Acyclic by Design: Interaction Nets for Non-Well-Founded Sets

Preface

The Monist Engine is a GPU-accelerated logic environment built to compute W.V.O. Quine's New Foundations (NF) set theory. The project originated as nf-sketches, a Lean 4 formalization using Bellman-Ford directed graphs to intercept cyclical paradoxes before call-stack exhaustion. The architecture was then rewritten into monist, a Rust-based compiler, to bypass the performance constraints of traditional hierarchical type-checkers. This implementation translates verified topologies directly into lock-free WebGPU compute shaders.

The engine coordinates a hybrid synthetic pipeline that maps logical intent into physical memory. To isolate theoretical validation from physical hardware constraints, the architecture partitions execution across five sequential stages:

1. **Natural Deduction (The Human Interface):** Users write high-level constraints and deploy interactive tactics (intro, apply, rewrite) via the CLI to capture mathematical intent.
2. **The Geometry Layer (CPU):** The interface flattens syntax into a topological matrix. The CPU traces paths via Bellman-Ford shortest-path evaluation to halt negative-weight cycles and exports verified SMT-LIB constraints to Lean 4.
3. **Graph Reduction (The Compiler):** Discarding variable substitution entirely, the compiler translates verified topologies into pure, untyped Interaction Nets driven by spatial combinators.
4. **The Physics Engine (GPU VRAM):** The hardware executor ships these combinator graphs to WGSL compute shaders, where lock-free atomic nodes physically collide and annihilate in parallel.
5. **The Holographic Co-processor:** Operating as an experimental continuous-math accelerator alongside the discrete engine, this system superposes arrays into a high-dimensional wave function using Vector Symbolic Architectures (HDC). It exploits destructive interference to cancel noise in $O(1)$ time before snapping results back to discrete boundaries.

1. The Human Interface

The architecture of the Monist Engine anchors itself at the semantic boundary, capturing abstract mathematical intent and structuring it for immediate physical computation. Developers orchestrate logic through the monist-cli Read-Eval-Print Loop (REPL) and the corresponding FormulaArena. This environment establishes the exact coordinate systems required to project virtual mathematics into strict hardware memory layouts. By adhering to Theodore Hailperin's foundational rules, the system guarantees that all standard mathematical theorems remain deducible within its structural framework (1944, 2).

To minimize computational friction, the engine processes abstract syntax trees using direct denotational semantics. Surpassing the rigid constraints of a base logic built exclusively on the Sheffer stroke (Hailperin 1944, 3–4), the system bifurcates the AST structure. It firmly isolates atomic set-theoretic relations—like equality (eq) and membership (mem)—from the standard first-order calculus of discrete boolean connectives and universal quantifiers. This architectural separation untangles mathematical bounding from logical branching, decisively resolving the historical conflict between natural deduction interfaces and Quinean stratification.

Interaction centers on the monist-cli REPL. Developers issue commands like assume and theorem to construct declarative axioms and set clear mathematical goals. The evaluation targets concrete spatial geometries, bypassing conditional hypotheticals. Tactical commands (intro, destruct) actively strip away logical scaffolding, extracting universal quantifiers and implications from the goal state to map them as isolated, concrete facts in the local context. The rewrite command then substitutes these targeted variables, collapsing high-level semantic abstractions into pure, foundational constraints composed exclusively of equality (=) and membership ($\in$) relations.

Once the tactics flatten the logic, developers initiate the topological handoff via the eval or step commands. The core graph engine instantly computes whether this raw geometry occupies finite spatial bounds or fundamentally collapses into a paradoxical, negative-weight loop. While traditional proof editors like PESCA live inside high-level functional languages like Haskell to assist in derivation construction (Negri and von Plato 2001, xv, 242), monist-cli aggressively replaces software-bound proof trees with tangible spatial structures. The engine pipes SMT-LIB verification witnesses straight into bare-metal WebGPU buffers. This establishes a frictionless transition from interactive theoretical editing to high-velocity hardware execution.

Samuel R. Buss's witnessing theorems for Bounded Arithmetic provide the definitive proof-theoretic anchor for this hardware transition. Buss proves that whenever a formal theory of bounded arithmetic (S_2^1) proves a Σ_1^b-formula $(\forall \vec{x})(\exists y) A(\vec{x}, y)$, a deterministic polynomial-time function g_A exists within the polynomial-time closure $\square_1^p$ to compute the witness y directly from the proof structure (1985). The compiler operationalizes this discovery. By converting natural deduction derivation trees into cut-free normal forms, it extracts the physical execution algorithm. During proof unwinding, the kernel evaluates Buss's canonical Witness predicate across the logical sequents, mapping structural cut-free steps cleanly into executable hardware transducer operations.

1.1. Trust, but Differentially Verify

The system deploys differential equivalence testing to guarantee operational integrity. As a user guides a proof through the monist-cli REPL, the CPU geometry layer (monist-core) processes the variables through a Bellman-Ford cycle detection algorithm. If the formula successfully stratifies, the engine generates an NfValidate.StratificationWitness trace, and the FormulaArena exports this witness as a standard SMT-LIB constraint block.

The system then pipes this SMT-LIB block into the Lean 4 parse-strat interpreter via a lake exe parse-strat --ingest-smt script. This script executes an independent topological Bellman-Ford pass to trace the exact SMT constraints generated by the Rust compiler. This independent validation mathematically proves that the compiled execution topology obeys the relative type-shifting bounds formalized in nf-sketches, specifically utilizing StratificationSoundness.lean and BellmanFordInvariants.lean. These formalized libraries act as capstone soundness bridges, proving that resolving localized constraint graphs satisfies Quine's rigid type-shifting limits without triggering scope leakage.

This geometric verification methodology differentiates the Monist Engine from adjacent proof architectures like Sky Wilshaw's Tangled Type Theory (TTT). Wilshaw establishes a formal model of TTT inside Lean 4, verifying a finite axiomatization (P 1–P 9) by indexing type sorts to a limit ordinal λ and formalizing type-level paths as morphisms in a quiver where a single arrow $\alpha \to \beta$ exists whenever $\alpha > \beta$ (2025, 5, 9–10, 50–52). The nf-sketches architecture captures these identical structural permutations, enforcing them dynamically through distance map relaxations to bypass direct axiomatic ingestion. Proving 1-to-1 equivalence between the Rust evaluator and the native Lean geometric verifier guarantees strict operational parity between bare-metal execution traces and ma-

chine-checked proof certificates. This confines the derivation search space directly to structural realities that are ready for immediate hardware evaluation.[1]

The core graph engine deploys an in_comp topological boundary flag to separate unstratifiable comprehensions from free-floating memory cycles. Unstratifiable comprehensions trigger an Extensionality Collision that halts execution. The topological filter isolates free-floating cycles, classifying them as Strongly Cantorian bedrock.

To process logic without bloating the bare-metal kernel, the current semantic parser monist-parser is restricted to a minimal functional subset: propositional connectives, quantifiers, atomic relations, and basic single-variable comprehensions ($\{x \vee P(x)\}$), leveraging deff macros to expand complex constructs.

Theoretical Syntactic Abstraction

To fully realize Hailperin's abstraction elimination theorems (1944, 2–6), the underlying architectural theory requires an extended interface capable of supporting circumflexed variable abstractions ($\hat{x}\, p$) corresponding to Hailperin's logic L_2. Under this theoretical model, converting these constructs into unique abstractionless transforms within the base logic L_1 would allow the parser to discard virtual class abstractions in favor of primitive quantification and membership relations via Theorem 1.1. Establishing this structural requirement ensures that any extended syntax must strictly reduce all derived mathematical theorems to flat topological matrices optimized for WebGPU execution.

The semantic layer demands a foundational syntax capable of mapping relational topologies without relying on outdated hierarchical type-checkers. The FormulaArena uses the formal syntax established in New Foundations for Mathematical Logic (Quine 1937, 71–72). This framework achieves architectural superiority for bare-metal execution through ontological monism and finite axiomatizability. Evaluating standard Zermelo-Fraenkel (ZFC) computationally requires an unbounded metalanguage or complex extrinsic predicate encoding because it relies on infinite axiom schemas of Separation and Replacement. While von Neumann-Bernays-Gödel (NBG) set theory achieves finite axiomatizability, it forces a two-sorted logic segregating entities into "sets" and

[1] Classical execution models suffer memory exhaustion by treating the entire logical structure as a passive, unified environment. Translating abstract mathematical constraints into exact nonlogical inference rules operates as an essential preprocessing step for continuous physics evaluation. This guarantees the resulting topological matrices map to hardware constraints.

structurally paralyzed "proper classes." The FormulaArena uses NF to preserve a strictly one-sorted, flat ontology where the universal set V and transfinite collections operate natively as first-class sets. Quinean stratification acts as an endogenous functional grammar. This structural filter allows the syntax engine to evaluate complex mathematical objects without allocating memory for distinct ontological sorts.

Users interacting with the FormulaArena establish the engine's functional limits through declarative structural axioms. The system guarantees proof integrity by verifying every tactical operation against foundational rules, such as Hailperin's Axiom R2 (1944, 3–4). Axiom R2 dictates that if $P(y)$ results from replacing each free occurrence of x in proposition $P(x)$ by y, and no bound occurrence of y in $P(y)$ results from this replacement, then $(x)P(x). \supset .P(y)$ operates as an immutable structural axiom. The engine tracks variable scopes and bound occurrences within the primitive syntax to establish logical consistency. It relies on a hybrid synthetic pipeline to fuse theoretical ontology with spatial execution. The human-facing interface permits users to assert classical extensional definitions, like W.V. Quine's Axiom P1 $((x \subset y) \cdot (y \subset x) \supset (x = y))$ (1937, 77), to capture mathematical intent and maintain theoretical familiarity. To translate this high-level semantics into physical topological matrices, the underlying geometry layer operationalizes exact combinatorial constraints, such as Hailperin's own Axiom P1 (Hailperin 1944, 10):

$$(u, v)(E\beta)(x)[x \epsilon \beta . \equiv . x \epsilon u \vee x \epsilon v).$$

Classical set theories enforce extensionality to simplify abstract ontology, collapsing computationally distinct pathways into identical logical objects whenever terminal outputs align. In an operational computing environment, enforcing this axiom globally introduces structural friction. The underlying parser systematically decouples stratified comprehension from extensionality before generating topological matrices. To unify the mathematical environment and process arbitrary sequences, the semantic layer leverages the principle of abstraction to define classes based on predicate satisfaction (Quine 1937, 77). The REPL operationalizes this formulation intensionally: given any condition, there is a class whose members fulfill the condition. The engine abandons global class identity checks at ingestion to preserve the distinct operational syntax required for physical hardware evaluation.

