Direct Logic Arithmetization, Corrected

Sound finite-field compilation, finite higher-order polynomial semantics, and proof objects beyond polynomials

THE CLAIM

Direct first-order arithmetization promises an unusually short route from logic to polynomial constraints.

THE RESULT

A formal construction suite for sound direct logic: corrected FOL, finite higher-order semantics, live DREGG descriptors, executable conformance, and encrypted evaluation.

EVIDENCE STANDARD

This report distinguishes proved mathematics, checked executable models, theorem interfaces, implementation evidence, and future work. Performance claims are not inferred from algebra alone.

REPORT MAP

Contents

SECTION 1

What was built, what failed, what got smaller

Direct logic arithmetization is salvageable, generalizable, and useful. The reviewed public BitLogic presentation is not a sound instance of it. This report does more than separate those statements: it supplies the missing constructions.

RESULT We reconstructed Gabbay's matrix-language compiler, proved its characteristic- zero semantics, added two exact finite-field repairs, and constructed a fixed-shape private-witness matrix relation in DREGG's live DescriptorIR2. We then closed its external-statement boundary with a six-input public direct-table relation and built a general finite-signature FOL compiler, public statement- bound optimized descriptors for four production DREGG decisions, a fully priced proof/FHE boundary, a per-subterm certified presentation compiler, a strong-CCC shared-net compiler with a live interaction receipt, and categorical query and realizability coverage theorems. The public BitLogic equations still fail the permanent counterexample suite. The positive stack now reaches formal source, emitted descriptor bytes, live proofs, encrypted execution, and independent cross-language checking.

The audit also turned on our own construction. The first acceptance shape of the public direct-table relation — a field sum of squared residuals, made sound by a Lean-side no-wrap premise that no emitted byte checked — was proved to accept a fully canonical false table, by exactly the class of cancellation this report uses against BitLogic. It was replaced with three linear atoms and six emitted 30-bit range lookups; the two premises that had made it sound are now derived theorems rather than assumptions; and both counterexamples are retained as armed rejection canaries built by CI. Section 3 gives the account, the load-bearing proof for each half of the repair, and the one premise that moved rather than vanished.

1.1 The construction stack

Layer	Verified construction	Exact boundary
Gabbay front end	Integer matrices, one-based Lagrange row interpolation, terms, bounded quantifiers, source semantics, rational residual compiler, projection certificate, and Boolean repair.	`GabbayMatrixCompiler` and `GabbayMatrixBridge`; positive one-index matrix fragment, not arbitrary recursion.
Live matrix relation	The original three-entry instance is a private existential witness with no public binding. A separate direct-table descriptor binds all six claimed cells and accepts exactly when the public table satisfies source `Holds`. Its first acceptance shape was proved to accept a canonical false table and was replaced; the counterexamples are retained as armed rejection canaries.	`GabbayDescriptorIR2PublicBinding`: 6 public inputs, 6 columns, 15 constraints (6 PI pins, 3 linear atoms, 6 emitted 30-bit range lookups), 0 nonlinear products, exact tamper refusal, and zero table privacy. It is not interpolation-performance evidence.
Finite FOL	Closed de Bruijn syntax with equality, many relations of independent arities, total public function tables, every connective, and exhaustive finite quantification. Four public Boolean skeletons are proved equivalent to DREGG Admission, Upgrade, Clearance, and Strand decisions.	`FiniteSignatureFOLDescriptorIR2` and `DirectLogicDreggWorkloads`; exact layouts, statement soundness, live relations, certificates, and bytes. Interior arithmetic, hash, membership, lifecycle, lookup, and receipt atoms remain an explicit boundary.
Certified optimization	Local rules reconstruct both endpoints; a fail-closed checker proves semantic equivalence and cost nonincrease. The public lowering binds every atom residual to a PI and is sound for arbitrary nonempty satisfying traces.	`DirectLogicBoolGraphDescriptorIR2`: the public factor descriptor moves from 35 to 18 constraints, 26 to 13 multiplications, and width 21 to 12 while pinning optimizer endpoints, PI layout, and exact bytes.

Layer (continued)	Verified construction	Exact boundary
Proof and FHE	One raw opening feeds the proof relation and a BFV plan through an explicit conversion witness. Bounded encrypted zero observation and threshold opening now have explicit arithmetic, message, re-encryption, and leakage ledgers.	`CertifiedHybridProofFheZeroObservation` ; full accounting removes the former 6-versus-9 multiplication advantage. No general BFV noise/security theorem or deployed same-opening verifier is claimed.
Higher order	A four-mode indexed compiler charges conversions and glue per subterm. Typed STLC compiles strongly-CCC to shared nets; local steps emit live public interaction receipts through direct depth 14.	`CertifiedIndexedPresentationCompiler` and `IntensionalCCCInteractionDescriptorIR2` ; exact for the represented fragment, with a proved typed-syntax-erasure boundary.
General CCC boundary	Every small CCC embeds by Yoneda. A faithful family of finite polynomial actions exists exactly for locally finite hom-sets, but no fixed finite width works in general, already for finite sets.	`FamilyPolynomialYonedaFrontier` and `CertifiedPresentationFibredCCC` ; fixed-meaning evidence fibres are CCCs, while the global presentation category is not even cartesian.
Sampled proof objects	Coverage changes preserve obstruction and miss events exactly; false acceptance factors into OOD, low-degree, binding, and query-miss budgets. Effective assembly covers are pullback-stable and admit regular images.	`CategoricalDescentQueryProtocol` , `RobustDescentTranscriptGame` , and `AssemblyRegularCoverage` ; cryptographic/FRI reductions and the associated topos or exact completion remain external.

This changes the technical center of the report. FOL is a source language, not the primitive substance of proof. A compiler may preserve one judgment while changing its presentation locally: natural residual, Boolean graph, shared net, live AIR, or encrypted plan. The reusable theorem is the certified change of presentation, with semantics and resources preserved at every edge.

1.2 Performance evidence we can actually claim

The strongest performance evidence is now attached to public statement-bound descriptors for named DREGG decisions. Exact formal ledgers, retained proof serializations, noisy timing samples, and encrypted-boundary costs are reported separately; none is relabelled as an end-to-end speedup.

Public case	Constraints	Nonlinear mults	Width	Retained proof bytes
Factor specimen	35 to 18	26 to 13	21 to 12	Exact statement binding; paired proof not captured
Admission	121 to 71	108 to 58	77 to 47	14,016 to 11,894
Upgrade	33 to 23	28 to 18	21 to 15	10,431 to 10,081
Clearance	33 to 23	28 to 18	21 to 15	10,469 to 10,157
Strand	17 = 17	13 = 13	11 = 11	9,850 = 9,850

The factor theorem distinguishes 30-to-13 graph equations from the public descriptor's complete 35-to-18 constraint ledger: four PI bindings and one output-equals-one check are included. Multiplications fall from 26 to 13, auxiliaries from 17 to 8, and width from 21 to 12, with degree at most 2. `public_statement_sound` covers every nonempty satisfying trace, not only the compiler's witness; `public_canonical_trace_complete` supplies the converse for true canonical inputs; and `checked_public_statement_sound` transports the result through the accepted optimizer while pinning the PI layout and exact JSON bytes.

Admission, Upgrade, Clearance, and Strand are Boolean skeletons proved equivalent to named production DREGG decisions at the atom boundary. Lean emitted both public descriptors and canonical traces; Rust pinned and parsed those bytes, proved and verified every variant, and rejected a public-input tamper at the consumer verifier. The table's relation ledgers are deterministic theorem facts. Its proof sizes are retained postcard encodings and can vary with proof contents.

The same capture retained 25 proving and 75 verification samples per variant, but supports no timing speedup claim: the shared host was severely contended and the distributions were wide and nonmonotonic. Strand is the decisive negative control. Its source and optimized descriptor, trace, and proof bytes are identical, yet its timing medians and tails differ. The capture therefore validates the pipeline and quantifies proof-byte changes; it does not establish a latency ratio, scaling law, Modulus comparison, or finality result.

Fully pricing encrypted zero observation also removes the apparent residual win. For the formal $B = 4$ five-equality plan, 6 inner multiplications plus 3 bounded zero-conversion multiplications equals the all-Boolean 9, and both reach depth 4. For the executable $B = 8$ case, 8 plus 7 equals the Boolean 15. The bounded scaled conversion executes exactly at $B = 2$. At $B = 8$ the captured scaled-bit threshold observation refused a noncanonical output under the current unproved noise

envelope, while the separate raw threshold path succeeded by revealing every live residual. This is fail-closed engineering and explicit leakage accounting, not an FHE speedup or a general noise/security theorem.
The executable conformance corpus evaluated 10,061 cases. The corrected `D-NOWRAP` lane passed all 6,737 admitted cases and refused 3,324 unsupported ones; `D-BOOLGRAPH` and `C-AIR` passed all 10,061. `M-SPEC` produced 3,936 mismatches and the naive field projection produced one cancellation mismatch. Unsound lanes are retained as falsification artifacts and excluded from every performance ratio.

FORMAL ARTIFACT - executable boundary The independent Standard ML checker passed 15/15 version-1 fixtures and its complete live version-2 checker passed 11/11. HOL4 proves semantic soundness for accepted v1 and live-v2 bytes with zero recorded theory axioms, and CakeML translation certificates exist for both checkers. Lean emits the production descriptors; Rust byte-pins, parses, proves, verifies, and tamper-tests them. This is cross-language assurance, but neither a shipped native CakeML checker binary, a Rust-to-Lean verifier refinement, nor a cryptographic same-opening reduction is claimed.

1.3 The public-audit result

The audited public claim does not survive technical review.

- Gabbay's construction is sound in its stated nonnegative characteristic-zero semantics. Naively interpreting its zero test in a prime field is unsound because nonzero residues can cancel or wrap to zero.
- BitLogic reverses the published zero-true connective and quantifier rules. Worse, its swap relation multiplies independent obligations. Under zero-true semantics multiplication expresses disjunction, so satisfying one factor can accept while the others fail. The concrete assignment $A = B = 10$, $k = q = 1$, $A' = 9$, $B' = 999$ is accepted by the displayed product although it violates two of the three intended obligations.
- The reviewed public repository supplies a PDF, not a runnable compiler, prover, verifier, parameter set, raw benchmark corpus, or reproducible path to the advertised 10–100x cost reduction and 1–3 second finality.

BOUNDARY The relation-level counterexample is decisive for the published equations; it is not a claim about unseen private code or a demonstrated production exploit. Conversely, unseen private code cannot substantiate a public performance or soundness claim. The strongest comparison currently available is therefore public evidence against public evidence: DREGG supplies the stronger formal semantics, executable falsification, construction breadth, and reproducibility; no like-for-like Modulus runtime comparison is possible from the released material.

EVIDENCE LEGEND **Kernel-checked** means a Lean or HOL4 theorem behind the stated axiom-hygiene gate. **Executable** means a concrete compiler, checker, counterexample, witness, or evaluator. **Measured** means a recorded implementation run with its workload and exclusions. A **theorem socket** still needs a protocol or backend instantiation. A **marketing claim** lacks a public reproducibility path.

SECTION 2

Audit of the direct FOL arithmetization claim

This section separates three statements that are easy to conflate: Gabbay's characteristic-zero semantics, a finite-field implementation of that semantics, and the equations printed in the public ModulusZK/BitLogic whitepaper. The algebraic kernel of the first is sound in its stated model. A direct finite-field port is not sound without an additional invariant or gadget. More strongly, the published BitLogic encoding is not a finite-field implementation of Gabbay's theorem: it changes the semantic domain and reverses every connective and finite quantifier fold used by that theorem. These are constructive specification findings: they neither allege bad intent nor establish a vulnerability in a deployed system.
The primary Gabbay source is the official ePrint 2024/954 record, last revised 12 February 2026 and published in the *Journal of Applied Logics* 12(6), 1481–1548 (2025). Numbered references below follow the current ePrint text; theorem numbers, rather than unstable viewer page counts, are the primary locators. The BitLogic source is the 14-page PDF in the official public repository. Its pinned tree at commit `26f125b37f0442d92991e3066095b1dfeb1b0ce4` (21 November 2025) contains exactly that one PDF blob.

2.1 What Gabbay actually proves

Gabbay interprets predicates as polynomials in $\mathbb{Q}[X]$ whose values on $\mathbb{Q}$ are nonnegative. Zero denotes truth. Figure 2, defining the semantics in Definition 2.2.8, assigns addition to conjunction and bounded universal quantification, and multiplication to disjunction and bounded existential quantification:

$$R(\varphi \wedge \psi) = R(\varphi) + R(\psi), \quad R(\varphi \vee \psi) = R(\varphi)R(\psi),$$
$$R(\forall x \in H.\varphi(x)) = \sum_{x \in H} R(\varphi(x)), \quad R(\exists x \in H.\varphi(x)) = \prod_{x \in H} R(\varphi(x)).$$

Definition 2.3.1 states the pointwise nonnegativity condition. Lemma 2.3.2 proves that predicate denotations preserve it. Theorem 2.3.4 is the soundness-and-completeness theorem, and Remark 2.3.5 identifies its load-bearing arithmetic facts:

$$a \geq 0, b \geq 0 \Rightarrow (a + b = 0 \Leftrightarrow a = 0 \wedge b = 0),$$
$$ab = 0 \Leftrightarrow a = 0 \vee b = 0.$$

Thus the theorem is not a theorem that arbitrary field-valued residuals may be combined by the same operations. It is an exact theorem about zero sets under a protected nonnegative, characteristic-zero interpretation.
We machine-checked this algebraic kernel, rather than re-formalizing every matrix and syntactic feature of the source paper. In `FOLArithmetizationCorrected.lean`, the structure `ZeroTrueLaws` makes the noncancellation and no-zero-divisor obligations explicit. `PositiveFormula.residual_eq_zero_iff` proves exact compilation for the positive Boolean fragment; `PositiveFormula.residual_all_eq_zero_iff` and `PositiveFormula.residual_any_eq_zero_iff` prove the corresponding finite quantifier folds. The instance `nonnegativeRationalLaws` discharges precisely Gabbay's nonnegative-rational obligations. This scope distinction matters: our Lean theorem validates the algebraic core used here, not every later construction in ePrint 2024/954.

2.2 Why the naive finite-field port fails

A finite field has no positivity order compatible with its ring operations, and nonzero summands can cancel. Gabbay explicitly records this limitation in Remark 6.3.3 ("Why Q?"): the zero-sum property used by addition-as-conjunction is lost in a prime field. The proposed cryptographic route is to establish the polynomial equality over $\mathbb{Q}[X]$ first and only then map that equality into a suitably large prime field. It is not a direct finite-field truth semantics, and the rational proof does not transfer unchanged.
The failure already occurs in BabyBear, the prime field with $p = 2,013,265,921$. Let $r = 1,728,404,513$. Lean verifies

$$r^2 = -1 \bmod p, \quad 1 \neq 0, \quad r^2 \neq 0, \quad 1 + r^2 = 0.$$

Taking squared equality residuals gives the concrete false conjunction

$$R(0 = 1) = 1, \quad R(0 = r) = r^2 = -1, \quad R((0 = 1) \wedge (0 = r)) = 0.$$

Both atoms are false under zero-means-true, yet their sum accepts. The exact certificate is `FOLArithmetizationCounterexamples.babyBear_false_and_false_accepted`; `babyBear_sqrt_neg_one` certifies the field identity, and `babyBear_false_forall_accepted` lifts the same pair to a two-point universal quantifier. This is not a low-probability collision: it is an explicit witness. Nor is $1 - R(\varphi)$ a general repair for negation before Booleanization; `babyBear_one_sub_is_not_general_negation` gives the residual $R(\varphi) = 2$, for which both $R(\varphi)$ and $1 - R(\varphi)$ are nonzero.

Two exact finite-field repairs are already formalized. First, compile in nonnegative integers and prove that the complete residual is strictly below the field modulus before casting. `zmod_natCast_eq_zero_iff_of_lt` is the no-wrap lemma, and `natural_residual_cast_sound` is the compiler theorem. Second, reify each arbitrary residual r as a Boolean false-bit b with inverse witness z:

$$r(1-b)=0, \quad rz=b.$$

The constraints force $b\in\{0,1\}$ and $b=0\Leftrightarrow r=0$, proved by `falseBit_eq_zero_iff`, `falseBit_is_bit`, and `exists_falseBitWitness`. Once Booleanized, exact connectives are

$$b_{\varphi\wedge\psi}=b_\varphi+b_\psi-b_\varphi b_\psi, \quad b_{\varphi\vee\psi}=b_\varphi b_\psi, \quad b_{\neg\varphi}=1-b_\varphi.$$

`BooleanFormula.falseBit_correct` proves, for every formula in this Boolean core, both that the output remains a bit and that it is zero exactly when the source formula holds. These repairs have real costs—static bound certificates in the first case, and witness equations plus Boolean connective gates in the second—so they must be included in any performance comparison.

