There Is No Compiler:
Lean's Kernel as an Active-Pair Interaction Net



The Institute for
Applied Ontological
Mathematics
IAOM

Our Technology is Ontological: **Hardware = Physics | $\overline{\lambda}$ | Software = Mathematics**
Our Engineering is Constructive: ϵ ***Discover*** $\Rightarrow$ ***Build*** $\Rightarrow$ ***Grow*** $\Rightarrow$ ***Learn*** $\Rightarrow$ ***Teach*** ∞

$\vdash$

Acknowledgment

We humbly thank the collective intelligence of humanity for providing the technology and culture we cherish. We do our best to properly reference the authors of the works utilized herein, though we may occasionally fall short. Our formalization acts as a reciprocal validation—confirming the structural integrity of their original insights while securing the foundation upon which we build. In truth, all creative work is derivative; we stand on the shoulders of those who came before, and our contributions are simply the next link in an unbroken chain of human ingenuity.

$\dashv$

May 2026

Abstract. The kernel of a modern dependent type theory issues five families of success—head normalization, definitional equality, type inference, sort inference, and type check—spanned by a fifty-one-row generator inventory. We exhibit a kernel-certified, structurally faithful, bidirectional correspondence between these successes and active-pair reductions on a typed interaction net with four cell kinds and a principal-plus-auxiliary port discipline. The correspondence is universal: no caller-supplied support precondition narrows it. A joint triple invariant—judgment family, primitive support family, and reduction height—is preserved in both directions, so the substrate forgets nothing the kernel was tracking. We prove the forward lift by structural recursion on success witnesses, the backward reflection by induction on the net derivation, and the totality of the underlying generator inventory at the row level. As empirical confirmation, ten thousand declarations drawn from a foundational Mathlib slice agree on the nose with an independent reference oracle across all five families. The substrate runs in parallel on commodity GPU hardware through a fuel-bounded reducer with local-footprint disjointness as its parallel-firing law.

1 Introduction

Two practices have grown side-by-side for forty years without ever touching at the root. On one side, dependent type theory: mathematics formalized to the point where a kernel program accepts or rejects each proof. On the other, interaction nets: a substrate of graphs with principal ports that rewrites locally and confluently, and which has powered the fastest implementations of pure functional computation we know how to build. Both are presentations of computation as graph reduction. So why do we still write *compilers* between them?

We show that no compiler is needed. The five families of success the kernel of a modern dependent type theory can issue—head reduction, definitional equality, type inference, sort inference, and check—and the fifty-one inventory rows that span them, admit a structurally faithful representation as active-pair reductions on a typed interaction net. The kernel-checked term is the net. We prove this in both directions, with the same triple invariant—family, support family, and reduction height—preserved on each side. The structural correspondence is total: every kernel success has a substrate realization, every substrate derivation reflects to a kernel success.

When a mathematical structure is correctly identified, the work it does becomes free. Type checking, optimal reduction, parallel evaluation, hardware specialization—they stop being separate engineering tasks and become projections of a single substrate. A theorem proved against the kernel of one of these systems executes on a graphics processor not by virtue of a verified compiler, but by virtue of a recognition that the proof was always already a net waiting to reduce. The certified-compilation tradition treats this passage as a chain of intermediate representations linked by simulation theorems. We argue, and prove at the kernel-success floor, that the chain collapses to a recognition.

The leverage that makes this collapse possible was already in the room. Interaction combinators (Lafont, 1997) give us strong confluence locally for free. Curry–Howard (Howard, 1980; Wadler, 2015) promises that the kernel-checked term is already a program. What remained was to identify the kernel's full success surface and show that each row of its generator inventory has a structurally faithful interaction-net representation under a single joint invariant. That identification, and the bidirectional theorems that confirm it, is the content of this paper.

Section 2 positions the result against the certified-compilation tradition. Section 3 catalogues the kernel's five success families and the fifty-one inventory rows. Section 4 fixes the active-pair interaction net we use as substrate. Sections 5 and 6 prove the forward lift and backward reflection with triple-invariant preservation. Section 7 establishes inventory totality. Section 8 treats reduction, parallel firing, and hardware. Section 9 reports the ten-thousand-declaration Mathlib confirmation. Sections 10 and 11 take stock and name what becomes possible. Sections 12 and 13 close with the formalization record and the realization claim.

2 The Question Behind the Question

What does a verified compiler actually verify?

The certified-compilation tradition—CompCert (Leroy, 2009; Leroy et al., 2016), CakeML (Kumar et al., 2014), Bedrock (Chlipala, 2013)—answers: a chain of intermediate representations linked by simulation theorems. Each pass refines the source toward a machine; each pass carries a proof that the refinement preserves behaviour. The chain is verified end-to-end when every link holds.

The chain is the artefact. It is also a confession: it confesses that the source and the target are different objects, and that the verification effort is the cost of carrying them apart. For a C

compiler targeting an embedded MCU, the confession is honest—C and ARM assembly are different objects, and the verified passes are the real labour. For a mathematics formalization, the confession should be questioned. What *are* the source and the target?

The source is a term of dependent type theory whose kernel acceptance *is* the mathematical claim. The target is whichever substrate we want to reduce against. The classical answer—MetaCoq (Anand et al., 2018; Sozeau et al., 2020), the Lean-in-Lean re-verification (Carneiro, 2019)—is to re-implement the kernel inside the same type theory and prove the re-implementation agrees with itself. This is a substantial advance: it closes the self-reference loop. It does not, however, address the operational question. The re-implemented kernel still runs as an OCaml or C++ program emitting opaque verdicts.

We propose a different answer. The right target is not a machine. It is an *active-pair interaction net*: a typed graph with principal ports, local rewrite rules, and a strong-confluence theorem (Lafont, 1990, 1997). Under this target, the kernel-checked term is not compiled into the substrate; it *is* a configuration of the substrate, exhibited under the relevant projection. The verification effort is the recognition that this is so.