To prevent structural contradictions and topological paradoxes, the system enforces the mechanical rule of stratification. A formula achieves stratification if the integers attached to variables separated by the membership operator ϵ ascend consecutively (Hailperin 1944, 2). To bypass the infinite, redundant series of null classes characteristic of classical hierarchical type theory, the mod-

ified abstraction axiom dictates that for a stratified condition ϕ lacking the variable x, the formal theorem $(\exists x)(y)((y \in x) \equiv \phi)$ holds permanently valid (Quine 1937, 78–79). The graph reduction engine represents Quine's $\in$-chains as directed physical edges residing directly in memory. An $\in$-chain consists of a contiguous sequence $\alpha_1 \in \alpha_2 \in \alpha_3 \ldots \in \alpha_n$. The system mandates uniform topological lengths for these evaluation paths, halting unstratified logical loops to maintain spatial integrity.[2]

The implementation of ordered sequences exposes the geometric constraints governing the human interface. Traditional mathematics defines the Kuratowski ordered pair as $(\{\{x\},\{x,y\}\})$. This structure forces a $+2$ typestate shift between variables, introducing computational bottlenecks into downstream hardware execution. To bypass this limitation, the monist-cli parser's architecture establishes a bifurcated roadmap for sequence handling, splitting future syntactic n-tuple abstraction from current bare-metal binary pairing. For generalized multi-variable sequences, the theoretical roadmap outlines a planned implementation of Hailperin's homogeneous ordered n-tuple representation (Hailperin 1944, 8-9). Formalized recursively via exact unit-class abstractions where ιX denotes $\hat{u}(u = X)$ and $\iota^n X$ denotes $\iota(\iota^{n-1} X)$, the Hailperin n-tuple is structured as:

$$\langle X_1, \ldots, X_n \rangle = \langle \iota^{2(n-2)} X_1, \langle X_2, \ldots, X_n \rangle \rangle$$
$$\text{for } n > 2.$$

Applying the $\iota^{2(n-2)}$ operator guarantees structural homogeneity by assigning identical typestate integers to every variable $x_1, \ldots, x_n$ during weak stratification evaluation.

Currently, for binary relations undergoing bare-metal evaluation, the monist-core geometry solver maps syntax directly into Quine ordered pairs $Q(a, b)$. The Quine pair maintains a rigid zero-weight topological shift, preserving absolute geometric flatness across the Directed Acyclic Graph. The engine relies on discrete primitive constructors (qpair and qproj) to guarantee this structural stability. This ensures variables within deep relational pipelines bypass type elevation limits during hardware execution. This physical routing bypasses the severe constraints of the semantic Wiener-Kuratowski ordered pair:

$$\langle x, y \rangle = \{z \mid z = \{z \mid z = x\} \vee z = \{z \mid z = x \vee z = y\}\},$$

…which demands that z reside exactly two type levels higher than x and y (Crabbé 1994, 486). While bare-metal relational evaluation utilizes Quine pairs to maintain flat topological rout-

[2] This strict bounding of the $\in$-chains functions as the precursor to dynamic shortest-path routing. By enforcing ascending integer values on the initial syntax, the engine guarantees that any paradoxical loop manifests as a detectable negative-weight cycle within the compiled Directed Acyclic Graph.

ing, semantic witness verifiers processing these +2 typestate elevations deploy Rosser's Axiom of Counting ($\forall x \in Nat, x = T(x)$). The verifiers apply Rosser's identity $T^3(sort(x)) = sort(x)$ to evaluate membership relations ($x \in_S y$) across elevated Wiener-Kuratowski tuples in $O(1)$ time without memory explosion (Crabbé 1994, 487). This dual architectural vision combines Hailperin coordinate homogeneity at the human interface with the established Quine typestate flatness in the physical execution arena.

The architecture secures internal coherence through the programmatic declaration of Strongly Cantorian sets. To map abstract sets to physical memory, the system replaces semantic descriptions with Lavinia Randall Holmes's concrete retraction architecture (Holmes 1995, 8–10; Holmes and Alves-Foss 2020, 9–10). Absolute data types, such as Booleans and natural numbers, are formalized as idempotent retractions τ satisfying $\tau(\tau(x)) = \tau(x)$ whose ranges constitute Strongly Cantorian domains (Holmes and Alves-Foss 2020, 43–44). This structural guarantee mandates the existence of a witness mapping κ_τ satisfying the operational identity $\kappa_\tau(\tau(x)) = K[\tau(x)]$, which converts value extensions directly into constant function constructors. This witness authorizes the compiler to execute relative typestate shifts mechanically via the inverse transformation $K_\tau^{-1}(K[\tau(x)])$. Physically and operationally, κ_τ specifies the exact memory storage layout of the data type within machine RAM, providing rigorous computational justification for these type-shifting invariants. This operational invariant isolates a specific sub-universe governed by standard Zermelo-Fraenkel axioms, permitting variables to bypass strict type elevation restrictions during complex pairing operations (Quine 1937, 77). By defining Strongly Cantorian sets via retractions at the semantic boundary, developers guarantee the required typestate flatness. This mechanic allows the unhindered construction of standard mathematical frameworks while satisfying the rigid stratification demands of the underlying computational engine.

To establish the theoretical validity of these typestate elevation invariants, the underlying mathematical framework relies on monotone Körner functions $f : N \to N$ satisfying the inequality $n \leq f(T n)$ (Forster 2025a, 88–89). In pure set theory, establishing that a Körner function commutes with the type-raising operator such that $T f(n) = f(T n)$ proves Rosser's Axiom of Counting. The monist-core architecture bypasses runtime algorithmic evaluation of these complex combinatorial functions. By flattening all typestate relationships into graph edge weights and evaluating them via Bellman-Ford relaxations, the engine structurally enforces uniform level-matching across deeply nested relational structures without incurring runtime functional verification overhead.

2. The Geometry Layer: Weak Stratification and Cycle Detection

Before the Monist Engine routes computation through bare-metal graph topologies, it verifies that the input logic avoids infinite loops and structural paradoxes. By treating abstract set theory as a spatial constraint problem, the Geometry Layer replaces traditional software parsers with topographical matrices, converting typestate validation directly into a shortest-path cycle detection algorithm.

2.1. Syntax is Dead

Transitioning to this geometric validation demands the total eradication of abstract syntax trees. Standard single-pass compiler front-ends rely on Context-Free Grammars (CFGs) and Pushdown Automata. These architectures fail to track Quine's relative integer type-shifts across nested expressions. Because a pushdown automaton uses a single LIFO memory stack, pumping a stratified formula inevitably breaks variable level-matching. This structural limitation causes standard parsers to either reject valid logic or generate unstratified paradoxes.

Theoretical proposals, like Calliope Ryan-Smith's syntax system for exo-stratified formulas, address this by introducing two-dimensional variables denoted as v_n^m, where n identifies the variable index and m embeds an integer typestate directly into the syntax (Ryan-Smith 2025, 310). By imposing strict rules—equality requiring $m_1 = m_2$ and membership requiring $m_1 = m_2 - 1$—this model produces a context-free language bypassable by standard pushdown automata.

Theoretical CFG for Exo-Stratified Set Theory Syntax:

Symbol		Expansions
F	→	$A \mid [F \to F) \mid [\forall V]F$
A	→	$\bot \mid PE \mid PM$
E	→	$\doteq P \mid \star E \star$
M	→	$\in P \star \mid \star M \star$
V	→	$P \mid \star V$
P	→	$v \mid P\sharp$

While this structural transformation into v_n^m allows theoretical parsers to ingest specific typed supersets in $O(1)$ time (Ryan-Smith 2025, 308–310), it fails to generalize to the broader logic. Ryan-Smith's subsequent proofs established that the full formal language of acyclic set theory is mathematically non-context-free. Because context-free parsers fundamentally cannot verify acyclicity during token ingestion, the monist-cli architecture averts the CFG approach. The system abandons formal language grammar at the semantic boundary, resolving typestates dynamically across raw syntax nodes via Bellman-Ford graph weights.

Once interactions strip away the logical scaffolding, the CPU ingests the remaining unvarnished formulas. The monist-core geometry solver initiates the conversion from high-level semantics into physical computation. It transforms raw mathematical expressions into a pure topological matrix. The system maps the logic as a Directed Acyclic Graph (DAG) constrained by integer weights, utilizing a context-sensitive, shortest-path evaluator (Crabbé 1994, 482). This geometric translation forces logical syntax out of semantic ambiguity, rendering the proof as an executable mathematical function.

Graphs form a strong structural abstraction for evaluating complex constraints. Directed graphs utilize ordered pairs of vertices to establish definitive directional flow from a tail to a head (Tarjan 1972, 146). Converting logical formulas into topological matrices maps every unique variable directly to a discrete vertex. Logical relations instantiate the directed edges connecting vertices, with each edge constrained by specific integer weights. The solver applies graph search algorithms to impose a strict directional hierarchy onto the logical graph (Tarjan 1972, 147).[3]

2.2. How Weak is Your Stratification?

Converting syntax into spatial architecture demands rigid adherence to type-level constraints. A weak stratification maps occurrences of pseudo-terms within an expression to specific integer values. These integers function as physical type assignments (Crabbé 1994, 482). This structural assignment obeys strict syntactic rules derived from the relational operators connecting the logical terms. The system evaluates logic based on structural compliance with these numerical constraints (Crabbé 1991, 217).

[3] Algorithms execute a depth-first search to traverse these structures. They select an unexplored edge emanating from the most recently reached vertex. Traversing the chosen edge leads the solver to a new vertex, continuing this process until all edges are explored once (Tarjan 1972, 147).

