcx**

lean/qtsl-aux/QtslAux/Repar.lean, line 247

```
theorem t05_soundness_gridcx {n : ℕ} {ι : Type*} (φs : ι → AffineWitness n)
  (N : MvPolynomial (Fin n) ℚ) (Nhat : ι → MvPolynomial (Fin n) ℚ)
  (B : ι → GridCx.Box n) (T : ∀ i, GridCx.PatchTree n (B i))
  (X : Set (Fin n → ℝ)) (Xp : ι → Set (Fin n → ℝ))
  (hX : X ⊆ ⋃ i, Xp i)
  (hreplay : ∀ i, Nhat i = (φs i).subst N)
  (hcorr : ∀ i, (φs i).invMapA '' Xp i ⊆ (B i).set)
  (hcells : ∀ i (c : (T i).CellIdx), ∀ w ∈ (T i).cell c,
    aeval w (Nhat i) ≤ 0) :
  ∀ z ∈ X, aeval z N ≤ 0
```

Qtsl.Repar.AffineWitness.**left\_certified**

lean/qtsl-aux/QtslAux/Repar.lean, line 77

```
| theorem left_certified :  $\phi.A_{\text{inv}} * \phi.A = 1$ 
```

Qtsl.BoxBern.**boxBern\_enclosure**

lean/qtsl-aux/QtslAux/BoxBern.lean, line 170

```
| theorem boxBern_enclosure (lo hi : Fin n →  $\mathbb{Q}$ ) (N : MvPolynomial (Fin n)  $\mathbb{Q}$ )
  (d : Fin n →  $\mathbb{N}$ ) (b : ( $\forall v$ , Fin (d v + 1)) →  $\mathbb{Q}$ ) (m M :  $\mathbb{Q}$ )
  (hexp : boxShift lo hi N =  $\sum \alpha$ , C (b  $\alpha$ ) * tensorBern d  $\alpha$ )
  (hm :  $\forall \alpha$ , m  $\leq$  b  $\alpha$ ) (hM :  $\forall \alpha$ , b  $\alpha \leq$  M) :
   $\forall x \in (\text{ratBox lo hi}).\text{set}$ , (m :  $\mathbb{R}$ )  $\leq$  aeval x N  $\wedge$  aeval x N  $\leq$  (M :  $\mathbb{R}$ )
```

Qtsl.BoxBern.**boxShift**

lean/qtsl-aux/QtslAux/BoxBern.lean, line 122

```
| noncomputable def boxShift (lo hi : Fin n →  $\mathbb{Q}$ ) :
  MvPolynomial (Fin n)  $\mathbb{Q} \rightarrow_a[\mathbb{Q}]$  MvPolynomial (Fin n)  $\mathbb{Q}$ 
```

Qtsl.BoxBern.**aeval\_boxShift**

lean/qtsl-aux/QtslAux/BoxBern.lean, line 126

```
| theorem aeval_boxShift (lo hi : Fin n →  $\mathbb{Q}$ ) (u : Fin n →  $\mathbb{R}$ )
  (N : MvPolynomial (Fin n)  $\mathbb{Q}$ ) :
  aeval u (boxShift lo hi N)
  = aeval (fun v => (lo v :  $\mathbb{R}$ ) + ((hi v :  $\mathbb{R}$ ) - (lo v :  $\mathbb{R}$ )) * u v) N
```

Qtsl.BoxBern.**boxBern\_nonpos**

lean/qtsl-aux/QtslAux/BoxBern.lean, line 195

```
| theorem boxBern_nonpos (lo hi : Fin n →  $\mathbb{Q}$ ) (N : MvPolynomial (Fin n)  $\mathbb{Q}$ )
  (d : Fin n →  $\mathbb{N}$ ) (b : ( $\forall v$ , Fin (d v + 1)) →  $\mathbb{Q}$ )
  (hexp : boxShift lo hi N =  $\sum \alpha$ , C (b  $\alpha$ ) * tensorBern d  $\alpha$ )
  (hcoeff :  $\forall \alpha$ , b  $\alpha \leq 0$ ) :
   $\forall x \in (\text{ratBox lo hi}).\text{set}$ , aeval x N  $\leq 0$ 
```

Qtsl.BoxBern.**boxBern\_enclosure\_minmax**

lean/qtsl-aux/QtslAux/BoxBern.lean, line 184

```
| theorem boxBern_enclosure_minmax (lo hi : Fin n →  $\mathbb{Q}$ )
  (N : MvPolynomial (Fin n)  $\mathbb{Q}$ )
  (d : Fin n →  $\mathbb{N}$ ) (b : ( $\forall v$ , Fin (d v + 1)) →  $\mathbb{Q}$ )
  (hexp : boxShift lo hi N =  $\sum \alpha$ , C (b  $\alpha$ ) * tensorBern d  $\alpha$ ) :
   $\forall x \in (\text{ratBox lo hi}).\text{set}$ ,
    ((Finset.univ.inf' Finset.univ_nonempty b :  $\mathbb{Q}$ ) :  $\mathbb{R}$ )  $\leq$  aeval x N  $\wedge$ 
    aeval x N  $\leq$  ((Finset.univ.sup' Finset.univ_nonempty b :  $\mathbb{Q}$ ) :  $\mathbb{R}$ )
```

Qtsl.BoxBern.**sum\_aeval\_tensorBern**

lean/qtsl-aux/QtslAux/BoxBern.lean, line 79

```
| theorem sum_aeval_tensorBern (d : Fin n →  $\mathbb{N}$ ) (u : Fin n →  $\mathbb{R}$ ) :
   $\sum \alpha : \forall v$ , Fin (d v + 1), aeval u (tensorBern d  $\alpha$ ) = 1
```

Qtsl.Enclosure.**encCheck**

lean/qtsl-aux/QtslAux/Enclosure.lean, line 93

```
| def encCheck (w : EncWitness) : Bool
```

Qtsl.Enclosure.**EncWitness**

lean/qtsl-aux/QtslAux/Enclosure.lean, line 82

```
| structure EncWitness where ...
```

Qtsl.Enclosure.**encCheck\_sound**

lean/qtsl-aux/QtslAux/Enclosure.lean, line 380

```
| theorem encCheck_sound (w : EncWitness) (h : encCheck w = true) :
   $\forall x : \mathbb{R}$ ,  $|x| \leq (w.r : \mathbb{R}) \rightarrow$ 
   $|w.\text{tag}.\text{eval } x - \text{evalPoly } w.\text{coeffs } x| \leq (w.R : \mathbb{R})$ 
```

Qtsl.Enclosure.**encCheck\_sound\_bracket**

lean/qtsl-aux/QtslAux/Enclosure.lean, line 426

```
| theorem encCheck_sound_bracket (w : EncWitness) (h : encCheck w = true) :
   $\forall x : \mathbb{R}$ ,  $|x| \leq (w.r : \mathbb{R}) \rightarrow$ 
   $\text{evalPoly } w.\text{coeffs } x - (w.R : \mathbb{R}) \leq w.\text{tag}.\text{eval } x \wedge$ 
   $w.\text{tag}.\text{eval } x \leq \text{evalPoly } w.\text{coeffs } x + (w.R : \mathbb{R})$ 
```

Qtsl.Enclosure.**encCheck\_point**

lean/qtsl-aux/QtslAux/Enclosure.lean, line 437

```
theorem encCheck_point (w : EncWitness) (h : encCheck w = true) (c : ℚ)
  (hc : |c| ≤ w.r) :
  evalPoly w.coeffs (c : ℝ) - (w.R : ℝ) ≤ w.tag.eval (c : ℝ) ∧
  w.tag.eval (c : ℝ) ≤ evalPoly w.coeffs (c : ℝ) + (w.R : ℝ)
```

Qtsl.Enclosure.**linear\_interp\_le**

lean/qtsl-aux/QtslAux/Enclosure.lean, line 445

```
theorem linear_interp_le {A C l h w : ℝ} (hlw : l ≤ w) (hwh : w ≤ h)
  (hl : A + C * l ≤ 0) (hh : A + C * h ≤ 0) : A + C * w ≤ 0
```

Qtsl.Enclosure.**integral\_exp\_neg\_sq**

lean/qtsl-aux/QtslAux/Enclosure.lean, line 459

```
theorem integral_exp_neg_sq (z : ℝ) :
  ∫ t in (0:ℝ)..z, Real.exp (-(t ^ 2)) = z * FnTag.erfT.eval (z ^ 2)
```

PPLC.Qtsl.Behavior.**qMorseDischarge\_sound**

lean/qtsl/Qtsl/Behavioral.lean, line 369

```
theorem qMorseDischarge_sound (B : Behavior X) (D : Set X)
  (p : Vtx → Set X) (cells : ι → Set X) (hmap : ι → Option Vtx)
  (hContain : ∀ i v, hmap i = some v → cells i ⊆ p v)
  (hCompact : ∀ v, IsCompact (B.MaxInv (fiberUnion cells hmap v)))
  (hDisj : ∀ v w, v ≠ w →
    Disjoint (B.MaxInv (fiberUnion cells hmap v))
      (B.MaxInv (fiberUnion cells hmap w)))
  (hCell : ∀ x ∈ D, B.ChainRecurrentIn D x → ∃ i, x ∈ B.MaxInv (cells i))
  (hTransit : ∀ i, hmap i = none → B.HasTransitCert (cells i))
  (hOrder : ∀ γ ∈ B.cpl, Set.range γ ⊆ D →
    (∀ v, ¬ Set.range γ ⊆ B.MaxInv (fiberUnion cells hmap v)) →
    ∃ v w, w < v ∧ alphaSet γ ⊆ B.MaxInv (fiberUnion cells hmap v) ∧
    omegaSet γ ⊆ B.MaxInv (fiberUnion cells hmap w)) :
  B.MorseSat D p
```

PPLC.Qtsl.Behavior.**MorseWitness**

lean/qtsl/Qtsl/Behavioral.lean, line 99

```
structure MorseWitness (B : Behavior X) (D : Set X) [PartialOrder Vtx]
  (p : Vtx → Set X) where ...
```

PPLC.Qtsl.Behavior.**HasTransitCert**

lean/qtsl/Qtsl/Behavioral.lean, line 124

```
def HasTransitCert (B : Behavior X) (C : Set X) : Prop
```

PPLC.Qtsl.Semiflow.**FlowDeriv0n**

lean/qtsl/Qtsl/MeanValue.lean, line 24

```
def FlowDeriv0n (φ : Semiflow X) (V W : X → ℝ) (C : Set X) : Prop
```

PPLC.Qtsl.Semiflow.**maxInv\_isCompact**

lean/qtsl/Qtsl/Discharge.lean, line 231

```
theorem maxInv_isCompact [T2Space X] {φ : Semiflow X} {C : Set X}
  (hflow : ∀ t, Continuous (φ.flow t)) (hC : IsCompact C) :
  IsCompact (φ.MaxInv C)
```

PPLC.Qtsl.Semiflow.**fiberUnion\_isCompact**

lean/qtsl/Qtsl/Discharge.lean, line 261

```
theorem fiberUnion_isCompact {Vtx ι : Type*} [Finite ι]
  {cells : ι → Set X} {hmap : ι → Option Vtx}
  (hcells : ∀ i, IsCompact (cells i)) (v : Vtx) :
  IsCompact (fiberUnion cells hmap v)
```

PPLC.Qtsl.Semiflow.HasTransitCert.**disjoint\_maxInv**

lean/qtsl/Qtsl/Discharge.lean, line 81

```
theorem HasTransitCert.disjoint_maxInv {φ : Semiflow X} {C1 C2 : Set X}
  (h : φ.HasTransitCert (C1 ∩ C2)) :
  Disjoint (φ.MaxInv C1) (φ.MaxInv C2)
```

PPLC.Qtsl.Behavior.crCover\_assemble

lean/qtsl/Qtsl/Behavioral.lean, line 352

```

theorem crCover_assemble {B : Behavior X} {D : Set X}
  {cells :  $\iota \rightarrow \text{Set } X$ } {hmap :  $\iota \rightarrow \text{Option } Vtx$ }
  (hCell :  $\forall x \in D, B.ChainRecurrentIn\ D\ x \rightarrow \exists i, x \in B.MaxInv\ (cells\ i)$ )
  (hTransit :  $\forall i, hmap\ i = none \rightarrow B.HasTransitCert\ (cells\ i)$ ) :
 $\forall x \in D, B.ChainRecurrentIn\ D\ x \rightarrow$ 
 $\exists v, x \in B.MaxInv\ (fiberUnion\ cells\ hmap\ v)$ 

```

PPLC.Qtsl.Behavior.HasTransitCert.maxInv\_empty

lean/qtsl/Qtsl/Behavioral.lean, line 144

```

theorem HasTransitCert.maxInv_empty {B : Behavior X} {C : Set X}
  (h : B.HasTransitCert C) : B.MaxInv C =  $\emptyset$ 

```

PPLC.Qtsl.Semiflow.qMorseDischarge\_sound\_orderAssembled

lean/qtsl/Qtsl/OrderAssembly.lean, line 250

```

theorem qMorseDischarge_sound_orderAssembled ( $\phi$  : Semiflow X) (D : Set X)
  (p :  $Vtx \rightarrow \text{Set } X$ ) (cells :  $\iota \rightarrow \text{Set } X$ ) (hmap :  $\iota \rightarrow \text{Option } Vtx$ )
  (hD : IsCompact D)
  (hflow :  $\forall t, \text{Continuous } (\phi.\text{flow } t)$ )
  (hcells :  $\forall i, \text{IsCompact } (cells\ i)$ )
  (hContain :  $\forall i\ v, hmap\ i = some\ v \rightarrow cells\ i \subseteq p\ v$ )
  (hOverlap :  $\forall v\ w, v \neq w \rightarrow$ 
 $\phi.HasTransitCert$ 
 $(fiberUnion\ cells\ hmap\ v\ n\ fiberUnion\ cells\ hmap\ w)$ )
  (hCell :  $\forall x \in D, ChainRecurrentIn\ \phi\ D\ x \rightarrow \exists i, x \in \phi.MaxInv\ (cells\ i)$ )
  (hTransitCert :  $\forall i, hmap\ i = none \rightarrow \phi.HasTransitCert\ (cells\ i)$ )
  (hConv :  $\forall i\ j\ k\ v, i \leq j \rightarrow j \leq k \rightarrow$ 
 $hmap\ i = some\ v \rightarrow hmap\ k = some\ v \rightarrow hmap\ j = some\ v$ )
  (hMono :  $\forall i\ j\ v\ w, i \leq j \rightarrow$ 
 $hmap\ i = some\ v \rightarrow hmap\ j = some\ w \rightarrow v \leq w$ )
  (hInterval :  $\forall \gamma : \mathbb{R} \rightarrow X, \phi.CompleteOrbit\ \gamma \rightarrow \text{Set.range } \gamma \subseteq D \rightarrow$ 
 $\exists\ b\alpha\ b\omega : \iota, b\omega \leq b\alpha \wedge$ 
 $\alpha\text{Set } \gamma \subseteq cells\ b\alpha \wedge \omega\text{Set } \gamma \subseteq cells\ b\omega \wedge$ 
 $\forall t : \mathbb{R}, \exists i, b\omega \leq i \wedge i \leq b\alpha \wedge \gamma\ t \in cells\ i$ ) :
MorseSat  $\phi\ D\ p$ 

```

PPLC.Qtsl.Semiflow.qMorseDischarge\_sound\_eventualMember

lean/qtsl/Qtsl/EventualMember.lean, line 274

```

theorem qMorseDischarge_sound_eventualMember ( $\phi$  : Semiflow X) (D : Set X)
  (p :  $Vtx \rightarrow \text{Set } X$ ) (cells :  $\iota \rightarrow \text{Set } X$ ) (hmap :  $\iota \rightarrow \text{Option } Vtx$ )
  (blocks :  $\beta \rightarrow \text{Set } X$ ) (emb :  $\iota \rightarrow \beta$ ) (cover :  $\iota \rightarrow \beta$ ) (bD :  $\beta$ )
  (hD : IsCompact D)
  (hflow :  $\forall t, \text{Continuous } (\phi.\text{flow } t)$ )
  (hcells :  $\forall i, \text{IsCompact } (cells\ i)$ )
  (hContain :  $\forall i\ v, hmap\ i = some\ v \rightarrow cells\ i \subseteq p\ v$ )
  (hOverlap :  $\forall v\ w, v \neq w \rightarrow$ 
 $\phi.HasTransitCert$ 
 $(fiberUnion\ cells\ hmap\ v\ n\ fiberUnion\ cells\ hmap\ w)$ )
  (hCell :  $\forall x \in D, ChainRecurrentIn\ \phi\ D\ x \rightarrow \exists i, x \in \phi.MaxInv\ (cells\ i)$ )
  (hTransitCert :  $\forall i, hmap\ i = none \rightarrow \phi.HasTransitCert\ (cells\ i)$ )
  (hConv :  $\forall i\ j\ k\ v, i \leq j \rightarrow j \leq k \rightarrow$ 
 $hmap\ i = some\ v \rightarrow hmap\ k = some\ v \rightarrow hmap\ j = some\ v$ )
  (hMono :  $\forall i\ j\ v\ w, i \leq j \rightarrow$ 
 $hmap\ i = some\ v \rightarrow hmap\ j = some\ w \rightarrow v \leq w$ )
  (hbD :  $D \subseteq blocks\ bD$ )
  (hJoin :  $\forall b : \beta, \forall x \in blocks\ b,$ 
 $\exists i : \iota, x \in blocks\ (emb\ i) \wedge blocks\ (emb\ i) \subseteq blocks\ b$ )
  (hcover :  $\forall i : \iota, blocks\ (cover\ i) \subseteq blocks\ (emb\ i)$ )
  (hcellsSup :  $\forall i : \iota, blocks\ (emb\ i) \setminus blocks\ (cover\ i) \subseteq cells\ i$ )
  (horder :  $\forall i\ j : \iota, blocks\ (emb\ i) \subseteq blocks\ (emb\ j) \rightarrow i \leq j$ )
  (hclosed :  $\forall i : \iota, \text{IsClosed } (blocks\ (emb\ i))$ )
  (hInv :  $\forall b : \beta, \phi.Invariant\ (blocks\ b)$ )

```

```

(hMeet :  $\forall a b : \beta, \exists c : \beta, \text{blocks } c \subseteq \text{blocks } a \cap \text{blocks } b \wedge$ 
 $\forall \gamma : \mathbb{R} \rightarrow X, \phi.\text{CompleteOrbit } \gamma \rightarrow \text{Set.range } \gamma \subseteq D \rightarrow$ 
 $\forall s t : \mathbb{R}, (\forall u, s \leq u \rightarrow u \leq t \rightarrow \gamma u \in \text{blocks } a \cap \text{blocks } b) \rightarrow$ 
 $\forall u, s \leq u \rightarrow u \leq t \rightarrow \gamma u \in \text{blocks } c)$ 
(hAbsorbOmega :  $\forall \gamma : \mathbb{R} \rightarrow X, \phi.\text{CompleteOrbit } \gamma \rightarrow \text{Set.range } \gamma \subseteq D \rightarrow$ 
 $\forall b : \beta, (\text{omegaSet } \gamma \cap \text{blocks } b). \text{Nonempty} \rightarrow$ 
 $\exists T : \mathbb{R}, \forall t, T \leq t \rightarrow \gamma t \in \text{blocks } b)$ 
(hAbsorbAlpha :  $\forall \gamma : \mathbb{R} \rightarrow X, \phi.\text{CompleteOrbit } \gamma \rightarrow \text{Set.range } \gamma \subseteq D \rightarrow$ 
 $\forall b : \beta, (\text{alphaSet } \gamma \cap \text{blocks } b). \text{Nonempty} \rightarrow$ 
 $\exists T : \mathbb{R}, \forall t, t \leq T \rightarrow \gamma t \in \text{blocks } b) :$ 
MorseSat  $\phi D \rho$ 

```

PPLC.Qtsl.Semiflow.qMorseDischarge\_sound\_cellCover

lean/qtsl/Qtsl/Crossing.lean, line 120

```

theorem qMorseDischarge_sound_cellCover ( $\phi : \text{Semiflow } X$ ) ( $D : \text{Set } X$ )
( $\rho : \text{Vtx} \rightarrow \text{Set } X$ ) ( $\text{cells} : \iota \rightarrow \text{Set } X$ ) ( $\text{hmap} : \iota \rightarrow \text{Option Vtx}$ )
(hflow :  $\forall t, \text{Continuous } (\phi.\text{flow } t)$ )
(htraj :  $\forall x, \text{Continuous fun } t : \mathbb{R}_{\geq 0} \Rightarrow \phi.\text{flow } t x$ )
(hcells :  $\forall i, \text{IsCompact } (\text{cells } i)$ )
(hContain :  $\forall i v, \text{hmap } i = \text{some } v \rightarrow \text{cells } i \subseteq \rho v$ )
(hCross :  $\forall v w, v \neq w \rightarrow \forall i j, \text{hmap } i = \text{some } v \rightarrow \text{hmap } j = \text{some } w \rightarrow$ 
 $\phi.\text{HasCrossingCert } (\text{cells } i \cap \text{cells } j)$ )
(hCell :  $\forall x \in D, \text{ChainRecurrentIn } \phi D x \rightarrow \exists i, x \in \phi.\text{MaxInv } (\text{cells } i)$ )
(hTransitCert :  $\forall i, \text{hmap } i = \text{none} \rightarrow \phi.\text{HasTransitCert } (\text{cells } i)$ )
(hOrder :  $\forall \gamma : \mathbb{R} \rightarrow X, \phi.\text{CompleteOrbit } \gamma \rightarrow \text{Set.range } \gamma \subseteq D \rightarrow$ 
 $(\forall v, \neg \text{Set.range } \gamma \subseteq \phi.\text{MaxInv } (\text{fiberUnion cells hmap } v)) \rightarrow$ 
 $\exists v w, w < v \wedge \text{alphaSet } \gamma \subseteq \phi.\text{MaxInv } (\text{fiberUnion cells hmap } v) \wedge$ 
 $\text{omegaSet } \gamma \subseteq \phi.\text{MaxInv } (\text{fiberUnion cells hmap } w)) :$ 
MorseSat  $\phi D \rho$ 

```

PPLC.Qtsl.Behavior.maxInv\_subset

lean/qtsl/Qtsl/Behavioral.lean, line 51

```

theorem maxInv_subset ( $B : \text{Behavior } X$ ) ( $C : \text{Set } X$ ) :  $B.\text{MaxInv } C \subseteq C$ 

```

PPLC.Qtsl.Behavior.maxInv\_isInvariant

lean/qtsl/Qtsl/Behavioral.lean, line 69

```

theorem maxInv_isInvariant ( $B : \text{Behavior } X$ ) ( $C : \text{Set } X$ ) :
 $B.\text{IsInvariantIn } (B.\text{MaxInv } C)$ 

```

PPLC.Qtsl.Semiflow.maxInv\_invariant

lean/qtsl/Qtsl/QtslSoundness.lean, line 63

```

theorem maxInv_invariant ( $\phi : \text{Semiflow } X$ ) ( $C : \text{Set } X$ ) :
 $\phi.\text{Invariant } (\phi.\text{MaxInv } C)$ 

```

PPLC.Qtsl.Semiflow.fiberUnion\_subset

lean/qtsl/Qtsl/QtslSoundness.lean, line 195

```

theorem fiberUnion_subset {cells :  $\iota \rightarrow \text{Set } X$ } {hmap :  $\iota \rightarrow \text{Option Vtx}$ }
{ $\rho : \text{Vtx} \rightarrow \text{Set } X$ } (hContain :  $\forall i v, \text{hmap } i = \text{some } v \rightarrow \text{cells } i \subseteq \rho v$ )
( $v : \text{Vtx}$ ) :  $\text{fiberUnion cells hmap } v \subseteq \rho v$ 

```

PPLC.Qtsl.Semiflow.maxInv\_forward\_confined

lean/qtsl/Qtsl/Discharge.lean, line 38

```

theorem maxInv_forward_confined { $\phi : \text{Semiflow } X$ } {C : Set X} {x : X}
(hx :  $x \in \phi.\text{MaxInv } C$ ) :  $\forall t : \mathbb{R}_{\geq 0}, \phi.\text{flow } t x \in C$ 

```

PPLC.Qtsl.Behavior.HasTransitCert.not\_forward\_confined

lean/qtsl/Qtsl/Behavioral.lean, line 129

```

theorem HasTransitCert.not_forward_confined {B : Behavior X} {C : Set X}
(h :  $B.\text{HasTransitCert } C$ ) { $\gamma : \mathbb{R}_{\geq 0} \rightarrow X$ } (h $\gamma : \gamma \in B.\text{fwd}$ ) :
 $\neg \forall t : \mathbb{R}_{\geq 0}, \gamma t \in C$ 

```

PPLC.Qtsl.Semiflow.transit\_maxInv\_empty

lean/qtsl/Qtsl/QtslSoundness.lean, line 114

```

theorem transit_maxInv_empty {φ : Semiflow X} {C : Set X} {V : X → ℝ}
  {ε : ℝ} (hε : 0 < ε) (hBdd : BddBelow (V '' C))
  (hDec : ∀ x ∈ C, ∀ t : ℝ≥0,
    (∀ s : ℝ≥0, s ≤ t → φ.flow s x ∈ C) → V (φ.flow t x) ≤ V x - ε * t) :
  φ.MaxInv C = ∅

```

PPLC.Qtsl.Behavior.**maxInv\_mono**

lean/qtsl/Qtsl/Behavioral.lean, line 55

```

theorem maxInv_mono {B : Behavior X} {C D : Set X} (h : C ⊆ D) :
  B.MaxInv C ⊆ B.MaxInv D

```

PPLC.Qtsl.Semiflow.**subset\_fiberUnion**

lean/qtsl/Qtsl/Discharge.lean, line 45

```

theorem subset_fiberUnion {Vtx ι : Type*} {cells : ι → Set X}
  {hmap : ι → Option Vtx} {i : ι} {v : Vtx} (hi : hmap i = some v) :
  cells i ⊆ fiberUnion cells hmap v