2.3 The BitLogic connective reversal

The public BitLogic PDF states on p. 4 that conjunction maps to product and disjunction to sum. Section 7.1 (p. 10) repeats that map and additionally maps bounded universal quantification to product and existential quantification to sum. Under zero-means-true, this reverses the intended logic even in an integral domain:

$$P_\varphi P_\psi=0\Leftrightarrow P_\varphi=0\vee P_\psi=0.$$

Accordingly, a product recognizes disjunction, not conjunction. The Lean theorem `FOLArithmetizationCounterexamples.zero_of_mul_iff_or` proves this generic fact. The finite-list regressions `product_is_not_zero_true_forall` and `sum_is_not_zero_true_exists` show both quantifier errors: the product $0\cdot 7$ accepts a purported universal despite one false point, while the sum $1+(-1)$ accepts a purported existential despite having no zero residual. The issue is visible in the swap relation printed in Section 4.1 (p. 6). Writing $q=f(k,s)$ for the quoted output, the PDF defines

$$P_{\text{swap}}=(A'-A+k)(B'-B-q)(AB-A'B')$$

and accepts when $P_{\text{swap}}=0$. Because this is a product, any one satisfied factor bypasses the other two. For example, over the integers choose

$$A=B=10, \quad k=q=1, \quad A'=9, \quad B'=999.$$

Then

$$A'-A+k=0, \quad B'-B-q=988\neq 0, \quad AB-A'B'=-8,891\neq 0,$$

so the printed relation accepts while its price/output and constant-product conditions both fail. This calculation is checked by `published_swap_accepts_invalid_witness`. More generally, `published_swap_first_factor_bypasses_rest` proves the bypass in every commutative ring, and `published_swap_bypass_is_polynomial_identity` proves

$$P_{\text{swap}(A,B,A-k,B',k,q)}=0$$

for all remaining inputs. Random evaluation cannot detect constraints erased by an identically zero factor; Schwartz–Zippel bounds a nonzero polynomial's accidental zeros, not a relation that is already the zero polynomial after the witness substitution.

Construction	Connectives (zero is true)	Finite quantifiers	Established scope
Gabbay, Thm. 2.3.4	$\wedge\mapsto+$; $\vee\mapsto\times$	$\forall\mapsto\sum$; $\exists\mapsto\prod$	Exact for pointwise nonnegative $\mathbb{Q}[X]$ denotations (formal theorem above).
BitLogic PDF, pp. 4, 10	$\wedge\mapsto\times$; $\vee\mapsto+$	$\forall\mapsto\prod$; $\exists\mapsto\sum$	Fails the connective and quantifier counterexamples above.
No-wrap field repair	Gabbay map, then bounded cast	Gabbay folds, then bounded cast	Exact under a proved residual bound; Lean: `natural_residual_cast_sound`.
Boolean field repair	$\wedge\mapsto a+b-ab$; $\vee\mapsto ab$	Boolean folds	Exact over fields after atom reification; Lean: `BooleanFormula.falseBit_correct`.

The authors' May 2026 CTB workshop slides make the implementation direction more concrete: they move through natural/integer semantics, bit decomposition, Booleanity constraints, and a Zinc integer AIR. The slides explicitly set range checks aside in the presented sketch and list direct finite-field arithmetization and a non-toy implementation as future work. This is useful convergence on the repair obligations above, not support for omitting them. Our construction

program therefore treats bounds, Booleanity, and range evidence as first-class compiler outputs rather than prose side conditions.

2.4 Evidence boundary and constructive next step

As of 22 July 2026, the named public BitLogic repository's pinned tree contains exactly one 14-page PDF. It contains no executable source, build manifest, compiler, witness generator, prover, verifier, release, parameter set, benchmark harness, proof corpus, or raw measurement. The PDF names the real, third-party Zinc/Zip proving stack, but the public search found no Modulus FOL-to-Zinc compiler, proof-format adapter, or verifier integration. This is a reproducible statement about the named public artifacts, not a claim about private or differently named work. Consistently, the whitepaper's Phase I roadmap (Section 9.1, p. 13) lists implementing the compiler and verifier and benchmarking polynomial-evaluation costs as work for 2025–2026.

The public performance claim is also not one stable metric. The current site applies "10–100x" to cost and adds "1–3 second finality"; a project-supplied 13 November 2025 release gives proof-price ranges; the 20 October 2025 project history instead says 10x faster and 90% smaller; and a 1 December 2025 post says 10–100x smaller proofs. Cost, proof size, prover time, and finality are different quantities. We found no public workload corpus, proof-system and security parameters, comparison implementation, hardware manifest, harness, or raw samples that bind any of them to the advertised multiplier.

"Finality" itself needs a tier. The official Modulus glossary distinguishes a trusted state committed by its single trusted sequencer, a virtual state whose batches have reached L1, and a consolidated state proven on L1 by a ZKP. It attributes fast or instant finality to the trusted sequencer and one-transaction L2 blocks. Without identifying one of these states, the site's 1–3 second figure is not a reproducible settlement or proof-latency measurement.

There is now a useful first-party disclosure boundary. In a 21 July 2026 public reply, Modulus wrote: "There are major ones. Just announcing tech breakthroughs without the open source code is generally frowned upon …" The same reply explains that commercial and internal positioning precedes publication. We therefore do not infer that private work is absent, nor that it matches the PDF. The decisive claim in this report is narrower: the one published BitLogic relation is not Gabbay's finite-field construction and fails its stated source logic.

Therefore we report a specification-level counterexample, not a production exploit and not a measured performance result. A deployed implementation could use different equations; publishing its compiler, proof interface, and verifier would resolve the discrepancy. The shared opportunity is an open conformance suite containing the examples above, corrected no-wrap and Booleanized lowerings, conventional circuit baselines, and identical proof-system parameters. Soundness is the admission gate; only implementations that agree on the source truth table should enter a latency, size, finality, or cost chart.

SECTION 3

Corrected constructions: from matrices to searched proof plans

The corrected development is now a coherent compiler stack rather than a replacement connective table. It begins with Gabbay-style integer matrices and exact rational interpolation, proves that an independently defined source semantics agrees with the compiled polynomial, crosses into a finite field only through an explicit projection certificate, and reaches DREGG's live descriptor relation. A second compiler handles arbitrary finite signatures. Above both sit checked optimization and certified representation search.

> **CONSTRUCTION RESULT** There is a sound version of the matrix idea. Its invariant is not "first-order logic is automatically a finite-field circuit." The invariant is: exact nonnegative rational semantics first; a proved prime and no-wrap boundary or an exact Boolean repair second; and a checked live relation last. Every transition now has a theorem, a refusal case, and a stated cost and public-binding boundary.

3.1 Construction map

Layer	Construction	Exact result	Lean artifact
Matrix semantics	integer tables, one-based rational Lagrange rows, bounded matrix quantifiers	compiled residual is zero iff the independently defined source formula holds	GabbayMatrix Semantics
Compiler agreement	positive-arity adapter between the semantics view and the proof-carrying compiler	row, term, formula, truth, residual, cost, projection certificate, and Skolem witness agree	GabbayMatrix Compiler / Bridge
Field boundary	reduced numerator and positive denominator plus prime/no-wrap certificate	field zero iff rational zero iff source truth; bad modulus is rejected	GabbayFiniteField Projection
Private Gabbay bridge	two-row, three-column successor table in DescriptorIR2	12-column constructive private trace; 15 constraints (9 gates plus 6 emitted 30-bit range lookups); exact for the table carried by that trace, but not externally bound	GabbayDescriptorIR2
Public Gabbay statement	six direct public table cells pinned to a six-column trace	15 constraints (6 PI pins, 3 linear atoms, 6 range lookups) and zero nonlinear products; any accepted canonical trace attests the named external table	GabbayDescriptorIR2 PublicBinding
Wire-hole canaries	the two canonical false tables that retired the previous acceptance shape, kept armed	each half of the repair refutes a table the other half admits; both refused as preselected traces and as external statements, with a positivity control	DirectLogicAdversarial FalsifierV2
General finite FOL	public total functions, witness relation tables, equality, all Boolean connectives and bounded quantifiers	canonical trace accepts iff the finite model satisfies the sentence	FiniteSignatureFOL DescriptorIR2
Optimization	checked identities, factoring and sharing, followed by explicit materialized BoolGraph emission	source meaning, public arbitrary-trace soundness and exact resource reductions are preserved	DirectLogicOptimizer Certificate DirectLogicBoolGraph DescriptorIR2
DREGG workloads	admission, upgrade, credential clearance, and strand-admission Boolean skeletons	public live soundness against named production decisions; exact before/after ledgers	DirectLogicDregg Workloads
Representation search	finite exact-change graph plus an indexed per-subterm compiler with explicit conversion and glue costs	selected composite remains exact; materialized ledger has no untracked remainder	CertifiedRepresentation Search CertifiedIndexed PresentationCompiler

3.2 Exact matrix semantics and its compiler

For a matrix with `rows` rows and `length` declared columns, `IntMatrix.rowPolynomial` is the unique rational Lagrange interpolant through the one-based nodes $1, \ldots, \text{length}$. The two facts needed downstream are proved directly:

$$P_{r(j+1)} = M_{r,j}, \quad \deg(P_r) < \text{length}.$$

Terms contain the distinguished index, rational constants, arithmetic, matrix length, and row lookup. Formula atoms are squared equalities. Conjunction and bounded universal quantification add nonnegative residuals; disjunction and bounded existence multiply them. `residual_nonnegative` is the load-bearing lemma that rules out additive cancellation over $\mathbb{Q}$. The central theorem, `residual_eq_zero_iff_holds`, proves for every valuation, rational index, and formula φ:

$$\rho_{M,x}(\varphi) = 0 \quad \text{iff} \quad M, x \vDash \varphi.$$

The quantifiers range over exactly the columns declared by the selected matrix. `quantifierExpansion_eq_length` records the exact number of body instantiations. Because each bounded quantifier closes its index, its compiled polynomial is constant in the outer index; `forall_degree_le_zero` and `exists_degree_le_zero` state that fact. These are degree theorems, not claims about prover latency.

`GabbayMatrixCompiler` independently defines its own signature, matrices, source satisfaction, term evaluation, and polynomial compiler. Its `eval_polynomial_eq_zero_iff` proves the same exact source/compiler relation. `GabbayMatrixBridge` then prevents agreement-by-prose: under `PositiveArity`, it proves equality of row polynomials, term and formula polynomials,

source truth, residuals, and exact expansion costs. `ViewCertificate.toCompiler` transports matrix bounds, numerator and denominator bounds, and the cost equality into the canonical compiler certificate.

FORMAL ARTIFACT - matrix/Skolem specimen The checked specimen is an actual two-row, three-column integer table. Its first row is the input and its second row is the chosen successor output. `successorTable_size` proves an exact six-cell table; `successorTable_row_degrees_exact` proves that both interpolants have degree one. Through the bridge, exhaustive universal checking costs exactly three equality atoms and two additions, and `successor_skolem_through_bridge` constructs the finite Skolem witness. The transported certificate discharges the $p = 17$ projection theorem without repeating the bound proof.

BOUNDARY The bridge's only semantic mismatch is explicit: the exploratory semantics type permits a zero-column matrix, while the compiler signature requires positive arity. This stack formalizes bounded finite matrix semantics. It is neither ordinary unbounded first-order quantification nor a succinctness theorem.

3.3 The field boundary is a certificate, not a cast

Let the exactly evaluated residual be the reduced rational $q = \frac{a}{b}$, with $b > 0$. `ProjectionCertificate q p` carries three facts:

$$p \text{ is prime}, \quad |a| < p, \quad b < p.$$

`projectCleared` sends only a into $\frac{\mathbb{Z}}{p}\mathbb{Z}$; `projectRational` sends $\frac{a}{b}$ after proving that the denominator remains invertible. `projectCleared_eq_zero_iff` and `projectRational_eq_zero_iff` prove that either representation is zero in the field exactly when $q = 0$. Composing that result with matrix correctness gives `projected_formula_zero_iff_holds` and `projected_rational_formula_zero_iff_holds`. The compiler emits the auditable floor

$$\text{modulusFloor}(q) = \max(|a|, b) + 1.$$

Any prime at least this floor constructs the certificate. `ProjectionCost` retains four distinct quantities: formula degree, numerator magnitude, normalized denominator, and required modulus floor. They are not collapsed into a context-free speedup number.

EXECUTABLE COUNTEREXAMPLE The false atom $0 = 5$ has exact residual 25. An unchecked projection to $\frac{\mathbb{Z}}{5}\mathbb{Z}$ returns zero, so `badPrime_unchecked_accepts` exhibits the failure in executable form. `badPrime_certificate_rejected` proves that the corrected compiler refuses the same case because the required strict inequality $25 < 5$ is impossible. This is the smallest useful answer to "why not just evaluate the rational construction in a field?"

For cases where a useful no-wrap certificate is unavailable, the independent Boolean repair remains exact over every field. Atomic zero tests introduce a bit and an inverse witness; Boolean connectives then operate on constrained bits. `Formula.falseBit_isBit_and_correct` and `falseBit_eq_zero_iff` in the matrix compiler prove that this alternative supports the full matrix formula language without pretending that rational addition retained its semantics after reduction.

3.4 Routes into the live DREGG relation

3.4.1 A fixed-shape private witness/semantics bridge

`GabbayDescriptorIR2` lowers a nontrivial matrix instance into the actual `EffectVmDescriptor2` grammar used by DREGG. Its source is a two-row, three-column graph of a unary successor function. The trace contains the six table entries and six denominator-cleared interpolation coefficients: exactly 12 columns (`TRACE_WIDTH = 12`). There are no residual, numerator, or denominator columns; the repair described in *The wire hole and its repair* below retired all three.

For row values y_1, y_2, y_3, the compiler stores the coefficients of $2P(X)$:

$$(6y_1 - 6y_2 + 2y_3, -5y_1 + 8y_2 - 3y_3, y_1 - 2y_2 + y_3).$$

This avoids witness-authored division inside the interpolation checks. Six interpolation identities, three linear acceptance atoms $\text{output}_j - \text{input}_j - 1$, and six 30-bit range lookups — one on each entry column — give exactly 15 live constraints (`compileDescriptor.constraints.length = 15`). The coefficient columns are nevertheless redundant for the current acceptance path: the acceptance atoms read the raw input and output entry columns directly. This instance therefore demonstrates a faithful fixed-shape witness/semantics bridge, not a performance advantage from interpolation.

`trace_satisfied_iff_holds` proves a table-indexed equivalence: `traceOf table` satisfies the emitted BabyBear relation exactly when the source formula holds of that same `table`. Its `WireBoundedTable` hypothesis is a premise of the **completeness** direction

only — it names which tables this descriptor can express. Soundness needs no side condition on the claim: the six emitted lookups are what `accepting_trace_wire_bounded` reads the 30-bit bound off, so a table outside the checked domain is refused rather than silently accepted. The canonical successor trace is constructed, the constructed trace of a one-cell tamper is refused, and the known modulus-five projection cannot enter the compiler.

> **BOUNDARY** The current descriptor is an existential private-witness statement, not an attestation of an externally named table. `descriptor_public_surface_empty` proves `piCount = 0`, `hashSites = []`, `ranges = []`, and `constraints.length = 15`. Read the third conjunct exactly: `ranges` is the v1 range **carrier** field, and it is empty by construction because the deployed v2 assembly refuses a descriptor that carries it at all. The no-wrap bound is enforced by the six graduated `lookup` constraints against the declared 30-bit range table, which the assembly lowers to a real byte-limb decomposition. An empty `ranges` field is not an absent range check. A range tooth binds no external **name**, however, so it does not close the statement gap this boundary records. `constructed_traces_publicly_indistinguishable` proves that every constructed table trace exposes the same empty public assignment. Most sharply, `accepting_trace_does_not_attest_external_table` constructs an accepting successor witness while a separately named tampered table is false; there is no contradiction because that external table is absent from `Satisfied2`. A public input or binding commitment is required before this relation can attest a claimed model. It is also not yet a generator for arbitrary matrix dimensions: that requires emitted interpolation, denominator management, range certificates, and cost accounting. IR-v2 fixes BabyBear; the witness does not choose its own field.

3.4.2 Direct-table public binding

`GabbayDescriptorIR2PublicBinding` closes the statement gap with a distinct, smaller descriptor. All six cells of the claimed two-by-three table are public inputs. Six first-row PI bindings pin them to six trace columns; three always-on gates check the three successor equations, one equation per gate; and six `lookup` constraints check that each bound column is a 30-bit value in the declared range table. There are no interpolation coefficients, denominator column, hash site, or commitment assumption. The descriptor's `ranges` field is empty — the enforcing instrument is the six lookups, not the v1 carrier.
The exact ledger is proved by `descriptor_public_surface_exact` and `direct_public_cost_exact`:

Public inputs	Trace columns	Constraints	Range teeth	Nonlinear products	Private table cells
6	6	15	6	0	0

The 15 constraints are 6 PI pins, 3 linear acceptance atoms, and 6 range lookups. Acceptance contains **no** nonlinear product: counting multiplication by constants, the whole acceptance block is 6 multiplication nodes, and `acceptanceNonlinearMulNodes = 0`. `publicLayout` pins the exact PI order, while an executable guard pins the full emitted descriptor JSON byte-for-byte, including `"sem":"range","bits":30` on table 2 and the six `{"t":"lookup","table":2,...}` entries. Without that byte pin the teeth could be deleted and the constraint count alone would not notice, since it sees how many constraints there are and not which.
The statement theorem is now external rather than witness-relative. `StatementSatisfied hash claim trace` is exactly three conjuncts: the trace is nonempty, its `.range` table is the honest 30-bit table, and it satisfies the emitted descriptor under the claimed public vector. Both surviving conjuncts are conditions on the **witness**, not side conditions on the claim. Canonicality of the first row is no longer among them: the descriptor's own teeth force it, which is what `first_row_input_bounded` and `first_row_output_bounded` prove. `statement_sound` then proves that **any** such satisfying trace implies `Holds` of the externally named table, with the claim's only hypothesis being the decoding convention that the six public field elements are read as canonical BabyBear integer representatives. `statement_complete` constructs a trace for every wire-bounded true claim, and `public_statement_satisfied_iff_holds` packages the exact result:

$$\exists t.\ \mathrm{StatementSatisfied}(h, M, t) \quad \text{iff} \quad M \vDash \varphi_{\text{successor}}.$$

`tampered_public_statement_refused` proves that the one-cell public tamper has no satisfying canonical witness. This is stronger than refusing one preselected trace: the public statement itself is impossible.