The system formalizes weak stratification by establishing a function $\sigma : X \to Z$ across the total set of variables X (Ryan-Smith 2025, 306–307). Evaluating a membership relation, denoted formally by $P \in Q$, demands that the integer type assigned to P must equal i if and only if the type assigned to Q is exactly $i+1$ (Crabbé 1994, 482). This syntax requires the containing set to occupy exactly one type level above its element (Crabbé 1992, 113). The topological matrix translates this condition into the spatial difference constraint $\sigma(y) - \sigma(x) = 1$ (Ryan-Smith 2025, 306). This spatial difference constraint operationalizes Marcel Crabbé's Collapsing Lemma (1992, 114–115). Crabbé proves that any typed model satisfying extensionality can be collapsed into an E-standard structure $\langle \langle M'_0, M'_1, \dots \rangle, \in \rangle$ where $M'_{i+1} \subseteq P(M'_i)$. By enforcing atomic stratification assignments σ as geometric edge weights, the bare-metal VRAM arena maps typed virtual sets into flat memory allocations, avoiding complex hierarchy tracking.

The equality relation $x = y$ demands identical type assignments across matched variables, producing the formal restriction $\sigma(x) = \sigma(y)$ (Ryan-Smith 2025, 306). Set abstracts demand further structural rigidity. Within formulas utilizing constructs such as $\{x \vee A\}$, all internal occurrences of the bound variable x receive a uniform integer value i, and the complete abstract term receives the value $i+1$ (Crabbé 1991, 216). Expressions governed by universal quantifiers dictate that all occurrences of the bound variable maintain a singular integer assignment throughout the graph topology.

2.3. Drawing the Lines

Geometric solvers evaluate these conditions by analyzing the logic as a constraint satisfaction problem. The system of difference constraints generated by the weak stratification function maps onto the directed graph geometry. Variables form the vertex coordinate space. Difference inequalities generate the weighted edges. This geometric framework translates syntactic validity into a quantifiable physical distance problem across the network topological space (Crabbé 1994, 482).

The structural mapping assigns precise directional edges based on foundational logical rules. When encountering the equality condition $P = Q$, the engine generates a bidirectional edge between node P and node Q possessing a weight of 0 (Crabbé 1994, 485). This locks the variables at identical typestate levels. The solver addresses the membership relation $P \in Q$ by generating a directed forward edge from node P to node Q bearing a weight of 1. The membership constraint also generates a reverse directed edge from node Q back to node P bearing a weight of -1. This assignment of directed forward edges ($+1$ weight) and reverse edges (-1 weight) structurally mirrors the discrete

representations of Wilshaw's quiver morphisms $\alpha \to \beta$ found in Tangled Type Theory (Wilshaw 2025, 9). The engine establishes its logical soundness by deploying the Bellman-Ford algorithm's shortest-path evaluation. This geometric pass verifies whether variables can populate a relative constraint graph without generating a negative-weight cycle. Resolving this local geometry satisfies the engine's internal IsStratifiedAux foundational bounds. It proves at design time that the parsed formula adheres to Quine's strict type-shifting limits without requiring translation into an infinite type ladder.

The resulting structure relies on an adjacency matrix where $M_{ij} = w$ represents the directed edge weight projecting from vertex i to vertex j, defaulting computationally to infinity for disconnected nodes (Ryan-Smith 2025, 308). Inequalities bounding the difference between two variables generate the corresponding directed edges where the integer limit itself serves as the physical edge weight. A constraint enforcing $x - y \leq c$ produces a directed edge from y to x bearing a weight of c. Membership relations map variables back to a universal zero vertex, establishing relative differences into integer coordinates and cementing strict syntactic law across the matrix.

To demonstrate the solver's algorithmic conversion mechanics, the following raw data set details the matrix parameters bridging continuous logic and graph geometry (Buss 1985, 30).

Logical Constraints and Graph Weights:

Logical Constraint	Source Node	Target Node	Integer Weight
$x - y \leq 5$	y	x	5
$y - z \leq -2$	z	y	-2
$w - x \leq 0$	x	w	0
$x = y$	x	y	0
$x = y$	y	x	0

2.4. Kosaraju's Two-Step

Resolving this matrix requires shortest-path algorithms to confirm logical consistency and calculate structural bounds. The initial graph traversal processes the zero-weight bidirectional edges generated by strict equality constraints. These edges cluster identical-type variables into cyclical rings. The monist-core solver executes Kosaraju's algorithm—utilizing dfsForward, dfsBackward,

findSCCs, and flattenGraph—to identify these Strongly Connected Components. This two-pass sweep uses matrix transposition to isolate distinct islands before shortest-path bounding begins.

To ground this depth-first traversal in structural proof theory, the geometry layer defines the intermediate graph skeletons built during the first pass as normal spanning trees (Diestel 2025, 15–16). When an algorithm explores a network, it lays down a tree-like skeleton connecting all vertices without loops. A spanning tree qualifies as "normal" if the leftover connections from the original graph obey a strict hierarchical rule: they can only link vertices possessing a direct ancestor-descendant relationship. In formal terms, the connected vertices must remain comparable in the tree-order $\leq_T$. By establishing this property during dfsForward, the compiler guarantees that every logical relation maps either to a direct parent-child descent or to a vertical back-edge. The structure prohibits horizontal "cross-edges" jumping between unrelated, parallel branches.

Because the engine enforces this strict verticality, it leverages Diestel's Lemma 1.5.4, proving that any two unrelated cousin branches (incomparable vertices x, y) remain separated. Any path between them must route backward through their shared ancestors, a geometric bottleneck known as the intersection of their up-closures and expressed algebraically as $\lceil x \rceil \cap \lceil y \rceil$. This topological guarantee provides the rigorous justification for Kosaraju's algorithm executing in rapid linear time across the syntax matrix. The structural impossibility of horizontal cross-edges ensures that 0-weight equality cycles (Strongly Connected Components) never straddle divergent subtrees. The contradictions remain isolated within discrete, vertical branches, allowing the engine to sweep and compress them without triggering a combinatorial explosion (Diestel 2025, 16).

```
Algorithm: Kosaraju's SCC Contraction via Two-Pass Depth-First Search
Input: Raw Logical Directed Graph G = (V, E)
Output: Condensed Acyclic Graph G' with 0-weight equality cycles
contracted

1. Execute dfsForward to compute finishing times for all vertices in
G.
2. Transpose the geometric matrix to create G_rev.
3. Execute dfsBackward on G_rev in decreasing order of finishing times
to isolate findSCCs.
4. Execute flattenGraph to collapse each SCC into a single
representative concept vertex c_p.
```

Deploying Kosaraju's two-pass algorithm contracts zero-weight semantic equality rings ($x = y$) into representative concept nodes. This flattens redundant topological clusters across the CPU ge-

ometry layer before Bellman-Ford cycle detection begins. Wilshaw's Tangled Type Theory manages these cyclical non-adjacent typestates mathematically by deploying fuzz maps ($f_{\beta,\gamma}$) and mapping unstratified extensions into discrete near-litter sets (2025, 8, 13). The monist-core engine bypasses these constructs in favor of Bowler's acyclic translation strategy. When Kosaraju's algorithm identifies a zero-weight cycle, the compiler applies an existential projection closure directly to the abstract syntax tree. It generates a structurally localized fresh variable to fuse the cyclical nodes deterministically, bypassing global namespace collisions. Hardware isolates these fused topological rings to manage unresolvable self-referential paradoxes, preventing anomalous memory structures from polluting valid type-level extensions. Weak extensionality confines absolute equivalence requirements strictly to designated objects, enabling non-extensional variables to advance through the solver without triggering widespread constraint failures (Crabbé 1992, 112). The topological matrix adapts dynamically to weak stratification by severing restrictive zero-weight edges. This dynamic adjustment grants specific variables structural immunity from rigid hierarchical typing. Crabbé proves that any total function p operating as a permutation on its range while collapsing all non-range atoms into a single concept C_p yields a model where weakly stratified comprehension holds valid (Crabbé 1992, 117).

To formalize this quasi-permutation, the geometry layer executes a graph minor contraction operation over partition classes (Diestel 2025, 18–19). When an interaction net undergoes strongly connected component contraction, the engine constructs an inflated graph (IX model) by establishing a vertex partition $P=\{V_x \mid x \in X\}$ across connected zero-weight equivalence rings. The resulting contraction minor, denoted formally as G/P, preserves every directed edge of G bridging distinct partition classes (Diestel 2025, 28). Multi-edges bridging identical partition pairs convert into parallel edges between representative concept vertices, while internal equality rings collapse without allocating residual loop buffers. This G/P contraction mapping executes the exact memory transformation necessary to collapse distinct but operationally equivalent nodes into a single representative concept vertex (c_p). This preserves strict comprehension syntax while eliminating redundant graph cycles in system RAM before Bellman-Ford evaluation.

Following the SCC contraction, the geometry layer executes the Bellman-Ford algorithm to compute the minimal bounds existing between all remaining variable nodes (Buss 1985, 30). The search for a valid mathematical stratification transforms into a physical single-source shortest-path problem. The solver assigns an absolute integer potential to every localized node while simultane-

ously satisfying all edge weight constraints. Validating these constraints yields an acyclic, highly optimized structural assignment confirming the expression as stratifiable.

This methodology ties logical definability directly to polynomial-time computational complexity through the foundational axioms of bounded arithmetic.[4]

2.5. Why Universal Evaluation Fails

This geometric execution derives from the foundational limitations of Quine's New Foundations (NF) theory. The category of sets in NF lacks Cartesian closedness (McLarty 1992, 555). A reliable definition of function mechanics demands the ordered pair formula $z = \langle x, y \rangle$ remain stratifiable. Preserving this stratifiability permits fundamental categorical properties like identity functions and valid composites to exist.

Universal evaluation functions violate core stratification constraints. Permitting an evaluation function transpose forces the generation of non-stratifiable relationships, preventing distinct sets from possessing actionable functional mapping (McLarty 1992, 555–556).[5] Colin McLarty provides rigorous algebraic proof of this failure. McLarty proves that any category possessing universal evaluation morphisms $ev : B^A \times A \to B$ permitting function transposes forces internal subobject classifiers to degenerate into trivial topoi. Evaluating the internal classifying function on the diagonal subobject R yields the direct self-referential contradiction $i(R) \in R \leftrightarrow \neg(i(R) \in R)$. This concrete algebraic proof demonstrates why the Monist Engine must prohibit continuous universal evaluation frameworks and execute localized, point-free combinator interactions across VRAM scratchpads.