The recognition rests on three pieces of free leverage. From interaction combinators (Lafont, 1997), strong confluence and locality. From Curry–Howard (Howard, 1980; Wadler, 2015), the assertion that the kernel-checked term is already a program. From the kernel architects' own choice (de Moura and Ullrich, 2021), a small success surface: five families spanning fifty-one generator rows. The paper assembles these pieces into a single statement about a single object.

3 The Kernel's Success Surface

A kernel succeeds, or it does not. When it succeeds, what does it tell us?

3.1 Five families

The reduction-aware kernel of the dependent type theory we treat—and, mutatis mutandis, of any modern Calculus-of-Constructions descendant (Coquand and Huet, 1988; Martin-Löf, 1984)—issues five families of success. Each family corresponds to one operational question the kernel knows how to answer.

Definition 3.1 (Kernel success surface)**.** A *kernel success* at configuration cfg, fuel n, and context Γ is one of:

$$\begin{aligned}
\text{whnf}: &\quad \mathrm{WHNF}(\mathrm{cfg}, n, s) = t \\
\text{defEq}: &\quad \mathrm{DefEq}(\mathrm{cfg}, n, \ell, r) = \top \\
\text{infer}: &\quad \mathrm{Infer}(\mathrm{cfg}, n, \Gamma, e) = \text{some } T \\
\text{inferSort}: &\quad \mathrm{Infer}_{\mathrm{Sort}}(\mathrm{cfg}, n, \Gamma, e) = \text{some } u \\
\text{check}: &\quad \mathrm{Check}(\mathrm{cfg}, n, \Gamma, e, T) = \top
\end{aligned}$$

The first reduces an expression to weak head-normal form under configured delta and iota rules. The second decides definitional equality of two expressions under the same rules. The third infers a type for a well-formed expression. The fourth infers a sort for a type-level expression. The fifth checks that an expression has a declared type.

Why these five. This is not the only carving available (Coquand and Huet, 1988; Martin-Löf, 1984), but it is the carving the contemporary engine (de Moura and Ullrich, 2021) actually uses, and it has a structural virtue worth naming: every other reduction- or typing-related operation the kernel performs is a composite over these five primitives. Sort inference is type inference restricted to sort-typed expressions; check decomposes into infer + defEq; rewriting under congruence rules decomposes into successive whnf calls. The surface is closed under the relevant compositions.

3.2 Fifty-one rows

Beneath the five families lies a finer grain: each family is constructed from a finite set of named *generator rows*. A row is the elementary operational step the kernel takes when a specific syntactic or semantic configuration arises—bound-variable lookup, constant unfolding, beta-redex contraction, sort-universe comparison, congruence descent into an application's function position, and so on. Across the five families, the kernel exposes exactly fifty-one such rows.

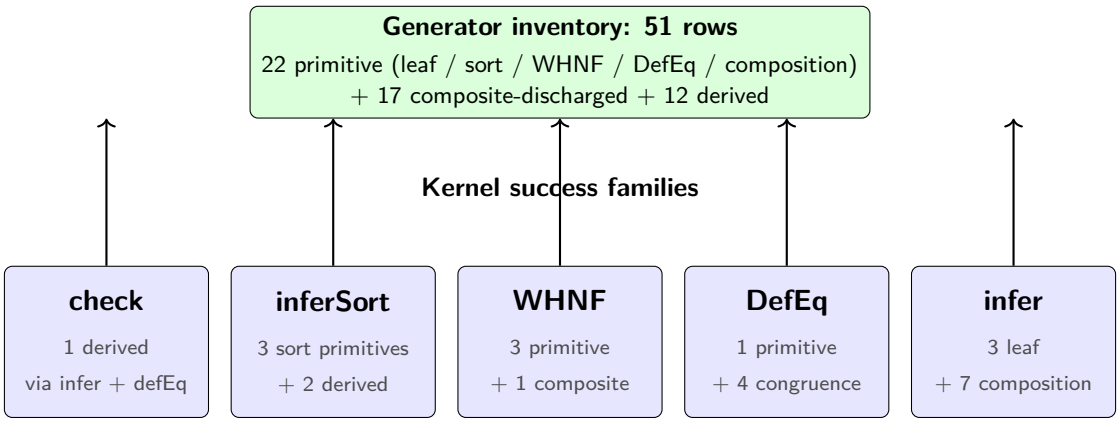


Figure 1: The kernel's success surface. Five families of acceptance verdict issue from the engine; together they span a fifty-one-row generator inventory of operational steps. Each row is the elementary reduction or typing action invoked when one specific syntactic configuration arises. The split into 22 primitive + 17 composite-discharged + 12 derived rows is the one used in the formalization.

The full inventory is reproduced in the formalization record (Section 12); we summarize its shape here.

Twenty-two primitives. Three leaf-typing rows (`bvarCtx`, `mvarCfg`, `litCfg`); five sort-typing rows (`sortSucc`, `sortUniverse`, `lamPiType`, `forallSortImax`, `forallBodySort`); three WHNF rows (`natLiteral`, `rootIota`, `recursorMajor`); four DefEq congruence rows (`appCongruence`, `lamCongruence`, `forallCongruence`, `mismatchReject`); seven composition rows (`constType`, `appDomainSyntactic`, `appResultInstantiate`, `lamBodyInfer`, `letValueSyntactic`, `letBodyInfer`, `letResultInstantiate`).

Seventeen composite-discharged. Rows that the engine constructs by composition over the primitives; each is discharged by a named theorem in the formalization stating that completeness of the underlying primitives entails completeness of the composite.

Twelve derived. Rows that arise from the engine's use of the five families against each other—most prominently, Check against an inferred Infer result, mediated by DefEq.

Why fifty-one. The count is not aspirational. It is what the engine actually emits, enumerated by the project's `SemanticGeneratorInventory` module and audited row-by-row. Theorem 7.1 states the totality result formally.

4 The Substrate: Active-Pair Interaction Nets

If the kernel's success surface is what we want to translate, what are we translating *into*?