```

PPLC.Qtsl.Semiflow.**qMorseDischarge\_sound\_modelClass**

lean/qtsl/Qtsl/Discharge.lean, line 297

```

theorem qMorseDischarge_sound_modelClass (φ : Semiflow X) (D : Set X)
  (ρ : Vtx → Set X) (cells : ι → Set X) (hmap : ι → Option Vtx)
  (hflow : ∀ t, Continuous (φ.flow t))
  (hcells : ∀ i, IsCompact (cells i))
  (hContain : ∀ i v, hmap i = some v → cells i ⊆ ρ v)
  (hOverlap : ∀ v w, v ≠ w →
    φ.HasTransitCert
      (fiberUnion cells hmap v n fiberUnion cells hmap w))
  (hCell : ∀ x ∈ D, ChainRecurrentIn φ D x → ∃ i, x ∈ φ.MaxInv (cells i))
  (hTransitCert : ∀ i, hmap i = none → φ.HasTransitCert (cells i))
  (hOrder : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    (∀ v, ¬ Set.range γ ⊆ φ.MaxInv (fiberUnion cells hmap v)) →
    ∃ v w, w < v ∧ alphaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap v) ∧
    omegaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap w)) :
  MorseSat φ D ρ

```

PPLC.Qtsl.Behavior.**MorseSat**

lean/qtsl/Qtsl/Behavioral.lean, line 118

```

def MorseSat (B : Behavior X) (D : Set X) [PartialOrder Vtx]
  (ρ : Vtx → Set X) : Prop

```

PPLC.Qtsl.Behavior.**EpsChainIn**

lean/qtsl/Qtsl/Behavioral.lean, line 80

```

inductive EpsChainIn (B : Behavior X) (D : Set X) (ε : ℝ) (T : ℝ≥0) :
  X → X → Prop
| step {x y : X} {γ : ℝ≥0 → X} {t : ℝ≥0} (hγ : γ ∈ B.fwd) (hγ0 : γ 0 = x)
  (hx : x ∈ D) (hy : y ∈ D) (ht : T ≤ t) (hjump : dist (γ t) y < ε) :
  EpsChainIn B D ε T x y
| trans {x y z : X} (h1 : EpsChainIn B D ε T x y)
  (h2 : EpsChainIn B D ε T y z) : EpsChainIn B D ε T x z

def ChainRecurrentIn (B : Behavior X) (D : Set X) (x : X) : Prop

```

PPLC.Qtsl.Behavior.**ChainRecurrentIn**

lean/qtsl/Qtsl/Behavioral.lean, line 88

```

def ChainRecurrentIn (B : Behavior X) (D : Set X) (x : X) : Prop

```

PPLC.Qtsl.Semiflow.**omegaSet**

lean/qtsl/Qtsl/QtslSoundness.lean, line 142

```

def omegaSet [TopologicalSpace X] (γ : ℝ → X) : Set X

```

PPLC.Qtsl.Semiflow.**alphaSet**

lean/qtsl/Qtsl/QtslSoundness.lean, line 145

```

def alphaSet [TopologicalSpace X] (γ : ℝ → X) : Set X

```

PPLC.Qtsl.Semiflow.**orderClause\_assemble**

lean/qtsl/Qtsl/OrderAssembly.lean, line 183

```

theorem orderClause_assemble (φ : Semiflow X) (D : Set X)
  (hD : IsCompact D) (hflow : ∀ t, Continuous (φ.flow t))
  (cells : ι → Set X) (hmap : ι → Option Vtx)
  (hConv : ∀ i j k v, i ≤ j → j ≤ k →
    hmap i = some v → hmap k = some v → hmap j = some v)
  (hMono : ∀ i j v w, i ≤ j →
    hmap i = some v → hmap j = some w → v ≤ w)
  (hTransit : ∀ i, hmap i = none → φ.HasTransitCert (cells i))
  (hInterval : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    ∃ βα bw : ι, bw ≤ βα ∧
      alphaSet γ ⊆ cells βα ∧ omegaSet γ ⊆ cells bw ∧
      ∀ t : ℝ, ∃ i, bw ≤ i ∧ i ≤ βα ∧ γ t ∈ cells i) :
  ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    (∀ v, ¬ Set.range γ ⊆ φ.MaxInv (fiberUnion cells hmap v)) →
    ∃ v w, w < v ∧
      alphaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap v) ∧
      omegaSet γ ⊆ φ.MaxInv (fiberUnion cells hmap w)

```

PPLC.Qtsl.Semiflow.eventualMember\_interval

lean/qtsl/Qtsl/EventualMember.lean, line 145

```

theorem eventualMember_interval (φ : Semiflow X) (D : Set X)
  (blocks : β → Set X) (emb : ι → β) (cover : ι → β) (bD : β)
  (cells : ι → Set X)
  (hbD : D ⊆ blocks bD)
  (hJoin : ∀ b : β, ∀ x ∈ blocks b,
    ∃ i : ι, x ∈ blocks (emb i) ∧ blocks (emb i) ⊆ blocks b)
  (hcover : ∀ i : ι, blocks (cover i) ⊆ blocks (emb i))
  (hcellsSup : ∀ i : ι, blocks (emb i) \ blocks (cover i) ⊆ cells i)
  (horder : ∀ i j : ι, blocks (emb i) ⊆ blocks (emb j) → i ≤ j)
  (hclosed : ∀ i : ι, IsClosed (blocks (emb i)))
  (hInv : ∀ b : β, φ.Invariant (blocks b))
  (hMeet : ∀ a b : β, ∃ c : β, blocks c ⊆ blocks a ∧ blocks b ∧
    ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
      ∀ s t : ℝ, (∀ u, s ≤ u → u ≤ t → γ u ∈ blocks a ∧ blocks b) →
      ∀ u, s ≤ u → u ≤ t → γ u ∈ blocks c)
  (hAbsorbOmega : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    ∀ b : β, (omegaSet γ n blocks b).Nonempty →
      ∃ T : ℝ, ∀ t, T ≤ t → γ t ∈ blocks b)
  (hAbsorbAlpha : ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    ∀ b : β, (alphaSet γ n blocks b).Nonempty →
      ∃ T : ℝ, ∀ t, t ≤ T → γ t ∈ blocks b) :
  ∀ γ : ℝ → X, φ.CompleteOrbit γ → Set.range γ ⊆ D →
    ∃ βα bw : ι, bw ≤ βα ∧
      alphaSet γ ⊆ cells βα ∧ omegaSet γ ⊆ cells bw ∧
      ∀ t : ℝ, ∃ i, bw ≤ i ∧ i ≤ βα ∧ γ t ∈ cells i

```

PPLC.Qtsl.Semiflow.prod

lean/qtsl/Qtsl/Product.lean, line 21

```

def prod (φ₁ : Semiflow X₁) (φ₂ : Semiflow X₂) : Semiflow (X₁ × X₂) where ...

```

PPLC.Qtsl.Semiflow.MorseWitness.prod

lean/qtsl/Qtsl/Product.lean, line 154

```

def MorseWitness.prod (W₁ : MorseWitness φ₁ D₁ ρ₁)
  (W₂ : MorseWitness φ₂ D₂ ρ₂) :
  MorseWitness (φ₁.prod φ₂) (D₁ ×ᵀ D₂)
  (fun p : Vtx₁ × Vtx₂ => ρ₁ p.1 ×ᵀ ρ₂ p.2) where ...

```

PPLC.Qtsl.Semiflow.qMorseProduct\_sound

lean/qtsl/Qtsl/Product.lean, line 235

```

theorem qMorseProduct_sound
  (h₁ : MorseSat φ₁ D₁ ρ₁) (h₂ : MorseSat φ₂ D₂ ρ₂) :
  MorseSat (φ₁.prod φ₂) (D₁ ×ᵀ D₂)
  (fun p : Vtx₁ × Vtx₂ => ρ₁ p.1 ×ᵀ ρ₂ p.2)

```



PPLC.Qtsl.Semiflow.prod\_flow

lean/qtsl/Qtsl/Product.lean, line 26

```
@[simp] theorem prod_flow (φ1 : Semiflow X1) (φ2 : Semiflow X2)
  (t : ℝ≥0) (p : X1 × X2) :
    (φ1.prod φ2).flow t p = (φ1.flow t p.1, φ2.flow t p.2)
```

PPLC.Qtsl.Semiflow.completeOrbit\_prod\_iff

lean/qtsl/Qtsl/Product.lean, line 39

```
theorem completeOrbit_prod_iff {γ : ℝ → X1 × X2} :
  (φ1.prod φ2).CompleteOrbit γ ↔
    φ1.CompleteOrbit (fun s => (γ s).1) ∧
    φ2.CompleteOrbit (fun s => (γ s).2)
```

PPLC.Qtsl.Semiflow.CompleteOrbit.prodMk

lean/qtsl/Qtsl/Product.lean, line 32

```
theorem CompleteOrbit.prodMk {γ1 : ℝ → X1} {γ2 : ℝ → X2}
  (h1 : φ1.CompleteOrbit γ1) (h2 : φ2.CompleteOrbit γ2) :
  (φ1.prod φ2).CompleteOrbit fun s => (γ1 s, γ2 s)
```

PPLC.Qtsl.Semiflow.omegaSet\_prod\_subset

lean/qtsl/Qtsl/Product.lean, line 55

```
theorem omegaSet_prod_subset (γ : ℝ → X1 × X2) :
  omegaSet γ ⊆
    (omegaSet fun s => (γ s).1) ×s (omegaSet fun s => (γ s).2)
```

PPLC.Qtsl.Semiflow.alphaSet\_prod\_subset

lean/qtsl/Qtsl/Product.lean, line 72

```
theorem alphaSet_prod_subset (γ : ℝ → X1 × X2) :
  alphaSet γ ⊆
    (alphaSet fun s => (γ s).1) ×s (alphaSet fun s => (γ s).2)
```

PPLC.Qtsl.Semiflow.EpsChainIn.fst

lean/qtsl/Qtsl/Product.lean, line 111

```
theorem EpsChainIn.fst {D1 : Set X1} {D2 : Set X2} {ε : ℝ} {T : ℝ≥0}
  {x y : X1 × X2}
  (h : EpsChainIn (φ1.prod φ2) (D1 ×s D2) ε T x y) :
  EpsChainIn φ1 D1 ε T x.1 y.1
```

PPLC.Qtsl.Semiflow.ChainRecurrentIn.fst

lean/qtsl/Qtsl/Product.lean, line 133

```
theorem ChainRecurrentIn.fst {D1 : Set X1} {D2 : Set X2} {x : X1 × X2}
  (h : ChainRecurrentIn (φ1.prod φ2) (D1 ×s D2) x) :
  ChainRecurrentIn φ1 D1 x.1
```

PPLC.Qtsl.Semiflow.alphaSet\_subset\_closure\_range

lean/qtsl/Qtsl/Product.lean, line 100

```
theorem alphaSet_subset_closure_range (γ : ℝ → X) :
  alphaSet γ ⊆ closure (Set.range γ)
```

PPLC.Qtsl.Semiflow.omegaSet\_subset\_closure\_range

lean/qtsl/Qtsl/Product.lean, line 95

```
theorem omegaSet_subset_closure_range (γ : ℝ → X) :
  omegaSet γ ⊆ closure (Set.range γ)
```

PPLC.Qtsl.Behavior.qBlockInv\_sound

lean/qtsl/Qtsl/Behavioral.lean, line 323

```
theorem qBlockInv_sound {B : Behavior X} {N X0 : Set X}
  (hInv : B.Invariant N) (hInit : X0 ⊆ N) : B.InvScopeSat N X0
```

PPLC.Qtsl.Behavior.invariant\_of\_boundaryDwell

lean/qtsl/Qtsl/Behavioral.lean, line 213

```
theorem invariant_of_boundaryDwell {B : Behavior X} {N : Set X}
  (hN : IsClosed N)
  (hcont : ∀ γ ∈ B.fwd, Continuous γ)
  (hdwell : ∀ γ ∈ B.fwd, γ 0 ∈ N → γ 0 ∈ frontier N →
    ∃ δ : ℝ≥0, 0 < δ ∧ ∀ s ≤ δ, γ s ∈ N) :
  B.Invariant N
```

PPLC.Qtsl.Behavior.HasFacetRows.boundaryDwell

lean/qtsl/Qtsl/Behavioral.lean, line 283

```

theorem HasFacetRows.boundaryDwell {B : Behavior X} {D N : Set X}
  (h : B.HasFacetRows D N) (hND : N ≤ D) (hD : B.Invariant D)
  (hcont : ∀ γ ∈ B.fwd, Continuous γ) :
  ∀ γ ∈ B.fwd, γ 0 ∈ N → γ 0 ∈ frontier N →
    ∃ δ : ℝ≥0, 0 < δ ∧ ∀ s ≤ δ, γ s ∈ N

```

PPLC.Qtsl.Behavior.invariant\_univ

lean/qtsl/Qtsl/Behavioral.lean, line 320

```

theorem invariant_univ (B : Behavior X) : B.Invariant Set.univ

```

PPLC.Qtsl.Behavior.HasFacetRows

lean/qtsl/Qtsl/Behavioral.lean, line 200

```

def HasFacetRows (B : Behavior X) [TopologicalSpace X] (D N : Set X) : Prop

```

PPLC.Qtsl.Semiflow.qBlockInv\_sound\_of\_rows

lean/qtsl/Qtsl/MeanValue.lean, line 166

```

theorem qBlockInv_sound_of_rows {φ : Semiflow X} {D N X0 : Set X}
  {n : ℕ} {ℓ : Fin n → X → ℝ} {ε : Fin n → ℝ} {F : Fin n → Set X}
  (W : Fin n → X → ℝ)
  (hN : IsClosed N)
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hND : N ≤ D) (hD : φ.Invariant D)
  (hε : ∀ i, 0 < ε i)
  (hℓ : ∀ i, ContinuousOn (ℓ i) (F i))
  (hW : ∀ i, φ.FlowDerivOn (ℓ i) (W i) (F i))
  (hrow : ∀ i, ∀ y ∈ F i, ε i ≤ W i y)
  (hlocal : ∀ x ∈ frontier N, ∃ (A : Set (Fin n)) (U : Set X), U ∈ ℱ x ∧
    (∀ i ∈ A, ℓ i x = 0 ∧ F i ∈ ℱ x) ∧
    ∀ y ∈ U, y ∈ D → (∀ i ∈ A, 0 ≤ ℓ i y) → y ∈ N)
  (hInit : X0 ≤ N) : φ.InvScopeSat N X0

```

PPLC.Qtsl.Grid.qBlockInv\_sound\_of\_grid

lean/qtsl/Qtsl/GridGeometry.lean, line 381

```

theorem qBlockInv_sound_of_grid (φ : Semiflow (Fin d → ℝ)) (B : Finset G.CellIdx)
  {X0 : Set (Fin d → ℝ)}
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hArena : φ.Invariant G.arena)
  (ε : G.FacetData → ℝ) (F : G.FacetData → Set (Fin d → ℝ))
  (hε : ∀ p, G.IsFacet B p → 0 < ε p)
  (hF : ∀ p, G.IsFacet B p → ∀ x ∈ G.facetBox p, F p ∈ ℱ x)
  (hrow : ∀ p, G.IsFacet B p → ∀ x ∈ F p, ∀ t : ℝ≥0,
    (∀ s ≤ t, φ.flow s x ∈ F p) →
    G.obs p x + ε p * (t : ℝ) ≤ G.obs p (φ.flow t x))
  (hInit : X0 ≤ G.blockRegion B) :
  φ.InvScopeSat (G.blockRegion B) X0

```

PPLC.Qtsl.Grid.qBlockInv\_sound\_of\_pointwiseRows

lean/qtsl/Qtsl/GridGeometry.lean, line 397

```

theorem qBlockInv_sound_of_pointwiseRows (φ : Semiflow (Fin d → ℝ))
  (B : Finset G.CellIdx) {X0 : Set (Fin d → ℝ)}
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hArena : φ.Invariant G.arena)
  (ε : G.FacetData → ℝ) (F : G.FacetData → Set (Fin d → ℝ))
  (W : G.FacetData → (Fin d → ℝ) → ℝ)
  (hε : ∀ p, G.IsFacet B p → 0 < ε p)
  (hF : ∀ p, G.IsFacet B p → ∀ x ∈ G.facetBox p, F p ∈ ℱ x)
  (hV : ∀ p, G.IsFacet B p → ContinuousOn (G.obs p) (F p))
  (hW : ∀ p, G.IsFacet B p → φ.FlowDerivOn (G.obs p) (W p) (F p))
  (hrow : ∀ p, G.IsFacet B p → ∀ y ∈ F p, ε p ≤ W p y)
  (hInit : X0 ≤ G.blockRegion B) :
  φ.InvScopeSat (G.blockRegion B) X0

```

PPLC.Qtsl.Semiflow.qBlockInv\_sound\_of\_dwell

lean/qtsl/Qtsl/Discharge.lean, line 168

```

theorem qBlockInv_sound_of_dwell {φ : Semiflow X} {N X0 : Set X}
  (hN : IsClosed N)
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hdwell : ∀ x ∈ N, x ∈ frontier N →
    ∃ δ : ℝ≥0, 0 < δ ∧ ∀ s ≤ δ, φ.flow s x ∈ N)
  (hInit : X0 ⊆ N) : φ.InvScopeSat N X0

```

PPLC.Qtsl.Behavior.qBlockInv\_sound\_of\_facetRows

lean/qtsl/Qtsl/Behavioral.lean, line 310

```

theorem qBlockInv_sound_of_facetRows {B : Behavior X} {D N X0 : Set X}
  (hN : IsClosed N)
  (hcont : ∀ γ ∈ B.fwd, Continuous γ)
  (hND : N ⊆ D) (hD : B.Invariant D) (hrows : B.HasFacetRows D N)
  (hInit : X0 ⊆ N) : B.InvScopeSat N X0

```

PPLC.Qtsl.Semiflow.invariant\_of\_invScopeSat

lean/qtsl/Qtsl/GridGeometry.lean, line 422

```

theorem invariant_of_invScopeSat {φ : Semiflow X} {D X0 : Set X}
  (h : φ.InvScopeSat D X0) : φ.Invariant D

```

PPLC.Qtsl.Semiflow.integrated\_ge\_of\_flowDeriv\_ge

lean/qtsl/Qtsl/MeanValue.lean, line 105

```

theorem integrated_ge_of_flowDeriv_ge {φ : Semiflow X} {C : Set X}
  {V W : X → ℝ} {c : ℝ}
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hV : ContinuousOn V C)
  (hW : φ.FlowDerivOn V W C)
  (hrow : ∀ y ∈ C, c ≤ W y)
  {x : X} {t : ℝ≥0} (hconf : ∀ s ≤ t, φ.flow s x ∈ C) :
  V x + c * t ≤ V (φ.flow t x)

```

PPLC.Qtsl.Grid.wedge\_containment

lean/qtsl/Qtsl/GridGeometry.lean, line 106

```

theorem wedge_containment (B : Finset G.CellIdx) (x : Fin d → ℝ)
  (hx : x ∈ G.blockRegion B) :
  ∃ U ∈ ℱ x, ∀ y ∈ U, y ∈ G.arena →
    (∀ p : G.FacetData, G.IsFacet B p → x ∈ G.facetBox p → 0 ≤ G.obs p y) →
    y ∈ G.blockRegion B

```

PPLC.Qtsl.Grid.qBlockInv\_sound\_of\_grid

lean/qtsl/Qtsl/GridGeometry.lean, line 381

```

theorem qBlockInv_sound_of_grid (φ : Semiflow (Fin d → ℝ)) (B : Finset G.CellIdx)
  {X0 : Set (Fin d → ℝ)}
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hArena : φ.Invariant G.arena)
  (ε : G.FacetData → ℝ) (F : G.FacetData → Set (Fin d → ℝ))
  (hε : ∀ p, G.IsFacet B p → 0 < ε p)
  (hF : ∀ p, G.IsFacet B p → ∀ x ∈ G.facetBox p, F p ∈ ℱ x)
  (hrow : ∀ p, G.IsFacet B p → ∀ x ∈ F p, ∀ t : ℝ≥0,
    (∀ s ≤ t, φ.flow s x ∈ F p) →
    G.obs p x + ε p * (t : ℝ) ≤ G.obs p (φ.flow t x))
  (hInit : X0 ⊆ G.blockRegion B) :
  φ.InvScopeSat (G.blockRegion B) X0

```

PPLC.Qtsl.Grid.qBlockInv\_sound\_of\_pointwiseRows

lean/qtsl/Qtsl/GridGeometry.lean, line 397

```

theorem qBlockInv_sound_of_pointwiseRows (φ : Semiflow (Fin d → ℝ))
  (B : Finset G.CellIdx) {X0 : Set (Fin d → ℝ)}
  (htraj : ∀ x, Continuous fun t : ℝ≥0 => φ.flow t x)
  (hArena : φ.Invariant G.arena)
  (ε : G.FacetData → ℝ) (F : G.FacetData → Set (Fin d → ℝ))
  (W : G.FacetData → (Fin d → ℝ) → ℝ)

```

```

(hε : ∀ p, G.IsFacet B p → 0 < ε p)
(hF : ∀ p, G.IsFacet B p → ∀ x ∈ G.facetBox p, F p ∈  $\mathcal{N}$  x)
(hV : ∀ p, G.IsFacet B p → ContinuousOn (G.obs p) (F p))
(hW : ∀ p, G.IsFacet B p →  $\phi$ .FlowDerivOn (G.obs p) (W p) (F p))
(hrow : ∀ p, G.IsFacet B p → ∀ y ∈ F p, ε p ≤ W p y)
(hInit : X0 ⊆ G.blockRegion B) :
 $\phi$ .InvScopeSat (G.blockRegion B) X0

```

PPLC.Qtsl.qLatticeInv\_exact

lean/qtsl/Qtsl/LatticeInv.lean, line 28

```

theorem qLatticeInv_exact
  (hdc : IsLowerSet C)
  (hJoin : ∀ b : β, ∀ x ∈ blocks b,
    ∃ i : ι, x ∈ blocks (emb i) ∧ blocks (emb i) ⊆ blocks b)
  (hcover : ∀ i : ι, blocks (cover i) ⊆ blocks (emb i))
  (hcellsSup : ∀ i : ι, blocks (emb i) \ blocks (cover i) ⊆ cells i)
  (horder : ∀ i j : ι, blocks (emb i) ⊆ blocks (emb j) → i ≤ j)
  (hcellSub : ∀ i : ι, cells i ⊆ blocks (emb i)) :
  (⋃ i ∈ C, cells i) = ⋃ i ∈ C, blocks (emb i)

```

PPLC.Qtsl.Semiflow.exists\_cellMem\_subset

lean/qtsl/Qtsl/EventualMember.lean, line 112

```

theorem exists_cellMem_subset
  {blocks : β → Set X} {emb : ι → β} {cover : ι → β} {cells : ι → Set X}
  (hJoin : ∀ b : β, ∀ x ∈ blocks b,
    ∃ i : ι, x ∈ blocks (emb i) ∧ blocks (emb i) ⊆ blocks b)
  (hcover : ∀ i : ι, blocks (cover i) ⊆ blocks (emb i))
  (hcellsSup : ∀ i : ι, blocks (emb i) \ blocks (cover i) ⊆ cells i)
  {x : X} (i : ι) (hmem : x ∈ blocks (emb i)) :
  ∃ j : ι, blocks (emb j) ⊆ blocks (emb i) ∧
    x ∈ blocks (emb j) ∧ x ∈ cells j

```

PPLC.Qtsl.Semiflow.qLatticeInv\_sound

lean/qtsl/Qtsl/LatticeInv.lean, line 65

```

theorem Semiflow.qLatticeInv_sound { $\phi$  : Semiflow X} {X0 : Set X}
  (hdc : IsLowerSet C)
  (hJoin : ∀ b : β, ∀ x ∈ blocks b,
    ∃ i : ι, x ∈ blocks (emb i) ∧ blocks (emb i) ⊆ blocks b)
  (hcover : ∀ i : ι, blocks (cover i) ⊆ blocks (emb i))
  (hcellsSup : ∀ i : ι, blocks (emb i) \ blocks (cover i) ⊆ cells i)
  (horder : ∀ i j : ι, blocks (emb i) ⊆ blocks (emb j) → i ≤ j)
  (hcellSub : ∀ i : ι, cells i ⊆ blocks (emb i))
  (hMember :  $\phi$ .InvScopeSat (⋃ i ∈ C, blocks (emb i)) X0) :
   $\phi$ .InvScopeSat (⋃ i ∈ C, cells i) X0

```

PPLC.Qtsl.Behavior.qLatticeInv\_sound

lean/qtsl/Qtsl/LatticeInv.lean, line 93

```

theorem Behavior.qLatticeInv_sound {B : Behavior X} {X0 : Set X}
  (hdc : IsLowerSet C)
  (hJoin : ∀ b : β, ∀ x ∈ blocks b,
    ∃ i : ι, x ∈ blocks (emb i) ∧ blocks (emb i) ⊆ blocks b)
  (hcover : ∀ i : ι, blocks (cover i) ⊆ blocks (emb i))
  (hcellsSup : ∀ i : ι, blocks (emb i) \ blocks (cover i) ⊆ cells i)
  (horder : ∀ i j : ι, blocks (emb i) ⊆ blocks (emb j) → i ≤ j)
  (hcellSub : ∀ i : ι, cells i ⊆ blocks (emb i))
  (hMember : B.InvScopeSat (⋃ i ∈ C, blocks (emb i)) X0) :
  B.InvScopeSat (⋃ i ∈ C, cells i) X0

```

PPLC.Qtsl.Behavior.qBasin\_sound

lean/qtsl/Qtsl/Behavioral.lean, line 327

```

theorem qBasin_sound {B : Behavior X} {S X0 Bad : Set X}
  (hscope : B.InvScopeSat S X0) (hAvoid : S ∩ Bad = ∅) :
  B.SafetySat X0 Bad

```

PPLC.Qtsl.member\_subset\_cells\_le

lean/qtsl/Qtsl/LatticeInv.lean, line 44

```

theorem member_subset_cells_le
  (hJoin :  $\forall b : \beta, \forall x \in \text{blocks } b,$ 
     $\exists i : \iota, x \in \text{blocks } (\text{emb } i) \wedge \text{blocks } (\text{emb } i) \subseteq \text{blocks } b$ )
  (hcover :  $\forall i : \iota, \text{blocks } (\text{cover } i) \subset \text{blocks } (\text{emb } i)$ )
  (hcellsSup :  $\forall i : \iota, \text{blocks } (\text{emb } i) \setminus \text{blocks } (\text{cover } i) \subseteq \text{cells } i$ )
  (horder :  $\forall i j : \iota, \text{blocks } (\text{emb } i) \subseteq \text{blocks } (\text{emb } j) \rightarrow i \leq j$ )
  (i :  $\iota$ ) :
   $\text{blocks } (\text{emb } i) \subseteq \bigcup j \in \text{Set.Iic } i, \text{cells } j$ 