---

[4] Theories of bounded arithmetic constitute weak fragments of Peano arithmetic. They restrict classical induction axioms to bounded formulas (Buss 1985, 2). The base theory S_2^1 defines the exact set of polynomial-time computable functions (Buss 1985, 65). Functions proving Σ_i^b-definability within S_2^i map to the polynomial-time closure of Σ_{i-1}^p (Buss 1985, 62). Predicates contained within Σ_1^p, representing NP logic, correspond to predicates expressible by Σ_1^b-formulas (Buss 1985, 20–21). Converting logical proofs into physical structures stripped of free cuts guarantees that valid proofs of Σ_i^b-formulas contain algorithms (Buss 1985, 74). These internal algorithms provide the actionable framework to compute formula witnesses through graph traversal (Buss 1985, 4).

[5] A trivial topos contains an isolated object T where every other object holds a monic directed to T (McLarty 1992, 555).

2.6. Catching Paradoxes in the Act

To expose the physical execution limits of the engine, developers construct topological anomalies like Russell's Paradox. This introduces the comprehension cycle. While Kosaraju's algorithm contracts zero-weight equality edges to optimize graph traversal, it operates to simplify the shortest-path coordinate space without imposing global extensional equivalence. When dynamic variable re-leveling under weak stratification collapses into an unresolvable cyclic dependency, normalization breaks down.

The evaluation forces the vertex R to simultaneously hold a $+1$ and -1 positional weight relative to its own coordinate point. Algorithmic rules dictate the relation generates a directed edge looping from R to R with weight 1, alongside a reverse edge bearing weight -1. The engine maps this contradiction natively: the breakdown surfaces within the Bellman-Ford matrix as a negative-weight cycle.

By calculating the Minimum Cycle Mean (MCM) and tracking K-Iteration depth bounds, the engine geometrically isolates paradoxes. Traversing an erratic cycle yields a continuous path length less than zero. The Bellman-Ford algorithm flags the negative-weight cycle, intercepting mathematical impossibilities prior to execution scaling. Detecting a negative-weight cycle geometrically proves the formula cannot resolve into a valid integer matrix. This halts execution structurally and proves unstratifiability through geometric constraint failure. This spatial interception enforces Crabbé's diagnosis of paraconsistent normalization failure within self-membership abstracts (Crabbé 1991, 216). Negative-weight cycles intercept non-normalizable proofs before bare-metal combinator reduction begins, isolating self-referential loops as bounded deadlocks.

2.7. Pulling the Plug

Classical logic engines attempt to evaluate non-well-founded sets and paradoxes through the application of recursive memory loops. Traditional compilers construct abstract syntax trees relying heavily on variable substitution and global memory environments. When forced to process self-membership anomalies, these outdated systems initiate infinite regression sequences. The machine continually allocates call stack memory to chase the unstratifiable relation down the substitution chain until total call-stack exhaustion crashes the operating system.

The monist-core architecture deters this catastrophic vulnerability through spatial interception. The geometry solver evaluates the boundaries of mathematics statically. The Bellman-Ford al-

gorithm's detection of the structural negative-weight cycle terminates the pathway mapping. The solver intercepts the extensionality collision before call-stack exhaustion can begin.[6]

This absolute barrier triggers the engine's topologically-guided evaluation bounds, initiating a hard structural K_ITERATION_HALT. This systemic command ceases all topological parsing operations and returns a detailed failure trace outlining the overlapping edge constraints that breached the typing hierarchy. The user witnesses the physical logic engine rejecting the topological tension natively. By anchoring evaluation to MCM thresholds ($\mu^* < 0$), the compiler freezes paradoxical regression directly into the syntax tree as a Comb.terminal "K_ITERATION_HALT" node.

This mechanism redefines computational paradoxes from volatile runtime vulnerabilities into safely observable geometric boundaries. Acyclic formulas require variables to maintain consistent quantifier bindings while forbidding cyclic relational sequences. While acyclic formulas prevent self-referential paradoxes, Ryan-Smith extends Al-Johar, Holmes, and Bowler's acyclic comprehension framework to prove that the formal language of acyclic set theory is mathematically non-context-free (2025, 308–309). Because context-free parsers cannot verify acyclicity during token ingestion, the monist-core geometry layer must execute Kosaraju's Strongly Connected Components algorithm and Bellman-Ford shortest-path traversal across physical RAM matrices to enforce Birkhoff's acyclic condition. By enforcing this rule physically, the engine ensures that only structurally verified, paradox-free Directed Acyclic Graphs receive full typestate certification. Validated DAGs shed their named variable hierarchies, moving forward to the untyped compiler layer and the bare-metal GPU execution arenas.

This enforcement of acyclicity realizes the topological comprehension constraint proposed by Garrett Birkhoff in 1937. Defining a formula as acyclic if it contains no membership chain beginning and ending with the same variable, Quine and Birkhoff identified the geometric boundary required to prevent self-referential paradoxes (Quine 1937, 80). By evaluating logic through Bellman-

[6] Holmes argues that the "extensionality collision" does not exist mathematically, framing stratification as a preliminary syntactic filter applied before logical evaluation. While this theoretical posture permits the construction of pure New Foundations syntax checkers that categorize collisions as parsing errors, it abstracts away the physical realities of hardware execution. The Monist Engine architecture translates logical intent into tangible spatial topologies where structural tension remains physically observable. Routing combinators through bare-metal interaction nets allows the engine to intercept these boundaries natively as calculable geometric workloads. This ensures the hardware isolates and resolves topological paradoxes through structural constraints.

Ford shortest-path matrices, the monist-core solver enforces Birkhoff's acyclic condition natively in hardware.

---

3. The Compiler: Destroying the Variable

Transitioning to bare-metal spatial computation demands the eradication of alphanumeric identifiers. Currently operating as a standalone low-level compilation target for Rust execution workflows (monist-comb), the compiler layer connects the validated directed acyclic graphs of the geometry solver with the physical execution units of the GPU. It destroys lexical environments, replacing them with nameless indexing and untyped combinatory logic to bypass physical bottlenecks.

3.1. Carrying Too Much Lexical Baggage

Standard computational architectures treat variables as persistent symbolic references, demanding continuous memory allocation, dynamic environment lookups, and hierarchical scope management. When a compiler evaluates a directed acyclic graph representing a set-theoretic proposition, retaining lexical identifiers introduces structural latency. Every function application triggers environment traversal to resolve variable bindings. Higher-order transformations force compilers to execute expensive α-conversions, preventing variable capture across nested scopes.

Moses Schönfinkel established the theoretical foundation for variable annihilation in 1924. Schönfinkel diagnosed the bound variable as an auxiliary, disposable construct, rather than an essential ontological component of logical expressions. Variables act as connective tags, coordinating argument positions with governing operators. Preserving these tags during physical hardware evaluation forces digital processors to waste cycle budgets on associative lookup tables. John Backus demonstrated that applicative systems founded on the lambda calculus introduce complex substitution rules and environment lookups, presenting severe execution barriers to hardware designers (1978, 616, 631). Realizing unrestricted lambda applications on physical machinery forces implementations to embed the applicative language inside inefficient von Neumann storage frameworks to manage variable binding histories.

The monist-comb compiler strips away the lexical environment. By abandoning lambda substitution in favor of fixed functional combining forms, the compiler eliminates variables (Backus 1978, 619–620). As Ryan-Smith notes regarding theoretical exo-stratified models, arbitrarily strip-

ping typestate limits degrades valid stratified expressions into paradoxical structures (2025, 309–310). Obliterating the integer bounds between variables x and y collapses the evaluation into unstratified self-membership $x \in x$. Before the monist-comb compiler strips away standard variable names to compile untyped S, K, I combinators, it transfers the dynamic typestate integer weights generated by the Bellman-Ford geometry solver into the physical interaction graph via T-functor wrappers. Preserving the solver's integer topological weights during variable annihilation prevents valid logic from degrading into unstratified computational deadlock.

Holmes establishes the mathematical validity of this transition via the Abstraction Theorem for synthetic combinatory logic. Holmes demonstrates that once variable binding is eradicated in favor of primitive combinators ($K[T]$, Abst, and the canonical retraction Λ), the foundational term formation rules and operational reduction axioms make zero reference to relative typestates (1995, 5–6). Translating this theory into physical hardware reveals that stratification exerts direct ontological consequences on the execution architecture. The monist-comb compiler extracts the Bellman-Ford distance bounds generated during CPU validation and burns them into the physical execution graph as Comb.t_inject topological friction markers. The GPU physics engine executes combinator collisions at hardware velocity without tracking dynamic typestate integers. It relies on these physical T-functor wrappers embedded in the static geometry to self-regulate recursion depths natively across the spatial architecture.

Traditional compilers exhaust memory bandwidth by copying substitution tables across function call stacks, embodying the physical manifestation of the von Neumann bottleneck. Maintaining named variables during non-well-founded set evaluation creates unresolvable topological bottlenecks. Processing self-referential membership relations such as $x \in x$ traps evaluation engines in recursive lookup loops. By translating verified directed acyclic graphs into nameless, point-free combinatory structures, the compiler converts semantic abstraction directly into localized spatial geometry.

3.2. A Tree Has No Name

To eliminate lexical overhead while preserving variable scope semantics, the compiler converts verified abstract syntax trees into nameless representations via De Bruijn indexing (1972, 381–392). De Bruijn indexing replaces alphanumeric variable tags with natural numbers, mapping the exact lexical depth between the variable occurrence and its governing λ-abstraction. During compilation, α-equivalent terms converge into identical physical data structures in memory. Function

evaluation drops reliance on string comparisons or associative environment queries. The engine resolves variable references through deterministic arithmetic offsets.