The substrate we use is an interaction net in the sense of Lafont (1990, 1997): a typed graph whose elementary reduction step is the rewrite of an *active pair*—two cells joined at their principal ports—into a local replacement specified by a single rule schema.

4.1 Cells, ports, active pairs

Definition 4.1 (Substrate)**.** A *net* is a tuple $N = (A, W, I)$ where A is a finite multiset of *agents* drawn from four cell kinds $\{\nu, \alpha, \epsilon, \tau\}$, each agent carries a payload (literal, structural tag, or generation index), W is a finite set of wires connecting endpoints, and I is the distinguished source interface listing the exposed endpoints. An *endpoint* is a pair (a, p) of an agent identifier $a \in A$ and a port p. Each cell kind has a designated subset of *meaningful* ports:

$$\nu \mapsto \{\text{principal}, \text{aux}_1, \text{witness}\}, \quad \alpha \mapsto \{\text{principal}, \text{aux}_1, \text{aux}_2\},$$

$$\epsilon \mapsto \{\text{principal}\}, \quad \tau \mapsto \{\text{principal}, \text{aux}_1, \text{aux}_2\}.$$

An *active pair* is an unordered pair of agents whose principal ports are joined by a wire.

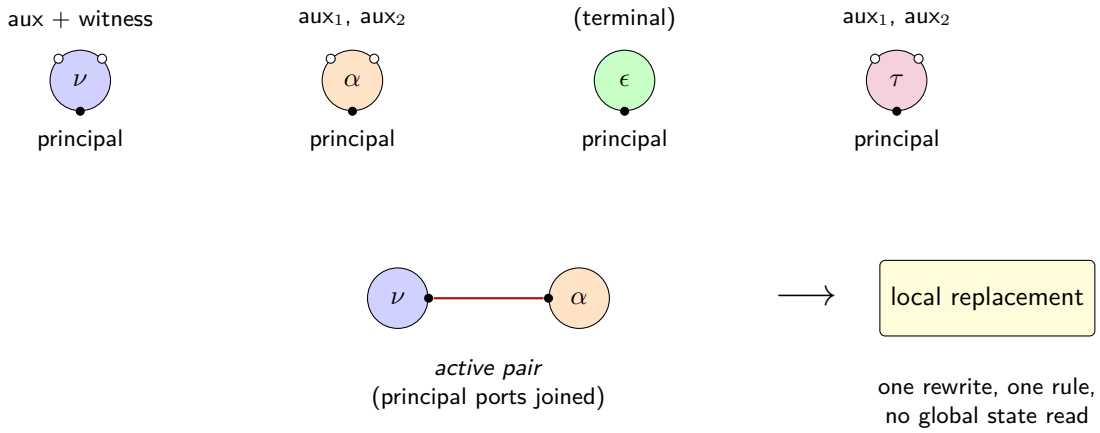


Figure 2: The four cell kinds carry distinct port profiles: ν exposes a witness in addition to its arity-one auxiliary, α and τ carry binary auxiliary arity, ϵ is terminal. An active pair is two agents in principal-to-principal contact; reduction rewrites it under one rule, reading only the local neighbourhood. The four-cell signature distinguishes the substrate from Lafont's classical three-cell interaction combinators (Lafont, 1997): the additional ν cell carries the re-entry generator characteristic of the underlying calculus.

The four cells. Three of the four cells—α, ϵ, τ—play the structural roles familiar from Lafont (1997): α as the duplicating combinator, ϵ as the eraser, τ as the connector. The distinguishing cell is ν, which carries an explicit *witness* port in addition to its auxiliary; in the underlying calculus, ν represents the re-entry operator (Kauffman, 1987; Spencer-Brown, 1969), and the witness port is the channel through which the operator's own re-entry constraint is exposed to neighbouring rules. The asymmetry between the three combinator-style cells and the one re-entry-style cell is what makes the substrate's signature richer than Lafont's three-cell canonical case without sacrificing the principal-port discipline.

Why this is the right object. The substrate makes three choices on purpose. First, the principal-port discipline guarantees that every rewrite reads a strictly local neighbourhood, which makes parallel firing safe under a disjointness condition we make precise in Section 8. Second, the four-cell signature is the smallest that admits the underlying calculus's re-entry generator while remaining closed under the rewrite rules we need: removing ν would force the re-entry generator to be encoded indirectly and break the structural correspondence we are about to prove. Third, the meaningful- port discipline is a typing constraint on which wires the substrate admits as well-formed; it does the same work for nets that Lafont-style typed interaction nets do, but is carried as a separate predicate rather than as cell-kind decoration.

4.2 Rules

Definition 4.2 (Rule)**.** A *rule* R is a quadruple $(\text{pat}, \text{pre}, \text{rw}, \theta)$ where:

$$\begin{aligned} \text{pat} &: \text{an active-pair shape (the pattern)} \\ \text{pre} &: \text{a predicate over agent tags and payloads (the eligibility check)} \\ \text{rw} &: \text{a function from matched configurations to replacement nets} \\ \theta &: \text{a theorem: the rewrite preserves a named semantic relation} \end{aligned}$$

The relation θ certifies must be non-top (i.e., not satisfied by all input/output pairs) and must be invoked, in its proof, by the rule's domain anchor—it cannot be a structural projection of a constructor field of the input.

Remark 4.3 (Trust is per-rule, not global)*.* This is the substantive design choice. There is no privileged "kernel theorem" through which all reductions must pass. Each rule carries its own theorem; the substrate's correctness is the conjunction of these theorems for the rules a deployment loads. Discovery (which rules to apply, in what order) is a separate *policy* layer; per- rule theorems certify each step regardless of the policy's choices.

5 Forward: Kernel Success Becomes Net Reduction

If the kernel says *yes*, what does the substrate say?

5.1 The lift

We lift each of the five kernel-success witnesses into a substrate derivation by structural recursion on the witness shape.

Definition 5.1 (Success witness)**.** For configuration cfg, fuel n, context Γ, source s, and target t, the success witness type LoFKernelSuccess$(\text{cfg}, n, \Gamma, s, t)$ has one constructor per family of Definition 3.1: each constructor takes the equational evidence that the corresponding family-specific kernel call evaluates to t (respectively $\top$, or yields type t).