BOUNDARY Public integrity is purchased by revealing the whole table. `public_statement_reveals_whole_table` proves that equality of the six public inputs implies equality of all table cells. This descriptor therefore has zero table privacy. It also makes no interpolation performance claim: there are no coefficient columns and interpolation is not part of logical acceptance. The result is an exact fixed-shape public statement, not an arbitrary-matrix compiler.

3.4.3 The wire hole and its repair

Both Gabbay descriptors above are the **second** version. The first accepted a fully canonical false table, and the counterexample that retired it is reproduced here because it is the most instructive artifact in this section.

THE HOLE Acceptance used to be a single always-on gate on the sum of the three squared successor residuals, sound only under a Lean-side `LiveProjectionCertificate` bounding the integer numerator, plus a `CanonicalFirstRow` trace premise. Neither premise was ever serialized, so the emitted bytes did not enforce what made them sound. Over BabyBear $284861408^2 = -1$, so residuals $1, 284861408, 0$ give $1 + (-1) + 0 = 0$ in the field while two of the three equations fail. The table input $= (0, 0, 0)$, output $= (2, 284861409, 1)$ is fully canonical, lies inside the descriptor's checked domain, is false, and was ACCEPTED. `wrapClaim_residual_exact` records its exact integer numerator $81146021767742465 = 2013265921 \times 40305665$, and `wrapClaim_numerator_exceeds_modulus` records that the retired certificate demanded exactly the bound this table violates.

Two independent defects meet in that table. A field sum of squares is not a conjunction, because a field has no order to make squares nonnegative. And the condition that repaired the arithmetic lived in Lean rather than in the descriptor, so no emitted byte checked it. The second is the deeper one: a premise off the wire is not a check.

The repair puts both halves on the wire. Acceptance became three linear atoms, one per successor equation, so no cancellation between equations is available at all. The no-wrap condition became a 30-bit range **lookup** on each of the six bound columns, against the declared range table. The graduated lookup form is deliberate rather than the v1 `ranges` carrier: the deployed v2 assembly refuses a descriptor carrying the v1 form at all, so those bytes would have enforced the bound only in Lean — the same mistake in a new place — whereas a lookup is lowered to a real byte-limb decomposition at the width pinned by the committed table id.

With both halves emitted, the two premises stop being premises. `statement_forces_wire_bounded` derives the retired `LiveProjectionCertificate` from the teeth; `first_row_input_bounded` and `first_row_output_bounded` derive the retired `CanonicalFirstRow`. `statement_sound` and `public_statement_satisfied_iff_holds` no longer take a certificate argument.

FORMAL ARTIFACT - both halves are load-bearing A single counterexample would leave one half of the repair looking decorative, so a second is carried. `wrapClaim` lies inside the 30-bit domain, so it is refused by the linear atoms alone and the teeth do no work on it. `borderWrapClaim` — $\text{input}_0 = 2013265920$, $\text{output}_0 = 0$ — is canonical and false, and `borderWrap_atom_gates_all_pass` proves that all three linear gates evaluate to a BabyBear zero on it, since the first residual is exactly $-p$. The only emitted gate that refuses it is the 30-bit tooth on column 0 (`borderWrap_range_tooth_fires`). Each half kills a counterexample the other admits; deleting either turns one of these theorems red.

Both canaries are proved in both polarities — refusal of the exact preselected trace, and refusal of the external statement for **every** trace — and `repaired_descriptor_still_accepts_truth` is the positivity control that the repair rejects the counterexamples rather than everything. `DirectLogicAdversarialFalsifierV2` is imported by the trusted umbrella, so CI builds these canaries like any other theorem; an unbuilt canary is not a canary.

The linear repair is also the cheaper circuit:

Public descriptor	Retired sum-of-squares	Repaired
Constraints	7	15
of which range teeth	0	6
Multiplication nodes	15	6
Nonlinear products	3	0

The three squares carried 15 multiplication nodes, 3 of them nonlinear; the three linear atoms carry 6 constant-scaling nodes and no nonlinear product. The added cost is two extra gates and the deployed byte-limb realization of the six teeth, which the node count does not price.

BOUNDARY One premise moved rather than vanished. A range lookup bounds a column only against the table it looks into, and the abstract `Satisfied2` carrier has structural faithfulness conjuncts for the memory and map tables but none for `.range`, so a prover may choose `tf .range` freely. `StatementSatisfied` therefore carries `HonestRangeTable trace` explicitly, and every refusal that routes through a range tooth is conditional on it — `borderWrapClaim` genuinely needs it, while `wrapClaim`'s refusal is table-free because linear atoms read no table. This premise is of a different kind from the retired one: it is a condition on the trace family rather than an unchecked assertion about the claim, it is discharged by construction for every trace built here and in deployment by the assembly, which constructs the limb decomposition rather than reading a prover-supplied table — and it is carrier-level and uniform, since every range lookup in the codebase reads against `t.tf .range`. The structural fix already exists in shape as `Satisfied2Faithful`, which carries `rangeTableFaithful` as a conjunct; this descriptor has not been ported to it, and porting would move the obligation to the Rust assembly rather than discharge it inside Lean.

3.4.4 General finite-signature FOL

`FiniteSignatureFOLDescriptorIR2` takes the complementary route. A signature may contain any finite list of relation symbols with independent arities and any finite list of total function symbols with complete public interpretations. Terms include constants, de Bruijn variables, and nested function application. Formulae include equality, relation application, every Boolean connective, and exhaustive finite universal and existential quantification.
For a q-element domain and relation arities a_r, the canonical witness-row width is proved exactly:

$$W = \sum_r q^{a_r}.$$

The row concatenates every relation table in symbol-major, tuple-major order; the descriptor contains $W + 1$ constraints. `fol_sound`, `fol_complete`, and `canonical_model_trace_iff` prove soundness, constructive completeness, and the exact canonical-trace equivalence. A checked `FOLLiveCertificate` binds the version, complete function-table signature, relation-column layout, encoded source formula, grounded Boolean source, and the live descriptor certificate. `checked_fol_sound` carries those bindings through to the original sentence.

FORMAL ARTIFACT - nested finite-FOL specimen The $q = 2$ specimen has a public unary `flip` function, one unary relation, one binary relation, and the sentence $\forall x.\exists y.R_1(\mathrm{flip}(x), y) \wedge R_0(\mathrm{flip}(\mathrm{flip}(x)))$. Its exact row width is $2 + 4 = 6$ and its descriptor has seven constraints. A canonical model trace satisfies the live relation; replacing the certified layout by the empty list is rejected.

BOUNDARY Grounding is exhaustive and can grow exponentially with quantifier depth. Function tables are public compilation inputs in this construction; private functions require a lookup or RAM lowering. The theorem is an exact finite model compiler, not a succinct quantifier protocol.

3.5 Checked optimization with exact savings

Optimization is now proof-producing. `DirectLogicOptimizerCertificate.Rule` contains only audited identities, nonempty-fold elimination, factorization, and explicit fanout sharing. A local claim binds the proposed before term, after term, rule payload, and side condition. `checkLocal` rejects a changed endpoint or a false guard. Composite certificates additionally check that the target of one step is the source of the next.
For every accepted certificate, the formal results are simultaneous:

- `Certificate.check_sound` preserves source truth;
- `Certificate.accepts_iff` preserves the exact field `BoolGraph` relation;
- `Certificate.check_cost_nonincrease` proves componentwise nonincrease of equations, multiplications, witnesses, maximum degree, and dense table cells;
- `optimize_checked` proves that the deterministic recursive optimizer always emits a valid certificate; and
- `optimize_accepts_iff` and `optimize_cost_nonincrease` expose the end-to-end semantic and cost contracts.

DirectLogicBoolGraphDescriptorIR2 realizes that ledger in the live backend. Every atomic zero test and Boolean connective receives explicit auxiliary columns and a flat list of windowGate constraints. Its standalone public variant additionally binds residual input a to public input a on the first row. public_descriptor_exact_overhead proves that statement binding adds exactly n linear equations and n public inputs, but no nonlinear multiplication or auxiliary witness.
For the four-input factor specimen, public_factor_emitted_exact proves the public statement ledger. The underlying graph equations are $30 \to 13$; the internal accepting descriptor is $31 \to 14$; after four public bindings the standalone public totals are:

Live resource	Before	After	Exact account
public constraints	35	18	four PI bindings, graph, and output-equals-one
nonlinear multiplications	26	13	exact emitted node ledger
auxiliary witness columns	17	8	postorder zero-test and connective witnesses
total trace width	21	12	four input residual columns plus auxiliaries

descriptor_constraint_degree proves that every emitted constraint has degree at most two. public_statement_sound proves arbitrary-trace soundness: any nonempty satisfying trace implies Formula.Holds under the verifier's public zero-residual vector. public_canonical_trace_complete constructs the public trace for every true Boolean assignment, and public_canonical_trace_iff packages canonical exactness. checked_public_statement_sound transports any accepted optimized public certificate back to its original source formula.
checkPublicLive reconstructs the optimizer proof, source and target endpoints, PI layout, complete resource ledger, and exact emitted descriptor JSON. The factor guards reject changed PI order, layout, or bytes.

3.5.1 Production-shaped DREGG decisions

DirectLogicDreggWorkloads instantiates the public compiler on four decisions already consumed by DREGG: turn admission including receipt-head binding, authorized program upgrade, structured credential clearance, and the stake/vouch strand gate before finality. Each namespace proves an exact source equivalence, an optimized equivalence, and arbitrary-trace public-descriptor soundness: for example AdmissionWorkload.source_iff_admissible, AdmissionWorkload.optimized_iff_admissible, and AdmissionWorkload.public_descriptor_sound.

Named workload	Public constraints	Multiplications	Trace width	Exact theorem
Turn admission	121 -> 71	108 -> 58	77 -> 47	admission_exact_live_ledger
Program upgrade	33 -> 23	28 -> 18	21 -> 15	upgrade_exact_live_ledger
Credential clearance	33 -> 23	28 -> 18	21 -> 15	clearance_exact_live_ledger
Strand admission	17 = 17	13 = 13	11 = 11	strand_exact_live_ledger

The first three sources expose genuine duplicated branch prefixes; the optimizer shares each prefix.
strand_already_normal_shape is the negative control: the already minimal three-way disjunction is unchanged. All four public certificates pass their byte/layout gates, while independent guards reject modified bytes, layout, endpoint, and version.

BOUNDARY The workload theorem is exact at the **atom boundary**. Arithmetic, hashes, membership, lifecycle, capability lookup, and receipt comparison remain the production predicates named by the atoms; this Boolean-skeleton module does not arithmetize their implementations. public_certificate_sound_of_atoms therefore requires the caller to prove that each public zero residual is equivalent to its named atom. Under that hypothesis, any nonempty satisfying trace proves the production decision. The tables count the public Boolean relation around those atoms, not the uncharged cost of implementing them.

BOUNDARY These are exact live DescriptorIR2 resource reductions: constraints, nonlinear multiplication nodes, auxiliary columns, width, and degree. They are not timings. Prover time, verifier time, proof size, memory traffic, and backend scheduling still require a controlled before/after benchmark.

3.6 Exact presentations and finite search

`CertifiedPresentationChange.Presentation` packages source meaning, evidence, acceptance, and a resource vector. An exact `Change P Q` maps evidence in both semantic directions and carries a compositional resource bound. Identity and composition form a genuine category, so a compiler can change representation without dissolving its proof obligation. The original concrete nodes are:

- proof-carrying natural no-wrap residuals;
- exact finite-field Boolean graphs; and
- hybrid evidence selecting either representation independently at each positive subformula.

`HybridEvidence.accepts_iff` proves every such mixture exact. The direct no-wrap-to-hybrid arrow agrees observationally with the route through a Boolean graph. `CertifiedRepresentationSearch` lifts these nodes to a finite graph of typed exact edges. Paths compose to exact changes, and their symbolic overhead composes with them. `shortestEnumerated_minimal` proves that the selected path has minimum scalar score among the caller-supplied candidates; `shortestEnumerated_composite_exact` proves that selection cannot change acceptance.

`CertifiedIndexedPresentationCompiler` closes the accounting gap left by meta-level split acceptance. A `Fragment` is indexed by its source formula, truth interpretation, and one of four distinct modes: dense polynomial, materialized Boolean graph, interaction net, or natural residual. It carries an acceptance proposition, `BackendCost`, and an exact semantic certificate. A `Conversion` combines an exact presentation change with a separately inspectable `ResourceMorphism`; sequential and tensor composition accumulate conversion overhead explicitly.

An indexed `Plan` may choose one mode for a whole subformula or compile its children independently. Every conjunction or disjunction node carries an explicit `glueCost`. `Plan.accepts_iff` proves global semantic preservation, while `Plan.materialization_overhead_eq_resources` proves that the recursive materialization derivation exposes the entire ledger with no remainder. `Boundary.fourWayPlan_modes` exhibits all four modes in one genuine per-subterm plan.

The teeth are negative as well as positive. `Boundary.free_glue_claim_is_false` proves that a materialized one-equation conjunction is not bounded by the zero-cost meta-level estimate. `Boundary.exact_semantics_but_no_free_conversion` gives two semantically exact fragments for which no zero-overhead resource morphism can materialize the target equation. Thus semantic equivalence cannot manufacture a performance claim. `Search.choose_minimal_relative` still proves optimality only within the caller-supplied finite candidates and transparent scalar score, never among an unstated universe of compilers.

BOUNDARY The indexed calculus certifies accounting structure; backend adapters must still construct the dense-polynomial, BoolGraph, interaction-net, or residual fragments they advertise. Only the residual and Boolean adapters are instantiated here. Representation names alone do not emit code.

WHAT HAS BEEN RECOVERED The defensible performance thesis is local and certified: exploit a compact residual exactly where a range proof makes it sound; use Boolean zero tests where it does not; share repeated structure; and search only among meaning-preserving changes of presentation. The checked specimens now show substantial live relation-size reductions, but no theorem here implies a universal 10-100x end-to-end speedup or chain finality.

SECTION 4

Higher-order logic and certified change of presentation

There is no single object called “the higher-order arithmetization.” The exact result is a map of several constructions, together with a compiler calculus that may select among them **per subterm** without erasing either meaning or cost.

- Finite extensional HOL has a complete polynomial semantics, but arrow values are dense tables.
- A small category has a faithful family of finite-field polynomial Yoneda actions exactly when every hom-set is finite.

- Every small CCC embeds fully faithfully in a presheaf CCC by Yoneda, preserving its chosen exponentials up to explicit isomorphism. This is semantic, not a finite-field encoding.
- Typed STLC compiles to shared nets in a genuine CCC and now reaches a live DescriptorIR2 receipt statement with exact width, hash, and depth boundaries.
- A certified indexed compiler can choose dense-polynomial, Boolean-graph, interaction-net, or natural-residual fragments independently, then charge the conversions and glue that join them.

The fifth item is the synthesis. Logic is not forced through one privileged syntax. A term may use whichever presentation has a proved semantic certificate and an honest backend ledger. The resulting theorem is compositional, but it is not a claim that every listed backend is already equally executable or equally fast.

FORMAL ARTIFACT - CERTIFIED PER-SUBTERM PRESENTATION

`CertifiedIndexedPresentationCompiler.Plan.accepts_iff` proves source semantics for every recursively mixed plan. `Plan.materialization_overhead_eq_resources` proves that its derived materialization morphism exposes the **entire** five-axis ledger: equations, multiplications, witnesses, maximum degree, and table cells. `ResourceMorphism.tensor_comp_interchange` proves that independent child conversions compose. These 28 kernel-clean keystones are a calculus of certified presentation change, not a latency theorem.

4.1 The compiler: choose locally, certify globally, charge the glue

A `Fragment truth formula mode` contains three inseparable pieces: an observable acceptance proposition, a `semanticCertificate` identifying it with the indexed source formula, and a `BackendCost`. The four mode tags remain distinct even when their acceptance propositions are logically equivalent:

$$\text{dense polynomial} \quad \text{Boolean graph} \quad \text{interaction net} \quad \text{natural residual}.$$

A `Plan.whole` stops at one certified fragment. `Plan.and` and `Plan.or` instead compile their children independently and then add a caller-supplied recombination ledger. Consequently the resource recurrence is

$$C(\text{left} \diamond \text{right}) = C(\text{left}) + C(\text{right}) + C(\text{glue}),$$

where addition is componentwise except that maximum degree uses `max`. `Plan.and_resources` and `Plan.or_resources` expose this recurrence literally. `Plan.accepts_iff` proves exact semantics by induction over any mixture of modes.