```

PPLC.Qtsl.Semiflow.qLatticeInv\_sound

lean/qtsl/Qtsl/LatticeInv.lean, line 65

```

theorem Semiflow.qLatticeInv_sound { $\phi : \text{Semiflow } X$ } { $X0 : \text{Set } X$ }
  (hdc : IsLowerSet C)
  (hJoin :  $\forall b : \beta, \forall x \in \text{blocks } b,$ 
     $\exists i : \iota, x \in \text{blocks } (\text{emb } i) \wedge \text{blocks } (\text{emb } i) \subseteq \text{blocks } b$ )
  (hcover :  $\forall i : \iota, \text{blocks } (\text{cover } i) \subset \text{blocks } (\text{emb } i)$ )
  (hcellsSup :  $\forall i : \iota, \text{blocks } (\text{emb } i) \setminus \text{blocks } (\text{cover } i) \subseteq \text{cells } i$ )
  (horder :  $\forall i j : \iota, \text{blocks } (\text{emb } i) \subseteq \text{blocks } (\text{emb } j) \rightarrow i \leq j$ )
  (hcellSub :  $\forall i : \iota, \text{cells } i \subseteq \text{blocks } (\text{emb } i)$ )
  (hMember :  $\phi.\text{InvScopeSat } (\bigcup i \in C, \text{blocks } (\text{emb } i)) X0$ ) :
   $\phi.\text{InvScopeSat } (\bigcup i \in C, \text{cells } i) X0$ 

```

PPLC.Qtsl.Behavior.qBasin\_batch

lean/qtsl/Qtsl/Behavioral.lean, line 336

```

theorem qBasin_batch {B : Behavior X} {S X0 : Set X} {J : Type*}
  {Bad : J  $\rightarrow$  Set X} (hscope : B.InvScopeSat S X0)
  (hAvoid :  $\forall j, S \cap \text{Bad } j = \emptyset$ ) :
  B.SafetySat X0 ( $\bigcup j, \text{Bad } j$ )

```

PPLC.Qtsl.Behavior.InvScopeSat

lean/qtsl/Qtsl/Behavioral.lean, line 45

```

def InvScopeSat (B : Behavior X) (S X0 : Set X) : Prop

```

PPLC.Qtsl.Behavior.SafetySat

lean/qtsl/Qtsl/Behavioral.lean, line 42

```

def SafetySat (B : Behavior X) (X0 Bad : Set X) : Prop

```

PPLC.Qtsl.Semiflow.InvScopeSat

lean/qtsl/Qtsl/QtslSoundness.lean, line 84

```

def InvScopeSat ( $\phi : \text{Semiflow } X$ ) (S X0 : Set X) : Prop

```

PPLC.Qtsl.Semiflow.SafetySat

lean/qtsl/Qtsl/QtslSoundness.lean, line 87

```

def SafetySat ( $\phi : \text{Semiflow } X$ ) (X0 B : Set X) : Prop

```

Qtsl.QtslCompose.System.qagTier

lean/qtsl-aux/QtslAux/QtslCompose/Presentation.lean, line 111

```

def System.qagTier {n :  $\mathbb{N}$ } (S : System n) : Tier

```

Qtsl.QtslCompose.admissible\_constrcuts

lean/qtsl-aux/QtslAux/QtslCompose/Presentation.lean, line 163

```

theorem admissible_constrcuts { $\iota \kappa \text{Idx } X : \text{Type}^*$ } [Fintype  $\iota$ ] [Fintype  $\kappa \text{Idx}$ ]
  {R : Set X} ( $\text{lam} : \iota \rightarrow \mathbb{R}$ ) (e :  $\iota \rightarrow X \rightarrow \mathbb{R}$ ) (nu :  $\kappa \text{Idx} \rightarrow \mathbb{R}$ ) (g :  $\kappa \text{Idx} \rightarrow X \rightarrow \mathbb{R}$ )
  (hlam :  $\forall i, 0 \leq \text{lam } i$ )
  (he :  $\forall i, \forall x \in R, e \ i \ x \leq 0$ )
  (hg :  $\forall k, \forall x \in R, g \ k \ x = 0$ ) :
   $\forall x \in R, (\sum i, \text{lam } i * e \ i \ x) + (\sum k, \text{nu } k * g \ k \ x) \leq 0$ 

```

Qtsl.AGCompose.Parallel.fail\_bound

lean/qtsl-aux/QtslAux/AGCompose/GshsParallel.lean, line 128

```

theorem fail_bound {ι : Type*} (s : Finset ι) {sat : ι → Set Ω} {satOut : Set Ω}
  {ε : ι → ℝ≥0∞}
  (h : ∀ i ∈ s, μ (sat i)c ≤ ε i)
  (hout : (⋂ i ∈ s, sat i) ⊆ satOut) :
  μ satOutc ≤ ∑ i ∈ s, ε i

```

Qtsl.QtslCompose.fail\_bound\_out

lean/qtsl-aux/QtslAux/QtslCompose/AGCompose.lean, line 131

```

theorem fail_bound_out [MeasurableSpace Ω] {μ : Measure Ω} {ε : Fin n → ℝ≥0∞}
  {satOut : Set Ω}
  (hw : ∀ j, conformExt n (⋂ i ∈ S.upstream j, sat i) ⊆ conform j)
  (hout : conformExt n (⋂ j, sat j) ⊆ satOut)
  (he : ∀ j, μ (conform j n (sat j)c) ≤ ε j) :
  μ (conformExt n satOutc) ≤ ∑ j, ε j

```

Qtsl.QtslCompose.sat\_of\_conform

lean/qtsl-aux/QtslAux/QtslCompose/AGCompose.lean, line 48

```

theorem sat_of_conform
  (hw : ∀ j, conformExt n (⋂ i ∈ S.upstream j, sat i) ⊆ conform j)
  {ω : Ω} (h : ∀ j, ω ∈ conform j → ω ∈ sat j)
  (hext : ω ∈ conformExt) : ∀ j, ω ∈ sat j

```

Qtsl.AGCompose.Series.fail\_subset

lean/qtsl-aux/QtslAux/AGCompose/GshsSeries.lean, line 45

```

theorem fail_subset
  (hw : conform1 n sat1 ⊆ conform2)
  (hout : conform1 n sat1 n sat2 ⊆ satOut) :
  conform1 n satOutc ⊆ (conform1 n sat1c) ∪ (conform2 n sat2c)

```

Qtsl.QtslCompose.System.qagTier\_admissible\_iff

lean/qtsl-aux/QtslAux/QtslCompose/Presentation.lean, line 116

```

theorem System.qagTier_admissible_iff {n : ℕ} (S : System n) :
  S.qagTier = .admissible ↔
  ∀ i, (S.charts i).profile = .ode v (S.charts i).profile = .discrete

```

Qtsl.QtslCompose.System.qagTier\_core\_iff

lean/qtsl-aux/QtslAux/QtslCompose/Presentation.lean, line 135

```

theorem System.qagTier_core_iff {n : ℕ} (S : System n) :
  S.qagTier = .core ↔
  ∃ i, (S.charts i).profile = .sde v (S.charts i).profile = .gshs

```

Qtsl.AGCompose.Series.certified

lean/qtsl-aux/QtslAux/AGCompose/GshsSeries.lean, line 132

```

theorem certified [IsFiniteMeasure μ]
  {Y1 Y2 : ℕ → Ω → ℝ} {X1 : ℕ → Ω → S1} {X2 : ℕ → Ω → S2}
  {B1 : Set S1} {B2 : Set S2} {a1 a2 : ℝ}

  (hY1 : Supermartingale Y1 ℱ μ) (hY10 : 0 ≤ Y1)
  (hY2 : Supermartingale Y2 ℱ μ) (hY20 : 0 ≤ Y2)

  (ha1 : 0 < a1) (ha2 : 0 < a2)
  (hbar1 : ∀ n ω, X1 n ω ∈ B1 → a1 ≤ Y1 n ω)
  (hbar2 : ∀ n ω, X2 n ω ∈ B2 → a2 ≤ Y2 n ω)

  (hwt1 : conform1 n sat1c ⊆ {ω | ∃ n, X1 n ω ∈ B1})
  (hwt2 : conform2 n sat2c ⊆ {ω | ∃ n, X2 n ω ∈ B2})

  (hw : conform1 n sat1 ⊆ conform2)
  (hout : conform1 n sat1 n sat2 ⊆ satOut) :
  μ (conform1 n satOutc) ≤
  ENNReal.ofReal (∫ ω, Y1 0 ω ∂μ) / ENNReal.ofReal a1 +
  ENNReal.ofReal (∫ ω, Y2 0 ω ∂μ) / ENNReal.ofReal a2

```

Qtsl.AGCompose.Parallel.certified

lean/qtsl-aux/QtslAux/AGCompose/GshsParallel.lean, line 58

```

theorem certified (s : Finset ι) [IsFiniteMeasure μ]
  {Y : ι → ℕ → Ω → ℝ} {lam : ι → ℝ} {X : ℕ → Ω → St} {Bad : ι → Set St} {a : ℝ}
  (hY : ∀ i ∈ s, Supermartingale (Y i) ℳ μ)
  (hY0 : ∀ i ∈ s, 0 ≤ Y i)
  (hlam : ∀ i ∈ s, 0 ≤ lam i)
  (hbar : ∀ i ∈ s, ∀ n ω, X n ω ∈ Bad i → a ≤ lam i * Y i n ω) :
  ENNReal.ofReal a * μ {ω | ∃ n, X n ω ∈ ⋃ i ∈ s, Bad i} ≤
  ENNReal.ofReal (∑ i ∈ s, lam i * ∫ ω, Y i 0 ω ∂μ)

```

Qtsl.AGCompose.Parallel.**certified\_pair**

lean/qtsl-aux/QtslAux/AGCompose/GshsParallel.lean, line 87

```

theorem certified_pair [IsFiniteMeasure μ]
  {Y1 Y2 : ℕ → Ω → ℝ} {X : ℕ → Ω → St} {B1 B2 : Set St} {lam a : ℝ}
  (hY1 : Supermartingale Y1 ℳ μ) (hY10 : 0 ≤ Y1)
  (hY2 : Supermartingale Y2 ℳ μ) (hY20 : 0 ≤ Y2) (hlam : 0 ≤ lam)
  (hbar1 : ∀ n ω, X n ω ∈ B1 → a ≤ Y1 n ω)
  (hbar2 : ∀ n ω, X n ω ∈ B2 → a ≤ lam * Y2 n ω) :
  ENNReal.ofReal a * μ {ω | ∃ n, X n ω ∈ B1 ∪ B2} ≤
  ENNReal.ofReal (∫ ω, Y1 0 ω ∂μ + lam * ∫ ω, Y2 0 ω ∂μ)

```

Qtsl.AGCompose.SumFormFeedback.**avoid\_bound**

lean/qtsl-aux/QtslAux/GshsFeedbackSum.lean, line 166

```

theorem avoid_bound {S : Type*} {X : ℕ → Ω → S} {B : Set S} {a : ℝ}
  (hbarrier : ∀ k ω, X k ω ∈ B → a ≤ F.W k ω) :
  ENNReal.ofReal a * μ {ω | ∃ k, X k ω ∈ B} ≤ ENNReal.ofReal (∫ ω, F.W 0 ω ∂μ)

```

Qtsl.AGCompose.**SumFormFeedback**

lean/qtsl-aux/QtslAux/GshsFeedbackSum.lean, line 79

```

structure SumFormFeedback (μ : Measure Ω) (ℳ : Filtration ℕ m0) where ...

```

Qtsl.LayerD.**sound\_GSHS\_safety\_barrier**

lean/qtsl-aux/QtslAux/MaxFormPDMP.lean, line 608

```

theorem sound_GSHS_safety_barrier [IsMarkovKernel M.seg] [IsMarkovKernel M.jump]
  {W : State → ℝ} {U : Set State} {m : ℝ} (hm : 0 < m)
  (hW : Measurable W) (hU : MeasurableSet U)
  (hSegD : SegmentDynkin M 0 W) (hForce : ForcedNonexpansion M W)
  (hNonneg : NonnegV W) {C : ℝ} (hC0 : 0 ≤ C) (hWC : ∀ y, W y ≤ C)
  (hFloor : ∀ y ∈ U, m ≤ W y) (x : State) :
  Battery.pathMeasure (skel M) (SkelState.alive x) {ω | ∃ n, ω n ∈ skelBad U}
  ≤ ENNReal.ofReal (W x / m)

```

Qtsl.QtslCompose.Discrete.**firstFail**

lean/qtsl-aux/QtslAux/QtslCompose/AGComposeDiscrete.lean, line 52

```

def firstFail (bad : Fin m → Set S) (j : Fin m) : Set (ℕ → S)

```

Qtsl.QtslCompose.Discrete.**measure\_firstFail\_le** lean/qtsl-aux/QtslAux/QtslCompose/AGComposeDiscrete.lean, line 86

```

theorem measure_firstFail_le {bad : Fin m → Set S} {V : S → ℝ} {C : ℝ} {j : Fin m}
  (hA : MeasurableSet (⋃ i, bad i)) (hV : Measurable V)
  (hV0 : ∀ y, 0 ≤ V y) (hVC : ∀ y, V y ≤ C)
  (hbar : ∀ y ∈ bad j, 1 ≤ V y)
  (hdrift : ∀ y, y ∉ (⋃ i, bad i) → ∫ z, V z ∂K y ≤ V y) (x : S) :
  pathMeasure K x (firstFail bad j) ≤ ENNReal.ofReal (V x)

```

Qtsl.QtslCompose.Discrete.**fail\_bound\_wired**

lean/qtsl-aux/QtslAux/QtslCompose/AGComposeDiscrete.lean, line 156

```

theorem fail_bound_wired (Sys : System m) {conf bad : Fin m → Set S}
  {V : Fin m → S → ℝ} {C : Fin m → ℝ}
  (hA : MeasurableSet (⋃ i, bad i))
  (hw : ∀ j, (⋂ i ∈ Sys.upstream j, (bad i)c) ≤ conf j)
  (hV : ∀ j, Measurable (V j)) (hV0 : ∀ j y, 0 ≤ V j y)
  (hVC : ∀ j y, V j y ≤ C j)
  (hbar : ∀ j, ∀ y ∈ bad j, 1 ≤ V j y)
  (hdrift : ∀ j, ∀ y ∈ conf j, y ∉ bad j → ∫ z, V j z ∂K y ≤ V j y) (x : S) :

```



```

pathMeasure K x {ω | ∃ n, ω n ∈ ⋃ i, bad i}
≤ ∑ j, ENNReal.ofReal (V j x)

```

MeasureTheory.Supermartingale.ville\_ineq

lean/qtsl-aux/QtslAux/Ville.lean, line 133

```

theorem Supermartingale.ville_ineq [IsFiniteMeasure μ] (hsup : Supermartingale f ℳ
  μ)
  (hnonneg : 0 ≤ f) (ε : ℝ ≥ 0) :
  ε * μ {ω | ∃ k, (ε : ℝ) ≤ f k ω} ≤ ENNReal.ofReal (∫ ω, f 0 ω ∂μ)

```

Qtsl.QtslCompose.Discrete.drift\_parallelComp

lean/qtsl-aux/QtslAux/QtslCompose/AGComposeDiscrete.lean, line 207

```

theorem drift_parallelComp {bad₁ : Set S₁} {bad₂ : Set S₂}
  {V₁ : S₁ → ℝ} {V₂ : S₂ → ℝ} {lam : ℝ} (hlam : 0 ≤ lam)
  (hint₁ : ∀ y, y ∉ bad₁ → Integrable V₁ (K₁ y))
  (hint₂ : ∀ y, y ∉ bad₂ → Integrable V₂ (K₂ y))
  (hd₁ : ∀ y, y ∉ bad₁ → ∫ z, V₁ z ∂K₁ y ≤ V₁ y)
  (hd₂ : ∀ y, y ∉ bad₂ → ∫ z, V₂ z ∂K₂ y ≤ V₂ y) :
  ∀ p : S₁ × S₂, p ∉ (Prod.fst ⁻¹' bad₁ ∪ Prod.snd ⁻¹' bad₂) →
  ∫ q, (V₁ q.1 + lam * V₂ q.2) ∂(K₁ ⋈ K₂) p
  ≤ V₁ p.1 + lam * V₂ p.2

```

Qtsl.QtslCompose.Discrete.avoid\_parallelComp

lean/qtsl-aux/QtslAux/QtslCompose/AGComposeDiscrete.lean, line 233

```

theorem avoid_parallelComp {bad₁ : Set S₁} {bad₂ : Set S₂}
  {V₁ : S₁ → ℝ} {V₂ : S₂ → ℝ} {C₁ C₂ lam : ℝ} (hlam : 0 ≤ lam)
  (hbad₁ : MeasurableSet bad₁) (hbad₂ : MeasurableSet bad₂)
  (hV₁ : Measurable V₁) (hV₂ : Measurable V₂)
  (hV₁0 : ∀ y, 0 ≤ V₁ y) (hV₂0 : ∀ y, 0 ≤ V₂ y)
  (hV₁C : ∀ y, V₁ y ≤ C₁) (hV₂C : ∀ y, V₂ y ≤ C₂)
  (hbar₁ : ∀ y ∈ bad₁, 1 ≤ V₁ y) (hbar₂ : ∀ y ∈ bad₂, 1 ≤ lam * V₂ y)
  (hd₁ : ∀ y, y ∉ bad₁ → ∫ z, V₁ z ∂K₁ y ≤ V₁ y)
  (hd₂ : ∀ y, y ∉ bad₂ → ∫ z, V₂ z ∂K₂ y ≤ V₂ y) (x : S₁ × S₂) :
  pathMeasure (K₁ ⋈ K₂) x
  {ω | ∃ n, ω n ∈ (Prod.fst ⁻¹' bad₁ ∪ Prod.snd ⁻¹' bad₂ : Set (S₁ × S₂))}
  ≤ ENNReal.ofReal (V₁ x.1 + lam * V₂ x.2)

```

Qtsl.QtslCompose.Discrete.smallGain\_drift

lean/qtsl-aux/QtslAux/QtslCompose/AGComposeDiscrete.lean, line 278

```

theorem smallGain_drift (hlam : 0 ≤ lam)
  (hint₁ : ∀ y, y ∉ A → Integrable V₁ (K y))
  (hint₂ : ∀ y, y ∉ A → Integrable V₂ (K y))
  (hd₁ : ∀ y, y ∉ A → ∫ z, V₁ z ∂K y ≤ V₁ y - alpha₁ y + sigma₁ y)
  (hd₂ : ∀ y, y ∉ A → ∫ z, V₂ z ∂K y ≤ V₂ y - alpha₂ y + sigma₂ y)
  (hm₁ : ∀ y, y ∉ A → lam * sigma₂ y ≤ (1 - theta) * alpha₁ y)
  (hm₂ : ∀ y, y ∉ A → sigma₁ y ≤ lam * ((1 - theta) * alpha₂ y)) :
  ∀ y, y ∉ A → ∫ z, (V₁ z + lam * V₂ z) ∂K y
  ≤ (V₁ y + lam * V₂ y) - theta * (alpha₁ y + lam * alpha₂ y)

```

Qtsl.QtslCompose.Discrete.avoid\_smallGain

lean/qtsl-aux/QtslAux/QtslCompose/AGComposeDiscrete.lean, line 310

```

theorem avoid_smallGain (hlam : 0 ≤ lam) (htheta : 0 ≤ theta)
  {C₁ C₂ : ℝ} (hA : MeasurableSet A)
  (hV₁ : Measurable V₁) (hV₂ : Measurable V₂)
  (hV₁0 : ∀ y, 0 ≤ V₁ y) (hV₂0 : ∀ y, 0 ≤ V₂ y)
  (hV₁C : ∀ y, V₁ y ≤ C₁) (hV₂C : ∀ y, V₂ y ≤ C₂)
  (hbar : ∀ y ∈ A, 1 ≤ V₁ y + lam * V₂ y)
  (halpha₁ : ∀ y, y ∉ A → 0 ≤ alpha₁ y)
  (halpha₂ : ∀ y, y ∉ A → 0 ≤ alpha₂ y)
  (hd₁ : ∀ y, y ∉ A → ∫ z, V₁ z ∂K y ≤ V₁ y - alpha₁ y + sigma₁ y)
  (hd₂ : ∀ y, y ∉ A → ∫ z, V₂ z ∂K y ≤ V₂ y - alpha₂ y + sigma₂ y)
  (hm₁ : ∀ y, y ∉ A → lam * sigma₂ y ≤ (1 - theta) * alpha₁ y)
  (hm₂ : ∀ y, y ∉ A → sigma₁ y ≤ lam * ((1 - theta) * alpha₂ y)) (x : S) :

```

```

    pathMeasure K x {ω | ∃ n, ω n ∈ A}
      ≤ ENNReal.ofReal (V1 x + lam * V2 x)

```

Qtsl.AGCompose.Series.**tower\_step** lean/qtsl-aux/QtslAux/AGCompose/GshsSeries.lean, line 81

```

theorem tower_step [IsFiniteMeasure μ] {mG : MeasurableSpace Ω} (hG : mG ≤ m0)
  {S : Set Ω} (hS : MeasurableSet[mG] S) {E : Set Ω} (hE : MeasurableSet[m0] E)
  {p : ℝ}
  (hcond : ∀m ω ∂μ, ω ∈ S → p ≤ (μ[E.indicator (fun _ => (1 : ℝ)) | mG]) ω) :
  ENNReal.ofReal p * μ S ≤ μ (S ∩ E)

```

Qtsl.AGCompose.Parallel.**supermartingale\_weighted\_sum** lean/qtsl-aux/QtslAux/AGCompose/GshsParallel.lean, line 31

```

theorem supermartingale_weighted_sum {ι : Type*} (s : Finset ι) [IsFiniteMeasure μ]
  {Y : ι → ℕ → Ω → ℝ} {lam : ι → ℝ}
  (hY : ∀ i ∈ s, Supermartingale (Y i) ℳ μ) (hlam : ∀ i ∈ s, 0 ≤ lam i) :
  Supermartingale (fun n ω => ∑ i ∈ s, lam i * Y i n ω) ℳ μ

```

Qtsl.AGCompose.SumFormFeedback.**debut\_avoid\_bound** lean/qtsl-aux/QtslAux/GshsFeedbackSum.lean, line 171

```

theorem debut_avoid_bound {S : Type*} {X : ℕ → Ω → S} {B : Set S} {a : ℝ}
  (hbarrier : ∀ k ω, X k ω ∈ B → a ≤ F.W k ω) {E : Set Ω}
  (hdebut : E ⊆ {ω | ∃ k, X k ω ∈ B}) :
  ENNReal.ofReal a * μ E ≤ ENNReal.ofReal (∫ ω, F.W 0 ω ∂μ)

```

Qtsl.QtslCompose.**qualitative** lean/qtsl-aux/QtslAux/QtslCompose/AGCompose.lean, line 69

```

theorem qualitative [MeasurableSpace Ω] {μ : Measure Ω}
  (hw : ∀ j, conformExt n (∩ i ∈ S.upstream j, sat i) ⊆ conform j)
  (h : ∀ j, ∀m ω ∂μ, ω ∈ conform j → ω ∈ sat j)
  {satOut : Set Ω} (hout : conformExt n (∩ j, sat j) ⊆ satOut) :
  ∀m ω ∂μ, ω ∈ conformExt → ω ∈ satOut

```

Qtsl.AGCompose.Series.**qualitative** lean/qtsl-aux/QtslAux/AGCompose/GshsSeries.lean, line 35

```

theorem qualitative
  (h1 : ∀m ω ∂μ, ω ∈ conform1 → ω ∈ sat1)
  (hw : conform1 ∩ sat1 ⊆ conform2)
  (h2 : ∀m ω ∂μ, ω ∈ conform2 → ω ∈ sat2)
  (hout : conform1 ∩ sat1 ∩ sat2 ⊆ satOut) :
  ∀m ω ∂μ, ω ∈ conform1 → ω ∈ satOut

```

Qtsl.AGCompose.Parallel.**supermartingale\_weighted\_sum** lean/qtsl-aux/QtslAux/AGCompose/GshsParallel.lean, line 31

```

theorem supermartingale_weighted_sum {ι : Type*} (s : Finset ι) [IsFiniteMeasure μ]
  {Y : ι → ℕ → Ω → ℝ} {lam : ι → ℝ}
  (hY : ∀ i ∈ s, Supermartingale (Y i) ℳ μ) (hlam : ∀ i ∈ s, 0 ≤ lam i) :
  Supermartingale (fun n ω => ∑ i ∈ s, lam i * Y i n ω) ℳ μ

```

Qtsl.AGCompose.**SkeletonCertificate** lean/qtsl-aux/QtslAux/GshsFeedbackSum.lean, line 39

```

structure SkeletonCertificate (μ : Measure Ω) (ℳ : Filtration ℕ m0) where ...

```

Qtsl.AGCompose.SkeletonCertificate.**step** lean/qtsl-aux/QtslAux/GshsFeedbackSum.lean, line 65

```

theorem step (C : SkeletonCertificate μ ℳ) (k : ℕ) :
  μ[C.val (k + 1) + C.alphaInt k - C.sigmaInt k | ℳ k] ≤m[μ] C.val k

```

Qtsl.AGCompose.SumFormFeedback.**drift** lean/qtsl-aux/QtslAux/GshsFeedbackSum.lean, line 124

```

theorem drift (k : ℕ) : μ[F.W (k + 1) + F.margin k | ℳ k] ≤m[μ] F.W k

```

Qtsl.AGCompose.SumFormFeedback.**compensated\_supermartingale** lean/qtsl-aux/QtslAux/GshsFeedbackSum.lean, line 196

```

theorem compensated_supermartingale
  (hα₁ : ∀ k, StronglyMeasurable[ℳ (k + 1)] (F.cert₁.alphaInt k))
  (hα₂ : ∀ k, StronglyMeasurable[ℳ (k + 1)] (F.cert₂.alphaInt k)) :
  Supermartingale F.compensated ℳ μ

```

Qtsl.AGCompose.SumFormFeedback.**margin\_budget**

lean/qtsl-aux/QtslAux/GshsFeedbackSum.lean, line 226

```

theorem margin_budget
  (hα₁ : ∀ k, StronglyMeasurable[ℳ (k + 1)] (F.cert₁.alphaInt k))
  (hα₂ : ∀ k, StronglyMeasurable[ℳ (k + 1)] (F.cert₂.alphaInt k)) (n : ℕ) :
  ∫ ω, (∑ j ∈ Finset.range n, F.margin j) ω ∂μ ≤ ∫ ω, F.W 0 ω ∂μ

```

Qtsl.QtslCompose.ChartShape.**profile**

lean/qtsl-aux/QtslAux/QtslCompose/Presentation.lean, line 57

```

def ChartShape.profile (c : ChartShape) : Profile

```

Qtsl.QtslCompose.**IsClassK0Inf**

lean/qtsl-aux/QtslAux/QtslCompose/WireCertDischarge.lean, line 38

```

structure IsClassK0Inf (κ : ℝ → ℝ) : Prop where ...