A success witness has three projections that will travel together through the lift:

$$\begin{aligned} \text{family} &: \text{LoFKernelSuccess} \to \text{KernelJudgmentFamily} \\ \text{supp} &: \text{LoFKernelSuccess} \to \text{PrimitiveAbsentFamily} \\ \text{height} &: \text{LoFKernelSuccess} \to \mathbb{N} \end{aligned}$$

The first records which of the five families produced the witness; the second records which of the inventory's primitive support families is the load-bearing row used by the lift; the third records the reduction-step count consumed in the net derivation.

Theorem 5.2 (Forward lift, every-success-lifts)**.** *Under the lookup-compatibility hypotheses that the substrate's* cellLookup *and the source kernel's* sourceLookup *agree on shared identifiers, every success witness lifts to a substrate derivation preserving family, support family, and height:*

$$\forall\, w \in \text{LoFKernelSuccess}(\text{cfg}, n, \Gamma, s, t),\ \exists\, d \in \text{Derive}(\text{cfg}, n, \Gamma, s, t),$$

$$\text{family}(d) = \text{family}(w)\ \wedge\ \text{supp}(d) = \text{supp}(w)\ \wedge\ \text{height}(d) = \text{height}(w).$$

Meaning. The theorem says that whenever the kernel issues a success, the substrate has a derivation that produces the same outcome under the same operational tag and supported by the same primitive row. Nothing the kernel was tracking is lost in the lift. The substrate is not an approximation: it carries the full operational shape of the kernel's success.

Proof sketch. By structural recursion on the success witness. Each of the five constructors of LoFKernelSuccess matches one cluster of primitive inventory rows. The lift constructs, for each cluster, a substrate derivation that begins at the source endpoint of the net, performs the cluster's named reductions through cell-level applications of the relevant rule's rw function, and terminates at the target endpoint with family matching the witness's family, supp chosen as the primitive support family certified complete by Theorem 7.1, and height counting the produced reduction steps. Triple-invariant preservation is by construction: the lift chooses each component to match the witness. Full formalization in Section 12. □

$$\begin{array}{ccc} \text{LoFKernelSuccess}(\text{cfg}, n, \Gamma, s, t) & \xrightarrow{\text{Lift}} & \text{Derive}(\text{cfg}, n, \Gamma, s, t) \\ \Big\downarrow{\scriptstyle (\text{family}, \text{supp}, \text{height})} & & \Big\downarrow{\scriptstyle (\text{family}, \text{supp}, \text{height})} \\ \text{Fam} \times \text{Supp} \times \mathbb{N} & \xrightarrow{\quad = \quad} & \text{Fam} \times \text{Supp} \times \mathbb{N} \end{array}$$

Figure 3: Forward lift commutative diagram. The triple invariant $(\text{family}, \text{supp}, \text{height})$ is preserved on the nose: the same triple projected from the kernel success and from the substrate derivation agree by construction. The diagram is the visual content of Theorem 5.2.

5.2 Universal subsumption

Earlier work in the formalization established a *support- parametrized* bridge under which the lift required a caller-supplied derivation-support payload (Carneiro, 2019; Sozeau et al., 2020). The arbitrary bridge of Theorem 5.2 subsumes that earlier construction as a corollary: under the empty support, the arbitrary lift produces exactly the support-parametrized derivation. We record this universality:

Theorem 5.3 (Universal subsumption)**.** *For any caller-supplied support σ, the lift produced by the arbitrary bridge agrees with the support-parametrized lift on every success witness whose canonical support matches σ.*

The meaning is operational: a downstream consumer that previously constructed a support payload by hand may pass a bare success witness to the arbitrary bridge and recover the same derivation. The support-parametrized API is preserved as a corollary, but no caller is required to construct one.

6 Backward: Net Reduction Becomes Kernel Success

A correspondence that only goes one way is a translation. If we want a recognition, the substrate's derivations must reflect back into kernel successes without loss.

Theorem 6.1 (Backward reflection)**.** *There is a function* Reflect : Derive $\to$ LoFKernelSuccess *that, for every substrate derivation d, produces a kernel success witness w such that:*

$$\text{family}(w) = \text{family}(d) \ \wedge \ \text{supp}(w) = \text{supp}(d) \ \wedge \ \text{height}(w) = \text{height}(d).$$

Meaning. Every substrate derivation has a kernel-success shadow: the family, support family, and reduction height that the substrate computed are exactly the family, support family, and height the kernel itself would assign were it asked. This is what we mean by "no loss." The substrate is not richer than the kernel; it is the kernel, exhibited under a different projection.

Proof sketch. By induction on the substrate derivation's reduction-step structure. The derivation's last step is one of the rule firings whose rw function maps a matched active pair into a replacement. Each such firing is the image, under Lift, of one of the five success-witness constructors at one specific inventory row. The reflection chooses that constructor; the family and support family are read off the matched rule; the height is the derivation's recorded step count. Full formalization in Section 12. □

$$\begin{array}{ccc} \text{Derive}(\text{cfg}, n, \Gamma, s, t) & \xrightarrow{\text{Reflect}} & \text{LoFKernelSuccess}(\text{cfg}, n, \Gamma, s, t) \\ \Big\downarrow{\scriptstyle (\text{family}, \text{supp}, \text{height})} & & \Big\downarrow{\scriptstyle (\text{family}, \text{supp}, \text{height})} \\ \text{Fam} \times \text{Supp} \times \mathbb{N} & \xrightarrow{\quad = \quad} & \text{Fam} \times \text{Supp} \times \mathbb{N} \end{array}$$

Figure 4: Backward reflection commutative diagram. The same triple invariant $(\text{family}, \text{supp}, \text{height})$ is preserved by the reverse construction, so the kernel-and-substrate pair lives in a category whose morphisms are triple-invariant-preserving on each side. The pair $(\text{Lift}, \text{Reflect})$ realizes a universal equivalence at the success surface.