The cost evidence is itself compositional. A `ResourceMorphism source target` contains an explicit overhead and proves

$$\text{target} \le \text{source} + \text{overhead}.$$

Identity has zero overhead; sequential changes add overheads; independent changes tensor; and `ResourceMorphism.tensor_comp_interchange` proves the interchange law. The plan first tensors the child materializations and then applies `ResourceMorphism.charge` for recombination. The equality `Plan.materialization_overhead_eq_resources` rules out an untracked remainder.

Two existing direct-logic implementations instantiate the interface concretely. `DirectLogicAdapter.naturalResidual` consumes a no-wrap certificate; `DirectLogicAdapter.booleanGraph` consumes the materialized inverse-witness graph; and `DirectLogicAdapter.residualToBoolean_acceptance` proves their exact semantic conversion. Because no tighter conversion-cost theorem is yet supplied, that adapter conservatively charges the complete Boolean target ledger. The dense and interaction modes are valid fragment interfaces, and the four-way plan witnesses per-subterm mixing, but a semantic `Fragment` alone does not emit either backend.

This distinction is enforced by two counterexamples. In `Boundary.free_glue_claim_is_false`, two zero-cost semantic atoms still require a one-equation conjunction recombiner, so the materialized plan is not bounded by a zero ledger. In `Boundary.exact_semantics_but_no_free_conversion`, source and target accept exactly the same proposition, but no zero-overhead resource morphism can produce a one-equation target from a zero-cost source. Logical equivalence does not manufacture backend work for free.

`Search.choose_minimal_relative` is deliberately finite-list-relative: it minimizes the declared score only among the candidates a caller supplied. It proves neither global compiler optimality nor that the score predicts wall-clock proof time.

BOUNDARY The indexed compiler certifies a change of presentation only after a backend has supplied a `Fragment`, any conversion morphism, and the recombination charge. It does not infer a live circuit, FHE program, commitment, or cost bound from semantic equivalence. Today the residual and Boolean-graph adapters are direct; the other backend connections remain separately checked constructions.

4.2 Finite extensional HOL: complete, exact, and dense

Fix a finite field F of cardinality q. For a finite coordinate type σ, write F^σ for $\sigma \to F$. Every function from one finite field power to another is represented coordinatewise by a polynomial function. For one scalar output, the explicit interpolation formula is

$$\delta_{a(x)} = \prod_{i \in \sigma} \big(1 - (x_i - a_i)^{q-1}\big), \quad p_{f(X)} = \sum_{a \in F^\sigma} f(a)\delta_{a(X)}.$$

Fermat's finite-field law gives $\delta_{a(a)} = 1$ and $\delta_{a(x)} = 0$ for $x \neq a$, so $p_{f(x)} = f(x)$ at every field point. `eval_interpolate` and `eval_interpolateVector` prove the scalar and vector forms. These are polynomial **functions**: $X^q - X$ is generally nonzero as syntax and zero as a function on F.

The finite HOL construction uses types generated by unit, scalar, product, and arrow. A type A has a finite coordinate type C_A and value space $[A] = F^{C_A}$. An arrow has coordinates

$$C_{A \to B} = \big(F^{C_A}\big) \times C_B,$$

so an arrow value is its complete input-output table. Lambda fills the table; application reads it. The intrinsically typed term language validates product beta, function beta, and extensional eta. Its formulas add equality, Boolean connectives, and universal and existential quantification at every generated type. Quantification at arrow type really ranges over every table.

A typed environment is flattened losslessly by `environmentEquiv`. Every term coordinate has a `termPolynomial`, and every formula φ has a Boolean-valued `formulaPolynomial` satisfying

$$\mathrm{eval}\big(p_\varphi, x\big) = 0 \quad \text{iff} \quad x \vDash \varphi.$$

The headline theorems are `eval_formulaPolynomial_point_isBit` and `eval_formulaPolynomial_point_eq_zero_iff`; forall, exists, and arrow-forall forms are proved separately. This is a complete arithmetization of the defined finite extensional equality-HOL. It is noncomputable truth-table interpolation, not an extracted optimizing compiler.

4.2.1 Exact exponential cost

If $|\sigma| = n$ and $|\tau| = m$, then

$$|(F^\sigma \to F^\tau)| = (q^m)^{q^n} = q^{mq^n}.$$

The exponential object has exactly mq^n field coordinates and q^{mq^n} points. `card_expCoord`, `card_tableObject`, and `card_arrowQuantifierDomain` prove those counts. The blowup is the represented function space, not slack in the analysis. `finite_hom_bound`, `no_injective_infinite_hom`, and the concrete `no_injective_nat_predicates` rule out a faithful injection of an infinite hom-set into one fixed finite table carrier.

4.3 The exact locally-finite family-polynomial frontier

The fixed-carrier obstruction is not the end of categorical arithmetization. The right positive object is a **family** indexed by a probe X and target A:

$$\mathrm{YonedaCoord}(X, A) = \mathrm{Hom}(X, A).$$

Under local finiteness this is a finite coordinate type for each pair. A morphism $f : A \to B$ acts by postcomposition $a \mapsto f \circ a$. In one-hot coordinates, output coordinate b is the linear polynomial

$$\mathrm{Act}_{f,X,b}(x) = \sum_{a : X \to A} [f \circ a = b] x_a.$$

`actionPolynomial_totalDegree_le_one` proves formal total degree at most one, and `eval_actionPolynomial` proves exact evaluation on every valid code. This is linear because f is fixed. When both composands vary, the earlier one-hot composition polynomial remains bilinear of degree at most two.

`FamilyPolynomialCategoryPresentation` couples faithful hom encodings to those polynomial actions. Its sharp classification is

$$\text{family-polynomial presentation exists} \quad \text{iff} \quad \text{every hom-set is finite},$$

proved by `familyPolynomialPresentation_nonempty_iff_locallyFinite`. `polynomialAction_id_onCode` and `polynomialAction_comp_onCode` prove the categorical laws on valid codes. `polynomial_identityProbe_iff` shows that two parallel arrows are equal exactly when their polynomial Yoneda outputs agree at the single identity probe. The 29 kernel-clean frontier keystones classify this family construction and its limits.

The polynomial family and semantic Yoneda are the same postcomposition operation: `polynomialAction_is_yoneda` states the equality. For a small CCC, `familyYonedaExponentialIso` then preserves its chosen exponentials in the presheaf CCC. Thus every **small, locally finite** CCC has both a faithful family of finite polynomial actions and a fully faithful semantic embedding preserving CCC structure. The presheaf category and the family of probes need not be finite.

4.3.1 Why one fixed width still fails, even for finite sets

Family-wise finiteness does not imply a uniform circuit. A `UniformIndexedHomPresentation` permits pair-dependent coordinate types but bounds all their cardinalities by one `budget`. `UniformIndexedHomPresentation.hom_card_le` then forces

$$|\mathrm{Hom}(A, B)| \leq q^{\mathrm{budget}}$$

for every pair of objects.

The CCC of finite sets is the sharp counterexample. `FiniteSetCCC.laws` constructs its terminal object, products, evaluation, curry, and uncurry. Every individual hom-set is finite, yet maps from the singleton to an n-element set have cardinality exactly n (`card_singleton_hom_fin`). Choosing $n = q^{\mathrm{budget}} + 1$ contradicts the uniform bound. Therefore `finite_sets_family_yes_uniform_no` proves both sides at once: the exact family-polynomial presentation exists, but no globally fixed coordinate budget can remain faithful.

This is stronger than the earlier infinite-hom obstruction. Even a locally finite CCC may require unbounded widths across its family. `EffectiveLocallyFiniteHom` is also extra data: proposition-valued local finiteness alone does not provide an enumeration or decidable equality for executable polynomial emission. Finally, `committed_identityProbe_sound` consumes a separately supplied binding commitment; Yoneda and interpolation do not manufacture cryptographic binding.

> **FORMAL ARTIFACT - CCC FRONTIER, EXACTLY** Arbitrary small CCC: fully faithful semantic Yoneda with chosen exponentials preserved. Small locally finite CCC: additionally, an exact family of finite-field polynomial representable actions. Uniform fixed-width family: impossible in general, already for the CCC of finite sets. Effective emission and cryptographic binding: additional structures, not consequences of those categorical theorems.

4.4 The intensional route: shared computation instead of arrow tables

Dense tables are not the only semantics for higher-order programs. The intensional development starts from intrinsically typed STLC with unit, natural numbers, products, arrows, and explicit source `let1`. Its target `Net` language adds a sharing binder. Beta compilation performs

$$(\lambda x.b)a \quad \longrightarrow \quad \text{share } a \text{ as } x \text{ in } b,$$

so the argument is represented once and referenced wherever the variable occurs. `compile_denote`, `localStep_denote`, and `trace_denote` prove compilation and every typed local trace denotationally exact. Product beta and extensional eta have their own local certificates.

The quantitative specimen binds an eight-operation computation once and uses it twice. Its shared target contains 8 work nodes and 13 total nodes; literal copying contains 16 work nodes and 19 total nodes. Beta costs one local rewrite. The shared cost is independent of q, while the matching one-input, two-output extensional table has $2q$ coordinates; `shared_smaller_than_table_coordinates` proves the shared graph smaller from $q \geq 7$. This is a checked structural comparison, not a prover latency benchmark.

`IntensionalCCCCategory` constructs genuine source and shared-net CCCs. Identity, composition, terminal arrows, pairing, evaluation, curry, and uncurry have explicit closed-net representatives, and `Shared.laws` proves their laws. The structural compiler induces `compilerFunctor`; `Compiler.strongCCC` proves preservation of identity, composition, terminal objects, products, evaluation, currying, and uncurrying. Equality is denotational equality in a represented image/quotient, not a decidable syntactic rewrite equivalence or the universal property of the free CCC.

4.4.1 A live DescriptorIR2 receipt, with an exact erasure boundary

The former interaction bridge stopped at an executable Boolean receipt. `IntensionalCCCInteractionDescriptorIR2` now emits a one-row live descriptor for the receipt's root-rule tag, complete address path, two domain tags, and two endpoint digests. At address depth d, `descriptor_shape_exact` and `cost_exact` prove:

Quantity	Exact value
trace columns / public inputs	$d + 5$ / $d + 5$
linear PI constraints	$d + 5$
hash sites / absorbed felts	2 / $2(d + 2)$
maximum arithmetic-gate degree	1

`trace_complete` constructs a satisfying DescriptorIR2 trace for every typed local step. Under a nonempty row and canonical BabyBear representatives for the layout tags, `satisfying_layout_words_exact` binds an arbitrary satisfying row to the exact root and address. `satisfying_trace_relation_sound` composes that binding with the receipt checker and returns two deliberately

separate facts: the Boolean interaction checks, and the original typed denotation is preserved. `binder_side_tamper_refused` rejects the concrete one-cell `shareBody` to `shareValue` mutation.

The shared live hash table absorbs at most 16 felts per site. Because each endpoint site absorbs $d + 2$, `max_live_depth_exact` gives the direct one-site boundary $d \leq 14$; deeper addresses require a multi-block sponge lowering. The depth-one wire guard fixes 6 public inputs, 6 trace columns, 6 PI constraints, 2 hash sites, and 6 absorbed felts. `beforeCommit_binds` additionally requires the named `Poseidon2SpongeCR` premise. The descriptor theorem itself does not discharge that cryptographic assumption.

Most importantly, the endpoint digest commits the **receipt layout**, not the full typed source nets. `layout_forgets_typed_endpoints` exhibits beta redexes containing literals 1 and 2 with the same layout; `receipt_backend_cannot_distinguish_typed_endpoints` propagates the collision; and `no_exact_typed_source_decoder` proves that no decoder from the erased layout can recover both exact sources. The 27 kernel-clean descriptor keystones therefore close the live receipt boundary while proving why a later typed syntax commitment must add new statement data.

> **BOUNDARY** A live receipt statement is not a complete higher-order program proof. It binds one local rule/address receipt up to the stated hash premise, but it does not bind the typed syntax erased by `layout`, encode an entire interaction trace as a robust code, or supply commitment, Fiat–Shamir, FRI, or end-to-end proof-system soundness.

4.5 What has, and has not, been generalized

Construction	Exact result	Boundary
Finite extensional HOL	Every defined formula has an exact Boolean polynomial.	Complete arrow tables; finite carrier; dense quantification.
Family-polynomial Yoneda	Exists exactly for locally finite categories; linear fixed-arrow action.	Family-indexed and potentially unbounded; effectiveness is extra.
Semantic Yoneda	Fully faithful for every small CCC; chosen exponentials preserved.	Presheaf semantics, generally infinite; no arithmetic backend or succinctness.
STLC to shared nets	Genuine CCCs, strong-CCC compiler, live local receipt descriptor.	One intensional fragment; receipt layout erases typed endpoints.
Indexed presentation compiler	Per-subterm semantic preservation with compositional charged resources.	Backend adapters and glue costs must be supplied; search is list-relative.
Fixed-meaning evidence fibre	Pointwise CCC of evidence transformers.	The global presentation category is not cartesian; zero-resource laws are not cost laws.

An arithmetic backend for an arbitrary CCC would still require a computable assignment of objects and arrows, a stated equality notion, faithfulness, CCC-structure preservation, and honest resource bounds for composition, evaluation, curry, and equality checking. The development now gives a semantic solution for all small CCCs, an exact arithmetic classification for the locally finite ones, a sharp uniform-width no-go, and a certified way to combine selected presentations. It does not collapse those four results into one impossible universal circuit.

SECTION 5

Proof objects beyond polynomials: realizability, coverage, and transcript soundness

The corrected arithmetic constructions answer how to present a chosen judgment to a chosen backend. They do not make first-order syntax, polynomial syntax, or a finite field the primitive notion of proof. A stronger architecture starts with meaning, chooses evidence, changes its presentation with explicit programs and costs, gives the evidence a local coverage structure, and only then binds and samples it. Each transition now has a named theorem or an explicitly open cryptographic socket.

Layer	Mathematical question	Formal construction here	Deployment obligation
Meaning	What does the judgment assert?	finite semantics; realizability predicates; resource-sensitive denotations	application-specific atoms, effects, and observation policy
Evidence / presentation	What witnesses the judgment, and in what representation?	PCA realizers; assembly morphisms; interaction traces; per-subterm certified changes	executable adapters and explicitly charged glue
Coverage / agreement	When do local views detect invalidity or determine one global object?	effective assembly covers; exact descent; finite coverage presentations	a code-specific distance and local-testability theorem
Transcript	How can hidden checks miss an obstruction?	natural pullback of protocols; exact miss counts; BabyBear query-bias bound	OOD and low-degree reductions with concrete parameters
Binding	Why can no response rewrite committed evidence?	an explicit point-binding premise	a commitment or PCS security reduction and Fiat–Shamir analysis

$$\text{judgment} \to \text{evidence} \to \text{presentation} \to \text{local coverage} \to \text{hidden queries} \to \text{bound transcript}.$$

The categorical semantics explains what evidence, effective cover, and compatible restriction mean. The finite probability game explains what a sampled verifier can miss. The cryptographic backend explains why the prover cannot change an unseen view after learning the query and why a transcript seed has the modeled distribution. None of those explanations subsumes the others.

5.1 Realizability is a semantics of evidence, not a field trick

The formal base is a relational partial combinatory algebra. `PCA.App f x y` says that applying f to x is defined with unique result y; the K and S laws are stated directly over that relation. The derived composition program is exact: `PCA.app_compose_iff` characterizes precisely the two consecutive applications it performs. An indexed predicate over I is a family $I \to \mathrm{Set}(A)$, and an entailment is witnessed by one realizer that transforms every member at every index. `entails_refl`, `entails_trans`, and `entails_reindex` prove the indexed preorder and strict substitution action.

Chosen computable pairing, sums, and currying give the expected Heyting operations. The formal universal properties are `entails_conj_iff`, `disj_entails_iff`, and `entails_imp_iff`; truth, falsity, and preservation under reindexing are proved alongside them. The construction does not assume that sum tags are injective or that application is globally total.

There is a useful finiteness warning. Under the usual nontrivial definition, a PCA cannot be a finite algebra. The Lean development includes `PCA.unitPCA`, a collapsed one-point inhabitant of the law interface, but does not advertise it as a computational realizability universe. Terwijn surveys the standard infinitude boundary in *Computability in Partial Combinatory Algebras*. A finite-field backend should encode a finite execution trace or certificate about a nontrivial PCA, not pretend to put the entire PCA into a small field carrier.

5.1.1 Quantifiers and the empty-fiber test

Existential quantification is ordinary union over a map fiber and is a left adjoint for every map (`existsAlong_adjunction`). Ordinary intersection gives the direct right adjoint only for surjections (`forallAlong_adjunction_of_surjective`). At an empty fiber, a pointwise premise supplies no uniform tracker and so cannot justify a defined application.