While standard de Bruijn indices measure depth relative to local λ-abstractions, high-performance proof engines adopt de Bruijn levels to stabilize term inspection across deep syntactic hierarchies. The compiler assigns each bound variable a fixed natural number corresponding to the exact nesting depth of its binding bracket relative to the root expression (Holmes and Alves-Foss 2020, 31). The outermost abstraction bracket binds the identifier ?1, an abstraction nested inside one bracket binds ?2, and an abstraction enclosed within $n-1$ brackets binds ?n. This global leveling preserves identical variable addresses across complex subterms, eliminating structural friction during visual pattern matching and graph rewriting.

Operating on nameless abstract syntax trees requires mechanical level-shifting during term substitution across divergent tree heights. Substituting a subterm T residing at level l_1 into a context at level l_2 within term U forces the compiler to adjust bound variable indices. This prevents structural corruption. Any bound variable ?n within T where $n > l_1$ undergoes direct arithmetic translation to $n - l_1 + l_2$ (Holmes and Alves-Foss 2020, 32). Holmes and Alves-Foss prove that this substitution operation becomes structurally impossible if any bound variable ?n appears within T satisfying the inequality $l_2 < n \leq l_1$. This rigid substitution boundary guarantees the compiler frontend enforces absolute scope safety.

These structural constraints execute algorithmically during foundational compiler operations like the BIND and UNEVAL tactics (Holmes and Alves-Foss 2020, 33). The BIND tactic shifts target subterms from level l to $l+1$, mechanically replaces instances of T with the new bound variable ?[l+1], encloses the expression within abstraction brackets, and verifies stratification compliance before generating an executable closure. The UNEVAL tactic performs reverse pattern matching to rewrite concrete values back into functional abstractions without evaluation overhead. Together, these tactics guarantee the frontend verifies binding limits prior to graph lowering.

```
Algorithm: Nameless_DeBruijn_Level_Shift
Input: Term T, Source Level l_1, Target Level l_2
Output: Shifted Term T' ready for context insertion

1. Match T:
2. Case BoundVar(?n):
 If n > l_1:
 Return BoundVar(?(n - l_1 + l_2))
 Else:
 Return BoundVar(?n)
3. Case Abstraction(Body):
 ShiftedBody := Nameless_DeBruijn_Level_Shift(Body, l_1, l_2)
 Return Abstraction(ShiftedBody)
4. Case Application(Op, Arg):
 ShiftedOp := Nameless_DeBruijn_Level_Shift(Op, l_1, l_2)
 ShiftedArg := Nameless_DeBruijn_Level_Shift(Arg, l_1, l_2)
 Return Application(ShiftedOp, ShiftedArg)
```

Combining de Bruijn representations with Quinean set theory requires the compiler to track lexical binding depth and relative typing integers simultaneously. A term $(\lambda x)(T)$ is structurally valid only when the bound variable x occurs within subterm T exclusively at relative type 0 (Holmes 1995, 3). The compiler computes relative types through recursive syntax traversal: function application $U(V)$ assigns relative type $n+1$ to operator U and relative type n to operand V, while pairing (U, V) preserves parallel type assignments. The compiler validates that every numerical index pointing to a binding abstraction satisfies uniform stratification criteria before generating execution graphs.

To support the physical monist-comb compilation pipeline, the nf-sketches verification environment mathematically proves the necessity of these theoretical boundaries within Lean 4. The nf-sketches architecture enforces scope safety dynamically, isolating variables into a fundamental Var structure (separating De Bruijn indices from string identifiers) and mapping them into a ScopedVar envelope that tags variables with their binder depth. The reduceCut diagnostic engine simulates structural level-shifts organically; when a simulated substitution violates the strict depth limits modeled by Nameless_DeBruijn_Level_Shift, the altered ScopedVar indices geometrically collide with Extensionality bounds to trigger a negative-weight cycle. By establishing this mathematical certification, nf-sketches proves that valid syntax naturally satisfies cut-free derivation complete-

ness. This allows the downstream monist-comb compiler to safely resolve dynamic bindings relative to its local environment stack and execute point-free interaction nets at bare-metal velocities.

3.3. Currying Favor with Combinators

While nameless indexing streamlines syntax traversal, GPU execution demands the eradication of abstraction boundaries. The monist-comb compiler translates nameless abstract syntax trees into synthetic combinatory logic via the primitive Comb inductive type. Isolating parameter dependencies through standard currying introduces structural violations in stratified λ-calculi (Holmes and Alves-Foss 2020, 9). In an uncurried function application $f(x, y)$, arguments x and y occupy identical relative typestates. Converting this structure into a curried application $f(x)(y)$ forces an immediate stratification violation because function application mechanically raises the relative typestate of the operator relative to its operand. Consequently, $f(x)$ must reside exactly one typestate higher than y, destroying the uniform typestate alignment required for multi-variable inputs (Holmes and Alves-Foss 2020, 9). To bypass this currying failure, the compiler processes parameter dependencies through uncurried binary Quine pairs or injects typestate wrappers before generating bare-metal interaction graphs. This enables the systematic deployment of primitive combinators without triggering typestate mismatches.

The compiler utilizes the constancy function K to discard extraneous arguments and introduce invariant values across execution paths. The fusion function S manages variable duplication, distributing application paths across sub-trees. This reduction serializes spatial topological matrices into flat memory words, eliminating syntax tree pointer allocation.

Relying exclusively on a minimal S and K basis triggers severe combinatorial explosion. Transforming deeply nested abstract syntax trees using only two combinators expands code size quadratically relative to the depth of bound variables. To enforce algorithmic stabilization, the compiler injects composition (B) and permutation (C) operators. If a target variable inhabits solely the right subtree N, the compiler routes evaluation through the composition operator B. If the variable resides solely in the left subtree M, the compiler utilizes the permutation operator C. Restricting the duplication operator S to expressions where the variable inhabits both branches maintains linear space complexity. It stabilizes intermediate term sizes and prevents memory exhaustion during physical graph reduction. This selective duplication blocks unconstrained expansion across non-linear abstractions, guaranteeing affine stability during evaluation.

To prevent memory explosion during combinator graph generation, the pipeline models its reduction rules on contraction-free sequent calculi (Negri and von Plato 2001, xii, 28–30). By enforcing shared contexts across two-premiss logical operations, the compiler ensures that the premisses of any reduction step remain uniquely determined by their conclusion. This structural determinism strips intermediate contraction nodes from the graph, keeping intermediate term sizes linear. To resolve self-referential cycles identified by Kosaraju's SCC algorithm, the engine utilizes the U combinator to project these unstratified loops into safe semantic configurations, converting raw paradoxes into bounded recursive evaluations. The compiler utilizes Buss's smash function $x \sharp y = 2^{|x| \cdot |y|}$ as a primitive sizing operator to bound sequence encodings and de Bruijn index shifts during hardware evaluation.

3.4. T-Weaking the Tension

Compiling untyped combinatory logic into stratified Quinean set theories generates topological tension. In cartesian closed categories, function spaces operate smoothly via standard evaluation arrows. In stratified set theories, standard cartesian closure fails because the classical evaluation morphism $ev_{A,B} : A \times (A \Rightarrow B) \to B$ defined by $ev(\langle x, f \rangle) = f(x)$ is unstratifiable (Forster, Lewicki, and Vidrine 2019, 8–9). The function object f resides exactly one type level higher than domain elements x, preventing the formation of a uniform evaluation mapping across unraised inputs.

To resolve this tension, the monist-comb compiler implements the categorical remedy governing the category of NF sets (N). While standard cartesian closure fails, N achieves pseudo-cartesian closure through relative adjunctions (Forster, Lewicki, and Vidrine 2019, 8–10). For any sets A and B, the compiler replaces the unstratifiable evaluation arrow with the $(TA \times -)$-co-universal pseudo-evaluation morphism $ev'_{A,B} : TA \times (A \Rightarrow B) \to TB$ defined by the action $\langle \{x\}, f \rangle \mapsto \{f(x)\}$. This mapping establishes the primary relative adjunction $(TA \times -) \to_T (A \Rightarrow -)$ and works alongside its secondary relative adjoint $(A \times -) \to_T (A \Rightarrow -)$. When compiling dependent products across localized slice categories N/A, the compiler operationalizes local pseudo-cartesian closure by treating the pullback functor f^* as a relative left adjoint to $\widetilde{\Pi}_f$ (Forster, Lewicki, and Vidrine 2019, 11–13). Injecting Comb.t_inject nodes instantiates these relative adjunctions in hardware memory, allowing uncurried combinator collisions to evaluate safely across bare-metal VRAM scratchpads. The compiler discards functional extensionality to preserve the distinct geometries governing topological recursion costs.

In New Foundations, this wrapper manifests as Ernst Specker's T-functor. The T-functor acts categorically as the unit of a relative KZ-pseudomonad. When the monist-comb compiler injects Comb.t_inject nodes at topological friction points, it executes a structural analogue to the Yoneda embedding. This embeds localized set-theoretic terms into a functorial network of relationships, bridging the stratification gap and permitting uncurried combinator collisions to execute safely across unraised input boundaries within VRAM scratchpads. Injecting Comb.t_inject wrappers (T-functors) constructs valid pseudo-evaluation morphisms across bare-metal graph nodes.

The compiler's T-Operator Compilation Pipeline operationalizes Thomas Forster's (2025b) algorithmic "T-weaking" to inject structural wrappers and enforce stratification boundaries natively. By feeding live integer variables derived from the Bellman-Ford witness directly into the algorithmicTWeaking function, the pipeline synthesizes T-operator injections on-the-fly. The algorithm traverses abstract syntax trees top-down, assigning integers to sub-formulas based on depth rules. When an expression violates weak stratification boundaries—such as evaluating a self-application $x(x)$—the compilation engine corrects the structural imbalance by dynamically injecting Comb.t_inject wrappers into the graph.

```
Algorithm: algorithmicTWeaking
Input: UntypedComb AST Node N, Target Level L
Output: Stratified Node N' with injected t_inject markers

1. Compute current relative level l := getWeightFromWitness(N)
2. If l > L:
 LevelDifference := l - L
 WrappedNode := N
 Loop LevelDifference times:
 WrappedNode := Comb.t_inject(WrappedNode)
 Return WrappedNode
3. Else If N is Application(Op, Arg):
 OpLevel := getWeightFromWitness(Op)
 ArgLevel := getWeightFromWitness(Arg)
 If OpLevel != ArgLevel + 1:
 ShiftedArg := algorithmicTWeaking(Arg, OpLevel - 1)
 Return Application(Op, ShiftedArg)
4. Return N
```

When compiling an unstratified implication across topological boundaries, the compiler injects these Comb.t_inject nodes directly into the friction points. This mathematical shim acts as a

geometric stabilizer during lazy reduction, shifting relative typestates to yield stabilized, executable expressions such as $T(A \to B) \to (T A \to T B)$. This guarantees bare-metal combinators remain type-safe through self-regulation without requiring a rigid runtime type-checker.