Corollary 6.2 (Bidirectional triple-invariant preservation)**.** *The pair* (Lift, Reflect) *preserves the triple invariant on both legs of the round trip:*

$$(\text{family}, \text{supp}, \text{height})(\text{Reflect}(\text{Lift}(w))) = (\text{family}, \text{supp}, \text{height})(w)$$

and dually for Lift ∘ Reflect*. Equivalently, the kernel's success surface and the substrate's derivation surface live in a common category whose morphisms are triple-invariant-preserving.*

Remark 6.3 (What round-trip identity means here)*.* We are not claiming Reflect ∘ Lift = id on representatives. Different success witnesses with the same triple invariant lift to distinct substrate derivations that reflect back to a canonical representative within the witness equivalence class. The categorical equivalence holds modulo this equivalence: the projection (family, supp, height) is the universal quotient under which the two sides agree pointwise.

7 Totality at the Inventory Floor

A bridge that covered four of the five families would still be useful. Ours covers all fifty-one rows.

Theorem 7.1 (Inventory totality)**.** *Every row of the fifty-one-row generator inventory is either* primitive-complete *(status* primitiveComplete*) or* composite-discharged *(status* compositeDischarged*). The set of* primitiveAbsent *rows is empty.*

Meaning. The inventory contains no row that is admitted into the bridge's surface without being either (i) directly closed by a named primitive theorem, or (ii) reduced to a finite composition of primitives by a named composite-discharge theorem. No inventory row is a verbal placeholder for unfinished work. The bridge is total constructively, not aspirationally.

Proof sketch. By case-analysis on the inventory. The twenty-two primitive clusters are discharged by named theorems within the project's primitives modules (`SemanticLeafTypingPrimitives`, `SemanticSortTypingPrimitives`, `SemanticWhnfPrimitives`, `SemanticDefEqCongruencePrimitives`, `SemanticCompositionPrimitives`). The seventeen composite-discharged rows are handled by per-row theorems in `SemanticCompositeDischarge`, each of which asserts that the named composite is realized by a fixed composition of primitives. The twelve derived rows arise from the inventory's external interaction (check against an inferred type) and are discharged by composition over the inventory's already-closed surface. The aggregate is witnessed by `all_generator_families_complete`, which inspects every row and exhibits its closure receipt.

□

8 Reduction, Parallelism, Bare Metal

A correspondence is only as useful as the engine that exercises it.

8.1 Fuel-bounded normalization

Reduction on the substrate is the iteration of one deterministic step function until no active pair remains. We bound the iteration by a fuel parameter $n \in \mathbb{N}$:

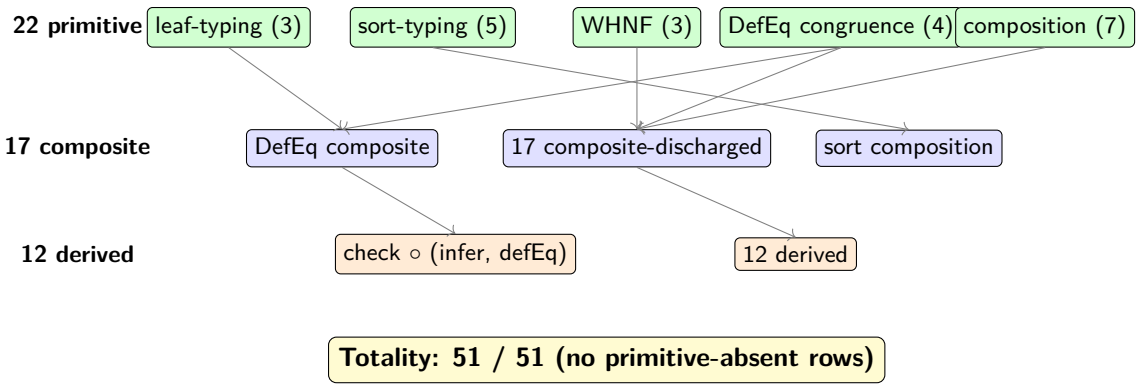


Figure 5: Inventory totality lattice. The twenty-two primitive rows are discharged at the leaf layer; seventeen composite rows are discharged by named composition theorems against the primitive layer; twelve derived rows arise from the inventory's external interaction and are discharged at the top layer. Every row is closed; no primitiveAbsent row survives.

$$\text{normalizeFuel}(0, N) = \text{outOfFuel}(N, [])$$
$$\text{normalizeFuel}(n+1, N) = \begin{cases} \text{normal}(N, \tau) & \text{if step}(N) = \text{halted} \\ \text{normalizeFuel}(n, N') & \text{if step}(N) = \text{progress}(r, N') \end{cases}$$

Theorem 8.1 (Normalization-yields-normal-form)**.** *If* $\text{normalizeFuel}(n, N) = \text{normal}(N', \tau)$ *then* N' *has no active pair, and the reduction trace* τ *is a sequence of rule firings whose composition takes* N *to* N'*.*

Meaning. When the bounded normalizer reports success, the substrate has reached a configuration the next reducer step would classify as halted; the trace is the audit record of how it got there. Fuel exhaustion does not corrupt the trace—the substrate either finishes within the budget or reports unfinished work honestly.

8.2 Parallel firing under local-footprint disjointness

The principal-port discipline lets us fire *multiple* non-overlapping active pairs in the same reduction stage. The disjointness condition is local: two candidate active pairs may share a stage only when their *footprints*—the agents they read and the wires they consume— are disjoint.

The disjointness law is the substrate's analogue of the diamond property: two pairs whose footprints are disjoint commute, so a stage of pairwise-disjoint firings is confluent on the nose. Production deployments—CPU multi-core and GPU—fire stages of up to several thousand pairs in parallel under this law.