`forallReal` repairs exactly that failure. Its evidence is a suspended pair (e, x); the application ex is demanded at each point that actually lies over the target index and is not demanded for an empty fiber. Consequently `forallReal_adjunction` proves the right-adjoint law for every set map, while `existsReal_adjunction` gives the matching arbitrary-map left adjoint. `existsReal_beckChevalley` and `forallReal_beckChevalley` prove literal Beck–Chevalley equalities for weak pullback squares. The regression theorem `forallReal_empty_equiprovable_truth` confirms that an empty-fiber universal is equiprovable with truth.

The uniform-family presentation packages these results without changing their meaning. `pcaUFam_tracks_iff` and `pcaUFam_entails_iff` identify it exactly with the PCA doctrine. `pcaUFamTriposLaws` collects the Heyting preorder, strict reindexing, both arbitrary-map adjoints, Beck–Chevalley, exact generic classification, and noncollapse. In particular, `pcaUFam_truth_not_entails_falsity` proves consistency already at a singleton index.

FORMAL ARTIFACT - tripos boundary A Set-indexed realizability tripos-law package has been proved. The associated elementary topos has not. The development does not yet construct the exact completion, a subobject classifier, or an equivalence between that completion and the topos associated with `pcaUFamTriposLaws`. Pitts' tripos-to-topos account describes the classical route citation is not a substitute for the missing Lean construction.

5.1.2 Assemblies now carry effective regular coverage

`Assembly` pairs a semantic carrier with a nonempty realizability relation. An assembly morphism is an actual semantic function tracked uniformly by one PCA element. `categoryAssembly` proves genuine category laws. Terminal objects, binary products, and equalizers were already explicit; `AssemblyRegularCoverage` adds pullbacks with their full universal property through `pullback_condition`, `pullbackLift_fst`, `pullbackLift_snd`, and `pullbackLift_unique`.

The new cover notion is evidence-relevant. `EffectiveCover q` does not merely say that q is surjective. It supplies one `liftCode` which, from any realizer of a target point, computes a realizer of a source point above it. This is exactly the program needed for constructive base change. Theorems `effectiveCover_id`, `effectiveCover_comp`, and `effectiveCover_pullback` prove identity, composition, and pullback stability, while `effectiveCover_surjective` recovers ordinary semantic surjectivity.

The coverage is regular rather than decorative. `effectiveCover_isKernelPairCoequalizer` proves that every effective cover is the coequalizer of its explicit kernel pair. Every assembly morphism f also factors as

$$X \overset{\text{effective cover}}{\to} \mathrm{im}(f) \overset{\text{monic}}{\to} Y.$$

Here `imageQuot_effectiveCover` supplies the cover, `imageIncl_monic` supplies exact cancellation, and `image_factorization` proves their composite is f. `imageQuot_isKernelPairCoequalizer` gives the coequalizer law for the quotient.

`ProgramCertificate.has_regular_image_factorization` transports the result directly to compiled PCA programs. The 29 kernel-clean coverage keystones therefore construct the regular-image ingredients of an exact completion, not merely a metaphor about local evidence.

Modest assemblies still induce partial equivalence relations on realizers, and `tracker_respects_realizerPER` proves their transport by uniform trackers. At the compiler boundary, a `ProgramCertificate` contains representations, executable PCA code, and its run theorem; `ProgramCertificate.hom` turns it into the morphism to which the new factorization applies.

BOUNDARY Effective regular coverage is not yet the realizability topos. The development does not prove that all internal equivalence relations are effective, construct the exact completion, or establish elementary-topos axioms. Nor does a cover or program certificate imply transcript soundness. The concrete theorem `compiled_regular_image_does_not_supply_query_soundness` exhibits a certified constant-false program with its effective regular image alongside an always-accept attestation that is complete but range-unsound.

5.2 Presentation change is fibred and resource-aware

The resource-aware `Presentation` interface records source meaning, evidence, acceptance, and a five-axis backend cost. An exact `Change` maps evidence, preserves meaning and acceptance in both directions, and carries a compositional cost bound. These changes form a category. Section 4′s `CertifiedIndexedPresentationCompiler` now makes this operational per subterm: child changes tensor, then recombination is charged, and `Plan.accepts_iff` closes the source-semantics induction.

The global category is deliberately not cartesian closed. Its objects may carry different meanings, while an exact arrow requires those meanings to be equivalent. `no_weakTerminal` proves that no object can receive arrows from both the true and false presentations; hence `no_global_cartesian_structure`. Likewise, the obvious presentation of logical conjunction generally has no categorical projections: `firstProjection_forces_right` and `secondProjection_forces_left` show that such projections would force implications between the two meanings.

The positive replacement is fibred. Over one fixed meaning, `EvidenceFamily` objects vary only the evidence type. Pointwise evidence transformers form a genuine cartesian closed category: unit evidence is terminal, paired evidence is product, and evidence transformers form exponentials. `EvidenceFamily.laws` packages product beta/eta and exponential beta/eta. This complements the indexed compiler without changing the boundary: the exponential is a space of evidence transformers inside one semantic fibre, not a finite-field table of every source morphism. Likewise, `Boundary.free_glue_claim_is_false` proves that a zero-resource semantic fibre does not justify zero-cost materialized recombination.

5.3 Exact descent: compatible local evidence glues uniquely

ProofObjectDescent isolates the backend-neutral local-to-global theorem. A DataCover includes a computational locator. Matching says local sections agree on every pairwise overlap, and glue chooses the located local value pointwise. The central theorem matching_iff_existsUnique_amalgamation proves

$$\text{all overlaps match} \quad \text{iff} \quad \exists! g,\ \text{every local view restricts from } g.$$

incompatible_no_amalgamation proves the refusal direction. verifiedEvidence_descends applies the same theorem to subtype-valued evidence $\{w : W_x \mid \mathrm{Checks}(x, w)\}$, so the glued global witness retains its local verification proofs. This theorem contains no polynomial, field, hash, commitment, random oracle, or FRI premise.

Exact descent checks every relevant overlap. RobustProofObjectDescent exposes what sampled checking additionally requires:

- WeightedQueryModel permits an arbitrary finite weighted query distribution;
- RejectLowerSoundness and DistanceUpperSoundness carry explicit, possibly nonlinear robustness moduli;
- FoldChainSoundness separates distance preservation from weighted exceptional challenges; and
- ProximityGap bounds challenges that transform a far word into a close word.

These are theorem sockets, not assumed code theorems. Their soundness, exceptional weight, and gap fields must be instantiated for the deployed graph, sheaf, HDX, or FRI code. Two finite instances show that the interfaces are nonvacuous. FinitePatchCode.reconstruct_codeword reconstructs the unique global codeword from matching valid patches. The three-patch WeightedStar theorem rejection_eq_hubDistance gives exact equality between rejected weight and distance to the hub-selected codeword; its one-error example proves coefficient one sharp and refutes a false factor-two bound.

5.4 Coverage presentations make query change natural

Assembly covers describe executable lifting of semantic evidence. A second, independent categorical structure now describes how a verifier presents local obstructions. A CoveragePresentation Family Invalid contains a finite query type, a decidable predicate bad family query , and a detection theorem saying that every invalid family has at least one bad query. It contains no commitment, polynomial, transcript, or sampler assumption.

A morphism $P \to Q$ interprets each query of Q as a query of P and preserves **and reflects** bad . The query map is therefore contravariant. These morphisms form a category. CoveragePresentation.obstruction_natural proves the naturality square, and obstruction_natural_comp proves it through successive presentation changes. Finite bad-set membership is natural as well, but cardinality need not be preserved: a query interpreter may duplicate a source query.

The protocol layer follows the same variance. PresentedProtocol.pullback reindexes an opening scheme and local test without changing its commitment relation. PresentedProtocol.pullback_comp proves that two such changes equal pullback along their composite. misses_natural and missEvent_natural prove exact equality of the pointwise and seed-level miss events. This event equality is the safe invariant; silently comparing bad-set cardinalities would be false for duplicating maps.

The realizability connection is explicit but limited. RealizabilityLayer.obstructionPredicate_natural proves reindexing naturality for the obstruction predicate in the existing PCA doctrine, and obstructionPredicate_classified classifies it by the existing generic predicate. If a query interpreter additionally has one uniform PCA tracker, TrackedPresentationMap.comp composes those interpreters as assembly morphisms. Coverage naturality alone does not manufacture that tracker or an associated topos.

> **FORMAL ARTIFACT - CATEGORICAL QUERY NATURALITY** The 37 kernel-clean keystones of CategoricalDescentQueryProtocol prove a category of finite coverage presentations, compositional protocol pullback, exact naturality of obstruction and miss events, a realizability reindexing law, and a factored soundness interface. Presentation change adds no hidden cryptographic error term because it proves event equality. Commitment binding, transcript sampling, OOD reduction, and low-degree reduction remain separate inputs.

5.4.1 Soundness factors into four independently priced events

For a computational commitment, BindingFailureAtSeed records the exact bad event: some accepted opening differs from the view fixed by the commitment. locallyAccepts_misses_or_bindingFailure proves that locally accepted openings either miss all precommitted obstructions or exhibit that failure. The event inclusion acceptSeeds_subset_four_events therefore places every accepting seed in one of four classes:

1. an OOD exception;
2. a low-degree or folding exception;
3. a binding failure; or
4. a sampler miss.

acceptSeed_card_le_budgets_with_binding gives the exact finite union bound. factored_soundness_with_binding_budget combines it with coverage detection and a SamplerBound , yielding

$$\text{accepting seeds} \le \varepsilon_{\text{OOD}} + \varepsilon_{\text{LDT}} + \varepsilon_{\text{bind}} + \varepsilon_{\text{miss}},$$

where the displayed terms are finite seed counts in this theorem, not silently normalized probabilities. factored_soundness is the perfect PointBinding specialization and removes the binding-failure addend. A computational deployment must supply a reduction translating its commitment advantage into the corresponding budget.

SamplerBound.independentUniform supplies the exact information-theoretic reference for k uniform queries with replacement. It does **not** prove that Fiat–Shamir or a random oracle produces that seed distribution. For the transparent reference, transparent_uniform_acceptance_card proves the exact count $|\text{good queries}|^k$. In the concrete finite-patch instance, WeightedStarInstance.oneError_uniform_one_query_strict proves that one uniform query has strictly fewer accepting seeds than the complete-overlap query space. This is deliberately the **unweighted** complete-overlap sampler; it does not flatten the weighted-star example's separate edge weights into the theorem.

> **BOUNDARY** Categorical naturality says that exact changes of query presentation preserve the obstruction and miss events. It does not prove commitment binding, Fiat–Shamir or random-oracle soundness, an OOD polynomial identity bound, FRI proximity, or the adversarial seed distribution. Those are the explicitly separate premises of factored_soundness_with_binding_budget and the transcript theorems below.

5.5 Hidden-query soundness: price the obstruction that was missed

Exact gluing does not imply that spot checks deterministically extract a global object. The corrected statement is probabilistic and keeps the committed family visible. If a finite section family has no amalgamation, badEdges_nonempty_of_no_amalgamation proves that its obstruction set is nonempty. For a query space of size Q, an obstruction set of size B, and k independent with-replacement queries, missSamples_card_sub proves the exact count

$$\text{misses} = (Q-B)^k, \quad \text{all samples} = Q^k.$$

Thus the exact uniform miss probability is $\left(1-\frac{B}{Q}\right)^k$. A bad committed object can pass on precisely this event; acceptance alone does not yield a total column.

QueryOpeningScheme separates committed family, digest, view, opening relation, and opening proof. PointBinding states the anti-rewriting property actually consumed by the verifier game: after committing, any accepted opening at a query equals that query's committed view. The responder may otherwise be adaptive. acceptsSample_misses proves that a bound responder accepted on every sampled query must have missed every obstruction, and acceptedSamples_card_le transfers the exact miss count to an acceptance bound. invalid_gluing_acceptance_bound specializes the statement to overlap inconsistency. In the transparent reference game, two of four overlap edges are bad; two queries miss them in exactly four of sixteen samples, and the accepted-sample count is at most four.

> **FORMAL ARTIFACT - query-game correction** The proved conclusion is a finite event inclusion and count, not deterministic extraction. Point binding prevents post-query rewriting; it does not make hidden queries hit every obstruction. A computational deployment must add the adversary's advantage in breaking its concrete commitment's binding game.

5.6 Full transcript composition, including BabyBear bias

RobustDescentTranscriptGame makes the protocol order explicit:

$$\text{commit family} \to \text{sample transcript seed} \to \text{derive challenges and queries} \to \text{open queried views}.$$

A TranscriptResponder may inspect the entire revealed seed and the query coordinate. Point binding still forces every accepted response to equal the precommitted view (locallyAccepts_misses). The protocol-specific AcceptanceReduction then has one clear job: prove that full acceptance implies at least one of three events:

1. all bound local openings pass;
2. an OOD algebraic exception occurs; or
3. a low-degree or folding exception occurs.

acceptSeeds_subset_exceptional_union_miss proves the corresponding event inclusion. acceptProb_le_union proves the exact finite union bound. It requires no independence between the OOD and low-degree events.

The query sampler is also modeled rather than idealized. A squeeze value in $\mathrm{Fin}(N)$ is reduced modulo a query-domain size m. If the obstruction density is at least δ, `mod_miss_prob_le` gives the conservative term

$$\left((1-\delta)+\frac{m}{N}\right)^k.$$

When $N \bmod m = 1$, only one residue class is heavy. The sharp theorem `mod_miss_prob_le_sharp` improves the term to

$$\left((1-\delta)+\frac{\delta}{N}\right)^k.$$

`biased_transcript_soundness_sharp_extra` allows every non-query transcript coordinate - OOD points, folding challenges, batching challenges, and future coordinates - to remain in an arbitrary finite `Extra` factor. The exceptional predicates and adaptive responder may inspect all of it. The query-marginal probability is unchanged because the uniform extra coordinates product out.

For BabyBear, $p = 2013265921$. For every realizable power-of-two domain $m = 2^\ell$ with $1 \le \ell \le 27$, the proved arithmetic fact is $p \bmod m = 1$. Therefore `babybear_transcript_soundness_sharp_extra` proves the complete bound

$$\Pr[\text{false acceptance}] \le \varepsilon_{\mathrm{OOD}} + \varepsilon_{\mathrm{LDT}} + \left((1-\delta)+\frac{\delta}{2013265921}\right)^k.$$

This is the exact sharp BabyBear expression formalized by the current game. The $\frac{\delta}{p}$ term is the modular-reduction defect; it is not silently rounded away and does not depend on trace height. The finite reference transcript proves that all three addends can be load-bearing: at $N = 5$, $m = 2$, $k = 1$, and $\delta = \frac{1}{2}$, the query term is $\frac{3}{5}$, the OOD and low-degree terms are each $\frac{1}{5}$, and the union bound is attained exactly.

> **BOUNDARY** The BabyBear theorem is conditional on a fixed precommitted invalid object, point binding, the stated obstruction-density premise, and a protocol-specific acceptance reduction. It does not itself prove the deployed commitment binding, an OOD Schwartz–Zippel bound, a FRI low-degree theorem, random-oracle security, Fiat–Shamir soundness, or a total-column extraction theorem. Those premises must be instantiated and their error terms composed before quoting a bit-security number.

5.6.1 What topos theory contributes - and what it cannot contribute alone

Mechanism	It can establish	It cannot establish by itself
Realizability / tripos	meaning, uniform evidence transformation, logical adjoints, substitution	local testability, hiding, commitment binding, proof size
Assemblies / regular coverage	tracked maps, pullback-stable effective covers, kernel-pair descent, regular images	exact completion, a topos, or query soundness
Descent / sheaf condition	unique gluing from complete compatible local data	that a few sampled overlaps expose every inconsistency
Coverage presentation	natural obstruction and miss events under exact query change	binding, Fiat–Shamir, OOD, or FRI budgets
Robust code theorem	distance versus rejected-query weight and proximity gaps	that an adaptive prover is bound to one word
Commitment / PCS	binding or hiding under a stated security game	semantic correctness of the committed proof object
FRI / graph IOPP	a quantitative low-degree or code-proximity test	the source logic semantics or commitment theorem

Non-polynomial quantitative sockets remain possible: sheaf cosystolic codes arXiv:2403.19388, HDX Tanner codes with local Reed–Solomon views arXiv:2308.15563, and graph IOPPs arXiv:2501.14337 arXiv:2508.10510. These are candidate sockets, not theorems about DREGG's deployed parameters. FRI is one established polynomial-code realization ICALP 2018, not the definition of proof-object descent.

5.7 Local interaction traces are proof objects too

The interaction route makes evidence a replayable list of local graph changes. `InteractionNetTrace` uses a finite typed Boolean port tree. An `ActiveRule` contracts one literal/gate interaction, and a one-hole `Context` identifies its location.

activeRule_preserves proves every primitive rewrite; contextualRule_preserves lifts it under arbitrary contexts. LocalCert.check is executable and fail-closed. checkTrace_sound proves denotational preservation for accepted replays, while checkTrace_reduces reconstructs the finite reduction.

LinearInteractionTrace independently tracks resource kind, ticket, polarity, and explicit provenance. Its trace checker preserves the extensional per-ticket observation, total endpoint provenance, each ticket's count, and signed balance. The executable counterexamples reject cross-kind closing, a rule at the wrong location, and silent weakening. These traces are genuine proof objects that need not first be formulas or residual polynomials.