```

Qtsl.QtslCompose.**polynomial\_discharge**

lean/qtsl-aux/QtslAux/QtslCompose/WireCertDischarge.lean, line 75

```

theorem polynomial_discharge (p : Polynomial ℝ)
  (h0 : p.coeff 0 = 0)
  (hderiv : ∀ x : ℝ, 0 ≤ x → 0 ≤ p.derivative.eval x)
  (hdeg : 0 < p.degree) (hlead : 0 ≤ p.leadingCoeff) :
  IsClassK0Inf (fun x => p.eval x) where
  zero_at_zero

```

Qtsl.QtslCompose.**slack\_triangle**

lean/qtsl-aux/QtslAux/QtslCompose/WireCertDischarge.lean, line 94

```

theorem slack_triangle {X : Type*} {f g h : X → ℝ} {c₁ c₂ : ℝ}
  (h₁ : ∀ x, f x ≤ g x + c₁) (h₂ : ∀ x, g x ≤ h x + c₂) :
  ∀ x, f x ≤ h x + (c₁ + c₂)

```

Qtsl.QtslCompose.**guarantee\_leg\_comp**

lean/qtsl-aux/QtslAux/QtslCompose/WireCertDischarge.lean, line 100

```

theorem guarantee_leg_comp {A B C : Type*} {w₁ : A → B} {w₂ : B → C}
  {γ₁ : A → ℝ} {γ₂ : B → ℝ} {γ₃ : C → ℝ}
  (h₁ : ∀ a, γ₂ (w₁ a) ≤ γ₁ a) (h₂ : ∀ b, γ₃ (w₂ b) ≤ γ₂ b) :
  ∀ a, γ₃ (w₂ (w₁ a)) ≤ γ₁ a

```

Qtsl.QtslCompose.IsClassK0Inf.**add**

lean/qtsl-aux/QtslAux/QtslCompose/WireCertDischarge.lean, line 52

```

theorem add (h₁ : IsClassK0Inf κ₁) (h₂ : IsClassK0Inf κ₂) :
  IsClassK0Inf (fun x => κ₁ x + κ₂ x) where
  zero_at_zero

```

Qtsl.QtslCompose.IsClassK0Inf.**comp**

lean/qtsl-aux/QtslAux/QtslCompose/WireCertDischarge.lean, line 63

```

theorem comp (h₁ : IsClassK0Inf κ₁) (h₂ : IsClassK0Inf κ₂) :
  IsClassK0Inf (κ₁ ∘ κ₂) where
  zero_at_zero

```

Qtsl.QtslCompose.IsClassK0Inf.**nonneg**

lean/qtsl-aux/QtslAux/QtslCompose/WireCertDischarge.lean, line 47

```

theorem nonneg (h : IsClassK0Inf κ) {x : ℝ} (hx : 0 ≤ x) : 0 ≤ κ x

```

BernsteinLogic.QtslCompose.Marks.**ConditionalDelivery**lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean,  
line 157

```

def ConditionalDelivery (N : MarkNet n V) (mark : Fin n → Set V) (i : Fin n) : Prop

```

BernsteinLogic.QtslCompose.Marks.**DepLocal**

lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line 109

```

| def DepLocal (N : MarkNet n V) : Prop

BernsteinLogic.QtslCompose.Marks.MarkFlowFeasible lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line
123
| def MarkFlowFeasible (N : MarkNet n V) (t : Fin n → ℕ) : Prop

BernsteinLogic.QtslCompose.Marks.CarvedDelivery lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.lean, line 69
| def CarvedDelivery (N : MarkNet n V) (mark P : Fin n → Set V) : Prop

BernsteinLogic.QtslCompose.Marks.CarveClosure lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.lean, line 76
| def CarveClosure (N : MarkNet n V) (P : Fin n → Set V) (t : Fin n → ℕ) : Prop

BernsteinLogic.QtslCompose.Marks.PinnedDelivery lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line 130
| def PinnedDelivery (N : MarkNet n V) (mark : Fin n → Set V) (a : Fin n → V) : Prop

BernsteinLogic.QtslCompose.Marks.InstantClosure lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line 148
| def InstantClosure (N : MarkNet n V) (a : Fin n → V) (t : Fin n → ℕ) : Prop

BernsteinLogic.QtslCompose.Marks.carvedDelivery_of_pinned lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.
lean, line 276
| theorem carvedDelivery_of_pinned (hpin : PinnedDelivery N mark a) :
  CarvedDelivery N mark (fun i => {a i})

BernsteinLogic.QtslCompose.Marks.carveClosure_of_pinned lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.
lean, line 290
| theorem carveClosure_of_pinned (hfeas : MarkFlowFeasible N t)
  (hpin : PinnedDelivery N mark a) (hclose : InstantClosure N a t) :
  CarveClosure N (fun i => {a i}) t

BernsteinLogic.QtslCompose.Marks.marks_of_pinned_delivery_via_carve lean/qtsl-aux/QtslAux/QtslCompose/
AGComposeCarves.lean, line 315
| theorem marks_of_pinned_delivery_via_carve (hser : Serial N)
  (hfeas : MarkFlowFeasible N t) (hpin : PinnedDelivery N mark a)
  (hclose : InstantClosure N a t) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i

BernsteinLogic.QtslCompose.Marks.carvedDelivery_mark_iff_conditional lean/qtsl-aux/QtslAux/QtslCompose/
AGComposeCarves.lean, line 332
| theorem carvedDelivery_mark_iff_conditional :
  CarvedDelivery N mark mark ↔ ∀ i, ConditionalDelivery N mark i

BernsteinLogic.QtslCompose.Marks.marks_of_carved_delivery lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.
lean, line 260
| theorem marks_of_carved_delivery (hser : Serial N) (hfeas : MarkFlowFeasible N t)
  (hpin : CarvedDelivery N mark P) (hclose : CarveClosure N P t) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i

BernsteinLogic.QtslCompose.Marks.marks_of_conditional_delivery_acyclic lean/qtsl-aux/QtslAux/QtslCompose/
AGComposeMarks.lean, line 421
| theorem marks_of_conditional_delivery_acyclic (hser : Serial N) (hloc : DepLocal N)
  (ord : Fin n → ℕ) (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (hfeas : MarkFlowFeasible N t) (hdel : ∀ i, ConditionalDelivery N mark i) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i

```

BernsteinLogic.QtslCompose.Marks.marks\_acyclic\_via\_maximal\_carve lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.lean, line 475

```
theorem marks_acyclic_via_maximal_carve (hser : Serial N) (hloc : DepLocal N)
  (ord : Fin n → ℕ) (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (hfeas : MarkFlowFeasible N t)
  (hdel : ∀ i, ConditionalDelivery N mark i) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i
```

BernsteinLogic.QtslCompose.Marks.JointRun lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line 104

```
def JointRun (N : MarkNet n V) (ω : ℕ → Fin n → V) : Prop
```

BernsteinLogic.QtslCompose.Marks.z1\_refutes\_walkless lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line 505

```
theorem z1_refutes_walkless :
  ¬ (∀ (m : ℕ) (W : Type) ( _ : Nonempty W) (N : MarkNet m W)
    (mark : Fin m → Set W) (a : Fin m → W) (t : Fin m → ℕ),
    Serial N → PinnedDelivery N mark a → InstantClosure N a t →
    ∃ ω : ℕ → Fin m → W, JointRun N ω ∧ ∀ i, ∃ k, ω k i ∈ mark i)
```

BernsteinLogic.QtslCompose.Marks.z1\_refutes\_walkless\_carve lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.lean, line 590

```
theorem z1_refutes_walkless_carve :
  ¬ (∀ (m : ℕ) (W : Type) ( _ : Nonempty W) (N : MarkNet m W)
    (mark P : Fin m → Set W) (t : Fin m → ℕ),
    Serial N → CarvedDelivery N mark P → CarveClosure N P t →
    ∃ ω : ℕ → Fin m → W, JointRun N ω ∧ ∀ i, ∃ k, ω k i ∈ mark i)
```

BernsteinLogic.QtslCompose.Marks.z5\_refutes\_pinless lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line 597

```
theorem z5_refutes_pinless :
  ¬ (∀ (m : ℕ) (W : Type) ( _ : Nonempty W) (N : MarkNet m W)
    (mark : Fin m → Set W) (t : Fin m → ℕ),
    Serial N → DepLocal N → (∀ i, ConditionalDelivery N mark i) →
    MarkFlowFeasible N t → W0Serial N → (∃ ω, JointRun N ω) →
    ∃ ω : ℕ → Fin m → W, JointRun N ω ∧ ∀ i, ∃ k, ω k i ∈ mark i)
```

BernsteinLogic.QtslCompose.Marks.z5\_refutes\_closureless lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.lean, line 530

```
theorem z5_refutes_closureless :
  ¬ (∀ (m : ℕ) (W : Type) ( _ : Nonempty W) (N : MarkNet m W)
    (mark P : Fin m → Set W) (t : Fin m → ℕ),
    Serial N → DepLocal N → CarvedDelivery N mark P →
    MarkFlowFeasible N t → W0Serial N → (∃ ω, JointRun N ω) →
    ∃ ω : ℕ → Fin m → W, JointRun N ω ∧ ∀ i, ∃ k, ω k i ∈ mark i)
```

BernsteinLogic.QtslCompose.Marks.z5\_carvedDelivery\_mark lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.lean, line 495

```
theorem z5_carvedDelivery_mark : CarvedDelivery z5Net z5Mark z5Mark
```

BernsteinLogic.QtslCompose.Marks.z5\_no\_carve\_closure lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.lean, line 509

```
theorem z5_no_carve_closure :
  ¬ ∃ P : Fin 2 → Set (Fin 3), (∀ i, P i ⊆ z5Mark i) ∧
  CarveClosure z5Net P ![0, 0]
```

BernsteinLogic.QtslCompose.Marks.z5\_no\_pin lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line 663

```

| theorem z5_no_pin : ¬ ∃ a : Fin 2 → Fin 3, PinnedDelivery z5Net z5Mark a

BernsteinLogic.QtslCompose.Marks.z1_walk_infeasible lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line
460
| theorem z1_walk_infeasible : ¬ ∃ t : Fin 2 → ℕ, MarkFlowFeasible z1Net t

BernsteinLogic.QtslCompose.Marks.allOnes_walk_feasible lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean,
line 469
| theorem allOnes_walk_feasible : MarkFlowFeasible allOnesNet ![0, 0]

BernsteinLogic.QtslCompose.Marks.allOnes_marks_realized lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean,
line 677
| theorem allOnes_marks_realized :
  ∃ ω : ℕ → Fin 2 → Bool, JointRun allOnesNet ω ∧
  ∀ i, ω (![0, 0] : Fin 2 → ℕ) i) i ∈ boolMark i

BernsteinLogic.QtslCompose.Marks.allOnes_direct lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.lean, line 711
| theorem allOnes_direct : JointRun allOnesNet fun _ => ![true, true]

BernsteinLogic.QtslCompose.Marks.fatCarve_marks_realized lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.
lean, line 666
| theorem fatCarve_marks_realized :
  ∃ ω : ℕ → Fin 2 → Fin 3, JointRun relay3Net ω ∧
  ∀ i, ω (![0, 0] : Fin 2 → ℕ) i) i ∈ relay3Mark i

BernsteinLogic.QtslCompose.Marks.marks_of_pinned_delivery lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.
lean, line 287
| theorem marks_of_pinned_delivery (hser : Serial N) (hfeas : MarkFlowFeasible N t)
  (hpin : PinnedDelivery N mark a) (hclose : InstantClosure N a t) :
  ∃ ω : ℕ → Fin n → V, JointRun N ω ∧ ∀ i, ω (t i) i ∈ mark i

BernsteinLogic.QtslCompose.Marks.pins_of_conditional_delivery lean/qtsl-aux/QtslAux/QtslCompose/
AGComposeMarks.lean, line 308
| theorem pins_of_conditional_delivery (hloc : DepLocal N) (ord : Fin n → ℕ)
  (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (hdel : ∀ i, ConditionalDelivery N mark i) :
  ∃ a : Fin n → V, PinnedDelivery N mark a

BernsteinLogic.QtslCompose.Marks.instantClosure_of_acyclic lean/qtsl-aux/QtslAux/QtslCompose/AGComposeMarks.
lean, line 357
| theorem instantClosure_of_acyclic (hser : Serial N) (hloc : DepLocal N)
  (ord : Fin n → ℕ) (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (a : Fin n → V) (t : Fin n → ℕ) :
  InstantClosure N a t

BernsteinLogic.QtslCompose.Marks.carveClosure_of_acyclic lean/qtsl-aux/QtslAux/QtslCompose/AGComposeCarves.
lean, line 357
| theorem carveClosure_of_acyclic (hser : Serial N) (hloc : DepLocal N)
  (ord : Fin n → ℕ) (hord : ∀ i, ∀ j ∈ N.dep i, ord j < ord i)
  (hfeas : MarkFlowFeasible N t) (hpin : CarvedDelivery N mark P) :
  CarveClosure N P t

Qtsl.QtslCompose.sim_expect lean/qtsl-aux/QtslAux/QtslCompose/SimTransport.lean, line 61

```

```

theorem sim_expect {σ : α → β} {K : α → α → ℝ} {Q : β → β → ℝ}
  (hσ : PushKernel σ K Q) (μ₀ : α → ℝ) (g : β → ℝ) (n : ℕ) :
  ∑ s, law K μ₀ n s * g (σ s) = ∑ b, law Q (push σ μ₀) n b * g b

```

Qtsl.QtslCompose.**sim\_expect\_bound**

lean/qtsl-aux/QtslAux/QtslCompose/SimTransport.lean, line 68

```

theorem sim_expect_bound {σ : α → β} {K : α → α → ℝ} {Q : β → β → ℝ}
  (hσ : PushKernel σ K Q) (μ₀ : α → ℝ) (g : β → ℝ) {n : ℕ} {b₀ : ℝ}
  (hb : ∑ b, law Q (push σ μ₀) n b * g b ≤ b₀) :
  ∑ s, law K μ₀ n s * g (σ s) ≤ b₀

```

Qtsl.QtslCompose.**sim\_region**

lean/qtsl-aux/QtslAux/QtslCompose/SimTransport.lean, line 77

```

theorem sim_region {σ : α → β} {K : α → α → ℝ} {Q : β → β → ℝ}
  (hσ : PushKernel σ K Q) {μ₀ : α → ℝ}
  (hK : ∀ s t, 0 ≤ K s t) (hμ₀ : ∀ s, 0 ≤ μ₀ s) {n : ℕ} {R' : Set β}
  (hR' : ∀ b, b ∉ R' → law Q (push σ μ₀) n b = 0) :
  ∀ s, σ s ∉ R' → law K μ₀ n s = 0

```

Qtsl.QtslCompose.**expect\_push**

lean/qtsl-aux/QtslAux/QtslCompose/SimTransport.lean, line 46

```

theorem expect_push (σ : α → β) (μ : α → ℝ) (g : β → ℝ) :
  ∑ b, push σ μ b * g b = ∑ s, μ s * g (σ s)

```

Qtsl.QRelChain.**push\_law**

lean/qtsl-aux/QtslAux/QRelChain.lean, line 152

```

theorem push_law [Fintype β] {π : α → β} {K : α → α → ℝ} {Q : β → β → ℝ}
  (h : PushKernel π K Q) (μ₀ : α → ℝ) :
  ∀ n, push π (law K μ₀ n) = law Q (push π μ₀) n
| 0 => rfl
| n + 1 => by rw [law_succ, push_step h, push_law h μ₀ n, law_succ]

end Push

section Drift

variable [Fintype α] {K : α → α → ℝ} {V μ₀ μ : α → ℝ}

theorem expect_step_le (hμ : ∀ s, 0 ≤ μ s)
  (hdrift : ∀ s, ∑ t, K s t * V t ≤ V s) :
  ∑ t, step K μ t * V t ≤ ∑ s, μ s * V s

```

Qtsl.QtslCompose.Meas.**sim\_expect\_lintegral**

lean/qtsl-aux/QtslAux/QtslCompose/SimTransportMeas.lean, line 91

```

theorem sim_expect_lintegral {σ : α → β} {K : Kernel α α} {Q : Kernel β β}
  (hσ : Measurable σ) (h : PushKernel σ K Q) (μ : Measure α)
  {g : β → ℝ≥0∞} (hg : Measurable g) (n : ℕ) :
  ∫⁻ s, g (σ s) ∂law K μ n = ∫⁻ b, g b ∂law Q (μ.map σ) n

```

Qtsl.QtslCompose.Meas.**pushKernel\_iff\_comp**

lean/qtsl-aux/QtslAux/QtslCompose/SimTransportMeas.lean, line 132

```

theorem pushKernel_iff_comp {σ : α → β} (hσ : Measurable σ)
  {K : Kernel α α} {Q : Kernel β β} :
  PushKernel σ K Q ↔
  Kernel.deterministic σ hσ ∘ₖ K = Q ∘ₖ Kernel.deterministic σ hσ

```

Qtsl.QRelChain.**law\_nonneg**

lean/qtsl-aux/QtslAux/QRelChain.lean, line 85

```

theorem law_nonneg (hK : ∀ s t, 0 ≤ K s t) (hμ₀ : ∀ s, 0 ≤ μ₀ s) :
  ∀ n s, 0 ≤ law K μ₀ n s
| 0, s => hμ₀ s
| n + 1, t => step_nonneg hK (law_nonneg hK hμ₀ n) t

theorem sum_step (hrow : ∀ s, ∑ t, K s t = 1) : ∑ t, step K μ t = ∑ s, μ s

```



Qtsl.QRelChain.**PushKernel**

lean/qtsl-aux/QtslAux/QRelChain.lean, line 130

```
def PushKernel (π : α → β) (K : α → α → ℝ) (Q : β → β → ℝ) : Prop
```

Qtsl.QtslCompose.**law\_prodKernel**

lean/qtsl-aux/QtslAux/QtslCompose/Flatten.lean, line 94

```
theorem law_prodKernel (K1 : α → α → ℝ) (K2 : β → β → ℝ)
  (μ1 : α → ℝ) (μ2 : β → ℝ) :
  ∀ n, law (prodKernel K1 K2) (prodDist μ1 μ2) n
    = prodDist (law K1 μ1 n) (law K2 μ2 n)
| 0 => rfl
| n + 1 => by
  rw [law_succ, law_prodKernel K1 K2 μ1 μ2 n, step_prodKernel, law_succ,
    law_succ]
```

```
section Projections
```

```
variable [DecidableEq α] [DecidableEq β]
```

```
theorem pushKernel_fst {K1 : α → α → ℝ} {K2 : β → β → ℝ}
  (h2 : ∀ b, ∑ b', K2 b b' = 1) :
  PushKernel Prod.fst (prodKernel K1 K2) K1
```

Qtsl.QtslCompose.**pushKernel\_fst**

lean/qtsl-aux/QtslAux/QtslCompose/Flatten.lean, line 109

```
theorem pushKernel_fst {K1 : α → α → ℝ} {K2 : β → β → ℝ}
  (h2 : ∀ b, ∑ b', K2 b b' = 1) :
  PushKernel Prod.fst (prodKernel K1 K2) K1
```

Qtsl.QtslCompose.**prodKernel**

lean/qtsl-aux/QtslAux/QtslCompose/Flatten.lean, line 34

```
def prodKernel (K1 : α → α → ℝ) (K2 : β → β → ℝ) :
  α × β → α × β → ℝ
```

Qtsl.QtslCompose.Meas.**law\_parallelComp**

lean/qtsl-aux/QtslAux/QtslCompose/FlattenMeas.lean, line 75

```
theorem law_parallelComp (μ : Measure α) (ν : Measure β)
  (K : Kernel α α) (Q : Kernel β β)
  [SFinite μ] [SFinite ν] [ISFiniteKernel K] [ISFiniteKernel Q] :
  ∀ n, law (K ⋈k Q) (μ.prod ν) n = (law K μ n).prod (law Q ν n)
| 0 => rfl
| n + 1 => by
  rw [law_succ, law_parallelComp μ ν K Q n, bind_parallelComp,
    ← law_succ, ← law_succ]
```

```
theorem pushKernel_fst {K : Kernel α α} {Q : Kernel β β}
  [ISFiniteKernel K] [ISMarkovKernel Q] :
  PushKernel Prod.fst (K ⋈k Q) K
```

Qtsl.QtslCompose.Meas.**PushKernel**

lean/qtsl-aux/QtslAux/QtslCompose/SimTransportMeas.lean, line 43

```
def PushKernel (σ : α → β) (K : Kernel α α) (Q : Kernel β β) : Prop
```

Qtsl.QtslCompose.Meas.**bind\_parallelComp**

lean/qtsl-aux/QtslAux/QtslCompose/FlattenMeas.lean, line 58

```
theorem bind_parallelComp (μ : Measure α) (ν : Measure β)
  (K : Kernel α α) (Q : Kernel β β)
  [SFinite μ] [SFinite ν] [ISFiniteKernel K] [ISFiniteKernel Q] :
  (μ.prod ν).bind (K ⋈k Q) = (μ.bind K).prod (ν.bind Q)
```

Qtsl.QtslCompose.**isMarkov\_prodKernel**

lean/qtsl-aux/QtslAux/QtslCompose/Flatten.lean, line 45

```
theorem isMarkov_prodKernel {K1 : α → α → ℝ} {K2 : β → β → ℝ}
  (h1 : ISMarkov K1) (h2 : ISMarkov K2) :
```

```

    IsMarkov (prodKernel K1 K2) where
    nonneg

```

Qtsl.QtslCompose.isDist\_prodDist lean/qtsl-aux/QtslAux/QtslCompose/Flatten.lean, line 61

```

theorem isDist_prodDist {μ1 : α → ℝ} {μ2 : β → ℝ}
  (h1 : IsDist μ1) (h2 : IsDist μ2) : IsDist (prodDist μ1 μ2) where
  nonneg

```

Qtsl.QtslCompose.push\_fst\_prodDist lean/qtsl-aux/QtslAux/QtslCompose/Flatten.lean, line 135

```

theorem push_fst_prodDist {μ1 : α → ℝ} {μ2 : β → ℝ} (h2 : IsDist μ2) :
  push Prod.fst (prodDist μ1 μ2) = μ1

```

Qtsl.QtslCompose.Meas.sfinite\_law lean/qtsl-aux/QtslAux/QtslCompose/FlattenMeas.lean, line 38

```

instance sfinite_law (K : Kernel α α) [IsSFiniteKernel K]
  (μ : Measure α) [SFinite μ] : ∀ n, SFinite (law K μ n)
| 0 => inferInstanceAs (SFinite μ)
| n + 1 => by
  have

```

Qtsl.QtslCompose.Meas.isProbabilityMeasure\_law lean/qtsl-aux/QtslAux/QtslCompose/FlattenMeas.lean, line 47

```

instance isProbabilityMeasure_law (K : Kernel α α) [IsMarkovKernel K]
  (μ : Measure α) [IsProbabilityMeasure μ] :
  ∀ n, IsProbabilityMeasure (law K μ n)
| 0 => inferInstanceAs (IsProbabilityMeasure μ)
| n + 1 => by
  have

```

Qtsl.QtslCompose.Meas.sim\_expect\_fst lean/qtsl-aux/QtslAux/QtslCompose/FlattenMeas.lean, line 106

```

theorem sim_expect_fst {K : Kernel α α} {Q : Kernel β β}
  [IsSFiniteKernel K] [IsMarkovKernel Q]
  (μ : Measure α) (ν : Measure β) [IsProbabilityMeasure ν]
  {g : α → ℝ≥0∞} (hg : Measurable g) (n : ℕ) :
  ∫- p, g p.1 dlaw (K  $\ll$  Q) (μ.prod ν) n = ∫- a, g a dlaw K μ n

```

Qtsl.QtslCompose.Meas.pushKernel\_iff\_stochSim lean/qtsl-aux/QtslAux/QtslCompose/FlattenMeas.lean, line 128

```

theorem pushKernel_iff_stochSim {σ : α → β} {K : Kernel α α}
  {Q : Kernel β β} : PushKernel σ K Q ↔ StochSim σ K Q

```

Qtsl.QtslCompose.prodDist lean/qtsl-aux/QtslAux/QtslCompose/Flatten.lean, line 40

```

def prodDist (μ1 : α → ℝ) (μ2 : β → ℝ) : α × β → ℝ

```

Qtsl.QtslCompose.step\_prodKernel lean/qtsl-aux/QtslAux/QtslCompose/Flatten.lean, line 75

```

theorem step_prodKernel (K1 : α → α → ℝ) (K2 : β → β → ℝ)
  (μ1 : α → ℝ) (μ2 : β → ℝ) :
  step (prodKernel K1 K2) (prodDist μ1 μ2)
  = prodDist (step K1 μ1) (step K2 μ2)

```

Qtsl.QtslCompose.pushKernel\_snd lean/qtsl-aux/QtslAux/QtslCompose/Flatten.lean, line 122

```

theorem pushKernel_snd {K1 : α → α → ℝ} {K2 : β → β → ℝ}
  (h1 : ∀ a, ∑ a', K1 a a' = 1) :
  PushKernel Prod.snd (prodKernel K1 K2) K2

```

Qtsl.QDistTransport.strict\_cell\_exact lean/qtsl-aux/QtslAux/QDistTransport.lean, line 193

```

theorem strict_cell_exact [DecidableEq β] {π : α → β} {K : α → α → ℝ}
  {Q : β → β → ℝ} (h : PushKernel π K Q) (μ0 : α → ℝ) (n : ℕ) :
  push π (law K μ0 n) = law Q (push π μ0) n

```

Qtsl.QtslCompose.aggReduce lean/qtsl-aux/QtslAux/QtslCompose/AggReduce.lean, line 37

```
theorem aggReduce {X A : Type*} (agg : X → A) (scope : Set X) (K : Set A)
  (G : A → ℝ) (hG : ∀ a ∈ K, G a ≤ 0) (hin : ∀ x ∈ scope, agg x ∈ K) :
  ∀ x ∈ scope, G (agg x) ≤ 0
```