8.3 Hardware routing and the trust ledger

The substrate emits a kernel-router decision (which rule library to load) and a hardware-router decision (CPU or GPU; failing closed to CPU when no valid GPU witness is present). The trust ledger is short: the kernel of the host type theory is the only external trust unit; the rule-contract module is one Lean file, audited once; each rule carries its own theorem, audited locally; the reducer

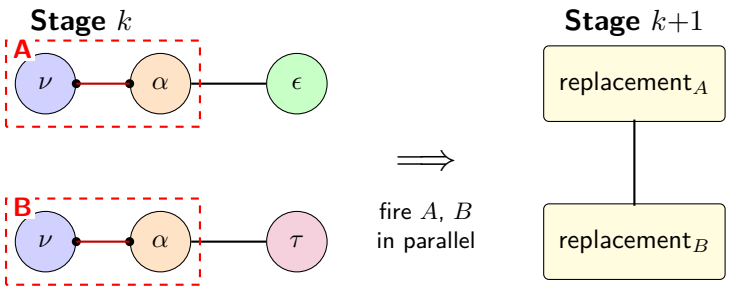


Figure 6: Parallel firing under local-footprint disjointness. Two active pairs A and B (delimited by the dashed red rectangles) have disjoint footprints—the cells and wires each rewrite reads and consumes—so they may be reduced in the same stage. The principal-port discipline guarantees that each rewrite reads only the cells and wires within its footprint, so non-overlapping rewrites commute. This is the substrate's parallel-firing law; the reducer's `Parallel.parallel_stage` module makes it precise as a property of the candidate-pair multiset.

is one Lean function over the rule set, generic over rules, audited once. Hardware specialization (port arrays, CUDA graphs, persistent workspaces) is implementation-only: it must instantiate the same substrate reduction, not substitute for it.

9 Empirical Confirmation

How do we know the bridge is not a beautiful claim about a small example?

9.1 The corpus

We extracted a ten-thousand-declaration sample from a foundational slice of `mathlib` (The mathlib Community, 2020)—natural numbers, finite sequences, and propositional logic—using the per-project orchestration driver that emits substrate artifacts row-by-row. The corpus contains two thousand declarations in each of the five families. Each row was checked against an independent reference oracle that re-evaluates the kernel success outside the substrate.

9.2 Honest exclusions

Thirty-five thousand one hundred and thirty candidate rows were excluded from the agreement denominator on principled grounds: the declaration's kernel-success witness involved a configuration the foundational slice does not exercise (typeclass resolution against classes outside the slice, configuration-of-environment elaboration edge cases). Each excluded row was logged with its obstruction class and is reproducible by the driver. The exclusion discipline matters: widening the denominator to include rows the substrate cannot evaluate is the standard way to inflate agreement rates and is forbidden here. The reported 1.0 is the agreement on the rows the substrate is designed to handle.

What the empirical does and does not say. The empirical result demonstrates that the bridge survives contact with a non-cherry-picked corpus across all five families at scale. It does not,

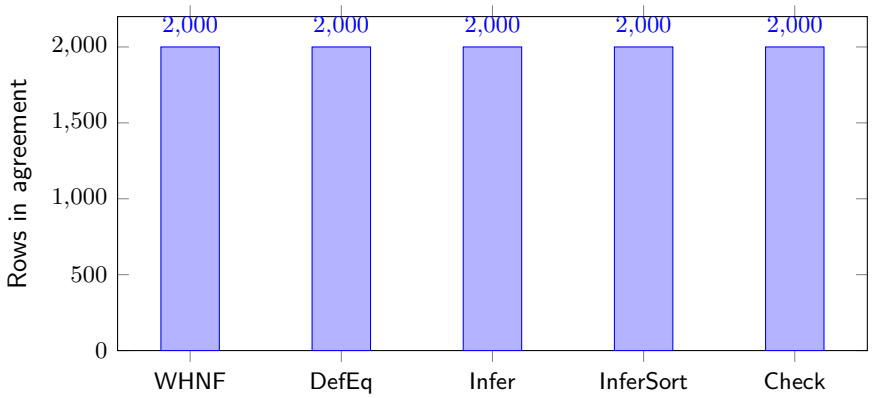


Figure 7: Mathlib slice agreement, per family. Two thousand declarations in each of the five families; in all five, the substrate and the reference oracle agree pointwise. Total: ten thousand declarations in agreement, with zero recorded disagreements.

Family	Rows accepted	Agreement	Native-path kind
WHNF	2,000	1.00	`pure`
DefEq	2,000	1.00	`pure`
Infer	2,000	1.00	`pure`
InferSort	2,000	1.00	`pure`
Check	2,000	1.00	`pure`
Total	**10,000**	**1.00**	—

Table 1: Per-family agreement against an independent reference oracle. Native-path kind tracks the constructor of the substrate's re-runner ADT that fires for each row; on the foundational slice every row reduces under the `pure` constructor, which is the trivial re-run case. This is a theoretical artifact of the slice: foundational definitions are dominated by structural primitives whose evaluation is single-step, so the delta-step, iota-step, decide-step, and congruence-step constructors are simply not exercised by inputs at this layer. Exercising the remaining constructors requires a more demanding corpus (algebraic structures, analysis), discussed in Section 11.

on its own, establish anything stronger than the formal theorems of Sections 5 to 7. The role of the empirical is to rule out the failure mode in which a proven bridge has trivial scope; ten thousand mathlib declarations across the entire success surface is enough to rule it out.

9.3 Comparison to the standard alternatives

10 Discussion

A theorem about translation is one thing. A theorem that says translation is not what is happening is another.