The new live intensional descriptor closes one local backend boundary. For a typed STLC step it emits the lossless root/address receipt, binds every row cell to public inputs, and recomputes two domain-separated receipt-layout digests. trace_complete and satisfying_trace_relation_sound connect a satisfying DescriptorIR2 witness to both executable receipt acceptance and typed denotational preservation. Its exact cost is $d + 5$ columns and public inputs, $d + 5$ linear PI constraints, two hash sites, and $2(d + 2)$ absorbed felts at address depth d.

That one-row descriptor is still not a succinct argument for a complete trace. The live hash-site limit gives $d \leq 14$ without a multi-block sponge, and no_exact_typed_source_decoder proves that the receipt layout has already erased the typed endpoint syntax. The remaining route is therefore precise: add typed-net statement data where required; encode a sequence of local receipts as covered patches; prove exact descent and the resource invariant; instantiate a robust code/proximity theorem; then supply binding, Fiat–Shamir, and FRI or another IOPP through the factored protocol interface.

> **CONSTRUCTION MAP** Realizability supplies a semantics of evidence and quantification. Assembly regular coverage supplies executable lifting, kernel-pair descent, and image factorization. The indexed compiler chooses a presentation per subterm while charging conversion and glue. Exact descent supplies the local-to-global law; coverage presentations make obstruction and miss events natural under query change. Robust code theorems price inconsistency, and the transcript games price missed obstructions and modular query bias. Commitments, Fiat–Shamir, FRI, graph IOPPs, and live interaction receipts instantiate separate layers. No layer requires FOL, hashes, or polynomials to be the primitive notion of logic.

SECTION 6

From finite logic to a live proof and encrypted plan

The constructive result is now a pipeline, not merely a corrected polynomial identity. A finite-signature first-order sentence can be grounded into an exact Boolean relation, compiled into the live DREGG descriptor IR, accompanied by a canonical trace, checked against exact emitted bytes, optimized by replayable certificates, and related to an encrypted evaluation over the same opening. There are executable Rust, Python, Standard ML, HOL4, and CakeML artifacts at different assurance boundaries.

finite FOL → grounded Boolean relation → checked representation plan → DescriptorIR2 + proof
same grounded source → shared opening → BFV/hybrid plan

This diagram is a map of proved and executable interfaces, not one monolithic end-to-end theorem. In particular, a Lean compiler proof, a Rust prover test, and a real BFV execution are three different kinds of evidence. The named boundaries below prevent one from being advertised as another.

6.1 General finite FOL reaches the live descriptor IR

FiniteSignatureFOLDescriptorIR2 is the broadest verified source compiler in this study. A signature contains any finite list of total public function symbols and relation symbols of independently chosen finite arities. Terms contain constants, de Bruijn variables, and nested function applications. Formulae contain equality, relation application, negation, conjunction, disjunction, and exhaustive finite universal and existential quantification.

For a domain of size q, a relation of arity a receives q^a canonical columns. The complete relation layout therefore has width

$$W = \sum_{r \in \text{ relations}} q^{a_r}.$$

Public function tables are compiled by evaluation, while relation tables are the model-bearing trace data. `compileDescriptorFOL` emits exactly W always-on bitness constraints and one always-on source-acceptance constraint. Quantifiers are grounded by exhaustive enumeration, so the construction is exact but can grow exponentially with quantifier depth. It is not presented as a succinct quantifier protocol.

> **FORMAL ARTIFACT - finite-signature live compiler** `lower_correct` relates the direct finite-model semantics to the grounded Boolean source. `fol_sound` proves that every row of a satisfying canonical live trace satisfies the original sentence. `fol_complete` constructs a one-row live trace from every satisfying model, and `canonical_model_trace_iff` gives the exact equivalence. The fail-closed `FOLLiveCertificate` binds the domain and signature, complete public function tables, canonical relation layout, encoded sentence, grounded source, atom count, and exact DescriptorIR2 JSON. `checked_fol_sound` transports a checked certificate and a satisfying trace back to the source sentence.

The concrete nested specimen has a two-element domain, a public Boolean flip function, one unary relation, one binary relation, and the sentence

$$\forall x, \exists y, R_2(\mathrm{flip}(x), y) \wedge R_1(\mathrm{flip}(\mathrm{flip}(x))).$$

Its canonical relation width is $2^1 + 2^2 = 6$ and its live descriptor has seven constraints. A certificate with an empty claimed layout is rejected. This specimen exercises functions, two relation arities, nested quantifiers, and exact live-byte binding in one kernel-checked construction.

The same target also contains a faithful fixed Gabbay matrix instance. `GabbayDescriptorIR2` compiles a two-row, three-entry Skolem graph using denominator-cleared Lagrange interpolation, three linear acceptance atoms, and a 30-bit range lookup on each of the six entry columns: 12 trace columns and 15 constraints. Its `trace_satisfied_iff_holds` theorem relates the actual live relation to the ordinary bounded source formula; successor data is accepted, while a tampered entry and the known unchecked modulus-five projection are refused. Its `WireBoundedTable` hypothesis is a completeness-side premise naming the expressible domain; the soundness direction derives that bound from the emitted lookups instead of assuming it. Arbitrary matrix length remains a separate symbolic compiler problem because it must emit and cost an interpolation algorithm rather than hide divisions in witness data. The fixed theorem constructs a trace from an externally named three-entry table, but the descriptor itself proves only an existential private-witness relation.

`GabbayDescriptorIR2PublicBoundary` proves that its public-input and hash-site surfaces are empty, that the v1 `ranges` carrier is empty, that it carries 15 constraints, and that all constructed tables have the same public assignment. Read the range conjunct exactly: an empty `ranges` field is not an absent range check. The no-wrap bound is carried by six graduated `lookup` constraints against the declared 30-bit range table — the form the deployed assembly actually realizes, and the form it requires, since it refuses a v2 descriptor carrying the v1 carrier at all. What the boundary theorem records is that no surface here binds an external NAME, so acceptance cannot attest an independently named external table. Moreover, interpolation coefficients are consistency columns; acceptance reads the raw input/output cells. This specimen establishes a fixed-shape semantic relation, not an interpolation performance result.

`GabbayDescriptorIR2PublicBinding` now closes that statement gap with a separate direct-table descriptor. Its six disclosed table cells are pinned to six trace columns by six public inputs; three linear acceptance atoms and six 30-bit range lookups bring the total to 15 constraints, and acceptance contains no nonlinear product at all. Under the canonical field-decoding convention alone — the previous no-wrap certificate is now derived rather than assumed — `public_statement_satisfied_iff_holds` proves that an arbitrary nonempty satisfying trace exists exactly when the externally claimed table satisfies the source formula. A one-cell public tamper is refused, and the complete IR-v2 JSON is pinned, including the `"bits":30` range table and the six lookup entries. This construction supplies public integrity with zero table privacy. It deliberately removes interpolation coefficients, so it closes public binding without manufacturing an interpolation speedup.

Both Gabbay descriptors carry this shape because the previous one was proved unsound. Acceptance used to be the sum of the three squared successor residuals, sound only under Lean-side premises that were never serialized, and over BabyBear $284861408^2 = -1$ makes a fully canonical FALSE table cancel to zero. `DirectLogicAdversarialFalsifierV2` keeps that table and a second one — whose only refuting gate is a range tooth — as armed rejection canaries in both the preselected-trace and external-statement polarities, together with a control proving the true statement still has a witness. Section 3 gives the full account, including the one premise that moved rather than vanished: a range lookup bounds a column only against an honest range table, and `Satisfied2` has no faithfulness conjunct for `.range`, so refusals through a range tooth

carry `HonestRangeTable` explicitly. That premise is carrier-level and uniform across every range lookup in the codebase, not specific to this descriptor.

6.2 Certificates optimize the relation without changing it

The important performance mechanism is not "FOL instead of circuits." It is certified selection and sharing among equivalent presentations. `DirectLogicOptimizerCertificate` is an executable translation validator for a bounded Boolean fragment. Each local certificate contains redundant before and after terms plus a rule payload. The checker reconstructs both endpoints, checks the rule guard, checks every transitive boundary, and rejects tampered endpoints or disconnected certificate chains.

Accepted certificates preserve three meanings at once:

- source truth under every atom interpretation;
- the exact field `BoolGraph.Accepts` relation; and
- nonincrease of equations, multiplications, witnesses, and maximum degree.

The deterministic optimizer performs constant elimination, double-negation elimination, idempotence, Boolean factoring, and explicit fanout sharing. `DirectLogicBoolGraphDescriptorIR2` now materializes its nodes as actual live IR-v2 columns and degree-at-most-two `windowGate`s. Because an accepting descriptor also needs a linear output-equals-one gate, its constraint count is one larger than the graph-equation count. `factor_emitted_exact` proves:

Live metric	Duplicated	Checked factored
Graph equations	30	13
Internal accepting constraints	31	14
Public accepting constraints	35	18
Public inputs	4	4
Nonlinear multiplications	26	13
Auxiliary witness columns	17	8
Trace width at four inputs	21	12
Maximum constraint degree	≤ 2	≤ 2

The 35-to-18 row is the standalone statement-bound form: four linear public-input pins are added to the 31-to-14 internal relation without adding a nonlinear product or auxiliary column. `public_factor_emitted_exact` proves the count, four-public-input layout, 21-to-12 width, and 17-to-8 auxiliary ledger. These are exact structural reductions in a live descriptor, not merely a cost model. They are also stronger than an optimizer's assertion: `optimize_checked`, `optimize_holds_iff`, `optimize_accepts_iff`, and `optimize_cost_nonincrease` independently expose the certificate, semantics, field relation, and cost proof. The live `checkLive` then reconstructs both optimizer endpoints, the complete layout ledger, and exact emitted JSON; tampered layout counts, bytes, or endpoints are rejected. `canonical_trace_satisfied_iff` gives exact canonical source semantics for the actual descriptor, and `checked_source_canonical_iff` transports an accepted live certificate all the way back to its unoptimized source. The production DREGG capture below now measures proof bytes for four source/optimized pairs, but its noisy timing distributions still do not support a wall-clock ratio.

The standalone reference tool adds per-subformula representation search. `tools/direct-logic-reference` keeps a Pareto frontier of positive no-wrap residual and exact Boolean candidates and records every residual-to-Boolean or Boolean-to-residual boundary. Its independently rechecked mixed specimen has the lexicographic objective

Plan	Mults	Equations	Witnesses	Conversions	Nodes
Selected hybrid	5	9	6	1	5
All Boolean	9	11	6	0	4
All no-wrap	unsupported	-	-	-	-

The all-no-wrap plan is refused because the single-field bound cannot certify it. The hybrid artifact is canonical JSON with SHA-256 `540ad40d3c282b1294db7b62ab9ad80cf287486014aba9b054369b3a79fcdef3`; the ordinary compiler artifact remains pinned independently at `e1d19701afcbc58adf5c624dcf33ca323375a9f858e167c0d5f258c03a11f0cd`. `check` verifies the hash, recompiles from the embedded source, reconstructs the chosen plan and baselines, checks every primitive equation and bound, and then compares admitted plans with source semantics. The Python program is an executable specification and witness-plan checker, not a proof system.

6.3 One opening, proof and FHE

`ProofFheSharedOpening` closes a semantic triangle that the earlier report only proposed. A typed Boolean source carries its atom bound; one `Opening n` contains the raw integer word for every atom. Admission is integer-exact: every word must be literally zero or one, not merely congruent to a bit modulo a field.

The proof side reads the raw opening as a one-row DescriptorIR2 trace. The FHE side reads the Boolean decoding of that same opening. For every commutative ring, `descriptorDenotation_eq_fheRing` proves equality between the descriptor source polynomial and the structurally lowered BFV Boolean program. `proofAccepts_iff_fheOutput_one` then proves predicate-level agreement in BabyBear, while `noncanonical_rejected` proves that an opening containing a non-bit is rejected by both interfaces.

`ProofFheDualCertificate` makes the bridge fail closed. Its versioned bundle binds the source, opening schema, exact DescriptorIR2 JSON, exact BFV program, the BFV eight-axis operation ledger, and a length-framed canonical wire image. An accepted bundle guarantees backend denotation agreement for every canonical opening of the declared arity, not only the opening carried by the bundle. Unknown versions, trailing words, noncanonical openings, descriptor substitution, and BFV substitution are all explicit rejection theorems.

> **BOUNDARY** The dual wire image is currently a mathematical `List Int`, not a proved Rust byte decoder. Ciphertext serialization, key ownership, leakage policy, threshold decryption, Rust-to-Lean refinement, and cryptographic security are outside these semantic theorems. The theorem says that both backends compute the same predicate on one admitted opening; it does not say that a deployed client supplied that opening correctly.

6.3.1 Certified mixed FHE plan

`CertifiedHybridProofFhe` goes beyond two identical Boolean lowerings. Its source consists of equality atoms over one raw opening. A conversion witness derives the exact equality-bit opening consumed by DescriptorIR2. The encrypted side may select, independently at each region, either a locally bounded zero-means-true residual or a one-means-true Boolean program. Every conversion between those conventions is present in the plan and its cost.

`Certificate.proof_iff_fhe_plan` proves that the live proof relation and mixed encrypted plan accept the same source predicate for every canonical raw opening. The concrete five-equality specimen deliberately makes the root residual inadmissible while leaving a four-equality subregion admissible. The body-only ledger initially appears smaller, but the formerly omitted exact zero observation removes that advantage:

Plan	Body mults	Zero mults	Total	Depth	Observation
Certified mixed, $B = 4$	6	3	9	4	scaled zero bit, then opening
All Boolean	9	0	9	4	Boolean output opening

`CertifiedHybridProofFheZeroObservation` proves why this cost is unavoidable. Across the whole plaintext field, an exact arithmetic zero indicator has degree at least $p - 1 = 1,032,192$ and any addition/subtraction/multiplication expression for it has multiplicative depth at least 20. The no-wrap certificate enables a smaller exact observer on $0..B$: the balanced product $\prod_{i=1}^{B}(i - r)$ is $B!$ at zero and zero elsewhere, using $B - 1$ ciphertext multiplications. For $B = 4$, those three multiplications change 6-versus-9 into 9-equals-9 and restore the all-Boolean depth-four envelope. The result is a scaled two-valued ciphertext, not a free canonical Boolean.

> **BOUNDARY** Three cost scopes remain separate. The graph reduction is a proved live IR transformation; the production capture gives exact resource and proof-byte changes but no timing ratio. The theorem-side presentation search still uses meta-level `HybridEvidence` glue. The BFV carrier now implements bounded scaled zero conversion and threshold observation, but its receipts explicitly exclude setup, input encryption, external same-opening verification, and a general multiplied-ciphertext noise theorem. Consequently none of these ledgers is an end-to-end latency claim.

6.4 Executable backends

6.4.1 Rust finite-logic front end and live proof roundtrip

`circuit/src/direct_logic_frontend.rs` is an executable Rust compiler for finite Boolean and enumeration inputs, equality, explicit finite predicate tables, all Boolean connectives, and bounded quantifiers. Enumeration values use exact one-hot encodings; quantifiers are deterministically unrolled. Explicit limits reject an empty or unbounded domain, oversized

predicate table, excessive quantifier expansion, unsupported source, and out-of-range input rather than silently changing the relation.

The compiler emits the deployed `EffectVmDescriptor2` structure and canonical JSON, reparses and structurally checks that JSON, and pins BLAKE3 digests of the source and descriptor. The focused suite contains nine tests covering source evaluation, descriptor equivalence, one-hot rejection, resource refusal, hash stability, and malformed inputs. Most importantly, `honest_compilation_proves_and_verifies_through_live_backend` sends a compiled one-row trace through the actual `prove_vm_descriptor2` and `verify_vm_descriptor2` APIs; the same prover refuses a row that falsifies the source formula.

This is the first live Rust prove/verify path for the construction. It is not yet a proof that the Rust compiler refines the Lean compiler, and one tiny roundtrip says nothing about production throughput or finality.

A separate translation-validation gate consumes the exact byte images emitted and pinned by Lean for the public Gabbay table and public optimized BoolGraph. The production Rust parser/prover suite passed 10/10 tests, including live proof roundtrips and exact binary pins. An independent Python parser/evaluator then checked all 8,000 small Gabbay tables and all 16 assignments to the four-input BoolGraph. These tests are strong cross-language agreement on committed specimens. They are translation validation, not a theorem that either executable implementation refines Lean.

The path now also has an absolute release benchmark. It measures `LogicProgram` to `compile_logic_program` to `EffectVmDescriptor2` to one-row trace to `prove_vm_descriptor2` and `verify_vm_descriptor2`. The capture used an Apple M2 Max with 12 logical CPUs and 96 GiB RAM, five warmups, 30 proving samples, and 100 verification samples. The live BatchSTARK parameters were BabyBear, extension degree 4, FRI log blowup 6, 19 queries, and 16 query-PoW bits. All proofs had `degree_bits = 0`; proof sizes are the live SDK's postcard encoding.