Holmes establishes that algorithmic stratification checking shares its underlying execution engine directly with term matching functions (Holmes 1995, 10–11). Both software routines traverse abstract syntax trees to enforce structural parallelism and integer assignment consistency across subterms. To prevent matching failures during complex graph elaborations, the compiler implements scin (strongly Cantorian input) and scout (strongly Cantorian output) declaration flags (Holmes and Alves-Foss 2020, 39–40). This instructs compilation pipelines to adjust relative types across operator input branches. Tagging logical operators and quantifiers with scout signals that their output resides within a Strongly Cantorian domain. This domain tagging authorizes the term matching engine to raise or lower relative typestates dynamically during unification without generating stratification faults, ensuring the smooth compilation of first-order logical sequents into flat combinator matrices.

This mechanical stabilization extends directly to the ingestion of domain-specific mathematical axioms. Conventional compiler theory assumes that introducing axioms into formal logic destroys cut eliminability, forcing execution engines to allocate unbounded memory for intermediate cut formulas. However, structural proof theory proves that free-variable mathematical axioms can be converted into nonlogical rules of inference that preserve cut-free normalization (Negri and von Plato 2001, 126–128). The Monist Engine injects domain axioms directly into the reduction graph as cut-free combinator transitions.

3.5. The Cost of Doing Business

Evaluating the compilation of non-well-founded sets through the lens of mechanics reveals that variable annihilation operates as a thermodynamic necessity. Rolf Landauer established that computation executes through physical degrees of freedom, proving that logical irreversibility mandates physical energy dissipation. Traditional compilers execute irreversible logical operations: overwriting variables, discarding stack frames, and executing global garbage collection sweeps destroy logical information, transforming potential computational work into thermal waste.

By compiling formulas into point-free interaction nets via untyped combinatory logic, the monist-comb architecture transitions from classical irreversible manipulation toward localized, conservation-oriented spatial geometry. Traditional variable substitution acts as a dissipative

process. Replacing named parameters across complex syntax trees demands copying memory structures and destroying prior pointer states. Combinatory graph reduction via S, K, and I operators eliminates these global environment overwrites.

Crucially, this framework introduces the concept of **Topological Recursion Cost**. The Monist pipeline deterministically tracks the exact number of graph-rewrite collisions required to normalize a combinator term into a stable fixpoint. This transforms logical execution from a purely informational process into a calculable geometric workload. The reduction of interaction nets proceeds through localized graph rewriting where combinator collisions deterministically route data flows along explicit topological pathways.

Annihilating variables into untyped logic allows the engine to bypass the physical memory illusions inherent in classical continuum mathematics. Classical static typing and unbounded hierarchical recursion implicitly assume an infinite memory substrate capable of storing arbitrary environment histories. The geometry solver flattens these assumptions into concrete, integer-weighted Directed Acyclic Graphs before compilation begins. Transforming abstract formulas into nameless de Bruijn offsets and primitive combinators compresses logical phase space down to its absolute physical minimum. The resulting interaction net represents a self-renewing structure ready for lock-free parallel execution across GPU threads, completely unburdened by the thermodynamic and computational friction of human-readable symbols.

---

4. The Physics Engine: Bare-Metal GPU Interaction Nets

Transitioning from semantic compilation into hardware execution demands abandoning classical computing paradigms. Classical systems rely on the von Neumann bottleneck: the single-word transmission channel connecting the CPU and memory store. This conduit forces an inefficient word-at-a-time traffic pattern. The primary traffic pumping through this transmission tube consists of data names, operational instructions, and computed addresses, prioritizing metadata over actual target data (Backus 1978, 615). Before a processor fetches or modifies a word, it must transmit the target address through the conduit, requiring prior instruction fetches and arithmetic computations to generate that address. This forces programmers to orchestrate vast administrative traffic concerning data location, consuming physical memory bandwidth on associative lookups. The architecture limits operations to generating storage addresses and moving individual data words, dividing computation into state-changing statements and evaluating expressions. Upholding this hi-

erarchy guarantees computational friction and memory exhaustion when processing non-well-founded sets. Eliminating symbolic variables and address calculations prevents GPU computing warps from wasting memory bandwidth on lexical overhead. Projecting logic directly into physical spatial coordinates resolves this structural bottleneck.

The Monist Engine adopts a functional style governed strictly by reduction semantics. Expressions reduce successively until reaching a final normal form (Backus 1978, 614–616). To accomplish history-sensitive computing without coupling semantics to word-at-a-time state transitions, the execution pipeline operates as an Applicative State Transition (AST) system (Backus 1978, 634–636). In an AST model, state transitions occur once per major computation via coarse-grained function application, discarding the baroque communication protocols required by conventional assignment statements. AST systems implement functional naming mechanisms where names operate as first-class functions composed via standard combining forms (Backus 1978, 638). This bypasses reliance on invisible hardware states and complex storage protocols. We map logic directly onto physical hardware environments, eliminating the von Neumann store and address registers to permit simultaneous bottom-up evaluation of independent functional applications. The architecture executes via bare-metal interaction nets across VRAM scratchpads in discrete state transitions without relying on hierarchical store protocols, translating set theory into tangible topological matrices.

4.1. Topological Graph Rewriting and Spatial Abstraction

Hardware execution shifts from static memory analysis to dynamic structural evolution. In topological graph rewriting, operations modify incidence relationships directly without triggering global reallocations of the underlying graph space. Contracting an edge $e = x\,y$ into a single representative vertex v_e mechanically converts parallel edges into localized loops, altering the graph minor at each step. The final evaluated output mathematically represents a topological minor of the initial compiled logic.

The monist-comb Interaction Net backend actualizes this topology within Video Random Access Memory (VRAM) arrays. High-performance systems languages enforce strict tree-structured ownership models and borrow-checking invariants that clash fundamentally with flat, self-referential memory graphs. Representing cyclic membership ($x \in x$) using reference-counted hierarchies introduces runtime overhead and ownership deadlocks. The Monist compiler bypasses tree-enforced memory validation by utilizing flat Arena allocators as first-class primitives. Unstratified

self-application ($x(x)$) forces physical memory addresses to simultaneously execute as scalar data values and higher-order execution kernels. This condition resolves natively within lock-free memory matrices.

The architecture compresses all combinator nodes into a localized contiguous VRAM arena utilizing 32-bit tagged pointers. The engine allocates cyclic structures inside unmanaged, immutable memory blocks without reference-counting overhead through lazy, call-by-need graph reduction. Lloyd Allison proves that infinite or circular data structures can operate in finite memory space without recursive call-stack overhead when evaluated under call-by-need graph reduction (2024, 1–2). Circular graph evaluation strictly demands a single-assignment memory model where values remain immutable once written. To verify this static allocation without runtime garbage collection pauses, the architecture integrates Deterministic Memory Management (DMM) coeffect frameworks. Integrating this coeffect taxonomy into the compiler's intermediate representation ensures that interaction net cells remain strictly arena-scoped. This guarantees at design time that cyclic structures never escape their allocated scratchpad buffers during hardware execution.

Every node possesses exactly one main port dedicated to active computational reduction (Taelin 2024, 3). Because the principal main port is represented implicitly within the memory layout, binary nodes require storage capacity exclusively for their two auxiliary ports (Taelin 2024, 5). We map each port into a compressed 32-bit word combining a 3-bit tag indicating the destination node type and a 29-bit value payload. This 29-bit addressable payload space constrains the global arena (GNet) to a maximum of 2^{29} active nodes, establishing a hard 4 GB memory boundary for the node buffer (APair) and a 2 GB boundary for the atomic substitution map (APort).

By allocating matrices within unmanaged memory arenas, the engine constructs raw, cyclic pointer graphs where variables freely reference their own memory addresses. This physical execution model maps directly to Holmes's semantic programming machine, where computation occurs across an infinite set of physical memory addresses X, and abstract programs map to machine states $X \to X^X$ (1995, 6–8). Under this architecture, unstratified self-referential terms like $\Delta = (\lambda x)(x(x))$ violate fundamental data type security. Executing Δ forces the machine to process physical address x simultaneously as raw storage location and as executable code instruction. A memory defragmentation sweep or address permutation alters the storage coordinate of x without changing its functional extension, permanently corrupting the unstratified execution output. Enforcing structural stratification provides memory security by preventing hardware operations from exploiting raw storage addresses across divergent computational levels. This rigid geometric con-

straint allows a complete binary node to occupy exactly one 64-bit memory word, discarding global garbage collection pauses. This extreme spatial compression enables simultaneous reductions across parallel compute units. Execution threads bypass CPU mutexes, utilizing strict Atomic Compare-And-Swap (CAS) spin-loops within WGSL compute shaders to guarantee safe parallel writes.

4.2. When Nodes Collide

Interaction combinatory logic discards the concept of a global clock. Time operates relativistically, allowing computations to distribute actively across multiple spatial locations simultaneously (Lafont 1997, 70). Node collisions constitute the sole mechanism of computational progress. Collisions occur exclusively when two agents establish a direct connection via their principal ports, forming a reducible expression, or redex (Taelin 2024, 3, 6). Exactly one deterministic interaction rule dictates each of the 64 possible pairwise collisions derived from the eight distinct node types (Taelin 2024, 8). Generating positive and negative directional operators computes exact path weights across binary redexes, providing a formal proof of execution invariance and time-reversibility during parallel hardware evaluation.