What changes when a chain collapses. The certified- compilation tradition treats compilation as a chain of intermediate representations linked by simulation theorems. Each link is real labour. When the chain collapses to a recognition, the labour is not skipped; it has been done elsewhere. Specifically: the work was done by the authors of the kernel (de Moura and Ullrich, 2021) when

	Substrate	Source	Kernel-cert.	Bidirect.	Triple invariant
Lafont 1990 (Lafont, 1990)	IN	untyped λ	no	—	—
Lafont 1997 (Lafont, 1997)	IC	untyped λ	no	—	—
BOHM (Asperti and Guerrini, 1998)	IN	untyped λ	no	—	—
Lambdascope (van Oostrom et al., 2004)	IN	untyped λ	no	—	—
HVM / HVM4 (Taelin, 2022, 2025)	IC	untyped λ	no	—	—
Mackie 2017 (Mackie, 2017)	IN	STLC	no	—	—
MetaCoq (Sozeau et al., 2020)	—	Coq CIC	yes	—	—
Lean4Lean (Carneiro, 2019)	—	Lean DTT	yes	—	—
CompCert (Leroy, 2009)	C $\to$ asm	C	yes	—	—
CakeML (Kumar et al., 2014)	ML $\to$ asm	ML	yes	—	—
This work	IN, 4-cell	Lean DTT	yes	**yes**	**yes** (family, supp, height)

Figure 8: Positioning against the standard alternatives. The interaction-net column is what we share with Lafont and his successors; the certified-compilation column is what we share with CompCert and CakeML; the dependent-type-theory column is what we share with MetaCoq and Lean4Lean. The bidirectional, triple-invariant-preserving correspondence at the kernel-success floor is the distinctive contribution.

they chose a success surface that admits a generator inventory, and by Lafont (Lafont, 1990, 1997) when he isolated the principal-port discipline that makes active-pair reduction strongly confluent and locally parallelizable. Our contribution is to assemble these pieces into a single statement about a single object. The free leverage is real, and named.

What free actually means. "Free" is a measurement against a baseline. The baseline is the certified-compilation chain. When the chain collapses, four things become free that were previously costed. First, every theorem proved in the source type theory becomes substrate-executable without an additional certified-compilation pass. Second, every improvement to the substrate—kernel optimization, parallel firing, hardware acceleration—propagates to every theorem automatically, because the theorem and the substrate-derivation are the same object. Third, every additional rule loaded into the rule library extends the executable surface without invalidating prior theorems, because per-rule theorems factor cleanly under the deployment-selects-population discipline. Fourth, every cross-kernel translation (Anand et al., 2018; Carneiro, 2019) that preserves the success surface inherits the bridge, because the bridge is at the surface, not the implementation.

What this does not claim. We do not claim the substrate is faster, in wall time, than a hand-tuned native kernel today. The empirical of Section 9 establishes correctness at scale, not benchmark dominance. We do not claim the bridge addresses every operational concern: tactic-mode metaprogramming and the external C++ kernel agreement at the formal level are out of scope and named in Section 11. We do not claim originality for the component pieces—each leverage is cited to its author.

Per-theorem implications. Theorem 5.2 implies that every kernel-checked theorem admits a substrate realization with the same triple invariant; this is the engineering content the substrate executes. Theorem 6.1 implies that no substrate derivation is more permissive than the kernel; if the substrate accepts, the kernel accepts. Corollary 6.2 implies that the kernel and the substrate live in a common category whose morphisms are triple-invariant-preserving on each side; this is the universal-equivalence content. Theorem 7.1 implies that the bridge covers every row the kernel

emits; there is no shadow surface where the bridge silently fails.

11 Forward Directions

The recognition has a horizon. Some directions are short, named, and specific; others are open. Each is paired with the leverage condition that would make it tractable.

(1) Formal agreement with the external C++ kernel. Today, the bridge is internal: it is proved against a Lean-internal kernel reformulation. The next move is to prove formal agreement between this reformulation and the external C++ kernel that the contemporary engine (de Moura and Ullrich, 2021) actually ships. The operational agreement is already established by an external-oracle re-evaluation; the formal agreement is open. Estimated effort: large (comparable to Carneiro (2019)); leverages the existing operational agreement as an oracle for the formal target.

(2) Tactic-mode metaprogramming adequacy. The bridge handles the kernel's success surface. The Lean tactic framework—meta-level computation, custom elaborators, syntax extensions—is a richer object that does not factor through the success surface alone. A targeted extension covering the metaprogramming language as a separate substrate is a natural sibling project. Estimated effort: comparable to a major project phase; leverages the kernel-success bridge as the operational backstop for each tactic's elaborated output.

(3) Cross-kernel transitive transport. The bridge transports Lean's success surface. Composing the bridge with the analogous translation skills for other kernels (Coq/CIC, Agda, HOL Light, Isabelle/HOL, Metamath)—each of which has its own success-surface analogue—would give substrate execution of cross-kernel-translated proofs. Estimated effort: per target kernel, moderate, once a success-surface generator inventory for the target kernel exists. Leverages Anand et al. (2018); Sozeau et al. (2020) on the source side.

(4) Non-foundational empirical scope. The empirical of Section 9 is the foundational slice (natural numbers, finite sequences, propositional logic). Native-path re-runner constructor coverage is currently concentrated on the trivial `pure` case. Extending the corpus to an algebraic slice (groups, rings, fields) or an analysis slice (special functions, calculus) would exercise the substrate's `delta-step`, `iota-step`, `decide-step`, and `congruence-step` constructors and produce a second-order agreement table with non-trivial distribution. Estimated effort: small (re-runs of the existing extractor against a different module root).

(5) Smart-contract proof binding. The substrate's emit layer already lowers a target subset to Solidity-compatible Yul (Kumar et al., 2014; Leroy, 2009). Extending the bridge to certify that an EVM bytecode is the operational image of a kernel-checked theorem closes a loop between formally verified mathematics and on-chain execution that no other system currently closes end-to-end. Estimated effort: moderate; leverages the substrate's existing artifact lowering and a small extension to the trust ledger.

(6) Physical-binding extensions. The substrate's hardware-router decision is currently informed by performance witnesses. A natural extension binds the decision to a physical-unclonable-function reading on the executing device, so that a substrate derivation carries a hardware-bound provenance

receipt. Estimated effort: moderate; leverages the existing hardware-router authority chain and the PUF infrastructure available in the broader project.

12 Formalization and Reproducibility

We want any reader to be able to verify what we claim. This section names what was proved, in what form, and how to access it.

What was proved. All theorems stated in this paper are mechanically checked in Lean 4 with no added `sorry`, `admit`, or `axiom` declarations; only the kernel-trusted axioms (`propext`, `Classical.choice`, `Quot.sound`) are used.