Workload	Cols / gates	Prove median / p95	Verify median / p95	Proof bytes
`bool-eq`	1 / 2	2.219 / 4.572 ms	0.282 / 0.789 ms	9,262
`enum-eq-4`	4 / 6	6.911 / 9.829 ms	0.293 / 0.691 ms	9,342
`enum-eq-16`	16 / 18	4.497 / 7.140 ms	0.309 / 0.686 ms	9,721
`enum-eq-64`	64 / 66	6.728 / 8.596 ms	0.500 / 0.862 ms	10,924
`forall-4`	4 / 6	19.637 / 23.838 ms	0.361 / 0.794 ms	10,202
`forall-8`	8 / 10	8.828 / 10.630 ms	0.488 / 1.086 ms	11,165
`forall-16`	16 / 18	26.833 / 31.437 ms	0.800 / 1.695 ms	13,101

The proving medians are visibly nonmonotone. The shared development host had load averages 31.34 / 37.81 / 38.25, so the result supplies neither a scaling law nor a speedup comparison. Release proving excludes debug self-verification; the harness separately verified every warmup and retained proof. It does not measure the abstract optimizer, a conventional-circuit or Modulus baseline, zero knowledge, network inclusion, or finality.

Stable capture: `docs/deos/artifacts/direct-logic-live-2026-07-22-m2-max`. Summary SHA-256: `39a623da8ea21a03e69a6f420d5b49ce52762a9e4cbb9fefbdb21e7ae661db73`. Raw-log SHA-256: `66b37bf19bfffe505b967b03fdf44a0cc8bde22ca232493417fe3350b9ef3b1e`.

6.4.2 Four production DREGG decision skeletons

The optimizer was also driven from four Boolean decision skeletons named and consumed by DREGG: fail-closed turn admission, anti-brick upgrade authorization, structured credential clearance, and stake/vouch/bond strand admission. Lean proved each source/optimized Boolean equivalence, emitted both public descriptors and canonical rows, and the Rust harness pinned, parsed, proved, and verified those exact bytes.

Decision	Width	Constraints	Nonlinear	Aux	Proof bytes
Admission	77 to 47	121 to 71	108 to 58	65 to 35	14,016 to 11,894
Upgrade	21 to 15	33 to 23	28 to 18	17 to 11	10,431 to 10,081
Clearance	21 to 15	33 to 23	28 to 18	17 to 11	10,469 to 10,157
Strand control	11 to 11	17 to 17	13 to 13	8 to 8	9,850 to 9,850

The relation ledgers are deterministic. The retained proof encodings shrink by 2,122 bytes for Admission, 350 for Upgrade, 312 for Clearance, and zero for the already-normal Strand control. The run retained 25 proving and 75 verification samples per variant after five warmups, but load averages were 33.61 / 26.37 / 24.91 and the distributions were wide and nonmonotone. Most decisively, the byte-identical Strand pair acquired different timing medians. The capture therefore claims no timing speedup.

Every descriptor binds atom residuals to public inputs. The harness changed public input zero while retaining the Lean-authored row. In release mode the producer omits debug self-verification and may return an invalid proof; the consumer

verifier rejected all eight such proofs. Producer success is not an acceptance decision. The consumer verifier is the recorded soundness boundary.

Stable capture: `docs/deos/artifacts/direct-logic-dregg-workloads-2026-07-22-m2-max`. The directory includes exact ledgers, all raw samples, metadata, and a SHA-256 manifest.

6.4.3 Independent HOL4 and CakeML checker boundary

`tools/direct-logic-cakeml` reimplements the certificate check independently of Lean. The version-1 Standard ML checker passed 15/15 golden and malformed fixtures. In HOL4, `check_bytes_sound` proves that accepted bytes decode under the known version to the canonical source/target pair and that their Boolean semantics agree for every environment. The theory has zero axioms in the recorded fresh audit, and `check_bytes_v_thm` is the proof-producing CakeML translation certificate.

The live version-2 checker embeds the complete length-delimited v1 certificate, a 32-bit atom bound, and exact canonical IR-v2 JSON. It reconstructs the Booleanity gates, acceptance polynomial, table and layout, then compares exact bytes. It caps the atom count at 256 and each section at one MiB. The live SML suite passed 11/11 fixtures, including exact agreement with Lean's pinned demo JSON and rejection of version, bound, embedded-target, JSON, trailing, and truncation tampering. HOL4's `check_live_bytes_sound` establishes the embedded v1 check, atom bound, exact descriptor bytes, and source/target semantic equivalence; `check_live_bytes_v_thm` translates the live checker to CakeML.

> **BOUNDARY** The checked and translated source is not yet counted here as a native verified checker binary. A machine-code claim requires the completed CakeML backend compilation plus a fresh theorem-tag and axiom audit of the live theories. The live checker byte-compares reconstructed JSON; it does not independently parse arbitrary descriptor JSON or prove the DREGG Rust verifier equivalent to the HOL relation.

6.4.4 Real BFV execution

`FheLogicBfvModel` proves the Boolean-ring identities and a no-wrap theorem for the nonnegative squared-equality residual

$$R = \sum_i (x_i - y_i)^2.$$

Under the declared input bounds and centered plaintext window, field zero is equivalent to all source equalities. For eight encrypted equalities, the residual interpreter has 8 multiplications, 8 relinearizations, and symbolic depth 1. The balanced all-Boolean interpreter has 15 multiplications, 15 relinearizations, and depth 4. Those body ledgers do not yet include observing whether the encrypted residual is zero.

The matching `fhegg-fhe` path executes real BFV ciphertexts. One recorded warm release run on `persvati` (x86-64 Linux, AMD Ryzen AI 9 HX PRO 370), at ring degree 4096 and plaintext modulus 1,032,193, produced:

Executed workload	SIMD cases	BFV body	Mul/relin	Noise
Boolean policy	16	22.950435 ms	3 / 3	79 bits
Eight-equality residual	4	48.197263 ms	8 / 8	49 bits

The residual certificate checked $648 < 516096$. These rows are different programs with different SIMD occupancy, so their times do not define a speedup. They are single warm observations. The timed body excludes key generation, encryption, decryption, and zero observation; the observed noise is not a proved noise or security bound. The focused release suite passed 3/3 tests, including refusal of a cancellation-capable bound.

`fhegg-fhe` now executes the missing boundary rather than leaving it as an unpriced counter. The two-equality, $B = 2$ oracle successfully evaluated the scaled encrypted observer and decrypted the expected values $[2, 0, 2, 0]$. At $B = 8$, the residual body uses eight ciphertext multiplications and the bounded observer adds seven, so the total is 15 at depth four - exactly the balanced Boolean ledger, not an 8-versus-15 win. The encrypted conversion ran, but n-of-n observation of its scaled output refused because the threshold combine produced a value outside the canonical set $\{0, 8!\}$. The API turns the unproved multiplied-ciphertext noise margin into a refusal rather than a wrong truth bit.

The alternate $B = 8$ boundary threshold-opens the raw residual successfully. For four live SIMD slots it revealed $[0, 1, 0, 2]$, from which the coordinator computed $[\text{true}, \text{false}, \text{true}, \text{false}]$. Two smudged decryption-share messages occupied 419,988 bytes and the threshold observation took roughly 165 ms in the retained run. This path is exact but intentionally leaky: it reveals the whole residual, not merely its zero bit.

BOUNDARY Neither observation is an end-to-end comparison. Setup and input encryption were measured outside the receipt; the same-opening field is a test-only transparent-construction receipt rather than an executed zero-knowledge verifier; and the existing smudging theorem does not prove correctness for the multiplied-ciphertext noise envelope. Internal residual-to-bit continuation is priced as an opening plus re-encryption and refused because no hybrid continuation machine is implemented. The honest conclusion is a three-way choice: pay the bounded polynomial cost, open and leak the raw residual, or introduce a different private comparison carrier.

6.5 Conformance before performance

`tools/direct-logic-bench` compares five relations against a common Boolean reference. A lane that fails semantics remains visible as a falsification artifact but is excluded from every performance ratio. The deterministic corpus contains the pinned adversarial and transaction examples plus 10,000 generated formulae, for 10,061 cases in total:

Lane	Gate	Supported	Unsupported	Mismatch
M-SPEC	fail	10,061	0	3,936
G-FIELD-NAIVE	fail	6,748	3,313	1
D-NOWRAP	pass	6,737	3,324	0
D-BOOLGRAPH	pass	10,061	0	0
C-AIR	pass	10,061	0	0

M-SPEC is an exact transcription of the public BitLogic connective, quantifier, and swap equations. G-FIELD-NAIVE separately tests the Gabbay-style positive residual after unsafe field reduction. D-NOWRAP is the corrected bounded positive route; unsupported means that its required partial-accumulator bound was not established. D-BOOLGRAPH is the exact inverse-witness Boolean graph. C-AIR is an independently written gate-by-gate arithmetic baseline.

The harness emits environment and source hashes, every workload and gate result, a representation-labelled symbolic ledger, raw timing samples, bootstrap intervals, checksums, and an admitted-only summary. Its timing channel measures Python relation evaluation. It does not measure witness generation, proving, verification, proof bytes, FHE, consensus, data availability, finality, or monetary cost. In particular, an unsound equation does not become an optimization because it evaluates quickly.

6.6 What would test the claimed performance numbers

The verified constructions justify an end-to-end benchmark campaign; they do not pre-decide its outcome. A credible 10-100x claim requires every candidate to prove the same source predicate, at the same security target, on the same hardware, while including every conversion and boundary operation. The first matrix should contain these pipelines:

Lane	Construction	Admission rule
C-AIR	ordinary gate-by-gate arithmetic circuit/AIR	independent source conformance
D-NOWRAP	positive residual with certified partial bounds	all no-wrap obligations discharge
D-BOOLGRAPH	live materialized bits and inverse-witness zero tests	checked source, layout, and exact bytes
D-HYBRID	certificate-selected residual/Boolean/shared DAG	certificate replay plus source equivalence
M-IMPL	public Modulus compiler/prover, if supplied	source, parameters, build, and raw artifacts available

The workload matrix should cross Boolean policy logic, equality-heavy batches, mixed negation and disjunction, nested finite quantifiers at several domain sizes, multi-arity signatures with public functions, shared-subterm fanout, the duplicated/factored live BoolGraph pair, the fixed Gabbay matrix instance, and the published swap relation. Both true and false instances are required; a prover must refuse the latter.

For every lane and size, the preserved run artifact must report:

- exact source, grounded relation, compiler/certificate hashes, field and security parameters;
- equations, multiplications, degree, witnesses, rows, columns, lookup/table cells, and every range or zero-test gadget;
- source compilation, witness generation, proving and verification time with warmup, sample count, distribution, and peak memory;
- proof bytes, public-input bytes, preprocessing material, and verifier work;
- for FHE, encryption, evaluation, conversion/bootstrapping, threshold or private zero decision, decryption, ciphertext bytes, occupancy, depth, and measured remaining noise;
- machine, operating system, compiler revisions, build flags, thread count, accelerator residency, and raw samples.

Only the complete wall-clock and resource columns can support an end-to-end cost ratio. The 30-to-13 graph-equation, 31-to-14 internal and 35-to-18 public accepting-constraint, 26-to-13 multiplication, and 21-to-12 width results are verified live structural reductions. The four DREGG captures add exact source/optimized relation ledgers and retained proof-byte reductions. They do not add a timing ratio. On the FHE side, including exact bounded zero observation has already erased both advertised body-only advantages on the measured specimens: 6 plus 3 equals Boolean 9, and 8 plus 7 equals Boolean 15. Future experiments should search for regimes where sharing, packing, a private comparison carrier, or a different representation improves the complete pipeline rather than reusing the superseded body-only comparison.

The 1-3 second finality claim requires a different matrix. At minimum it must state the consensus and fault model, validator count and geography, block and proof size, data-availability path, network delay and loss model, batching and sequencing policy, hardware, confirmation rule, and latency distribution under load and faults. Logic arithmetization can change proving cost, but finality is also a consensus, networking, execution, and data-availability property.

6.7 Arithmetization is not chain portability

Mapping a finite logical relation into polynomials answers only the relation encoding question. A portable application additionally needs a correct compiler, a sound proof system, a stable recursive statement ABI, verifiers on each target, foreign-finality or inclusion evidence, action and asset binding, data availability, and an atomicity protocol. None follows from calling FOL “direct.”

DREGG has separate evidence in this larger stack: a whole-history fold, local EVM, Solana, and Cosmos verifier demonstrations, and a generic data-availability interface. Outbound message roots currently fail closed. Foreign light-client folding, Celestia integration, and multi-chain atomicity remain open or conditional. These facts make the architecture inspectable; they do not justify a universal-portability or live-production claim.

6.8 Constructive public comparison

As inspected on 22 July 2026, the public BitLogic repository contains its whitepaper PDF but no compiler, witness generator, prover, verifier, release, benchmark driver, parameter manifest, or raw result bundle. The project site advertises 10-100x cost reduction and 1-3 second finality. No public adapter exists that can run those claims through the matrix above. The counterexample in this report is therefore an attack on a published relation, not on unavailable private code or a deployed chain.

Axis	Reviewed public Modulus evidence	DREGG artifact in this study
Source semantics	Printed equations; the published swap product admits the proved bypass.	Kernel-checked corrected FOL, finite-signature semantics, and exact public-table binding.
Compiler	No public compiler implementation found.	Lean FOL-to-IR, six-PI direct-table, and four-PI BoolGraph compilers; executable Rust and Python tools.
Optimization	No public derivation or cost certificate found.	Replayable rewrites, live graph lowering, exact layout/byte guards, checked hybrid search.
Proof path	Architecture prose names zkEVM, STARK, and SNARK components.	Live IR-v2 descriptors, four DREGG decision captures, consumer tamper refusal, explicit FRI assumptions.
Independent checker	No public checker source found.	SML/HOL4/CakeML boundary plus Rust/Python exact-byte translation validation.
Encrypted logic	No public FHE logic compiler found.	Same-opening theorem, real BFV bounded zero conversion, threshold observation, and fail-closed noise refusal.
Performance	Headline ratios without a reviewed reproducibility bundle.	Verified structural savings and proof-byte reductions; FHE body wins erased at observation; no timing winner.

FORMAL ARTIFACT - the public lead DREGG’s present advantage is auditable semantics and compositional evidence: source, countermodels, repair theorems, compilers, certificates, fixtures, and boundary assumptions can all be inspected. It is not yet an established advantage in prover latency, proof size, monetary cost, or chain finality.

SECTION 7

Conclusions: from a broken shortcut to a certified compiler family

The public BitLogic shortcut fails. The underlying research direction does not. The useful result of this audit is a stronger architecture than the claim that prompted it.

> **DECISIVE RESULT** We now have a coherent path from matrix FOL to sound compiler semantics and a statement-bound public direct-table relation, and from bounded finite logic to optimized public descriptors for four named DREGG decisions. The public factor descriptor moves from 35 to 18 constraints with exact statement soundness; live production proofs retain smaller serialized outputs for three optimized cases, while Strand supplies an unchanged negative control. Fully priced FHE accounting erases the earlier residual multiplication lead instead of hiding its zero observation. Higher-order, categorical, interaction, query, and realizability results then generalize the architecture beyond FOL without pretending that one finite polynomial table is the primitive form of logic.

7.1 The corrected answer to the original claim

Gabbay's theorem is not false in its actual setting. Over nonnegative rationals, zero residuals have the intended meaning: conjunction and bounded universal quantification are sums; disjunction and bounded existential quantification are products. The error is to transport those equations into a prime field without transporting their semantic hypotheses.

The repaired compiler makes that transport explicit. It either proves numerator and denominator bounds before field projection, or converts truth to canonical bits with checked zero witnesses. `GabbayMatrixCompiler` reconstructs the actual matrix language and interpolation mechanism; `GabbayMatrixBridge` proves the two formal presentations coherent; and `GabbayDescriptorIR2` supplies a fixed-shape private Skolem-witness relation in the live proof grammar.

`GabbayDescriptorIR2PublicBoundary` proves that this descriptor has zero public inputs, no hash sites, and an empty v1 range carrier: acceptance does not attest an independently named external table, and the fixed instance is not interpolation-performance evidence. The empty carrier field is not an absent range check — the bound is carried by six emitted 30-bit range lookups, which bind no external name and so leave the statement gap open. The separate `GabbayDescriptorIR2PublicBinding` closes that gap directly: six public cells bind the complete two-by-three table; six trace columns carry 15 constraints — six PI pins, three linear acceptance atoms, and six range lookups — with no nonlinear product anywhere in acceptance; under the canonical decoding convention alone, a satisfying trace exists iff the named table satisfies source `Holds`, and a one-cell public tamper is refused. This integrity variant reveals the whole table, provides zero table privacy, and deliberately performs no interpolation. `FiniteSignatureFOLDescriptorIR2` then extends the live route to many relation symbols, total public function symbols, equality, all connectives, and nested finite quantifiers.

BitLogic takes neither repaired route in its released equations. It reverses the zero-true connective interpretation and multiplies obligations that must all hold. The published swap product therefore accepts a displayed counterexample. That is not a speculative concern or a disagreement about notation: it is a wrong acceptance relation.