Qtsl.QtslCompose.coupling\_row lean/qtsl-aux/QtslAux/QtslCompose/AggReduce.lean, line 80

```
theorem coupling_row {m : Multiset C} {f : C → ℝ} {l u : ℝ}
  (hl : ∀ c ∈ m, l ≤ f c) (hu : ∀ c ∈ m, f c ≤ u) :
  l * m.card ≤ query m f ∧ query m f ≤ u * m.card
```

Qtsl.QtslCompose.count\_row\_nonneg lean/qtsl-aux/QtslAux/QtslCompose/AggReduce.lean, line 96

```
theorem count_row_nonneg (m : Multiset C) : (0 : ℝ) ≤ m.card
```

Qtsl.QtslCompose.count\_row\_le lean/qtsl-aux/QtslAux/QtslCompose/AggReduce.lean, line 99

```
theorem count_row_le {m : Multiset C} {M : ℕ} (h : m.card ≤ M) :
  (m.card : ℝ) ≤ M
```

Qtsl.QtslCompose.moment\_row lean/qtsl-aux/QtslAux/QtslCompose/AggReduce.lean, line 105

```
theorem moment_row (m : Multiset C) (f : C → ℝ) :
  (query m f) ^ 2 ≤ m.card * query m (fun c => f c ^ 2)
```

Qtsl.QtslCompose.query\_count lean/qtsl-aux/QtslAux/QtslCompose/AggReduce.lean, line 66

```
theorem query_count [DecidableEq C] (m : Multiset C) (f : C → ℝ) :
  query m f = ∑ c ∈ m.toFinset, (m.count c : ℝ) * f c
```

Qtsl.QtslCompose.query\_const\_one lean/qtsl-aux/QtslAux/QtslCompose/AggReduce.lean, line 55

```
theorem query_const_one (m : Multiset C) :
  query m (fun _ => (1 : ℝ)) = m.card
```

Qtsl.Structural.countAtomRounding\_sound lean/qtsl-aux/QtslAux/Structural.lean, line 236

```
theorem countAtomRounding_sound (k : X → ℝ) (j : ℤ) (Reach : Set X)
  (hbridge : ∀ x ∈ Reach, ∃ n : ℤ, k x = (n : ℝ)) :
  ∀ x ∈ Reach, x ∈ {x | k x ≤ (j : ℝ)}c → (j : ℝ) + 1 ≤ k x
```

Qtsl.Structural.countAtomRounding\_tighter lean/qtsl-aux/QtslAux/Structural.lean, line 250

```
theorem countAtomRounding_tighter (k : X → ℝ) (j : ℤ) :
  {x | (j : ℝ) + 1 ≤ k x} ⊆ {x | (j : ℝ) < k x}
```

Qtsl.Structural.countAtomRounding\_eq lean/qtsl-aux/QtslAux/Structural.lean, line 259

```
theorem countAtomRounding_eq (k : X → ℝ) (j : ℤ) (Reach : Set X)
  (hbridge : ∀ x ∈ Reach, ∃ n : ℤ, k x = (n : ℝ)) :
  Reach n {x | k x ≤ (j : ℝ)}c = Reach n {x | (j : ℝ) + 1 ≤ k x}
```

Qtsl.QRelChain.sound\_QRelTV lean/qtsl-aux/QtslAux/QRelChain.lean, line 736

```
theorem sound_QRelTV [Fintype L] [DecidableEq L] {f : σ₁ → L} {g : σ₂ → L}
  {V : σ₁ × σ₂ → ℝ} {b : ℝ}
  (hK : IsMarkov K) (hμ₀ : IsDist μ₀) (hK₁ : IsMarkov K₁) (hK₂ : IsMarkov K₂)
  (hm₁ : PushKernel Prod.fst K K₁) (hm₂ : PushKernel Prod.snd K K₂)
  (hfreeze : ∀ p ∈ A₁ ×s A₂, K p p = 1)
  (hfr₁ : ∀ a ∈ A₁, K₁ a a = 1) (hfr₂ : ∀ a ∈ A₂, K₂ a a = 1)
  (habs₁ : Tendsto (fun n => ∑ a ∈ A₁, law K₁ (push Prod.fst μ₀) n a)
    atTop (ℵ 1))
  (habs₂ : Tendsto (fun n => ∑ a ∈ A₂, law K₂ (push Prod.snd μ₀) n a)
    atTop (ℵ 1))
```

```

(hV0 : ∀ p, 0 ≤ V p) (hdrift : ∀ p, ∑ q, K p q * V q ≤ V p)
(hterm : ∀ p ∈ A1 ×s A2, f p.1 ≠ g p.2 → 1 ≤ V p)
(hb : ∑ p, μ0 p * V p ≤ b) :
tvDist (push f (finalLaw K1 (push Prod.fst μ0) A1))
(push g (finalLaw K2 (push Prod.snd μ0) A2)) ≤ b

```

Qtsl.QRelChain.sound\_QRelExpGap

lean/qtsl-aux/QtslAux/QRelChain.lean, line 777

```

theorem sound_QRelExpGap {v1 : σ1 → ℝ} {v2 : σ2 → ℝ} {V : σ1 × σ2 → ℝ} {b : ℝ}
(hK : IsMarkov K) (hμ0 : IsDist μ0) (hK1 : IsMarkov K1) (hK2 : IsMarkov K2)
(hm1 : PushKernel Prod.fst K K1) (hm2 : PushKernel Prod.snd K K2)
(hfreeze : ∀ p ∈ A1 ×s A2, K p p = 1)
(hfr1 : ∀ a ∈ A1, K1 a a = 1) (hfr2 : ∀ a ∈ A2, K2 a a = 1)
(habs1 : Tendsto (fun n => ∑ a ∈ A1, law K1 (push Prod.fst μ0) n a)
atTop (ℵ 1))
(habs2 : Tendsto (fun n => ∑ a ∈ A2, law K2 (push Prod.snd μ0) n a)
atTop (ℵ 1))
(hV0 : ∀ p, 0 ≤ V p) (hdrift : ∀ p, ∑ q, K p q * V q ≤ V p)
(hterm : ∀ p ∈ A1 ×s A2, |v1 p.1 - v2 p.2| ≤ V p)
(hb : ∑ p, μ0 p * V p ≤ b) :
| (∑ a1 ∈ A1, finalMass K1 (push Prod.fst μ0) a1 * v1 a1) -
∑ a2 ∈ A2, finalMass K2 (push Prod.snd μ0) a2 * v2 a2 | ≤ b

```

Qtsl.QRelChain.tendsto\_absorbedMass\_one

lean/qtsl-aux/QtslAux/QRelChain.lean, line 392

```

theorem tendsto_absorbedMass_one (hK : IsMarkov K) (hμ0 : IsDist μ0)
(hfr : ∀ a ∈ A, K a a = 1)
(hreach : ∀ s, ∃ k, 0 < ∑ a ∈ A, law K (pointMass s) k a) :
Tendsto (fun n => ∑ a ∈ A, law K μ0 n a) atTop (ℵ 1)

```

Qtsl.QRelChain.tvDist

lean/qtsl-aux/QtslAux/QRelChain.lean, line 453

```

def tvDist [Fintype L] (p q : L → ℝ) : ℝ

```

Qtsl.QRelChain.push\_step

lean/qtsl-aux/QtslAux/QRelChain.lean, line 133

```

theorem push_step [Fintype β] {π : α → β} {K : α → α → ℝ} {Q : β → β → ℝ}
(h : PushKernel π K Q) (μ : α → ℝ) :
push π (step K μ) = step Q (push π μ)

```

Qtsl.QRelChain.expect\_step\_le

lean/qtsl-aux/QtslAux/QRelChain.lean, line 164

```

theorem expect_step_le (hμ : ∀ s, 0 ≤ μ s)
(hdrift : ∀ s, ∑ t, K s t * V t ≤ V s) :
∑ t, step K μ t * V t ≤ ∑ s, μ s * V s

```

Qtsl.QRelChain.expect\_law\_le

lean/qtsl-aux/QtslAux/QRelChain.lean, line 175

```

theorem expect_law_le (hK : ∀ s t, 0 ≤ K s t) (hμ0 : ∀ s, 0 ≤ μ0 s)
(hdrift : ∀ s, ∑ t, K s t * V t ≤ V s) :
∀ n, ∑ s, law K μ0 n s * V s ≤ ∑ s, μ0 s * V s
| 0 => le_rfl
| n + 1 =>
(expect_step_le (law_nonneg hK hμ0 n) hdrift).trans
(expect_law_le hK hμ0 hdrift n)

end Drift

section Absorb

variable [Fintype α] {K : α → α → ℝ} {μ0 : α → ℝ}

theorem law_mono_absorbing (hK : ∀ s t, 0 ≤ K s t) (hμ0 : ∀ s, 0 ≤ μ0 s)

```

```

{a :  $\alpha$ } (haa : K a a = 1) :
Monotone fun n => law K  $\mu_0$  n a

```

Qtsl.QRelChain.law\_mono\_absorbing

lean/qtsl-aux/QtslAux/QRelChain.lean, line 189

```

theorem law_mono_absorbing (hK :  $\forall s t, 0 \leq K s t$ ) (h $\mu_0$  :  $\forall s, 0 \leq \mu_0 s$ )
{a :  $\alpha$ } (haa : K a a = 1) :
Monotone fun n => law K  $\mu_0$  n a

```

Qtsl.QRelChain.finalMass

lean/qtsl-aux/QtslAux/QRelChain.lean, line 198

```

def finalMass (K :  $\alpha \rightarrow \alpha \rightarrow \mathbb{R}$ ) ( $\mu_0$  :  $\alpha \rightarrow \mathbb{R}$ ) (a :  $\alpha$ ) :  $\mathbb{R}$ 

```

Qtsl.QRelChain.finalExpect\_le

lean/qtsl-aux/QtslAux/QRelChain.lean, line 226

```

theorem finalExpect_le {V gv :  $\alpha \rightarrow \mathbb{R}$ } {A : Finset  $\alpha$ } {b :  $\mathbb{R}$ }
(hK : IsMarkov K) (h $\mu_0$  : IsDist  $\mu_0$ ) (hfr :  $\forall a \in A, K a a = 1$ )
(hgV :  $\forall a \in A, gv a \leq V a$ ) (hV0 :  $\forall s, 0 \leq V s$ )
(hdrift :  $\forall s, \sum t, K s t * V t \leq V s$ ) (hb :  $\sum s, \mu_0 s * V s \leq b$ ) :
 $\sum a \in A, finalMass K \mu_0 a * gv a \leq b$ 

```

Qtsl.QRelChain.tvDist

lean/qtsl-aux/QtslAux/QRelChain.lean, line 453

```

def tvDist [Fintype L] (p q : L  $\rightarrow \mathbb{R}$ ) :  $\mathbb{R}$ 

```

Qtsl.QRelChain.tvDist\_push\_le\_disagree

lean/qtsl-aux/QtslAux/QRelChain.lean, line 455

```

theorem tvDist_push_le_disagree { $\sigma_1 \sigma_2$  : Type*} [Fintype  $\sigma_1$ ] [Fintype  $\sigma_2$ ]
[Fintype L] [DecidableEq L] ( $\gamma$  :  $\sigma_1 \times \sigma_2 \rightarrow \mathbb{R}$ ) (h $\gamma$  :  $\forall p, 0 \leq \gamma p$ )
(f :  $\sigma_1 \rightarrow L$ ) (g :  $\sigma_2 \rightarrow L$ ) :
tvDist (push (fun p :  $\sigma_1 \times \sigma_2 \Rightarrow f p.1$ )  $\gamma$ ) (push (fun p :  $\sigma_1 \times \sigma_2 \Rightarrow g p.2$ )  $\gamma$ )  $\leq$ 
 $\sum p \in \text{univ.filter fun p : } \sigma_1 \times \sigma_2 \Rightarrow f p.1 \neq g p.2, \gamma p$ 

```

Qtsl.QRelChain.sum\_finalMass\_snd\_eq

lean/qtsl-aux/QtslAux/QRelChain.lean, line 532

```

theorem sum_finalMass_snd_eq (hK : IsMarkov K) (h $\mu_0$  : IsDist  $\mu_0$ )
(hK $_1$  : IsMarkov K $_1$ ) (hK $_2$  : IsMarkov K $_2$ )
(hm $_1$  : PushKernel Prod.fst K K $_1$ ) (hm $_2$  : PushKernel Prod.snd K K $_2$ )
(hfreeze :  $\forall p \in A_1 \times^s A_2, K p p = 1$ ) (hfr $_1$  :  $\forall a \in A_1, K_1 a a = 1$ )
(habs $_2$  : Tendsto (fun n =>  $\sum a \in A_2, law K_2 (push Prod.snd \mu_0) n a$ )
atTop ( $\mathcal{N} 1$ ))
{a $_1$  :  $\sigma_1$ } (ha $_1$  : a $_1$   $\in A_1$ ) :
 $\sum a_2 \in A_2, finalMass K \mu_0 (a_1, a_2) = finalMass K_1 (push Prod.fst \mu_0) a_1$ 

```

Qtsl.QRelChain.sum\_finalMass\_fst\_eq

lean/qtsl-aux/QtslAux/QRelChain.lean, line 585

```

theorem sum_finalMass_fst_eq (hK : IsMarkov K) (h $\mu_0$  : IsDist  $\mu_0$ )
(hK $_1$  : IsMarkov K $_1$ ) (hK $_2$  : IsMarkov K $_2$ )
(hm $_1$  : PushKernel Prod.fst K K $_1$ ) (hm $_2$  : PushKernel Prod.snd K K $_2$ )
(hfreeze :  $\forall p \in A_1 \times^s A_2, K p p = 1$ ) (hfr $_2$  :  $\forall a \in A_2, K_2 a a = 1$ )
(habs $_1$  : Tendsto (fun n =>  $\sum a \in A_1, law K_1 (push Prod.fst \mu_0) n a$ )
atTop ( $\mathcal{N} 1$ ))
{a $_2$  :  $\sigma_2$ } (ha $_2$  : a $_2$   $\in A_2$ ) :
 $\sum a_1 \in A_1, finalMass K \mu_0 (a_1, a_2) = finalMass K_2 (push Prod.snd \mu_0) a_2$ 

```

Qtsl.QRelChain.sum\_finalMass\_prod\_eq\_one

lean/qtsl-aux/QtslAux/QRelChain.lean, line 638

```

theorem sum_finalMass_prod_eq_one (hK : IsMarkov K) (h $\mu_0$  : IsDist  $\mu_0$ )
(hK $_1$  : IsMarkov K $_1$ ) (hK $_2$  : IsMarkov K $_2$ )
(hm $_1$  : PushKernel Prod.fst K K $_1$ ) (hm $_2$  : PushKernel Prod.snd K K $_2$ )
(hfreeze :  $\forall p \in A_1 \times^s A_2, K p p = 1$ ) (hfr $_1$  :  $\forall a \in A_1, K_1 a a = 1$ )
(habs $_1$  : Tendsto (fun n =>  $\sum a \in A_1, law K_1 (push Prod.fst \mu_0) n a$ )
atTop ( $\mathcal{N} 1$ ))
(habs $_2$  : Tendsto (fun n =>  $\sum a \in A_2, law K_2 (push Prod.snd \mu_0) n a$ )

```

```

    atTop ( $\mathcal{N}$  1)) :
 $\sum p \in A_1 \times^s A_2, \text{finalMass } K \mu_0 p = 1$ 

```

Qtsl.QRelChain.push\_finalLaw\_fst

lean/qtsl-aux/QtslAux/QRelChain.lean, line 667

```

theorem push_finalLaw_fst [Fintype L] [DecidableEq L]
  (hK : IsMarkov K) (hμ₀ : IsDist μ₀) (hK₁ : IsMarkov K₁) (hK₂ : IsMarkov K₂)
  (hm₁ : PushKernel Prod.fst K K₁) (hm₂ : PushKernel Prod.snd K K₂)
  (hfreeze :  $\forall p \in A_1 \times^s A_2, K p p = 1$ ) (hfr₁ :  $\forall a \in A_1, K_1 a a = 1$ )
  (habs₂ : Tendsto (fun n =>  $\sum a \in A_2, \text{law } K_2 (\text{push Prod.snd } \mu_0) n a$ )
    atTop ( $\mathcal{N}$  1))
  (f :  $\sigma_1 \rightarrow L$ ) :
  push (fun p :  $\sigma_1 \times \sigma_2 \Rightarrow f p.1$ ) (finalLaw K μ₀ ( $A_1 \times^s A_2$ )) =
    push f (finalLaw K₁ (push Prod.fst μ₀) A₁)

```

Qtsl.QRelChain.sound\_QRelTV\_of\_reach

lean/qtsl-aux/QtslAux/QRelChain.lean, line 815

```

theorem sound_QRelTV_of_reach [Fintype L] [DecidableEq L] {f :  $\sigma_1 \rightarrow L$ }
  {g :  $\sigma_2 \rightarrow L$ } {V :  $\sigma_1 \times \sigma_2 \rightarrow \mathbb{R}$ } {b :  $\mathbb{R}$ }
  (hK : IsMarkov K) (hμ₀ : IsDist μ₀) (hK₁ : IsMarkov K₁) (hK₂ : IsMarkov K₂)
  (hm₁ : PushKernel Prod.fst K K₁) (hm₂ : PushKernel Prod.snd K K₂)
  (hfreeze :  $\forall p \in A_1 \times^s A_2, K p p = 1$ )
  (hfr₁ :  $\forall a \in A_1, K_1 a a = 1$ ) (hfr₂ :  $\forall a \in A_2, K_2 a a = 1$ )
  (hreach₁ :  $\forall s, \exists k, 0 < \sum a \in A_1, \text{law } K_1 (\text{pointMass } s) k a$ )
  (hreach₂ :  $\forall s, \exists k, 0 < \sum a \in A_2, \text{law } K_2 (\text{pointMass } s) k a$ )
  (hV0 :  $\forall p, 0 \leq V p$ ) (hdrift :  $\forall p, \sum q, K p q * V q \leq V p$ )
  (hterm :  $\forall p \in A_1 \times^s A_2, f p.1 \neq g p.2 \rightarrow 1 \leq V p$ )
  (hb :  $\sum p, \mu_0 p * V p \leq b$ ) :
  tvDist (push f (finalLaw K₁ (push Prod.fst μ₀) A₁))
    (push g (finalLaw K₂ (push Prod.snd μ₀) A₂))  $\leq b$ 

```

Qtsl.QRelChain.sound\_QRelExpGap\_of\_reach

lean/qtsl-aux/QtslAux/QRelChain.lean, line 833

```

theorem sound_QRelExpGap_of_reach {v₁ :  $\sigma_1 \rightarrow \mathbb{R}$ } {v₂ :  $\sigma_2 \rightarrow \mathbb{R}$ }
  {V :  $\sigma_1 \times \sigma_2 \rightarrow \mathbb{R}$ } {b :  $\mathbb{R}$ }
  (hK : IsMarkov K) (hμ₀ : IsDist μ₀) (hK₁ : IsMarkov K₁) (hK₂ : IsMarkov K₂)
  (hm₁ : PushKernel Prod.fst K K₁) (hm₂ : PushKernel Prod.snd K K₂)
  (hfreeze :  $\forall p \in A_1 \times^s A_2, K p p = 1$ )
  (hfr₁ :  $\forall a \in A_1, K_1 a a = 1$ ) (hfr₂ :  $\forall a \in A_2, K_2 a a = 1$ )
  (hreach₁ :  $\forall s, \exists k, 0 < \sum a \in A_1, \text{law } K_1 (\text{pointMass } s) k a$ )
  (hreach₂ :  $\forall s, \exists k, 0 < \sum a \in A_2, \text{law } K_2 (\text{pointMass } s) k a$ )
  (hV0 :  $\forall p, 0 \leq V p$ ) (hdrift :  $\forall p, \sum q, K p q * V q \leq V p$ )
  (hterm :  $\forall p \in A_1 \times^s A_2, |v_1 p.1 - v_2 p.2| \leq V p$ )
  (hb :  $\sum p, \mu_0 p * V p \leq b$ ) :
  |(  $\sum a_1 \in A_1, \text{finalMass } K_1 (\text{push Prod.fst } \mu_0) a_1 * v_1 a_1$ ) -
     $\sum a_2 \in A_2, \text{finalMass } K_2 (\text{push Prod.snd } \mu_0) a_2 * v_2 a_2$  |  $\leq b$ 

```

Qtsl.QRelChain.exists\_uniform\_reach

lean/qtsl-aux/QtslAux/QRelChain.lean, line 334

```

theorem exists_uniform_reach (hK : IsMarkov K) (hfr :  $\forall a \in A, K a a = 1$ )
  (hreach :  $\forall s, \exists k, 0 < \sum a \in A, \text{law } K (\text{pointMass } s) k a$ ) :
   $\exists N \epsilon, 0 < \epsilon \wedge 1 \leq N \wedge \forall s, \epsilon \leq \sum a \in A, \text{law } K (\text{pointMass } s) N a$ 

```

Qtsl.QRelChain.absorbed\_decay

lean/qtsl-aux/QtslAux/QRelChain.lean, line 354

```

theorem absorbed_decay {μ :  $\alpha \rightarrow \mathbb{R}$ } (hK : IsMarkov K) (hμ : IsDist μ)
  (hfr :  $\forall a \in A, K a a = 1$ ) {N :  $\mathbb{N}$ } {ε :  $\mathbb{R}$ }
  (hε :  $\forall s, \epsilon \leq \sum a \in A, \text{law } K (\text{pointMass } s) N a$ ) :
   $1 - \sum a \in A, \text{law } K \mu N a \leq$ 
     $(1 - \epsilon) * (1 - \sum s \in A, \mu s)$ 

```

Qtsl.QFixBall.sound\_QFixBall

lean/qtsl-aux/QtslAux/QFixBall.lean, line 88

```

theorem sound_QFixBall [CompleteSpace  $\alpha$ ] (hK :  $K < 1$ )
  (hlip : LipschitzOnWith K f (closedBall  $x_0$  r))
  (hres :  $\text{dist } (f x_0) x_0 \leq \delta$ ) (hrad :  $\delta + K * r \leq r$ ) :
   $\exists! y, y \in \text{closedBall } x_0 r \wedge \text{IsFixedPt } f y$ 

```

Qtsl.QFixBall.dist\_fixedPoint\_le

lean/qtsl-aux/QtslAux/QFixBall.lean, line 59

```

theorem dist_fixedPoint_le (hK :  $K < 1$ )
  (hlip : LipschitzOnWith K f (closedBall  $x_0$  r))
  (hres :  $\text{dist } (f x_0) x_0 \leq \delta$ ) (hrad :  $\delta + K * r \leq r$ )
  {y :  $\alpha$ } (hy :  $y \in \text{closedBall } x_0 r$ ) (hfy :  $\text{IsFixedPt } f y$ ) :
   $\text{dist } x_0 y \leq \delta / (1 - K)$ 

```

Qtsl.QFixBall.anchored\_le\_radius

lean/qtsl-aux/QtslAux/QFixBall.lean, line 103

```

theorem anchored_le_radius (hK :  $K < 1$ )
  (hrad :  $\delta + K * r \leq r$ ) :  $\delta / (1 - K) \leq r$ 

```

Qtsl.QFixBall.sound\_QFixBall\_affine

lean/qtsl-aux/QtslAux/QFixBall.lean, line 120

```

theorem sound_QFixBall_affine {M :  $\mathbb{N}$ }
  (A : EuclideanSpace  $\mathbb{R}$  (Fin M)  $\rightarrow$  L[ $\mathbb{R}$ ] EuclideanSpace  $\mathbb{R}$  (Fin M))
  (c  $x_0$  : EuclideanSpace  $\mathbb{R}$  (Fin M)) {K :  $\mathbb{R}_{\geq 0}$ } { $\delta$  r :  $\mathbb{R}$ }
  (hK :  $K < 1$ ) (hA :  $\|A\|_+ \leq K$ )
  (hres :  $\text{dist } (A x_0 + c) x_0 \leq \delta$ ) (hrad :  $\delta + K * r \leq r$ ) :
   $\exists! y, y \in \text{closedBall } x_0 r \wedge \text{IsFixedPt } (\text{fun } x \Rightarrow A x + c) y$ 

```

Qtsl.Wasserstein.sound\_QFixBall\_wasserstein

lean/qtsl-aux/QtslAux/Wasserstein/Complete.lean, line 449

```

theorem sound_QFixBall_wasserstein {T :  $P_2 X \rightarrow P_2 X$ } { $\mu_0$  :  $P_2 X$ } {K :  $\mathbb{R}_{\geq 0}$ }
  { $\delta$  r :  $\mathbb{R}$ } (hK :  $K < 1$ )
  (hlip : LipschitzOnWith K T (closedBall  $\mu_0$  r))
  (hres :  $\text{dist } (T \mu_0) \mu_0 \leq \delta$ ) (hrad :  $\delta + K * r \leq r$ ) :
   $\exists! v, v \in \text{closedBall } \mu_0 r \wedge \text{IsFixedPt } T v$ 

```

Qtsl.QFixBall.delta\_nonneg

lean/qtsl-aux/QtslAux/QFixBall.lean, line 34

```

theorem delta_nonneg (hres :  $\text{dist } (f x_0) x_0 \leq \delta$ ) :  $0 \leq \delta$ 

```

Qtsl.QFixBall.radius\_nonneg

lean/qtsl-aux/QtslAux/QFixBall.lean, line 37

```

theorem radius_nonneg (hK :  $K < 1$ ) (hres :  $\text{dist } (f x_0) x_0 \leq \delta$ )
  (hrad :  $\delta + K * r \leq r$ ) :  $0 \leq r$ 

```

Qtsl.QFixBall.mapsTo\_closedBall

lean/qtsl-aux/QtslAux/QFixBall.lean, line 45

```

theorem mapsTo_closedBall (hK :  $K < 1$ )
  (hlip : LipschitzOnWith K f (closedBall  $x_0$  r))
  (hres :  $\text{dist } (f x_0) x_0 \leq \delta$ ) (hrad :  $\delta + K * r \leq r$ ) :
  Set.MapsTo f (closedBall  $x_0$  r) (closedBall  $x_0$  r)

```

Qtsl.QFixBall.fixedPoint\_unique

lean/qtsl-aux/QtslAux/QFixBall.lean, line 76

```

theorem fixedPoint_unique (hK :  $K < 1$ )
  (hlip : LipschitzOnWith K f (closedBall  $x_0$  r))
  {y z :  $\alpha$ } (hy :  $y \in \text{closedBall } x_0 r$ ) (hz :  $z \in \text{closedBall } x_0 r$ )
  (hfy :  $\text{IsFixedPt } f y$ ) (hfz :  $\text{IsFixedPt } f z$ ) :  $y = z$ 

```

Qtsl.QFixBall.lipschitzWith\_affine

lean/qtsl-aux/QtslAux/QFixBall.lean, line 113

```

theorem lipschitzWith_affine {E : Type*} [NormedAddCommGroup E]
  [NormedSpace  $\mathbb{R}$  E] (A :  $E \rightarrow$  L[ $\mathbb{R}$ ] E) (c : E) (hA :  $\|A\|_+ \leq K$ ) :
  LipschitzWith K fun x  $\Rightarrow$  A x + c

```

Qtsl.QDistTransport.pushKernel\_iff\_eps\_zero

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 183



```

theorem pushKernel_iff_eps_zero [DecidableEq β] {π : α → β}
  {K : α → α → ℝ} {Q : β → β → ℝ} :
  PushKernel π K Q ↔ EpsPushKernel π K Q 0

```

Qtsl.QDistTransport.**epsPushKernel\_comp**

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 168

```

theorem epsPushKernel_comp [DecidableEq β] [DecidableEq γ]
  {π₁ : α → β} {π₂ : β → γ} {K₁ : α → α → ℝ} {K₂ : β → β → ℝ}
  {K₃ : γ → γ → ℝ} {ε₁ ε₂ : ℝ}
  (h₁ : EpsPushKernel π₁ K₁ K₂ ε₁) (h₂ : EpsPushKernel π₂ K₂ K₃ ε₂) :
  EpsPushKernel (π₂ ∘ π₁) K₁ K₃ (ε₁ + ε₂)

```

Qtsl.QDistTransport.**sound\_QDistSelfTransport**

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 74

```

theorem sound_QDistSelfTransport [DecidableEq β] (π : α → β)
  {μ ν : α → ℝ} {ε : ℝ} (h : tvDist μ ν ≤ ε) :
  tvDist (push π μ) (push π ν) ≤ ε

```

Qtsl.QRelChain.**IsMarkov**

lean/qtsl-aux/QtslAux/QRelChain.lean, line 55

```

structure IsMarkov [Fintype α] (K : α → α → ℝ) : Prop where ...