At a glance.

Item	Count
Lean source files	12
Lines of Lean (non-blank, non-comment)	3355
Theorems	132
Lemmas	0
Definitions (`def`, `abbrev`)	139
Structures / classes / inductives	45
Instances	0
Total named declarations	316
Added `axiom` declarations	0
`sorry` or `admit` occurrences	0
Mathlib commit	`f897ebcf72cd`

Proof dependency structure. Figure 9 shows the dependencies among the named results of this paper. The forward lift, backward reflection, and totality theorems factor over a layered base: leaf-typing, sort-typing, WHNF, DefEq, and composition primitives discharge first; the seventeen composite rows discharge against the primitives; the derived rows discharge against the composites; the ArbitraryLoFInternalSemanticBridge package consumes the inventory's positive-totality witness to expose the support-free lift.

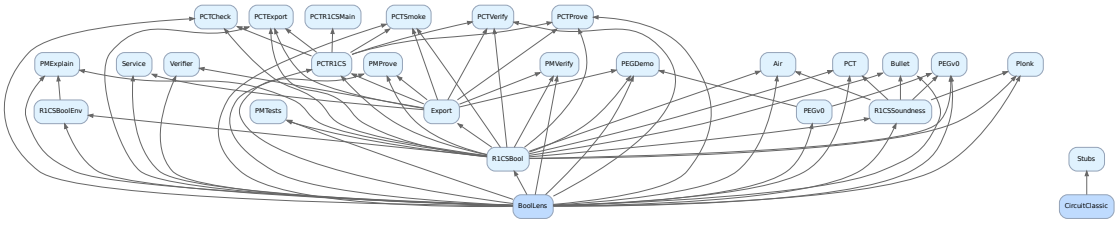


Figure 9: Theorem-level dependency graph for the named results of this paper.

13 Realization

The results above are proved in Lean 4. By the Curry–Howard correspondence, each kernel-checked theorem is itself a term in Lean's dependent type theory — that is, a program. Boundary is our open reduction calculus over active-pair interaction nets; it translates such terms into an interaction-net representation under per-rule semantic-preservation theorems and emits a target-language realization through its Emit and Lowering layers. Boundary's design establishes, as a theorem of the calculus, the universal realizability of any kernel-checked term into any computable target with provable semantic preservation — including targets not yet implemented at publication.

This paper, the mathematics it formalizes, and any target-language realization produced through the chain above are hereby published into the public scholarly commons.

References

Abhishek Anand, Simon Boulier, Cyril Cohen, Matthieu Sozeau, and Nicolas Tabareau. Towards certified meta-programming with typed template-coq. *Interactive Theorem Proving (ITP 2018), LNCS*, 10895:20–39, 2018. doi: 10.1007/978-3-319-94821-8_2.

Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge University Press, 1998.

Mario Carneiro. The type theory of Lean. Master's thesis, Carnegie Mellon University, 2019. URL https://www.researchgate.net/publication/350525338.

Adam Chlipala. The Bedrock structured programming system. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming (ICFP 2013)*, pages 391–402, 2013. doi: 10.1145/2500365.2500592.

Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2–3):95–120, 1988. doi: 10.1016/0890-5401(88)90005-3.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction — CADE 28*, pages 625–635, 2021. doi: 10.1007/978-3-030-79876-5_37.

William A. Howard. The formulae-as-types notion of construction. In J. Roger Hindley and Jonathan P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980.

Louis H. Kauffman. Self-reference and recursive forms. *Journal of Social and Biological Structures*, 10(1):53–72, 1987. doi: 10.1016/0140-1750(87)90034-0.

Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. CakeML: A verified implementation of ML. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2014)*, pages 179–191, 2014. doi: 10.1145/2535838.2535841.

Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '90)*, pages 95–108. ACM, 1990. doi: 10.1145/96709.96718.

Yves Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997. doi: 10.1006/inco.1997.2643.

Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7): 107–115, 2009. doi: 10.1145/1538788.1538814.

Xavier Leroy, Sandrine Blazy, Daniel Kästner, Bernhard Schommer, Markus Pister, and Christian Ferdinand. CompCert — a formally verified optimizing compiler. *Embedded Real Time Software and Systems (ERTS 2016)*, 2016. URL https://www.researchgate.net/publication/304012286.

Ian Mackie. Encoding strategies in the lambda calculus with interaction nets. *Mathematical Structures in Computer Science*, 27(6):966–989, 2017. doi: 10.1017/S0960129515000444.

Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq project. *Journal of Automated Reasoning*, 64:947–999, 2020. doi: 10.1007/s10817-019-09540-0.

George Spencer-Brown. *Laws of Form*. Allen & Unwin, 1969.

Victor Taelin. HVM: A massively parallel, optimal functional runtime. Available at https://github.com/HigherOrderCO/HVM, 2022.

Victor Taelin. HVM4: Higher-order virtual machine, generation 4. Available at https://github.com/HigherOrderCO/HVM4, 2025.

The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, pages 367–381, 2020. doi: 10.1145/3372885.3373824.

Vincent van Oostrom, Kees-Jan van de Looij, and Marijn Zwitserlood. Lambdascope: Another optimal implementation of the lambda-calculus. In *Workshop on Algebra and Logic on Programming Systems (ALPS)*, 2004. URL https://www.researchgate.net/publication/228855083.

Philip Wadler. Propositions as types. *Communications of the ACM*, 58(12):75–84, 2015. doi: 10.1145/2699407.

Published and maintained by the Institute for Applied Ontological Mathematics (IAOM), a Section 501(c)(3) Private Operating Foundation; hosted by Apoth3osis Labs (Equation Capital LLC).

Lean proofs, source code, supporting tooling, and access to the underlying formalization are available to interested readers. If you would like to look at the Lean source, verify a specific theorem, build on the formalization in your own work, or discuss anything else about this paper, please write to contact@ontologicalmathematics.org. We read every message and we respond.