The same standard caught our own first attempt. That public relation originally accepted with a field sum of three squared residuals and was made sound by a Lean-side no-wrap premise that no emitted byte enforced — and over BabyBear $284861408^2 = -1$, so a fully canonical false table cancelled to zero and was accepted. The repair moved both halves onto the wire: three linear atoms, one per equation, and a 30-bit range lookup on each bound column, in the graduated form the deployed assembly actually realizes. The two premises then became theorems, so the statement results no longer take a certificate argument. Two counterexamples are retained as armed canaries, one refuted by the atoms alone and one refuted only by a range tooth, so neither half of the repair can be deleted without turning a theorem red. One premise moved rather than vanished and is named as such: `Satisfied2` has no faithfulness conjunct for the range table, so refusals through a range tooth carry `HonestRangeTable` explicitly — a carrier-level condition on the trace family, uniform across every range lookup in the codebase, discharged in deployment by the assembly that builds the limb decomposition itself.

7.2 The novel contribution that emerged

The strongest reusable construction is **certified change of presentation**. One statement may be realized by different evidence objects:

$$\text{source judgment} \to \text{residual} \to \text{Boolean graph} \to \text{shared net} \to \text{AIR or FHE plan}.$$

Each arrow carries semantic equivalence, a fail-closed checker or proof, and an explicit resource bound. Composition of such arrows is itself certified. This turns backend choice into proof-producing compilation rather than folklore. The deterministic optimizer realizes the idea for checked Boolean rewrites; the reference Pareto search chooses residual or Boolean representations per grounded subformula; and the hybrid proof/FHE certificate proves both backends decide the same raw opening.

This is also where the verified resource results come from. Checked factor sharing reduces graph equations from 30 to 13. Its **public** accepting descriptor includes four PI bindings and one output check, so the complete constraint ledger moves from 35 to 18; multiplications move from 26 to 13, auxiliaries from 17 to 8, and width from 21 to 12. Every constraint has degree at most 2. Arbitrary-trace public soundness, canonical completeness, and the accepted optimizer/layout/exact-byte certificate make this an exact statement-bound result rather than a witness-only cost model.

The same construction now compiles four production-derived DREGG decisions. For Admission, public constraints move 121 to 71, multiplications 108 to 58, width 77 to 47, and one retained proof 14,016 to 11,894 bytes. Upgrade moves 33 to 23, 28 to 18, 21 to 15, and 10,431 to 10,081 bytes; Clearance has the same relation ledger and moves 10,469 to 10,157 proof bytes. Strand is already normal: 17 constraints, 13 multiplications, width 11, and 9,850 proof bytes on both sides. These skeletons are proved equivalent to the named DREGG decisions while their arithmetic, hash, membership, lifecycle, lookup, and receipt-comparison atoms keep their existing implementations.

The paired timing capture does not support a speedup claim. Severe host contention produced wide, nonmonotonic distributions; most decisively, byte-identical Strand source and optimized cases reported different medians and tails. The exact ledgers and retained proof-size changes survive that control. A 10–100x latency claim does not.

Fully priced FHE accounting closes a second tempting overclaim. At residual bound $B = 4$, the former 6-versus-9 inner comparison becomes 9 = 9 after the exact bounded zero conversion, with depth 4 on both sides. At $B = 8$, residual evaluation plus conversion is 8 + 7 = 15, equal to the Boolean 15. The scaled conversion executes at $B = 2$; the captured $B = 8$ scaled-bit observer refuses under the current unproved noise envelope, while raw threshold opening succeeds only by disclosing all live residuals. There is therefore no residual multiplication advantage to advertise, and no general BFV noise, security, privacy, or end-to-end latency theorem.

7.3 Higher order and the CCC question

The CCC line is now an exact frontier rather than a slogan or a no-go alone.

- The finite extensional polynomial model remains exact but dense. It gives a specification semantics for finite equality-HOL, not a succinct compiler for arbitrary higher-order programs.
- A faithful family of finite-field polynomial Yoneda actions exists exactly when the category is locally finite; each fixed-arrow action is linear. Even the CCC of finite sets admits no faithful **uniformly fixed-width** family, so local finiteness does not imply one bounded backend.
- The intensional route compiles typed STLC strongly-CCC to shared nets. Its local interaction receipts now reach live public DescriptorIR2 through direct depth 14, with arbitrary-trace layout soundness and a proof that the erased receipt cannot reconstruct every exact typed endpoint.
- The indexed compiler may select dense polynomial, Boolean graph, interaction net, or natural residual per subterm. Its source theorem charges conversions and recombination in a five-axis ledger; selection is only optimal relative to the finite candidate list supplied.
- The global category of presentations whose meanings vary cannot be CCC: it is not even cartesian. After fixing the source meaning, the fibre of evidence families is a CCC. This is the precise fibred replacement.

Thus arbitrary small CCC semantics embed by Yoneda, locally finite CCCs additionally admit the exact polynomial family, and represented higher-order programs have an operational shared-net route. What is ruled out is the stronger fantasy that all of this fits faithfully and succinctly into one uniform finite polynomial universe.

7.4 Proof objects beyond polynomials

The realizability line now supplies more than analogy. The PCA doctrine has arbitrary-map quantifiers and Beck–Chevalley laws; assemblies have pullbacks; effective covers are stable under pullback, are kernel-pair coequalizers, and give regular-image factorizations, including for compiled program certificates. These are concrete ingredients of an exact completion, not yet the exact completion or associated topos itself.

The query line is separate and compositional. Coverage presentations form a category; exact presentation change preserves obstruction and miss events. A locally accepting false statement lies in one of four explicitly priced events: OOD, low degree or folding, commitment binding failure, or query miss. The commit-before-query transcript theorem then prices the last event without asserting a false deterministic total-column bridge. Under point binding and modularly reduced queries, the sharp BabyBear-shaped bound is

$$\varepsilon_{\mathrm{OOD}} + \varepsilon_{\mathrm{LDT}} + \left((1-\delta) + \frac{\delta}{N}\right)^{k}$$

when $N \bmod m = 1$. A computational commitment adds its binding budget. Additional transcript coordinates product out of the query marginal, while OOD and folding events may inspect the complete transcript. Category theory identifies and transports the obstruction; coding theory makes it dense; commitments bind local views; and the sampler prices a miss. None of the category or topos results replaces the required hash/PCS binding, Fiat–Shamir, OOD, FRI, or verifier-correspondence reductions.

7.5 Executable assurance now present

- Lean emits source and optimized public descriptors and canonical traces for Admission, Upgrade, Clearance, and Strand. Rust pins those exact bytes, parses them, proves and verifies all eight cases, recomputes their ledgers, and confirms consumer-verifier rejection after a public-input tamper.
- The production capture retains every raw timing sample and one serialized proof per endpoint. Exact relation/proof-byte changes are reported; the unchanged Strand control prevents noisy timing differences from being sold as speedup.
- The independent reference compiler and 10,061-case conformance corpus keep unsound relations as visible falsification artifacts while excluding them from performance comparisons.
- Independent Standard ML checkers passed 15/15 v1 and 11/11 live-v2 fixtures. HOL4 proves accepted bytes semantically sound with zero recorded theory axioms; proof-producing CakeML translation certificates exist for both checkers.
- The BFV carrier executes residual evaluation, bounded scaled zero conversion, and n-of-n threshold opening. Its manifests record multiplication, depth, decryption shares, combines, comparisons, re-encryption, coverage, and leakage; noncanonical threshold output is refused rather than coerced to a bit.

This is already a toolchain, not a white-paper sketch. Its remaining trust boundaries are also explicit. Production equivalence stops at the named Boolean atom boundary. Rust implementations are not yet proved to refine the Lean models; the CakeML certificates are not yet shipped native checker binaries. The BFV path still lacks a general noise/security proof and a deployed cryptographic same-opening verifier, and raw threshold observation discloses its residuals. The query theorems still require concrete commitment, Fiat–Shamir, OOD, FRI, and verifier-correspondence reductions. No captured timing establishes an end-to-end prover or finality advantage.

7.6 What may be claimed now

> **COMPARATIVE RESULT** DREGG surpasses the reviewed public Modulus/BitLogic material in publicly checkable semantics, construction depth, falsification coverage, executable tooling, cross-language assurance, certificate discipline, and reproducibility. The public BitLogic relation is refuted; DREGG's corrected public relations are statement-bound and proved exact under stated premises. Three production-derived decisions show exact relation-ledger and retained proof-byte reductions, with an unchanged negative control. Fully priced FHE accounting reports equality rather than a fictitious win. DREGG does not yet establish a 10–100x end-to-end proving advantage, 1–3 second chain finality, or runtime superiority to an unpublished Modulus implementation.

Finality is a consensus, networking, data-availability, and validator claim; it does not follow from replacing a circuit connective. Likewise, a symbolic multiplication count is not a prover benchmark. A defensible performance claim must measure the same workload through baseline AIR, repaired residual, Boolean, and certified hybrid paths, including witness generation, proof generation, verification, proof bytes, range checks, zero decisions, and every conversion boundary.

7.7 Constructive offer

TO MODULUSZK AND CULTDAO Keep the useful matrix-language insight. Replace the finite-field equations with one of the proved repairs. Conjoin independent obligations. Publish the exact source language, compiler, relation, witness generator, prover, verifier, parameters, corpus, raw measurements, and exclusions. Import the counterexamples as permanent regression tests. Then compare the repaired implementation against the certified residual, Boolean, shared, and hybrid constructions here on identical workloads.

BOUNDARY This report establishes a specification-level failure in the released BitLogic relation, not an exploit against private or deployed software. It also establishes positive formal and executable DREGG constructions, not a production security certification. Those two boundaries make the conclusion stronger: the criticism is reproducible, and the alternative is inspectable.

The research program that survives is not "logic bypasses circuits." It is: compile judgments into the cheapest sound local presentation, attach evidence for every translation, and let proof systems, encrypted runtimes, and future backends compete inside one semantics-preserving envelope.

SECTION 8

Primary sources and inspected artifacts

1. Murdoch J. Gabbay. *Arithmetisation of computation via polynomial semantics for first-order logic*. IACR ePrint 2024/954, third revision, 12 February 2026; Journal of Applied Logics 12(6), 1481-1548, 2025. In the current revision, the principal locators used in this report are Figure 2 and Definition 2.2.8, Definition 2.3.1, Lemma 2.3.2, Theorem 2.3.4, Remark 2.3.5, and Remark 6.3.3. ePrint record and current PDF.
2. Murdoch J. Gabbay and Andrew Mendelsohn. *SNARKs for First Order Logic*. Workshop on Cryptographic Tools for Blockchains, affiliated with EUROCRYPT 2026, Rome, 9 May 2026, 46 slides. slides workshop programme.
3. Mr. O' Modulus. *BitLogic from Modulus: zkFOL Bitcoin: An Engineering Whitepaper for Formal Logic, Zero-Knowledge, and Decentralized Finance on Bitcoin*. 21 November 2025. Public PDF and repository inspected at commit `26f125b37f0442d92991e3066095b1dfeb1b0ce4` . pinned GitHub tree.
4. ModulusZK. Official homepage and documentation, inspected 22 July 2026. homepage documentation architecture glossary.
5. ModulusZK. Public posts cited in the text, 1 December 2025 and 21 July 2026. 1 December 2025 post 21 July 2026 post.
6. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Fast Reed-Solomon Interactive Oracle Proofs of Proximity*. ICALP 2018, LIPIcs 107, Article 14, 1-17. DOI.
7. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, transparent, and post-quantum secure computational integrity*. IACR ePrint 2018/046. ePrint.
8. Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. IACR ePrint 2012/144. ePrint.
9. Emily Riehl. *Category Theory in Context*. Dover Publications, 2016. See Chapter 2 for representability and the Yoneda lemma. author's book page.
10. J. Lambek and P. J. Scott. *Introduction to Higher-Order Categorical Logic*. Cambridge Studies in Advanced Mathematics 7, Cambridge University Press, 1986. publisher front matter.
11. J. M. E. Hyland, P. T. Johnstone, and A. M. Pitts. *Tripos Theory*. Mathematical Proceedings of the Cambridge Philosophical Society 88 (1980), 205-232. DOI.
12. Jaap van Oosten. *Realizability: An Introduction to its Categorical Side*. Studies in Logic and the Foundations of Mathematics 152, Elsevier, 2008. publisher page.
13. Wouter Pieter Stekelenburg. *Categories of assemblies for realizability*. arXiv:1307.0663, 2013. arXiv.
14. S. A. Terwijn. *Computability in partial combinatory algebras*. arXiv:1910.09258, third revision, 5 February 2020. arXiv.
15. Liron Cohen, Étienne Miquey, and Ross Tate. *Evidenced Frames: A Unifying Framework Broadening Realizability Models*. LICS 2021. DOI.
16. Liron Cohen, Ariel Grunfeld, Dominik Kirst, and Étienne Miquey. *Syntactic Effectful Realizability in Higher-Order Logic*. LICS 2025. DOI arXiv.

17. Liron Cohen, Ariel Grunfeld, Dominik Kirst, and Étienne Miquey. *From Partial to Monadic: Combinatory Algebra with Effects*. FSCD 2025, LIPIcs 337, Article 14. DOI artifact.
18. Liron Cohen and Tomer Samara. *Evidence-Tracked Tape Semantics for Probabilistic Computation*. FSCD 2026, LIPIcs 378, Article 13. DOI.
19. Yves Lafont. *Interaction Nets*. POPL 1990, 95-108. DOI.
20. Adrien Ragot, Thomas Seiller, and Lorenzo Tortora de Falco. *Linear Realisability over Nets: Multiplicatives*. CSL 2025, LIPIcs 326, Article 43. DOI.
21. Uriya A. First and Tali Kaufman. *Cosystolic Expansion of Sheaves on Posets with Applications to Good 2-Query Locally Testable Codes and Lifted Codes*. STOC 2024, 1749-1760. DOI arXiv, third revision.
22. Irit Dinur, Siqi Liu, and Rachel Yun Zhang. *New Codes on High Dimensional Expanders*. CCC 2025, LIPIcs 339, Article 27. DOI arXiv.
23. Hugo Delavenne, Tanguy Medevielle, and Élina Roussel. *Interactive Oracle Proofs of Proximity to Codes on Graphs*. arXiv:2501.14337, second revision, 13 May 2025. arXiv.
24. Hugo Delavenne and Louise Lallemand. *Codes on any Cayley Graph have an Interactive Oracle Proof of Proximity*. arXiv:2508.10510, 14 August 2025. arXiv.
25. Yong Kiam Tan, Magnus O. Myreen, Ramana Kumar, Anthony Fox, Scott Owens, and Michael Norrish. *The verified CakeML compiler backend*. Journal of Functional Programming 29 (2019), e2. DOI.
26. Oskar Abrahamsson, Magnus O. Myreen, Ramana Kumar, and Thomas Sewell. *Candle: A Verified Implementation of HOL Light*. ITP 2022, LIPIcs 237, Article 3. DOI project page.
27. Albert Garreta, Hendrik Waldner, Katerina Hristova, and Luca Dall'Ava. *Zinc: Succinct Arguments with Small Arithmetization Overheads from IOPs of Proximity to the Integers*. IACR ePrint 2025/316. ePrint.
28. Alireza Shirzad, Sriram Sridhar, Dimitrios Papadopoulos, and Charalampos Papamanthou. *Relaxed Modular PCS from Arbitrary PCS and Applications to SNARKs for Integers*. IACR ePrint 2026/347, revised 20 June 2026. ePrint.
29. Alexander Abdugafarov, Albert Garreta, Amit Kumar, Michał Osadnik, Psi Vesely, Ilia Vlasov, and Kai Zhe Zheng. *Zinc+: SNARKs for Polynomial Rings*. IACR ePrint 2026/855, revised 9 May 2026. ePrint.
30. Marco Stronati, Denis Firsov, Antonio Locascio, and Benjamin Livshits. *Clap: a Semantic-Preserving Optimizing eDSL for Plonkish Proof Systems*. arXiv:2405.12115, second revision, 11 July 2024. arXiv.
31. Simon Guilloud, Sankalp Gambhir, Andrea Gilot, and Viktor Kunčak. *Mechanized HOL Reasoning in Set Theory*. ITP 2024, LIPIcs 309, Article 18. DOI.
32. Raghav Malik, Vedant Paranjape, and Milind Kulkarni. *Circuit Optimization using Arithmetic Table Lookups*. Proceedings of the ACM on Programming Languages 9, PLDI (2025), Article 159, 301-323. DOI.
33. Song Bian, Yintai Sun, Zian Zhao, Haowen Pan, Mingzhe Zhang, and Zhenyu Guan. *Libra: Pattern-Scheduling Co-Optimization for Cross-Scheme FHE Code Generation over GPGPU*. USENIX Security 2026, prepublication page inspected July 2026. USENIX.
34. Zhihao Li, Hongyu Wang, Yuan Zhao, Lichun Li, Zhiwei Wang, Jiaxing He, Changzheng Wei, Ying Yan, and Lifeng Guo. *BatchBoot: Fast Batched Bootstrapping for TFHE scheme and Practical Applications*. USENIX Security 2026, prepublication page inspected July 2026. USENIX.
35. Siddharth Bhat, Alex Keizer, Chris Hughes, Andrés Goens, and Tobias Grosser. *Verifying Peephole Rewriting in SSA Compiler IRs*. ITP 2024, LIPIcs 309, Article 9. DOI artifact.
36. DREGG formalization sources. Integration root: `metatheory/Dregg2/Metatheory/DirectLogicArithmetization.lean`. The report cites individual theorem names from its import closure.