When identical binary types collide, the hardware triggers the Annihilate rule, destroying both parent nodes and establishing two new redexes connecting their internal auxiliary wires (Taelin 2024, 7). Specifically, the physical execution cross-wires auxiliary port 1 of the first cell to port 2 of the second cell, and auxiliary port 2 of the first cell to port 1 of the second cell. Distinct binary types trigger the Commute rule, commanding the hardware to clone both structures across each other's auxiliary ports to reorganize the local topology dynamically.[7]

The bare-metal runtime processes these specific physical collisions through sequential hardware commands: allocating resource capacities, loading node structures from global memory into processor registers, initializing fresh substitution variables, storing generated nodes back into memory buffers, and atomically linking outgoing wires (Taelin 2024, 17). To demonstrate the concrete programming reality of these spatial rewrites, we observe the execution kernel's memory manipulation during a standard commutation collision:

[7] Commutation preserves mathematical intent by duplicating and routing logic symmetrically. This allows parallel sub-evaluations to proceed deeper within the interaction net without central synchronization.

```
pub fn interact_comm(&mut self, net: &GNet, a: Port, b: Port) -> bool
{
    // Allocates needed resources.
    if !self.get_resources(net, 4, 4, 4) {
        return false;
    }
    // Loads nodes from global memory.
    let a_ = net.node_take(a.get_val() as usize);
    let a1 = a_.0;
    let a2 = a_.1;
    let b_ = net.node_take(b.get_val() as usize);
    let b1 = b_.0;
    let b2 = b_.1;
    // Stores new vars.
    net.vars_create(self.v0, NONE);
    net.vars_create(self.v1, NONE);
    net.vars_create(self.v2, NONE);
    net.vars_create(self.v3, NONE);
    // Stores new nodes.
    net.node_create(self.n0, pair(port(VAR, self.v0), port(VAR,
self.v1)));
    net.node_create(self.n1, pair(port(VAR, self.v2), port(VAR,
self.v3)));
    net.node_create(self.n2, pair(port(VAR, self.v0), port(VAR,
self.v2)));
    net.node_create(self.n3, pair(port(VAR, self.v1), port(VAR,
self.v3)));
    // Links.
    self.link_pair(net, pair(port(b.get_tag(), self.n0), a1));
    self.link_pair(net, pair(port(b.get_tag(), self.n1), a2));
    self.link_pair(net, pair(port(a.get_tag(), self.n2), b1));
    self.link_pair(net, pair(port(a.get_tag(), self.n3), b2));
    return true;
}
```

4.3. Look Ma, No Locks

This geometric execution proceeds without external operating system locks. The architecture relies on Atomic Compare-And-Swap (CAS) spin-loops directly accessing VRAM addresses without central synchronization. To mathematically secure the physical feasibility of this lock-free routing across parallel compute warps, the architecture invokes Nash-Williams's arboricity and tree packing theorems (Diestel 2025, 50–51). To prevent memory bus contention during parallel redex evaluations, the monist-comb compiler calculates the arboricity of the localized interaction net.

Theorem 2.4.3 establishes that the edge set of a multigraph $G=(V,E)$ can be covered by at most k forests if and only if every non-empty subset $U \subseteq V$ induces at most $k(|U|-1)$ edges. By verifying that localized VRAM scratchpads maintain bounded arboricity, the compiler proves that reduction redexes never exceed the linear memory access limits of the underlying hardware threads. Furthermore, Corollary 2.4.2 proves that every $2k$-edge-connected multigraph contains k edge-disjoint spanning trees. Guaranteeing edge-disjoint tree structures within saturated subgraphs allows parallel GPU compute warps to traverse and reduce distinct variable paths simultaneously without encountering CAS spin-loop deadlocks.

Variables occur precisely twice within the topological net (Taelin 2024, 9). Parallel reductions involving shared variable wires must manage deferred substitutions securely to prevent data corruption. The engine routes simultaneous substitutions through a global, atomic substitution map constructed in a flat memory buffer. To navigate these interaction nets without central synchronization, GPU threads operate as reversible two-stack execution machines (Lafont 1997, 91–93). Threads carry a pair of γ- and δ-stacks, pushing directional tokens when entering auxiliary ports and popping tokens when colliding at principal ports. This traversal model ensures asynchronous warps execute graph transformations deterministically across localized scratchpads.

The lock-free linking algorithm probes the substitution buffer indexed by the active variable (Taelin 2024, 9–10). If the atomic query encounters an empty slot, the wire registers its counterpart and suspends local action until the secondary variable arrives in memory. Upon arrival, the parallel thread retrieves the stored port, deletes the map coordinate, and permanently links the topological endpoints. The reference mechanism models this exact physical linking progression using autonomous spin-loops:

```
# Attempts to link A and B.
def link(subst: Dict[str, Node], A: Node, B: Node):
    while True:
        # If A is not a VAR: swap A and B, and continue.
        if type(A) != VAR:
            swap(A, B)
        # If A is not a VAR: both are non-vars. Create a new redex.
        if type(A) != VAR:
            push_redex(A, B)
        # Here, A is a VAR. Create a A: B entry in the map.
        got: Port = subst.set_atomic(A, B)
```

```
        # If there was no A entry, stop.
        if got is None:
            break
        # Otherwise, delete A and link got to B.
        del subst[A]
        A = got
```

Workload distribution operates identically to the execution pathways. The system utilizes 64-bit task bags to store active redexes (Taelin 2024, 19). Starving hardware threads execute a single atomic_exchange command against neighboring thread queues to steal older redexes dynamically. This secures maximum computational density across hardware limits with minimal contention. The structural topology of the underlying network guarantees strong confluence. If a localized net reduces into distinct intermediate states, the interaction rules guarantee those diverging branches will eventually reduce to a common final normal form (Lafont 1997, 73). This absolute determinism allows bare-metal engines to achieve extreme concurrency, aggressively modifying memory space regardless of parallel scheduling order.

4.4. VRAM Autonomous Reclamation

Erasing topological information constitutes a dissipative event. Traditional tracing garbage collectors generate unacceptable memory latency and interrupt asynchronous hardware warps. The architecture resolves this by reclaiming memory autonomously through the linear mechanics of interaction combinators, removing global tracing sweeps. This provides physical justification for why asynchronous, lock-free VRAM spin-loops avoid thermal throttling. The system frees memory words the exact moment sub-nets become unreachable. Two specific topological interaction rules control this dissipation: Void and Erase (Taelin 2024, 7, 21–22).

The Void rule executes upon the direct connection of two nullary nodes. The collision consumes both structures instantly, clearing their memory footprint without residual traces. The Erase rule triggers when an explicit erasure node connects with a binary node. The interaction dismantles the parent binary structure and propagates fresh erasure nodes down both associated auxiliary channels. When identical binary cells collide under the Annihilate rule, the hardware cross-wires their auxiliary wires and uncopies the parent nodes. This uncopying scales dissipation linearly with execution velocity rather than generating fixed thermal waste per node collision (Landauer 1991, 24–25). Any bounded tree configuration interacting with an eraser node completely reduces into

independent eraser cells, cascading deletion autonomously across dead subgraphs (Lafont 1997, 85).[8]

Bare-metal GPU execution isolates operations to block-local scratchpads mapped directly within hardware L1 shared memory buffers (Taelin 2024, 21). A 96KB reserve secures exactly 8192 local nodes, shielding active computational loops from global VRAM latency limits. When a physical interaction forces a localized block thread to establish a connection with an external global variable, standard graph rewriting risks invalidating pointers across multiprocessor boundaries. We secure this boundary by extending the linking process with a specialized GPU LEAK sub-interaction. The operation synthesizes a global view of the local node within primary VRAM using paired placeholder variables. This preserves memory safety while allowing localized block threads to maintain uninterrupted execution.

4.5. Agentic Reflection and Cycle Sweeping

Non-well-founded sets naturally generate topological anomalies during execution. Interaction nets permit the construction of vicious circles, mathematically defined as closed principal paths (Lafont 1997, 78–79). Lacking a triggering initial combinator collision, these closed loops form permanent geometric deadlocks within the matrix. The system restricts classical mathematical reductions to specific, computationally verified domains, preventing these topological deadlocks from polluting the main graph. By identifying when local graph execution operates completely inside a Strongly Connected Component where $T(m) = m$, the evaluation structure dynamically wraps these retractions in an SC_CUT isolation boundary. Before returning this data to the volatile exterior, it mechanically triggers Knaster-Tarski least fixpoint calculations ($\mathrm{lfp}(F)$) via the verifySCStability function. These "islands of classicality" allow choice functions and constant-time inductive logic to bypass global cycle detectors without generating unverified paradoxes.

The capstone of this architecture is the native combinatorial execution of the Stratified Yoneda Lemma:

$$\mathrm{Nat}(C(U,-),F) \cong T(F(U)).$$

The system directly queries the set of natural transformations from the hom-functor of the Universal Set (V) to a covariant presheaf F. This operational litmus test confirms the structural ro-

[8] Autonomous erasure acts as a physical thermodynamic sink within the graph. It absorbs isolated logic states without demanding secondary CPU scheduling intervention or halting the primary evaluation path.

bustness of the compiler stack. Rather than shattering the unstratified graph into a fatal loop across , the engine isolates the evaluation inside an SC_CUT boundary, forces the required T-relative adjunctions, and resolves the query in polynomial time. The engine structurally evaluates the universe containing itself and returns the isomorphic bounds $T(F(V))$.

The pedagogical capability of the engine surfaces when users deploy an unstratified Y combinator to orchestrate a coreclusive, self-referential agent loop. The physics engine processes this unstratified interaction natively. When the compiler processes a coreclusive agent loop or an unstratified membership structure ($x \in x$), the execution runtime initially binds the identifier ds to an unevaluated functional recipe node (Allison 2024, 1–2). Upon execution of a compute shader forcing the recipe evaluation, the hardware graph reduction mechanism computes the target value and atomically overwrites the recipe node in memory with the resulting structure. For expressions containing recursive references to themselves, this overwriting embeds a direct physical pointer back to the root node of the data structure. This guarantees that structural paths traverse exclusively through data constructors back to the root node, forming an explicit memory graph. This single-assignment recipe-overwriting mechanism safely isolates paradoxical memory rings before standard combinator resolution occurs. By stripping away typestates and hierarchical identifiers, these paradoxes manifest physically within VRAM as disconnected floating memory rings. The system bypasses stack overflows and logic panics. Dedicated background hardware warps traverse the active memory arenas utilizing local pointer reversal.