```

Qtsl.QDistTransport.**sound\_QDistTransport\_eps**

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 152

```

theorem sound_QDistTransport_eps [DecidableEq β] {π : α → β}
  {K : α → α → ℝ} {Q : β → β → ℝ} {ε : ℝ} {μ₀ : α → ℝ}
  (hK : IsMarkov K) (hQ : IsMarkov Q) (hμ₀ : IsDist μ₀)
  (h : EpsPushKernel π K Q ε) (n : ℕ) :
  tvDist (push π (law K μ₀ n)) (law Q (push π μ₀) n) ≤ n * ε

```

Qtsl.QDistTransport.**tvDist\_push\_le**

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 60

```

theorem tvDist_push_le [DecidableEq β] (π : α → β) (p q : α → ℝ) :
  tvDist (push π p) (push π q) ≤ tvDist p q

```

Qtsl.QDistTransport.**tvDist\_step\_le**

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 86

```

theorem tvDist_step_le {K : α → α → ℝ} (hK : IsMarkov K) (p q : α → ℝ) :
  tvDist (step K p) (step K q) ≤ tvDist p q

```

Qtsl.QDistTransport.**EpsPushKernel**

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 103

```

def EpsPushKernel [DecidableEq β] (π : α → β) (K : α → α → ℝ)
  (Q : β → β → ℝ) (ε : ℝ) : Prop

```

Qtsl.QDistTransport.**push\_step\_defect**

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 107

```

theorem push_step_defect [DecidableEq β] {π : α → β} {K : α → α → ℝ}
  {Q : β → β → ℝ} {ε : ℝ} {μ : α → ℝ} (hμ : IsDist μ)
  (h : EpsPushKernel π K Q ε) :
  tvDist (push π (step K μ)) (step Q (push π μ)) ≤ ε

```

Qtsl.QDistTransport.**tvDist\_triangle**

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 40

```

theorem tvDist_triangle {L : Type*} [Fintype L] (p q r : L → ℝ) :
  tvDist p r ≤ tvDist p q + tvDist q r

```

SuFunctor.Universe.BelState.**push\_comp**

lean/su-functor/SuFunctor/Universe/TotalCitizenship.lean, line 89

```

theorem BelState.push_comp (f : V → U) (g : U → W) (x : BelState V) :
  BelState.push (g ∘ f) x = BelState.push g (BelState.push f x)

```

Qtsl.QDistTransport.**eq\_of\_tvDist\_le\_zero**

lean/qtsl-aux/QtslAux/QDistTransport.lean, line 48

```

theorem eq_of_tvDist_le_zero {L : Type*} [Fintype L] {p q : L → ℝ}
  (h : tvDist p q ≤ 0) : p = q

```

SuFunctor.Adequacy.**dtmcAbsorb\_mass\_le\_reach** lean/su-functor/SuFunctor/Adequacy/DtmcAbsorb.lean, line 484

```
theorem dtmcAbsorb_mass_le_reach (M : FinDTMC S) {C : Finset S}
  {T t' : ℕ} (h : T ≤ t') :
  massOn M t' C ≤ massOn M T C + massOn M T (M.reachSet C \ C)
```

SuFunctor.Adequacy.**dtmcAbsorbBracket\_reach\_sound** lean/su-functor/SuFunctor/Adequacy/DtmcAbsorb.lean, line 512

```
theorem dtmcAbsorbBracket_reach_sound (M : FinDTMC S)
  {C Abs : Finset S} (hAbs : ∀ s ∈ Abs, M.Absorbing s)
  (hCA : C ⊆ Abs) {T t' : ℕ} (h : T ≤ t') :
  massOn M T C ≤ massOn M t' C ∧
  massOn M t' C ≤ massOn M T C + massOn M T (M.reachSet C \ C)
```

SuFunctor.Adequacy.**dtmcAbsorb\_reach\_residual\_le** lean/su-functor/SuFunctor/Adequacy/DtmcAbsorb.lean, line 528

```
theorem dtmcAbsorb_reach_residual_le (M : FinDTMC S)
  {C Abs : Finset S} (hAbs : ∀ s ∈ Abs, M.Absorbing s) (T : ℕ) :
  massOn M T (M.reachSet C \ C) ≤ 1 - massOn M T Abs
```

SuFunctor.Adequacy.**dtmcAbsorb\_eventual\_bracket\_reach** lean/su-functor/SuFunctor/Adequacy/DtmcAbsorb.lean, line 550

```
theorem dtmcAbsorb_eventual_bracket_reach (M : FinDTMC S)
  {C Abs : Finset S} (hAbs : ∀ s ∈ Abs, M.Absorbing s)
  (hCA : C ⊆ Abs) {T : ℕ} {lo hi : ℚ}
  (hlo : lo ≤ massOn M T C)
  (hhi : massOn M T C + massOn M T (M.reachSet C \ C) ≤ hi) :
  ∀ t', T ≤ t' → lo ≤ massOn M t' C ∧ massOn M t' C ≤ hi
```

SuFunctor.Adequacy.FinDTMC.**ForwardClosed** lean/su-functor/SuFunctor/Adequacy/DtmcAbsorb.lean, line 229

```
def FinDTMC.ForwardClosed (M : FinDTMC S) (D : Finset S) : Prop
```

SuFunctor.Adequacy.**dtmc\_mass\_mono\_of\_forwardClosed** lean/su-functor/SuFunctor/Adequacy/DtmcAbsorb.lean, line 276

```
theorem dtmc_mass_mono_of_forwardClosed (M : FinDTMC S)
  {D : Finset S} (hD : M.ForwardClosed D) {T t' : ℕ} (h : T ≤ t') :
  massOn M T D ≤ massOn M t' D
```

SuFunctor.Adequacy.**idBool\_strict\_improvement** lean/su-functor/SuFunctor/Adequacy/DtmcAbsorb.lean, line 769

```
theorem idBool_strict_improvement {T t' : ℕ} (h : T ≤ t') :
  massOn idBool t' {true} = 0
```

SuFunctor.Adequacy.**dtmcPosteriorBracket\_sound** lean/su-functor/SuFunctor/Adequacy/DtmcAbsorb.lean, line 574

```
theorem dtmcPosteriorBracket_sound {ι : Type} [Fintype ι]
  (N Mv : ι → ℚ) (R : ℚ) (hR : 0 ≤ R)
  (hN : ∀ c, 0 ≤ N c)
  (hS : 0 < ∑ c', N c')
  (hle : ∀ c, N c ≤ Mv c)
  (hgap : (∑ c', (Mv c' - N c')) ≤ R) (c : ι) :
  N c / ((∑ c', N c') + R) ≤ Mv c / (∑ c', Mv c') ∧
  Mv c / (∑ c', Mv c') ≤ (N c + R) / ((∑ c', N c') + R)
```

SuFunctor.Adequacy.**massAt\_expect\_le\_of\_contraction\_of\_weighted** lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 164

```
theorem massAt_expect_le_of_contraction_of_weighted (M : FinDTMC S)
  (V : S → ℚ) {c : ℚ} (hc : 0 ≤ c)
  (hcontr : ∀ s, (∑ s', M.step s s' * V s') ≤ c * V s) (t : ℕ) :
  (∑ s, massAt M t s * V s) ≤ c ^ t * ∑ s, massAt M 0 s * V s
```

SuFunctor.Adequacy.qGenFormContraction\_subsumes\_genform lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 181

```
theorem qGenFormContraction_subsumes_genform [DecidableEq S]
  (M : FinDTMC S) (V : S → ℚ)
  (hg : ∀ s, genRow (compile M) V s ≤ 0) (t : ℕ) :
  expectQ (M.pathLaw t) (atLast V) ≤
  expectQ (M.pathLaw 0) (atLast V)
```

SuFunctor.Adequacy.wmassAt lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 63

```
def wmassAt (init : S → ℚ) (W : S → S → ℚ) : ℕ → S → ℚ
| 0 => init
| t + 1 => fun s' => ∑ s, wmassAt init W t s * W s s'

@[simp] theorem wmassAt_zero (init : S → ℚ) (W : S → S → ℚ) :
  wmassAt init W 0 = init
```

SuFunctor.Adequacy.massAt\_eq\_wmassAt lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 85

```
theorem massAt_eq_wmassAt (M : FinDTMC S) (t : ℕ) :
  massAt M t = wmassAt M.init M.step t
```

SuFunctor.Adequacy.wmassAt\_expect\_le\_of\_contraction lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 116

```
theorem wmassAt_expect_le_of_contraction {init : S → ℚ} {W : S → S → ℚ}
  (V : S → ℚ) (hinit : ∀ s, 0 ≤ init s) (hW : ∀ s s', 0 ≤ W s s')
  {c : ℚ} (hc : 0 ≤ c)
  (hcontr : ∀ s, (∑ s', W s s' * V s') ≤ c * V s) (t : ℕ) :
  (∑ s, wmassAt init W t s * V s) ≤ c ^ t * ∑ s, init s * V s
```

SuFunctor.Adequacy.massAt\_expect\_on\_le\_of\_regionContraction lean/su-functor/SuFunctor/Adequacy/DtmcAliveTail.lean, line 60

```
theorem massAt_expect_on_le_of_regionContraction (M : FinDTMC S)
  (A : Finset S) (V : S → ℚ) {c : ℚ} (hc : 0 ≤ c)
  (hclosed : ∀ x ∉ A, ∀ y ∈ A, M.step x y = 0)
  (hcontr : ∀ x ∈ A, (∑ y ∈ A, M.step x y * V y) ≤ c * V x) (t : ℕ) :
  (∑ x ∈ A, massAt M t x * V x) ≤ c ^ t * ∑ x ∈ A, M.init x * V x
```

SuFunctor.Adequacy.dtmcAliveTail\_of\_regionContraction lean/su-functor/SuFunctor/Adequacy/DtmcAliveTail.lean, line 93

```
theorem dtmcAliveTail_of_regionContraction (M : FinDTMC S)
  (A : Finset S) (V : S → ℚ) {c : ℚ} (hc : 0 ≤ c)
  (hfloor : ∀ x ∈ A, 1 ≤ V x)
  (hclosed : ∀ x ∉ A, ∀ y ∈ A, M.step x y = 0)
  (hcontr : ∀ x ∈ A, (∑ y ∈ A, M.step x y * V y) ≤ c * V x) (t : ℕ) :
  massOn M t A ≤ c ^ t * ∑ x ∈ A, M.init x * V x
```

SuFunctor.Adequacy.aliveTail\_needs\_floor lean/su-functor/SuFunctor/Adequacy/DtmcAliveTail.lean, line 128

```
theorem aliveTail_needs_floor :
  ¬ (massOn unitChain 1 Finset.univ ≤
    (1 : ℚ) ^ 1 * ∑ x ∈ Finset.univ, unitChain.init x * ((1 : ℚ) / 2))
```

SuFunctor.Adequacy.aliveTail\_needs\_no\_reentry lean/su-functor/SuFunctor/Adequacy/DtmcAliveTail.lean, line 159

```
theorem aliveTail_needs_no_reentry :
  ¬ (massOn reentryChain 1 {true} ≤
    (1 : ℚ) ^ 1 * ∑ x ∈ ({true} : Finset Bool),
    reentryChain.init x * (1 : ℚ))
```

SuFunctor.Adequacy.**genFormTemplate\_collapse** lean/su-functor/SuFunctor/Adequacy/DtmcTemplate.lean, line 35

```
theorem genFormTemplate_collapse {ι : Type} [Fintype ι]
  (w : ι → ℚ) (d : ι → ℤ) {M γ c : ℚ} (hM : 0 ≤ M) (hγ : 0 < γ)
  (h : (∑ i, w i * γ ^ d i) ≤ c) (n : ℤ) :
  (∑ i, w i * (M * γ ^ (n + d i))) ≤ c * (M * γ ^ n)
```

SuFunctor.Adequacy.**aliveStep\_factor** lean/su-functor/SuFunctor/Adequacy/DtmcProductProfile.lean, line 264

```
theorem aliveStep_factor {g : List ℚ} {p : ℚ} (hg : ∀ q ∈ g, 0 < q)
  (hp : 0 < p) (p : ℚ) {d : ℕ} (x : Fin d → ℕ)
  (hx : ∀ i, downMove x i ≠ 0) :
  aliveStep g p p x
  = (∑ i, transferRatio g p p (x i)) / (d : ℚ)
    * prodProfileV g p x
```

SuFunctor.Adequacy.**transferRatio\_of\_ge** lean/su-functor/SuFunctor/Adequacy/DtmcProductProfile.lean, line 180

```
theorem transferRatio_of_ge {g : List ℚ} {p : ℚ} (hg : ∀ q ∈ g, 0 < q)
  (hp : 0 < p) (p : ℚ) {v : ℕ} (hv : g.length + 1 ≤ v) :
  transferRatio g p p v = rInf p p
```

SuFunctor.Adequacy.**productProfile\_collapse** lean/su-functor/SuFunctor/Adequacy/DtmcProductProfile.lean, line 297

```
theorem productProfile_collapse {g : List ℚ} {p : ℚ}
  (hg : ∀ q ∈ g, 0 < q) (hp : 0 < p) (p : ℚ) {d : ℕ} {c : ℚ}
  (hclass : ∀ k ∈ Finset.Icc 1 d,
    ((d : ℚ) - (k : ℚ)) * rMax g p p + (k : ℚ) * rInf p p
    ≤ (d : ℚ) * c)
  (x : Fin d → ℕ) (j : Fin d) (hgeo : g.length + 1 ≤ x j)
  (h2 : 2 ≤ x j) :
  aliveStep g p p x ≤ c * prodProfileV g p x
```

SuFunctor.Adequacy.**rampProfile\_needs\_window\_reach** lean/su-functor/SuFunctor/Adequacy/DtmcProductProfile.lean, line 370

```
theorem rampProfile_needs_window_reach :
  (∀ k ∈ Finset.Icc 1 (N : ℕ) 3,
    ((3 : ℚ) - (k : ℚ)) * rMax rampG (7/4) (1/4)
    + (k : ℚ) * rInf (7/4) (1/4) ≤ (3 : ℚ) * (193/200)) ∧
  (3 : ℚ) * (193/200)
  < 2 * transferRatio rampG (7/4) (1/4) 1
    + transferRatio rampG (7/4) (1/4) 5
```

SuFunctor.Adequacy.**sum\_range\_mul\_pow\_mul\_eq** lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 279

```
theorem sum_range_mul_pow_mul_eq (c : ℚ) (n : ℕ) :
  (∑ i ∈ Finset.range n, (i : ℚ) * c ^ i) * (1 - c) ^ 2 =
  c - (n : ℚ) * c ^ n + ((n : ℚ) - 1) * c ^ (n + 1)
```

SuFunctor.Adequacy.**sum\_Ico\_pow\_le** lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 318

```
theorem sum_Ico_pow_le {c : ℚ} (hc : 0 ≤ c) (hc1 : c < 1) (T T' : ℕ) :
  (∑ t ∈ Finset.Ico T T', c ^ t) ≤ c ^ T / (1 - c)
```

SuFunctor.Adequacy.**sum\_Ico\_mul\_pow\_le** lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 335

```
theorem sum_Ico_mul_pow_le {c : ℚ} (hc : 0 ≤ c) (hc1 : c < 1)
  (T T' : ℕ) :
  (∑ t ∈ Finset.Ico T T', (t : ℚ) * c ^ t) ≤
  c ^ T * ((T : ℚ) * (1 - c) + c) / (1 - c) ^ 2
```

SuFunctor.Adequacy.**wmassAt\_exit\_tail\_le** lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 374

```

theorem wmassAt_exit_tail_le {init : S → ℚ} {W : S → S → ℚ}
  (V e : S → ℚ) (hinit : ∀ s, 0 ≤ init s) (hW : ∀ s s', 0 ≤ W s s')
  {c : ℚ} (hc : 0 ≤ c) (hc1 : c < 1)
  (hcontr : ∀ s, (∑ s', W s s' * V s') ≤ c * V s)
  {κ : ℚ} (hk : 0 ≤ κ) (hexit : ∀ s, e s ≤ κ * V s)
  (hIV : 0 ≤ ∑ s, init s * V s) (T T' : ℕ) :
  (∑ t ∈ Finset.Ico T T', ∑ s, wmassAt init W t s * e s) ≤
  κ * c ^ T * (∑ s, init s * V s) / (1 - c)

```

SuFunctor.Adequacy.wmassAt\_exit\_moment\_tail\_le lean/su-functor/SuFunctor/Adequacy/DtmcContraction.lean, line 403

```

theorem wmassAt_exit_moment_tail_le {init : S → ℚ} {W : S → S → ℚ}
  (V e : S → ℚ) (hinit : ∀ s, 0 ≤ init s) (hW : ∀ s s', 0 ≤ W s s')
  {c : ℚ} (hc : 0 ≤ c) (hc1 : c < 1)
  (hcontr : ∀ s, (∑ s', W s s' * V s') ≤ c * V s)
  {κ : ℚ} (hk : 0 ≤ κ) (hexit : ∀ s, e s ≤ κ * V s)
  (hIV : 0 ≤ ∑ s, init s * V s) (T T' : ℕ) :
  (∑ t ∈ Finset.Ico T T', (t : ℚ) * ∑ s, wmassAt init W t s * e s) ≤
  κ * (∑ s, init s * V s) * c ^ T * ((T : ℚ) * (1 - c) + c) /
  (1 - c) ^ 2

```

Qtsl.Acceptance.towerCitation\_sound\_of\_wf

lean/qtsl-aux/QtslAux/Acceptance.lean, line 56

```

theorem towerCitation_sound_of_wf
  (C : ι → Row σ J ι) (baseTrue : σ → Prop) (sem : J → Prop)
  (SideOK : ι → Prop)
  (hbase : ∀ i s, C i = Row.base s → baseTrue s)
  (hside : ∀ i, SideOK i)
  (hstep : ∀ i prems concl, C i = Row.ext prems concl → SideOK i →
    (∀ j ∈ prems, Row.holds baseTrue sem (C j)) → sem concl)
  (hwf : WellFounded (cites C)) :
  ∀ i, Row.holds baseTrue sem (C i)

```

Qtsl.Acceptance.towerCitation\_sound

lean/qtsl-aux/QtslAux/Acceptance.lean, line 74

```

theorem towerCitation_sound [Finite ι]
  (C : ι → Row σ J ι) (baseTrue : σ → Prop) (sem : J → Prop)
  (SideOK : ι → Prop)
  (hbase : ∀ i s, C i = Row.base s → baseTrue s)
  (hside : ∀ i, SideOK i)
  (hstep : ∀ i prems concl, C i = Row.ext prems concl → SideOK i →
    (∀ j ∈ prems, Row.holds baseTrue sem (C j)) → sem concl)
  (hacyc : Acyclic C) :
  ∀ i, Row.holds baseTrue sem (C i)

```

Qtsl.Acceptance.towerCitation\_root

lean/qtsl-aux/QtslAux/Acceptance.lean, line 86

```

theorem towerCitation_root [Finite ι]
  (C : ι → Row σ J ι) (baseTrue : σ → Prop) (sem : J → Prop)
  (SideOK : ι → Prop)
  (hbase : ∀ i s, C i = Row.base s → baseTrue s)
  (hside : ∀ i, SideOK i)
  (hstep : ∀ i prems concl, C i = Row.ext prems concl → SideOK i →
    (∀ j ∈ prems, Row.holds baseTrue sem (C j)) → sem concl)
  (hacyc : Acyclic C)
  {Jroot : J} (hroot : ∃ i prems, C i = Row.ext prems Jroot) :
  sem Jroot

```

Qtsl.Acceptance.Row.cited

lean/qtsl-aux/QtslAux/Acceptance.lean, line 36

```

def Row.cited : Row σ J ι → List ι
| .base _ => []
| .ext prems _ => prems

```

```

def Row.holds (baseTrue :  $\sigma \rightarrow \text{Prop}$ ) (sem :  $J \rightarrow \text{Prop}$ ) : Row  $\sigma$  J  $\iota \rightarrow \text{Prop}$ 
| .base s => baseTrue s
| .ext _ concl => sem concl

def cites (C :  $\iota \rightarrow \text{Row } \sigma \text{ J } \iota$ ) (j i :  $\iota$ ) : Prop

```

Qtsl.Acceptance.**acyclic\_wellFounded**

lean/qtsl-aux/QtslAux/Acceptance.lean, line 50

```

theorem acyclic_wellFounded [Finite  $\iota$ ] (C :  $\iota \rightarrow \text{Row } \sigma \text{ J } \iota$ )
(hacyc : Acyclic C) : WellFounded (cites C)

```

Qtsl.Acceptance.**citePos\_sound**

lean/qtsl-aux/QtslAux/Acceptance.lean, line 101

```

theorem citePos_sound {X : Type*} {R D : Set X} {P : Set X  $\rightarrow$  Prop}
(hmono : Monotone P) (hsub :  $R \subseteq D$ ) (hR : P R) : P D

```

Qtsl.Acceptance.**citeCmpl\_subset**

lean/qtsl-aux/QtslAux/Acceptance.lean, line 105

```

theorem citeCmpl_subset {X : Type*} {R D : Set X} (hsub :  $R \subseteq D$ ) :
 $D^c \subseteq R^c$ 

```

Qtsl.Acceptance.**citeCmpl\_sound**

lean/qtsl-aux/QtslAux/Acceptance.lean, line 109

```

theorem citeCmpl_sound {X : Type*} {R D : Set X} {P : Set X  $\rightarrow$  Prop}
(hanti : Antitone P) (hsub :  $R \subseteq D$ ) (hR : P  $R^c$ ) : P  $D^c$ 

```

Modules cited: QtslAux.QGuard, QtslAux.LayerD.MPDischarge, SuFunctor.Site.FellAcceptance, SuFunctor.Site.FellBehav, QtslAux.Enclosure.

Also cited in this chapter, listed under an earlier one: ProofObject.sound, SuFunctor.Plain.CheckerFace, semanticallyValid\_iff\_face.

## A.8 Partial Completeness

Citations from Chapter 16.

PPLC.Qtsl.**sound\_u10f**

lean/qtsl/Qtsl/TimedCore.lean, line 49

```

theorem sound_u10f (B : Behavior X) (T :  $\mathbb{R}_{\geq 0}$ ) ( $\nu \chi$  : Set X) (x : X)
(hbox : B.BoxInv  $\nu$  x) :
B.AService T ( $\chi \cap \nu$ ) x  $\leftrightarrow$  B.AService T  $\chi$  x

```

PPLC.Qtsl.**sound\_firstService**

lean/qtsl/Qtsl/TimedCore.lean, line 138

```

theorem sound_firstService (hB3 : B.FusionClosed) (hn :  $0 < n$ )
( $\alpha$  : Fin n  $\rightarrow$  Set X) (T : Fin n  $\rightarrow \mathbb{R}_{\geq 0}$ ) (x : X)
(hatt :  $\forall i$ , B.FirstVisits ( $\alpha$  i))
(hx :  $\forall i$ , B.AService (T i) ( $\alpha$  i) x) :
B.AService (Finset.univ.inf' (Finset.univ_nonempty_iff.mpr  $\langle \{0, hn\} \rangle$ )) T
(B.fsTarget  $\alpha$  T) x

```

PPLC.Qtsl.**sound\_fsResidual**

lean/qtsl/Qtsl/TimedCore.lean, line 149

```

theorem sound_fsResidual (hB3 : B.FusionClosed) (hn :  $0 < n$ )
( $\alpha$  : Fin n  $\rightarrow$  Set X) (T : Fin n  $\rightarrow \mathbb{R}_{\geq 0}$ ) (c :  $\mathbb{R}_{\geq 0}$ ) (x : X)
(hc :  $\forall i$ ,  $c < T$  i)
(hatt :  $\forall i$ , B.FirstVisits ( $\alpha$  i))
(hx :  $\forall i$ , B.AService (T i) ( $\alpha$  i) x) :
B.AService c (B.fsTarget  $\alpha$  T  $\cup$  B.residualCell  $\alpha$  T c) x

```

PPLC.Qtsl.TimedCoreCountermodel.**firstService\_attainment\_necessary**

lean/qtsl/Qtsl/TimedCore.lean, line 496

```

theorem firstService_attainment_necessary :
  ∃ (S : Type) (B : Behavior S) (α : Fin 2 → Set S) (x : S),
    B.FusionClosed ∧
    (∀ i, B.AService 2 (α i) x) ∧
    ¬ B.AService 2 (B.fsTarget α (fun _ => 2)) x

```



# References

---

- [1] *A Certificate-Based Approach to Formally Verified Approximations*. DOI: [10.4230/LIPIcs.ITP.2019.8](https://doi.org/10.4230/LIPIcs.ITP.2019.8). URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2019.8>.
- [2] M. Abadi and L. Lamport. “Composing Specifications”. In: *ACM Transactions on Programming Languages and Systems* 15.1 (1993), pp. 73–132.
- [3] Alessandro Abate, Daniel Contro, Mirco Giacobbe, Agustín Martínez-Suñé, and Diptarko Roy. *Value Functions as Supermartingale Certificates*. 2026. arXiv: [2605.31524](https://arxiv.org/abs/2605.31524).
- [4] Alessandro Abate, Alec Edwards, Mirco Giacobbe, Hashan Punchihewa, and Diptarko Roy. *Quantitative Verification with Neural Networks*. 2023. arXiv: [2301.06136](https://arxiv.org/abs/2301.06136).
- [5] Alessandro Abate, Mirco Giacobbe, Sergey Ichtchenko, and Diptarko Roy. *Complete  $\omega$ -Regular Supermartingale Certificates*. 2026. arXiv: [2605.21134](https://arxiv.org/abs/2605.21134).
- [6] Alessandro Abate, Mirco Giacobbe, and Diptarko Roy. *Stochastic Omega-Regular Verification and Control with Supermartingales*. 2024. arXiv: [2405.17304](https://arxiv.org/abs/2405.17304).
- [7] Marco Abbadini, Guram Bezhanishvili, and Luca Carai. *Vietoris endofunctor for closed relations and its de Vries dual*. 2023. arXiv: [2308.16823](https://arxiv.org/abs/2308.16823).
- [8] Mario Alvarez-Picallo, Dan R. Ghica, David Sprunger, and Fabio Zanasi. *Functorial String Diagrams for Reverse-Mode Automatic Differentiation*. 2021. arXiv: [2107.13433](https://arxiv.org/abs/2107.13433).
- [9] K. J. Åström and K. Furuta. “Swinging up a pendulum by energy control”. In: *Automatica* 36.2 (2000), pp. 287–295.
- [10] Karl Johan Åström. “Optimal Control of Markov Processes with Incomplete State Information”. In: *Journal of Mathematical Analysis and Applications* 10.1 (1965), pp. 174–205. DOI: [10.1016/0022-247X\(65\)90154-X](https://doi.org/10.1016/0022-247X(65)90154-X).
- [11] T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Křetínský, et al. “The Hanoi omega-automata format”. In: *Computer Aided Verification (CAV 2015), Part I*. Vol. 9206. Lecture Notes in Computer Science. (The  $\text{Fin}(\cdot)/\text{Inf}(\cdot)$  acceptance atoms as an interchange standard.) Springer, 2015, pp. 479–486.
- [12] John C. Baez and Jade Master. *Open Petri Nets*. 2018. arXiv: [1808.05415](https://arxiv.org/abs/1808.05415).
- [13] Christel Baier, Luca de Alfaro, Vojtěch Forejt, and Marta Kwiatkowska. “Model Checking Probabilistic Systems”. In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. Springer, 2018, pp. 963–999. DOI: [10.1007/978-3-319-10575-8\\_28](https://doi.org/10.1007/978-3-319-10575-8_28).
- [14] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.
- [15] S. Banach. “Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales”. In: *Fundamenta Mathematicae* 3 (1922), pp. 133–181.
- [16] Jon Bentley. *Programming Pearls*. 2nd ed. Addison-Wesley, 2000.
- [17] V. van den Berk. “Wafer fab performance analysis”. Master’s thesis. Eindhoven University of Technology, 2004.
- [18] S. N. Bernstein. “Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités”. In: *Communications de la Société Mathématique de Kharkov*. 2nd ser. 13 (1912), pp. 1–2.
- [19] Raven Beutner, C.-H. Luke Ong, and Fabian Zaiser. *Guaranteed Bounds for Posterior Inference in Universal Probabilistic Programming*. 2022. arXiv: [2204.02948](https://arxiv.org/abs/2204.02948).
- [20] Martin E. Bidlingmaier. *Algebraic Semantics of Datalog with Equality*. 2023. arXiv: [2302.03167](https://arxiv.org/abs/2302.03167).
- [21] G. Birkhoff. *Lattice Theory*. 3rd ed. American Mathematical Society Colloquium Publications 25. ( $M_3$ , the diamond, as the canonical modular non-distributive lattice.) 1967.
- [22] G. Birkhoff. “Rings of sets”. In: *Duke Mathematical Journal* 3.3 (1937). (The representation theorem for finite distributive lattices via join-irreducibles.) pp. 443–454.
- [23] F. Blanchini. “Set invariance in control”. In: *Automatica* 35.11 (1999), pp. 1747–1767.

- [24] E. G. Boman, D. Chen, O. Parekh, and S. Toledo. “On factor width and symmetric H-matrices”. In: *Linear Algebra and its Applications* 405 (2005). (Factor-width-2 = scaled diagonal dominance — the class the /DD tier’s congruence certificates inhabit.), pp. 239–248.
- [25] Filippo Bonchi and Cipriano Junior Cioffo. *Completeness for Probabilistic Boolean Tapes*. 2026. arXiv: [2606.19017](#).
- [26] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. *String Diagram Rewrite Theory I. Rewriting with Frobenius Structure*. 2020. arXiv: [2012.01847](#).
- [27] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. *String Diagram Rewrite Theory II. Rewriting with Symmetric Monoidal Structure*. 2022. arXiv: [2104.14686](#).
- [28] Marvin Brieger, Stefan Mitsch, and André Platzer. *Complete Dynamic Logic of Communicating Hybrid Programs*. 2024. arXiv: [2408.05012](#).
- [29] Romeo Brunetti, Klaus Fredenhagen, and Rainer Verch. *The generally covariant locality principle. A new paradigm for local quantum physics*. 2001. DOI: [10.1007/s00220-003-0815-7](#). arXiv: [math-ph/0112041](#).
- [30] Carlos E. Budde, Arnd Hartmanns, Tobias Meggendorfer, Maximilian Weininger, and Patrick Wienhöft. *Sound Statistical Model Checking for Probabilities and Expected Rewards (extended version)*. 2024. arXiv: [2411.00559](#).
- [31] Mihai Budiu, Frank McSherry, Leonid Ryzhyk, and Val Tannen. *DBSP: Automatic Incremental View Maintenance for Rich Query Languages*. 2022. arXiv: [2203.16684](#).
- [32] Manuela L. Bujorianu, John Lygeros, and Marius C. Bujorianu. *Towards a General Theory of Stochastic Hybrid Systems*. CTIT Technical Report Series, No. TR-CTIT-08-25, Centre for Telematics and Information Technology, University of Twente, Enschede. Mar. 12, 2008. URL: <https://ris.utwente.nl/ws/portalfiles/portal/5100731/journal8.pdf>.
- [33] R. Bunel, J. Lu, I. Turkaslan, P. H. S. Torr, P. Kohli, and M. P. Kumar. “Branch and bound for piecewise linear neural network verification”. In: *Journal of Machine Learning Research* 21 (2020).
- [34] C. P. Burgess. *Introduction to Effective Field Theory*. 2007. DOI: [10.1146/annurev.nucl.56.080805.140508](#). arXiv: [hep-th/0701053](#).
- [35] Zhipeng Cao, Peixin Wang, Luke Ong, Đorđe Žikelić, Dominik Wagner, and Bai Xue. *Comparative Analysis of Barrier-like Function Methods for Reach-Avoid Verification in Stochastic Discrete-Time Systems*. 2025. arXiv: [2512.05348](#).
- [36] Rocco Caprio, Adrien Corenflos, and Sam Power. *Wasserstein Contraction of Coordinate Ascent Variational Inference*. 2026. arXiv: [2605.30253](#).
- [37] M. Capucci and D. J. Myers. *Certified lenses and compositional system certification*. (The wiring-square conditions, the  $\mathcal{K}_\infty^0$  slack discipline, and the compose-by-lens-composition-and-slack-addition law cited in the composition sections.) 2026. arXiv: [2604.03017](#).
- [38] Matteo Capucci and David Jaz Myers. *Compositionality of Lyapunov functions via assume-guarantee reasoning*. 2026. arXiv: [2604.03017](#).
- [39] G. T. Cargo and O. Shisha. “The Bernstein form of a polynomial”. In: *Journal of Research of the National Bureau of Standards* 70B (1966). (Bernstein coefficients enclose the range on the simplex — the sound direction this calculus’s kernel uses.), pp. 79–81.
- [40] P. de Casteljau. *Outillages méthodes calcul*. Technical report, A. Citroën, Paris. (The founding subdivision/evaluation algorithm for polynomials in Bernstein form.) 1959.
- [41] A. Chakarov and S. Sankaranarayanan. “Probabilistic program analysis with martingales”. In: *Computer Aided Verification (CAV 2013)*. Vol. 8044. Lecture Notes in Computer Science. (Ranking supermartingales.) Springer, 2013, pp. 511–526.
- [42] *Chebyshev Interpolation Polynomial-based Tools for Rigorous Computing*. URL: <https://ens-lyon.hal.science/ensl-00472509v2>.
- [43] Leihao Chen, Tobias Fritz, Tomáš Gonda, Andreas Klingler, and Antonio Lorenzin. *The Aldous–Hoover Theorem in Categorical Probability*. (Plates over Kolmogorov products: the unique morphism whose finite marginals are the body’s instances.) arXiv: [2411.12840](#).
- [44] Rance Cleaveland, A. W. Roscoe, and Scott A. Smolka. “Process Algebra and Model Checking”. In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. Springer, 2018, pp. 1149–1195. DOI: [10.1007/978-3-319-10575-8\\_32](#).
- [45] E. F. Codd. “Missing information (applicable and inapplicable) in relational databases”. In: *ACM SIGMOD Record* 15.4 (1986). DOI: [10.1145/16301.16303](#).
- [46] Earl A. Coddington and Norman Levinson. *Theory of Ordinary Differential Equations*. New York: McGraw-Hill, 1955.

## REFERENCES

- [47] C. Conley. *Isolated Invariant Sets and the Morse Index*. CBMS Regional Conference Series in Mathematics 38. American Mathematical Society, 1978.
- [48] C. Conley and R. Easton. “Isolated invariant sets and isolating blocks”. In: *Transactions of the American Mathematical Society* 158 (1971), pp. 35–61.
- [49] Gert de Cooman and Filip Hermans. *Imprecise probability trees. Bridging two theories of imprecise probability*. 2008. arXiv: [0801.1196](#).
- [50] Fábio Gagliardi Cozman. “Credal Networks”. In: *Artificial Intelligence* 120.2 (2000), pp. 199–233. doi: [10.1016/S0004-3702\(00\)00029-1](#).
- [51] Gavin E. Crooks. *Field Guide to Continuous Probability Distributions*. Feb. 15, 2022. URL: <https://threeplusone.com/pubs/FieldGuide.pdf>.
- [52] Geoffrey Cruttwell, Michael Lambert, Dorette Pronk, and Martin Szyld. *Double Fibrations*. 2022. arXiv: [2205.15240](#).
- [53] Keri D’Angelo and Sophie Libkind. *Dependent Directed Wiring Diagrams for Composing Instantaneous Systems*. 2025. arXiv: [2503.05457](#).
- [54] David “davidad” Dalrymple. *Imprecise beliefs. A tiny introduction*. Post on LessWrong, 2026–07–29. 2026. URL: <https://www.lesswrong.com/posts/e7Pd4Q9TF7jFdmPgZ/imprecise-beliefs-a-tiny-introduction> (visited on 07/31/2026).
- [55] David “davidad” Dalrymple. *Safeguarded AI. Constructing guaranteed safety (programme thesis)*. Programme thesis. The Advanced Research and Invention Agency (ARIA).
- [56] David “davidad” Dalrymple. *Certifying strict positive definiteness via an approximate triangular inverse*. Post, X (formerly Twitter), status 2022316806094913669. (The triangular-inverse method: sign-and-pattern checks on  $A \cdot Z$  for an untrusted triangular witness  $Z$ ; no divisions, witness accuracy never essential.) 2026.
- [57] Swaraj Dash, Younesse Kaddar, Hugo Paquet, and Sam Staton. *Affine Monads and Lazy Structures for Bayesian Programming*. 2022. arXiv: [2212.07250](#).
- [58] Swaraj Dash and Sam Staton. *Monads for Measurable Queries in Probabilistic Databases*. 2021. arXiv: [2112.14048](#).
- [59] S. Dashkovskiy, B. S. Rüffer, and F. R. Wirth. “An ISS small gain theorem for general networks”. In: *Mathematics of Control, Signals, and Systems* 19.2 (2007). (The  $n$ -component ISS small-gain theorem over arbitrary interconnection graphs whose certificate form the SMALLGAINN section catalogues; its monotone-operator fixpoint machinery lives producer-side as existence theory for the  $(\lambda, B, \theta)$  side data.), pp. 93–122. doi: [10.1007/s00498-007-0014-8](#).
- [60] M. H. A. Davis. *Markov Models and Optimization*. Monographs on Statistics and Applied Probability 49. London: Chapman & Hall, 1993.
- [61] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. “Metrics for labelled Markov processes”. In: *Theoretical Computer Science* 318.3 (2004). (Behavioral pseudometrics for probabilistic systems.), pp. 323–354.
- [62] Alin Deutsch, Alan Nash, and Jeff Remmel. “The Chase Revisited”. In: *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS ’08)*. 2008, pp. 149–158. doi: [10.1145/1376916.1376938](#).
- [63] W. Doeblin. “Exposé de la théorie des chaînes simples constantes de Markov à un nombre fini d’états”. In: *Revue Mathématique de l’Union Interbalkanique* 2 (1938). (The origin of the coupling method for Markov chains.), pp. 77–105.
- [64] J. L. Doob. *Stochastic Processes*. (Systematic supermartingale theory; maximal inequalities and optional sampling.) New York: Wiley, 1953.
- [65] R. Durrett. *Probability: Theory and Examples*. 5th ed. (Borel–Cantelli.) Cambridge University Press, 2019.
- [66] R. Ehlers. “Formal verification of piece-wise linear feed-forward neural networks”. In: *Proceedings of ATVA 2017 (Automated Technology for Verification and Analysis)*. (The “triangle” linear relaxation of ReLU.) 2017.
- [67] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. “On Probabilistic Automata in Continuous Time”. In: *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS 2010)*. IEEE Computer Society, 2010, pp. 342–351.
- [68] E. A. Emerson and C.-L. Lei. “Modalities for model checking: branching time logic strikes back”. In: *Science of Computer Programming* 8.3 (1987). (Acceptance as positive boolean combinations of finitely-often/infinitely-often atoms — Emerson–Lei conditions.), pp. 275–306.

## REFERENCES

- [69] Stewart N. Ethier and Thomas G. Kurtz. *Markov Processes: Characterization and Convergence*. Wiley Series in Probability and Mathematical Statistics. New York: Wiley, 1986.
- [70] Lawrence C. Evans. *Partial Differential Equations*. 2nd ed. Vol. 19. Graduate Studies in Mathematics. Providence: American Mathematical Society, 2010.
- [71] Mariken H. C. Everdij and Henk A. P. Blom. “Hybrid state Petri nets which have the analysis power of stochastic hybrid systems and the formal verification power of automata”. In: *Petri Nets*. Ed. by Pawel Pawlewski. InTech, 2010. Chap. 12, pp. 227–252. arXiv: [1003.4812](#).
- [72] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. “Data Exchange. Semantics and Query Answering”. In: *Theoretical Computer Science* 336.1 (2005), pp. 89–124. DOI: [10.1016/j.tcs.2004.10.033](#).
- [73] J. Farkas. “Theorie der einfachen Ungleichungen”. In: *Journal für die reine und angewandte Mathematik* 124 (1902), pp. 1–27.
- [74] R. T. Farouki. “The Bernstein polynomial basis: a centennial retrospective”. In: *Computer Aided Geometric Design* 29.6 (2012), pp. 379–419.
- [75] J. M. G. Fell. “A Hausdorff topology for the closed subsets of a locally compact non-Hausdorff space”. In: *Proceedings of the American Mathematical Society* 13.3 (1962), pp. 472–476. DOI: [10.1090/S0002-9939-1962-0139135-6](#).
- [76] Christopher J. Fewster and Rainer Verch. *Algebraic quantum field theory in curved spacetimes*. 2015. arXiv: [1504.00586](#).
- [77] Christopher J. Fewster and Rainer Verch. *Dynamical locality and covariance. What makes a physical theory the same in all spacetimes?* 2011. DOI: [10.1007/s00023-012-0165-0](#). arXiv: [1106.4785](#).
- [78] Damir Filipović and Martin Larsson. “Polynomial diffusions and applications in finance”. In: *Finance and Stochastics* 20.4 (2016), pp. 931–972.
- [79] Hans Föllmer and Alexander Schied. “Convex Measures of Risk and Trading Constraints”. In: *Finance and Stochastics* 6.4 (2002), pp. 429–447. DOI: [10.1007/s007800200072](#).
- [80] Brendan Fong. “Decorated Cospans”. In: *Theory and Applications of Categories* 30.33 (2015), pp. 1096–1120. arXiv: [1502.00872](#).
- [81] F. G. Foster. “On the stochastic matrices associated with certain queuing processes”. In: *Annals of Mathematical Statistics* 24.3 (1953). (The additive-drift criterion.), pp. 355–360.
- [82] John W. Fowler and Jennifer Robinson. *Measurement and Improvement of Manufacturing Capacity (MIMAC). Final Report*. Technology Transfer Report 95062861A-XFR. Austin, TX: SEMATECH, 1995.
- [83] David H. Fremlin. *Measure Theory. Volume 4: Topological Measure Spaces*. Colchester: Torres Fremlin, 2006.
- [84] Tobias Fritz and Wendong Liang. *Free gs-monoidal categories and free Markov categories*. 2022. DOI: [10.1007/s10485-023-09717-0](#). arXiv: [2204.02284](#).
- [85] J. Garloff. “Convergent bounds for the range of multivariate polynomials”. In: *Interval Mathematics 1985*. Ed. by K. Nickel. Vol. 212. Lecture Notes in Computer Science. (Subdivision sharpens Bernstein enclosures — the completeness partner of the enclosure soundness used here.) Springer, 1986, pp. 37–56.
- [86] Murray Gell-Mann and James B. Hartle. *Classical Equations for Quantum Systems*. 1992. DOI: [10.1103/PhysRevD.47.3345](#). arXiv: [gr-qc/9210010](#).
- [87] G. Gentzen. “Untersuchungen über das logische Schließen I, II”. In: *Mathematische Zeitschrift* 39 (1935). English translation in M. E. Szabo (ed.), *The Collected Papers of Gerhard Gentzen*, North-Holland, 1969., pp. 176–210, 405–431.
- [88] S. Gerschgorin. “Über die Abgrenzung der Eigenwerte einer Matrix”. In: *Bulletin de l'Académie des Sciences de l'URSS, Classe des sciences mathématiques et naturelles* 6 (1931), pp. 749–754.
- [89] Dan R. Ghica, Koko Muroya, and Todd Waugh Ambridge. “A Robust Graph-Based Approach to Observational Equivalence”. In: *Logical Methods in Computer Science* 21.2 (2025). DOI: [10.46298/lmcs-21\(2:8\)2025](#). arXiv: [1907.01257](#).
- [90] Mirco Giacobbe, Daniel Kroening, Abhinandan Pal, and Michael Tautschnig. *Let a Neural Network Be Your Invariant*. Advances in Neural Information Processing Systems 38 (NeurIPS 2025), pp. 74713–74740. 2025. URL: [https://pure-oai.bham.ac.uk/ws/portalfiles/portal/281309250/Let\\_a\\_Neural\\_Network\\_Be\\_Your\\_Invariant.pdf](https://pure-oai.bham.ac.uk/ws/portalfiles/portal/281309250/Let_a_Neural_Network_Be_Your_Invariant.pdf).
- [91] Mirco Giacobbe, Daniel Kroening, Abhinandan Pal, and Michael Tautschnig. *Neural Model Checking*. 2024. arXiv: [2410.23790](#).



## REFERENCES

- [92] Itzhak Gilboa and David Schmeidler. “Updating Ambiguous Beliefs”. In: *Journal of Economic Theory* 59.1 (1993), pp. 33–49. doi: [10.1006/jeth.1993.1003](https://doi.org/10.1006/jeth.1993.1003).
- [93] A. Girard and G. J. Pappas. “Approximation metrics for discrete and continuous systems”. In: *IEEE Transactions on Automatic Control* 52.5 (2007). (Approximate simulation with quantitative degradation — the analytic ancestor of the  $\varepsilon$ -witness transport cell.), pp. 782–798.
- [94] J.-Y. Girard. “Linear logic”. In: *Theoretical Computer Science* 50.1 (1987). (Source of the additive/multiplicative vocabulary for context-sharing vs. context-splitting rule formats.), pp. 1–101.
- [95] Michèle Giry. “A Categorical Approach to Probability Theory”. In: *Categorical aspects of topology and analysis*. Vol. 915. Lecture Notes in Mathematics. Springer, 1982. doi: [10.1007/bfb0092872](https://doi.org/10.1007/bfb0092872).
- [96] Jean Goubault-Larrecq. “Continuous Previsions”. In: *Computer Science Logic (CSL 2007)*. Vol. 4646. Lecture Notes in Computer Science. Springer, 2007, pp. 542–557. doi: [10.1007/978-3-540-74915-8\\_40](https://doi.org/10.1007/978-3-540-74915-8_40).
- [97] Jean Goubault-Larrecq. *Just Previsions*. 2026. arXiv: [2605.10173](https://arxiv.org/abs/2605.10173).
- [98] Jean Goubault-Larrecq. “Prevision Domains and Convex Powercones”. In: *Foundations of Software Science and Computational Structures (FoSSaCS 2008)*. Vol. 4962. Lecture Notes in Computer Science. Springer, 2008, pp. 318–333. doi: [10.1007/978-3-540-78499-9\\_23](https://doi.org/10.1007/978-3-540-78499-9_23).
- [99] A. Granas and J. Dugundji. *Fixed Point Theory*. Springer Monographs in Mathematics. (The closed-ball condition  $d(Tx_0, x_0) \leq (1 - \rho)r$  and the a-priori bound  $d(x_0, x^*) \leq \delta/(1 - \rho)$ ; §I.1.) Springer, 2003.
- [100] Marco Grandis and Robert Paré. “Adjoint for double categories”. In: *Cahiers de Topologie et Géométrie Différentielle Catégoriques* 45.3 (2004), pp. 193–240.
- [101] Marco Grandis and Robert Paré. “Limits in double categories”. In: *Cahiers de Topologie et Géométrie Différentielle Catégoriques* 40.3 (1999), pp. 162–220.
- [102] Dennis Guck, Hassan Hatefi, Holger Hermanns, Joost-Pieter Katoen, and Mark Timmer. *Analysis of Timed and Long-Run Objectives for Markov Automata*. 2014. arXiv: [1407.7356](https://arxiv.org/abs/1407.7356).
- [103] Rudolf Haag and Daniel Kastler. “An Algebraic Approach to Quantum Field Theory”. In: *Journal of Mathematical Physics* 5.7 (1964), pp. 848–861. doi: [10.1063/1.1704187](https://doi.org/10.1063/1.1704187).
- [104] Amar Hadzahasanovic. *Combinatorics of higher-categorical diagrams*. 2024. arXiv: [2404.07273](https://arxiv.org/abs/2404.07273).
- [105] Amar Hadzahasanovic and Alex Kavvos. *alifib: a programming language and proof assistant founded on higher-dimensional rewriting*. 2026. URL: <https://github.com/ahadziha/alifib>.
- [106] Amar Hadzahasanovic and Diana Kessler. *Higher-dimensional subdiagram matching*. 2023. arXiv: [2304.09216](https://arxiv.org/abs/2304.09216).
- [107] Jack K. Hale and Sjoerd M. Verduyn Lunel. *Introduction to Functional Differential Equations*. Vol. 99. Applied Mathematical Sciences. New York: Springer, 1993.
- [108] Joseph Y. Halpern and Samantha Leung. *Weighted Sets of Probabilities and Minimax Weighted Expected Regret. New Approaches for Representing Uncertainty and Making Decisions*. 2013. arXiv: [1302.5681](https://arxiv.org/abs/1302.5681).
- [109] Xiaoying Han and Peter E. Kloeden. *Random Ordinary Differential Equations and Their Numerical Solution*. Vol. 85. Probability Theory and Stochastic Modelling. Singapore: Springer, 2017. doi: [10.1007/978-981-10-6265-0](https://doi.org/10.1007/978-981-10-6265-0).
- [110] D. Handelman. “Representing polynomials by positive linear functions on compact convex polyhedra”. In: *Pacific Journal of Mathematics* 132.1 (1988), pp. 35–62.
- [111] Philip Hartman. *Ordinary Differential Equations*. 2nd ed. Vol. 38. Classics in Applied Mathematics. Philadelphia: Society for Industrial and Applied Mathematics, 2002.
- [112] Arnd Hartmanns and Benjamin Lucien Kaminski. *Optimistic Value Iteration*. 2019. arXiv: [1910.01100](https://arxiv.org/abs/1910.01100).
- [113] S. W. Hawking, A. R. King, and P. J. McCarthy. “A new topology for curved space-time which incorporates the causal, differential, and conformal structures”. In: *Journal of Mathematical Physics* 17 (1976), pp. 174–181. doi: [10.1063/1.522874](https://doi.org/10.1063/1.522874).
- [114] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. “The probabilistic model checker Storm”. In: *International Journal on Software Tools for Technology Transfer* 24.4 (2022), pp. 589–610. doi: [10.1007/s10009-021-00633-z](https://doi.org/10.1007/s10009-021-00633-z).
- [115] R. A. Horn and C. R. Johnson. *Matrix Analysis*. 2nd ed. (Sylvester’s criterion, congruence, and Gershgorin discs, as cited in the certificate proofs.) Cambridge University Press, 2013.
- [116] J. Hsu. “Probabilistic Couplings for Probabilistic Reasoning”. (Couplings as certificates in relational program logics — the pre/post-style sibling of the judgment form used here.) PhD thesis. University of Pennsylvania, 2017. arXiv: [1710.09951](https://arxiv.org/abs/1710.09951).

## REFERENCES

- [117] Mathieu Huot, Alexander K. Lew, Vikash K. Mansinghka, and Sam Staton.  *$\omega$ PAP Spaces. Reasoning Denotationally About Higher-Order, Recursive Probabilistic and Differentiable Programs*. 2023. arXiv: [2302.10636](#).
- [118] Mathieu Huot, Alexander K. Lew, Vikash K. Mansinghka, and Sam Staton.  *$\omega$ PAP Spaces. Reasoning Denotationally About Higher-Order, Recursive Probabilistic and Differentiable Programs*. 2023. arXiv: [2302.10636](#).
- [119] Leonid Hurwicz. *A Class of Criteria for Decision-Making under Ignorance*. Cowles Commission Discussion Paper: Statistics 356. Cowles Commission for Research in Economics, 1951.
- [120] IEEE. *IEEE Standard for Floating-Point Arithmetic*. IEEE Std 754–2019. Revision of IEEE Std 754–2008. Institute of Electrical and Electronics Engineers, 2019.
- [121] *Independent products in infinite spaces (Miranda & Zaffalon, JMAA 2015)*. URL: <https://people.idsia.ch/~zaffalon/papers/2015jmaa-indprods.pdf>.
- [122] G. Irving. *aks*. Lean formalization repository: the Random library’s diagonal-dominance PSD bridge and a certified 65536-node spectral expander. 2026. URL: <https://github.com/girving/aks>.
- [123] G. Irving. *Answer to: Certificate for strict positive definiteness of a sparse matrix*. Computational Science Stack Exchange (question 45365, answer 45400). (The rounding-tolerant integer variant and its certificate-size analysis.) 2026.
- [124] Bart Jacobs. *Categorical Logic and Type Theory*. Vol. 141. Studies in Logic and the Foundations of Mathematics. North-Holland, 1999.
- [125] Z.-P. Jiang, I. M. Y. Mareels, and Y. Wang. “A Lyapunov formulation of the nonlinear small-gain theorem for interconnected ISS systems”. In: *Automatica* 32.8 (1996), pp. 1211–1215.
- [126] Z.-P. Jiang, A. R. Teel, and L. Praly. “Small-gain theorem for ISS systems and applications”. In: *Mathematics of Control, Signals and Systems* 7 (1994), pp. 95–120.
- [127] Peter T. Johnstone. *Sketches of an Elephant. A Topos Theory Compendium*. Geometric logic: Part D1. Oxford: Oxford University Press, 2002.
- [128] Aditi Kabra, Jonathan Laurent, Ruben Martins, Stefan Mitsch, and André Platzer. *LLM-Powered Automatic Theorem Proving and Synthesis for Hybrid Systems and Games*. 2026. arXiv: [2603.00737](#).
- [129] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. “Planning and Acting in Partially Observable Stochastic Domains”. In: *Artificial Intelligence* 101.1–2 (1998), pp. 99–134. DOI: [10.1016/S0004-3702\(98\)00023-X](#).
- [130] W. D. Kalies, K. Mischaikow, and R. C. A. M. van der Vorst. “An algorithmic approach to chain recurrence”. In: *Foundations of Computational Mathematics* 5.4 (2005), pp. 409–449.
- [131] W. D. Kalies, K. Mischaikow, and R. C. A. M. van der Vorst. “Lattice structures for attractors II”. In: *Foundations of Computational Mathematics* 16 (2016), pp. 1151–1191.
- [132] W. D. Kalies, K. Mischaikow, and R. C. A. M. van der Vorst. “Lattice structures for attractors III”. In: *Journal of Dynamics and Differential Equations* 34 (2022), pp. 1729–1768. arXiv: [1911.09382](#).
- [133] W. D. Kalies and T. Wehbe. *Compositionality of Global Dynamics in Product and Skew-Product Systems*. 2026. arXiv: [2606.30925](#).
- [134] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. “Reluplex: an efficient SMT solver for verifying deep neural networks”. In: *Proceedings of CAV 2017 (Computer Aided Verification)*. 2017.
- [135] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Princeton: Van Nostrand, 1960.
- [136] Hassan K. Khalil. *Nonlinear Systems*. 3rd ed. Prentice Hall, 2002.
- [137] Aleks Kissinger. *Abstract Tensor Systems as Monoidal Categories*. 2013. arXiv: [1308.3586](#).
- [138] Martin Kleppmann. *Making CRDTs Byzantine Fault Tolerant*. 9th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC’22), Rennes, France. ACM. doi:10.1145/3517209.3524042. 2022. URL: <https://martin.kleppmann.com/papers/bft-crdt-papoc22.pdf>.
- [139] Martin Kleppmann. *Thinking in Events. From Databases to Distributed Collaboration Software*. Keynote, 15th ACM International Conference on Distributed and Event-Based Systems (DEBS ’21), Virtual Event, Italy. ACM. doi:10.1145/3465480.3467835. 2021. URL: <https://martin.kleppmann.com/papers/debs21-keynote.pdf>.
- [140] Joachim Kock. *Whole-grain Petri nets and processes*. 2020. arXiv: [2005.05108](#).
- [141] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.
- [142] Denny Kopp, Michael Hassoun, Adar Kalir, and Lars Mönch. “SMT2020—A Semiconductor Manufacturing Testbed”. In: *IEEE Transactions on Semiconductor Manufacturing* 33.4 (2020), pp. 522–531. DOI: [10.1109/TSM.2020.3001933](#).

## REFERENCES

- [143] Vanessa Kosoy and Alex Appel. *Infra-Bayesianism*. Sequence on the AI Alignment Forum; Appel posts under the handle “Diffraction”. 2020. URL: <https://www.alignmentforum.org/s/CmrW8fCmSLK7E25sa> (visited on 07/21/2026).
- [144] J.-L. Krivine. “Anneaux préordonnés”. In: *Journal d’Analyse Mathématique* 12 (1964), pp. 307–326.
- [145] P. R. Kumar and Thomas I. Seidman. “Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems”. In: *IEEE Transactions on Automatic Control* 35.3 (1990), pp. 289–298. doi: [10.1109/9.50339](#).
- [146] Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. “CakeML: A Verified Implementation of ML”. In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL ’14)*. 2014, pp. 179–191. doi: [10.1145/2535838.2535841](#).
- [147] H. J. Kushner. *Stochastic Stability and Control*. Mathematics in Science and Engineering 33. (Supermartingale safety certificates for stochastic systems.) New York: Academic Press, 1967.
- [148] Marta Kwiatkowska, Gethin Norman, and David Parker. “PRISM 4.0: Verification of Probabilistic Real-Time Systems”. In: *Computer Aided Verification (CAV)*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 585–591. doi: [10.1007/978-3-642-22110-1\\_47](#).
- [149] Michael Lambert. *Discrete Double Fibrations*. 2021. arXiv: [2101.06734](#).
- [150] L. H. Landweber. “Decision problems for  $\omega$ -automata”. In: *Mathematical Systems Theory* 3.4 (1969). (Deterministic Büchi automata recognize exactly the limit languages.), pp. 376–384.
- [151] J. M. Lane and R. F. Riesenfeld. “A theoretical development for the computer generation and display of piecewise polynomial surfaces”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2.1 (1980), pp. 35–46.
- [152] Jonathan Laurent and André Platzer. *Learning to Find Proofs and Theorems by Learning to Refine Search Strategies. The Case of Loop Invariant Synthesis*. 2022. arXiv: [2205.14229](#).
- [153] Jonathan Laurent and André Platzer. *Oracular Programming. A Modular Foundation for Building LLM-Enabled Software*. 2025. arXiv: [2502.05310](#).
- [154] Elena Di Lovere, Giovanni de Felice, and Mario Román. *Monoidal Streams for Dataflow Programming*. 2022. arXiv: [2202.02061](#).
- [155] F. William Lawvere. “Adjointness in Foundations”. In: *Dialectica* 23 (1969), pp. 281–296.
- [156] F. William Lawvere. “Equality in Hyperdoctrines and Comprehension Schema as an Adjoint Functor”. In: *Applications of Categorical Algebra*. Vol. 17. Proceedings of Symposia in Pure Mathematics. American Mathematical Society, 1970, pp. 1–14.
- [157] F. William Lawvere. “Metric Spaces, Generalized Logic, and Closed Categories”. In: *Rendiconti del Seminario Matematico e Fisico di Milano* 43 (1973). Reprinted in *Reprints in Theory and Applications of Categories* 1 (2002), 1–37, pp. 135–166.
- [158] Kwanwoo Lee, Siyong Song, Daesoon Chang, and Sangchul Park. “A New AMHS Testbed for Semiconductor Manufacturing”. In: *Proceedings of the 2022 Winter Simulation Conference*. 2022, pp. 3318–3325. doi: [10.1109/WSC57314.2022.10015476](#).
- [159] Pierre-David Letourneau, Manish Kumar Singh, Hsin-Pai Cheng, Shizhong Han, Yunxiao Shi, et al. *PADRe: A Unifying Polynomial Attention Drop-in Replacement for Efficient Vision Transformer*. (Degree-2–3 polynomial attention matches softmax accuracy at 11–43 $\times$  speed; softmax itself only rational-approximable.) arXiv: [2407.11306](#).
- [160] D. A. Levin and Y. Peres. *Markov Chains and Mixing Times*. 2nd ed. With E. L. Wilmer. (Total-variation distance, couplings, and the data-processing/projection inequality: Ch. 4–5.) American Mathematical Society, 2017.
- [161] Ziyun Li and Henrik Boström. “FlowGuard: Guarding Flow Matching via Conformal Sampling”. In: *Proceedings of the Fourteenth Symposium on Conformal and Probabilistic Prediction with Applications*. Vol. 266. Proceedings of Machine Learning Research. 2025, pp. 775–777.
- [162] Sophie Libkind, Andrew Baas, Evan Patterson, and James Fairbanks. *Operadic Modeling of Dynamical Systems. Mathematics and Computation*. 2021. arXiv: [2105.12282](#).
- [163] Sophie Libkind and David Jaz Myers. *Towards a double operadic theory of systems*. 2025. arXiv: [2505.18329](#).
- [164] Jack Liell-Cock and Sam Staton. *Compositional imprecise probability*. 2024. arXiv: [2405.09391](#).
- [165] Feng Lin and W. Murray Wonham. “On Observability of Discrete-Event Systems”. In: *Information Sciences* 44 (1988), pp. 173–198. doi: [10.1016/0020-0255\(88\)90001-1](#).
- [166] T. Lindvall. *Lectures on the Coupling Method*. Corrected reprint, Dover, 2002. (The coupling inequality  $TV \leq \Pr[X \neq Y]$ : §1.2.) Wiley, 1992.
- [167] Robin Lorenz and Sean Tull. *Causal models in string diagrams*. 2023. arXiv: [2304.07638](#).



## REFERENCES

- [168] N. Lynch and F. Vaandrager. “Forward and Backward Simulations, Part I: Untimed Systems”. In: *Information and Computation* 121.2 (1995), pp. 214–233.
- [169] Michael Makkai and Gonzalo E. Reyes. *First Order Categorical Logic. Model-Theoretical Methods in the Theory of Topoi and Related Categories*. Vol. 611. Lecture Notes in Mathematics. Springer, 1977. doi: [10.1007/BFb0066201](https://doi.org/10.1007/BFb0066201).
- [170] David Malament. “The class of continuous timelike curves determines the topology of spacetime”. In: *Journal of Mathematical Physics* 18.7 (1977), pp. 1399–1404. doi: [10.1063/1.523436](https://doi.org/10.1063/1.523436).
- [171] Keye Martin and Prakash Panangaden. *A domain of spacetime intervals in general relativity*. 2004. arXiv: [gr-qc/0407094](https://arxiv.org/abs/gr-qc/0407094).
- [172] Jade Master and Joe Moeller. *Colored Petri Nets are Monoidal Double Functors*. 2025. arXiv: [2510.01946](https://arxiv.org/abs/2510.01946).
- [173] Jade Master, Evan Patterson, Shahin Yousfi, and Arquimedes Canedo. *String Diagrams for Assembly Planning*. 2019. arXiv: [1909.10475](https://arxiv.org/abs/1909.10475).
- [174] Cristina Matache, Sean Moss, Sam Staton, and Ariadne Si Suo. *Denotational semantics for languages for inference. semirings, monads, and tensors*. 2023. arXiv: [2312.16694](https://arxiv.org/abs/2312.16694).
- [175] Georges Matheron. *Random Sets and Integral Geometry*. New York: John Wiley & Sons, 1975.
- [176] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.
- [177] K. L. McMillan. “Circular Compositional Reasoning about Liveness”. In: *Correct Hardware Design and Verification Methods (CHARME 1999)*. Vol. 1703. Lecture Notes in Computer Science. Springer, 1999, pp. 342–346.
- [178] Paul-André Mellès. “Functorial Boxes in String Diagrams”. In: *Computer Science Logic (CSL 2006)*. Vol. 4207. Lecture Notes in Computer Science. Springer, 2006, pp. 1–30. doi: [10.1007/11874683\\_1](https://doi.org/10.1007/11874683_1).
- [179] Paul-André Mellès and Noam Zeilberger. *A bifibrational reconstruction of Lawvere’s presheaf hyperdoctrine*. 2016. arXiv: [1601.06098](https://arxiv.org/abs/1601.06098).
- [180] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. (Foster–Lyapunov drift conditions, systematized.) Springer, 1993.
- [181] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. 2nd ed. (Foster–Lyapunov drift conditions; the geometric drift form behind the deadline battery.) Cambridge University Press, 2009.
- [182] Matteo Mio, Ralph Sarkis, and Valeria Vignudelli. *Combining nondeterminism, probability, and termination. equational and metric reasoning*. 2020. arXiv: [2012.00382](https://arxiv.org/abs/2012.00382).
- [183] K. Mischaikow and M. Mrozek. “Conley index”. In: *Handbook of Dynamical Systems*. Vol. 2. North-Holland, 2002, pp. 393–460.
- [184] P. J. Moylan and D. J. Hill. “Stability criteria for large-scale systems”. In: *IEEE Transactions on Automatic Control* 23.2 (1978), pp. 143–149.
- [185] Jean-Michel Muller. *Elementary Functions. Algorithms and Implementation*. 3rd ed. Boston: Birkhäuser, 2016. doi: [10.1007/978-1-4899-7983-4](https://doi.org/10.1007/978-1-4899-7983-4).
- [186] T. Murata. “Petri nets: Properties, analysis and applications”. In: *Proceedings of the IEEE* 77.4 (1989). (Place invariants — the linear conservation laws entering as free count rows.), pp. 541–580.
- [187] M. Nagumo. “Über die Lage der Integralkurven gewöhnlicher Differentialgleichungen”. In: *Proceedings of the Physico-Mathematical Society of Japan*. 3rd ser. 24 (1942), pp. 551–559.
- [188] G. Nelson and D. C. Oppen. “Fast decision procedures based on congruence closure”. In: *Journal of the ACM* 27.2 (1980), pp. 356–364.
- [189] R. Nieuwenhuis and A. Oliveras. “Fast congruence closure and extensions”. In: *Information and Computation* 205.4 (2007), pp. 557–574.
- [190] J. R. Norris. *Markov Chains*. Cambridge: Cambridge University Press, 1997.
- [191] V. P. Orevkov. “Lower bounds for lengthening of proofs after cut-elimination”. In: *Zapiski Nauchnykh Seminarov LOMI* 88 (1979). English translation in *Journal of Soviet Mathematics* (1982), pp. 137–162.
- [192] Kalyanapuram R. Parthasarathy. *Probability Measures on Metric Spaces*. New York: Academic Press, 1967.
- [193] Evan Patterson. *Structured and Decorated Cospans from the Viewpoint of Double Category Theory*. 2023. arXiv: [2304.00447](https://arxiv.org/abs/2304.00447).
- [194] Evan Patterson, Owen Lynch, and James Fairbanks. *Categorical Data Structures for Technical Computing*. 2021. arXiv: [2106.04703](https://arxiv.org/abs/2106.04703).
- [195] Judea Pearl. *Causality. Models, Reasoning, and Inference*. 2nd ed. Cambridge University Press, 2009. doi: [10.1017/CB09780511803161](https://doi.org/10.1017/CB09780511803161).

## REFERENCES

- [196] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo: Morgan Kaufmann, 1988.
- [197] Riccardo Penco. *An Introduction to Effective Field Theories*. 2020. arXiv: [2006.16285](#).
- [198] Benedikt Peterseim and Milan Lopuszanski-Zwakenberg. *Fixed-parameter tractable inference for discrete probabilistic programs, via string diagram algebraisation*. 2026. arXiv: [2604.25321](#).
- [199] Andrew M. Pitts. “Categorical Logic”. In: *Handbook of Logic in Computer Science*. Ed. by Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum. Vol. 5. Oxford University Press, 2000, pp. 39–128.
- [200] George Pólya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton University Press, 1945.
- [201] S. Prajna and A. Jadbabaie. “Safety verification of hybrid systems using barrier certificates”. In: *Hybrid Systems: Computation and Control (HSCC 2004)*. Vol. 2993. LNCS. Springer, 2004, pp. 477–492.
- [202] S. Prajna, A. Jadbabaie, and G. J. Pappas. “A framework for worst-case and stochastic safety verification using barrier certificates”. In: *IEEE Transactions on Automatic Control* 52.8 (2007), pp. 1415–1428.
- [203] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley, 1994.
- [204] Tim Quatmann and Joost-Pieter Katoen. *Sound Value Iteration*. 2018. arXiv: [1804.05001](#).
- [205] M. O. Rabin. “Decidability of second-order theories and automata on infinite trees”. In: *Transactions of the American Mathematical Society* 141 (1969), pp. 1–35.
- [206] Peter J. Ramadge and W. Murray Wonham. “Supervisory Control of a Class of Discrete Event Processes”. In: *SIAM Journal on Control and Optimization* 25.1 (1987), pp. 206–230. doi: [10.1137/0325013](#).
- [207] Aaditya Ramdas, Peter Grünwald, Vladimir Vovk, and Glenn Shafer. *Game-theoretic statistics and safe anytime-valid inference*. 2022. arXiv: [2210.01948](#).
- [208] Aaditya Ramdas, Johannes Ruf, Martin Larsson, and Wouter Koolen. *Testing exchangeability, fork-convexity, supermartingales, and e-processes*. 2021. arXiv: [2102.00630](#).
- [209] Oliver Richardson and Joseph Y Halpern. *Probabilistic Dependency Graphs*. 2020. arXiv: [2012.10800](#).
- [210] Oliver E Richardson. *Loss as the Inconsistency of a Probabilistic Dependency Graph. Choose Your Model, Not Your Loss Function*. 2022. arXiv: [2202.11862](#).
- [211] Oliver E. Richardson, Joseph Y. Halpern, and Christopher De Sa. *Inference for Probabilistic Dependency Graphs*. 2023. arXiv: [2311.05580](#).
- [212] Oliver E. Richardson, Mandana Samiei, Mehran Shakerinava, Joseph D. Viviano, Abdessamad El Kabid, et al. *Local Inconsistency Resolution. The Interplay between Attention and Control in Probabilistic Models*. 2026. arXiv: [2604.17140](#).
- [213] R. Tyrrell Rockafellar and Stanislav Uryasev. “Optimization of Conditional Value-at-Risk”. In: *The Journal of Risk* 2.3 (2000), pp. 21–41. doi: [10.21314/JOR.2000.038](#).
- [214] W. Rudin. *Principles of Mathematical Analysis*. 3rd ed. (Taylor’s theorem with the Lagrange form of the remainder: Theorem 5.15; the nonnegative-remainder enclosure at nonnegative argument used by the TAYLORBOUND section.) McGraw-Hill, 1976.
- [215] A. van der Schaft and D. Jeltsema. “Port-Hamiltonian Systems Theory: An Introductory Overview”. In: *Foundations and Trends in Systems and Control* 1.2–3 (2014), pp. 173–378.
- [216] A. Schrijver. *Theory of Linear and Integer Programming*. (Farkas variants and multiplier sign conventions: §7.) Wiley, 1986.
- [217] Patrick Schultz and David I. Spivak. *Temporal Type Theory. A topos-theoretic approach to systems and behavior*. 2017. arXiv: [1710.10258](#).
- [218] Glenn Shafer and Vladimir Vovk. *Game-Theoretic Foundations for Probability and Finance*. Wiley Series in Probability and Statistics. Hoboken, NJ: Wiley, 2019. doi: [10.1002/9781118548035](#).
- [219] Michael Shulman. “Framed bicategories and monoidal fibrations”. In: *Theory and Applications of Categories* 20.18 (2008), pp. 650–738.
- [220] G. Singh, T. Gehr, M. Püschel, and M. Vechev. “An abstract domain for certifying neural networks”. In: *Proceedings of the ACM on Programming Languages* 3 (2019). POPL 2019, Article 41. (DeepPoly.)
- [221] Richard D. Smallwood and Edward J. Sondik. “The Optimal Control of Partially Observable Markov Processes over a Finite Horizon”. In: *Operations Research* 21.5 (1973), pp. 1071–1088. doi: [10.1287/opre.21.5.1071](#).

## REFERENCES

- [222] D. I. Spivak, R. Garner, and A. D. Fairbanks. *Functorial aggregation*. (Aggregation as commutative-monoid folds along a classifying map; the multiset as the universal aggregate.) arXiv: [2111.10968](#).
- [223] David Sprunger and Shin-ya Katsumata. *Differentiable Causal Computations via Delayed Trace*. 2019. arXiv: [1903.01093](#).
- [224] R. Statman. “Lower bounds on Herbrand’s theorem”. In: *Proceedings of the American Mathematical Society* 75.1 (1979). (Non-elementary lower bounds on quantifier-inference elimination; with the Orevkov bound, the classical witness for the cut-elimination blow-up.), pp. 104–107.
- [225] G. L. Steele Jr. *It’s Time for a New Old Language*. Keynote, ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP). (The “Computer Science Metanotation” this book’s rule cells are set in.) 2017.
- [226] R. S. Streett. “Propositional dynamic logic of looping and converse is elementarily decidable”. In: *Information and Control* 54.1–2 (1982). (The complemented-pairs acceptance condition.), pp. 121–141.
- [227] Daniel W. Stroock and S. R. Srinivasa Varadhan. *Multidimensional Diffusion Processes*. Vol. 233. Grundlehren der mathematischen Wissenschaften. Berlin: Springer, 1979.
- [228] Sean Summers and John Lygeros. “Verification of discrete time stochastic hybrid systems: A stochastic reach-avoid decision problem”. In: *Automatica* 46.12 (2010), pp. 1951–1961. doi: [10.1016/j.automatica.2010.08.006](#).
- [229] Natan T’Joens, Jasper De Bock, and Gert de Cooman. *A Particular Upper Expectation as Global Belief Model for Discrete-Time Finite-State Uncertain Processes*. 2020. doi: [10.1016/j.ijar.2020.12.017](#). arXiv: [2008.03258](#).
- [230] O. Taussky. “A recurring theorem on determinants”. In: *American Mathematical Monthly* 56.10 (1949). (The Levy–Desplanques strict-dominance nonsingularity theorem and its history.), pp. 672–676.
- [231] B. D. H. Tellegen. “A general network theorem, with applications”. In: *Philips Research Reports* 7 (1952), pp. 259–269.
- [232] W. Thomas. “Automata on infinite objects”. In: *Handbook of Theoretical Computer Science*. Vol. B. Elsevier, 1990, pp. 133–191.
- [233] Yde Venema, Steve Vickers, and Jacob Vosmaer. *Generalized powerlocales via relation lifting*. 2012. arXiv: [1202.3264](#).
- [234] J. Ville. *Étude critique de la notion de collectif*. Monographies des Probabilités. (The maximal inequality for nonnegative supermartingales — the probability engine of every battery rule.) Paris: Gauthier-Villars, 1939.
- [235] N. N. Vorob’ev. “Consistent families of measures and their extensions”. In: *Theory of Probability and its Applications* 7.2 (1962). Witness: the three-cell frustrated ring of SuFunctor. Adequacy. MarginalGlue is the minimal instance of the paper’s non-regular-cover obstruction, machine-checked over the subprobability stock., pp. 147–163. doi: [10.1137/1107014](#).
- [236] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer, 2005.
- [237] Ian Waudby-Smith and Aaditya Ramdas. *Estimating means of bounded random variables by betting*. 2020. arXiv: [2010.09686](#).
- [238] H. Weyl. *The Classical Groups: Their Invariants and Representations*. (Polarized power sums generate the multisymmetric invariants in characteristic 0.) Princeton University Press, 1939.
- [239] J. C. Willems. “Dissipative dynamical systems, Part I: General theory”. In: *Archive for Rational Mechanics and Analysis* 45 (1972), pp. 321–351.
- [240] Jan C. Willems. “The behavioral approach to open and interconnected systems”. In: *IEEE Control Systems Magazine* 27.6 (2007), pp. 46–99. doi: [10.1109/MCS.2007.906923](#).
- [241] Paul Wilson and Fabio Zanasi. *Data-Parallel Algorithms for String Diagrams*. 2023. arXiv: [2305.01041](#).
- [242] E. Wong and J. Z. Kolter. “Provable defenses against adversarial examples via the convex outer adversarial polytope”. In: *Proceedings of ICML 2018*. Vol. 80. PMLR. 2018.
- [243] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, et al. “Automatic perturbation analysis for scalable certified robustness and beyond”. In: *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. (The LiRPA framework and name.) 2020.
- [244] Bai Xue, Luke Ong, Dominik Wagner, and Peixin Wang. *Refined Barrier Conditions for Finite-Time Safety and Reach-Avoid Guarantees in Stochastic Systems*. 2025. arXiv: [2509.18518](#).
- [245] Fabian Zaiser, Andrzej S. Murawski, and C.-H. Luke Ong. *Guaranteed Bounds on Posterior Distributions of Discrete Probabilistic Programs with Loops*. 2024. arXiv: [2411.10393](#).

## REFERENCES

- [246] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. “Efficient neural network robustness certification with general activation functions”. In: *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. (CROWN: adaptive two-sided linear bounds.) 2018.
- [247] Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, et al. *Better Together. Unifying Datalog and Equality Saturation*. 2023. arXiv: [2304.04332](https://arxiv.org/abs/2304.04332).
- [248] Zhugen Zhou and Oliver Rose. “A Bottleneck Detection and Dynamic Dispatching Strategy for Semiconductor Wafer Fabrication Facilities”. In: *Proceedings of the 2009 Winter Simulation Conference*. 2009, pp. 1646–1653. URL: <https://www.informs-sim.org/wsc09papers/155.pdf>.

## Source attachment

The complete  $\text{\LaTeX}$  source of this book — the same single-file assembly served at <https://maitria.org/book.tex> — rides inside this PDF as an embedded file attachment named `book.tex`, byte-exact. Recovery:

```
pdftdetach -list book.pdf
pdftdetach -save 1 book.pdf
```

(or `pdftdetach -saveall book.pdf`; or `reader.attachments` in `pypdf`). `pdftotext` extracts the book's rendered text.