This hardware evaluation of self-referential agent loops operationalizes the model-theoretic distinction between term models and co-term models in base theories like NF_0 and TZT_0 (Forster 2025a, 67–68). While discrete inductive AST evaluations correspond to standard term models omitting the 1-type Σ^i, the GPU's execution of infinite coreclusive reduction streams instantiates co-term models generated by omitting the 1-type Σ^c. Because the inclusion embedding from the inductive term model into the coinductive co-term model is elementary for weakly stratified formulas (Forster 2025, 68–69), coreclusive recipe-overwriting in VRAM preserves absolute first-order logical equivalence. The hardware physics engine executes infinite self-referential streams safely within flat memory arenas without violating the semantic invariants established during compiler verification.

Evaluating logic on hardware consumes quantifiable degrees of freedom. Algorithmic complexity, memory consumption, and thermodynamic dissipation represent material realities. Enforcing extensional equivalence at the hardware level would collapse computationally distinct interac-

tion nets into identical memory states, destroying the engine's ability to measure localized structural costs. Because the VRAM arena operates without rigid hierarchical universes or extensional identity constraints, physical hardware execution remains purely intensional.

The WGSL compute shaders execute an $O(1)$ cycle garbage collection sweep within reduce.wgsl. While the CPU-bound monist-core geometry layer utilizes Bellman-Ford and Minimum Cycle Mean algorithms to intercept paradoxical negative-weight loops prior to compilation, the GPU runtime encounters structurally valid but coreclusive sub-graphs during execution. In runtime execution arenas processing saturated, non-well-founded datasets stabilized via T-functors or unstratified Y-combinator loops, the engine generates dense, isolated interaction nets that disconnect from the primary computational root.

To physically reclaim these "floating" memory rings spawned by unstratified self-reference, the cycle garbage collection kernel (reduce.wgsl) maps flat VRAM arenas directly to the edge space $E(G)$ and vertex space $V(G)$ over the two-element field $F_2=\{0,1\}$ (Diestel 2025, 23). Diestel's Proposition 1.9.1 proves that an edge set belongs to the algebraic cycle space $C(G)$ if and only if every vertex in the induced subgraph maintains an even degree, physically manifesting as disjoint unions of cycles (Diestel 2025, 24). Furthermore, Proposition 1.9.6 establishes that $C(G)$ forms the exact kernel of the incidence matrix B over F_2 (Diestel 2025, 27). During background garbage collection sweeps, the shader evaluates this matrix product against local pointer allocations in $O(1)$ hardware steps. Because Tutte's theorem proves that the cycle space of a 3-connected topology is generated exclusively by non-separating induced cycles (Diestel 2025, 69), the hardware kernel instantly detects coreclusive floating rings by isolating kernel vectors in $C(G)$ that lack external cutset dependencies.

Evaluating infinite coreclusive reduction chains via permutation models proves that self-referential sequences force parity collisions across singletons (Forster 2025a, 81). This theoretical parity collision manifests physically within the GPU physics engine, allowing the background warp to pinpoint these isolated topological loops and immediately tag them with TAG_ERA memory structures. The anomalous, infinitely recurring logic is structurally neutralized and reallocated back into the lock-free allocation pool without interrupting parallel hardware warps. This demonstrates the definitive triumph of intensional operational semantics over classical extensional set theory and proves the system capable of hosting unstratified sets as continuous geometric facts.

5. Going Holographic

Executing discrete interaction combinators across binary graph topologies provides a verifiable, acyclic substrate for scalar logic and structural proof reduction. However, scaling this execution engine to high-energy relational datasets requires expanding beyond point-to-point connections. To avoid routing bottlenecks, the architecture embeds logic into continuous Vector Symbolic Architectures (VSA), compressing sprawling relational queries into continuous spatial structures.

5.1. Beyond Binary Topology

While binary Program Semantic Graphs represent point-to-point data dependencies, they trigger structural latency when processing irreducible multi-way operations across sprawling datasets. To capture these hardware constraints natively, the Monist Engine's Holographic Co-processor API promotes discrete relations into continuous, high-dimensional tensors. The system evaluates massive discrete search bounds programmatically, generating quasi-orthogonal vectors via HDCVector::random_basis() and folding thousands of data points into a single tensor via element-wise superposition. Treating massive datasets as continuous wave functions maps multi-way relational logic directly onto spatial execution grids. This routing bypasses the memory exhaustion inherent in sequential von Neumann pipelines and discrete hierarchical searches.

5.2. Clifford Algebra to the Rescue

Evaluating multi-way relational transformations across spatial architectures demands an algebraic substrate capable of processing geometric operations without runtime interpretation overhead (Haynes 2026a, 1–5). The Program Hypergraph embeds the graded structure of Clifford algebras into the compiler's semantic type graph as compile-time dimension axes governed by a Dimensional Type System (Haynes 2026a, 3; Haynes 2026b, 9). Clifford algebras unify scalar, vector, and higher-order exterior structures under the geometric product generated by a real vector space and a quadratic form (Haynes 2026b, 7). Restricted to the outer product, multivector grade forms a finitely generated abelian group isomorphic to integer addition governed by standard integer arithmetic. This structural isomorphism satisfies the operational axioms of a static dimension axis. It empowers the compiler to evaluate grade consistency across complex relational expressions in constant time ($O(1)$) per operation.

The draft models geometric algebra embeddings through a generalization known as the Program Hypergraph (PHG). This promotes binary edges $e = (u, v)$ to k-ary hyperedges $f = (S_f, t_f, \lambda_f)$ (Haynes 2026b, 4). Decomposing a three-way geometric join into sequential binary operations destroys algebraic provenance, generates semantically empty intermediate nodes, and accumulates floating-point numerical residue across successive evaluations (Haynes 2026b, 2, 8–10). Encoding a k-simplex join of $k+1$ vertices into a plane or volume as a single, indivisible hyperedge preserves exact algebraic identity across the relational transformation. The compiler's unification engine evaluates multivector grade consistency prior to code generation, identifying structural zeros directly from source annotations. In three-dimensional Projective Geometric Algebra, static grade inference determines that approximately ninety-five percent of Cayley table entries evaluate to structural zero for standard grade combinations. Pruning these null branches during graph elaboration enables the compiler to emit sparse machine code containing only non-zero arithmetic instructions. This structural pruning achieves a twenty-fold improvement in computational throughput directly from hyperedge grade annotations without demanding runtime branching or dynamic memory allocation. All dimensional and grade annotations operate as compilation metadata through progressive lowering stages, dissolving before physical machine code emission occurs (Haynes 2026a, 5–6).

5.3. Don't Compress the Phase Space

Scaling graph evaluation to continuous phase spaces demands reconciling logical graph transformations with physical thermodynamic limits (Landauer 1991, 23–24). Standard digital logic gates compress phase space during execution by merging distinct input states into single output channels, discarding operational history and dissipating thermal energy. Decoupling physical computation from irreversible energy dissipation requires state transformations where each sequential step results from a bijective one-to-one mapping, preserving trajectory identification across every operation. Reversible computing architectures isolate computational pathways inside high potential barriers to prevent lateral tunneling between parallel execution tracks, maintaining distinct physical trajectories throughout extended operational sequences (Landauer 1991, 27). Avoiding phase-space compression across relational transformations requires embedding multi-input logical gates inside higher-dimensional bijective manifolds governed by spin and projection operators.[9]

[9] Erasing a single bit of information in a classical digital system at thermal equilibrium requires an unavoidable energy dissipation of $k\,T \ln 2$ into the environment, where k is Boltzmann's constant and T is absolute temperature (Landauer 1991, 24). Maintaining computation inside reversible interaction fab-

To process massive spatial arrays without phase-space compression or thermal exhaustion, the Holographic Co-processor embeds discrete graph structures into a ten-thousand-dimensional continuous wave function using HDCVector primitives. High-energy datasets map into hyperdimensional vector spaces by encoding categorical records as orthogonal multivectors along Clifford algebra dimension axes (Haynes 2026a, 3; Haynes 2026b, 7). The co-processor bundles millions of relational events into a single composite phase space via element-wise superposition. This compresses dense logical configurations while preserving the structural identities of all constituent inputs. Navigating this superposed volume relies on Codebook associative memory structures rather than discrete memory pointers, empowering custom WGSL compute shaders to evaluate complex multiway relational predicates across the continuous phase space.

5.4. Canceling the Noise

To observe the Holographic Co-processor executing physical continuous math, a developer feeds a dense, distributed swarm dataset into the FormulaArena pipeline. Because the engine processes a closed, saturated universe governed by Syntactic Monism, it natively supports operations based on the absolute complement ($\{x \mid x \notin A\}$)—a construct forbidden in standard ZFC. Bypassing $O(N)$ list sweeps, the compiler constructs "exclusion-first" logic gate wrappers. It maps target identities into the complement space and superposes the distributed swarm graph into a ten-thousand-dimensional continuous wave function via HDCVector::superpose() accumulation.

Retrieving the targeted anomaly executes through physical wave mechanics and Successive Interference Cancellation (SIC) via the holographic_exclusion_query() pipeline. The Interaction Net triggers a localized topological sweep that filters out non-matching paths natively in $O(1)$ time. Unselected background telemetry structures and uncalibrated parameter variations act as massive amorphous sets of controlled junk. By injecting phase-inverted vectors, these amorphous background structures superimpose out of phase, canceling out physically through destructive interference within the continuous computing substrate.

This destructive interference isolates anomalies before snapping the residual wave function back into discrete ScopedVar entries for the monist-core geometry solver to evaluate. Passing this residual data back through the established $O(V+E)$ DAG topological sort allows the pipeline to isolate unresolvable contradictions—specifically, negative-weight cycles—across the distributed network in linear time. The developer receives the extracted non-well-founded cycle as an exact, verifi-

---

rics bypasses this erasure limit by preserving logical history across bidirectional combinator ports.

able Directed Acyclic Graph. This final interaction completes the operational synthesis of continuous physical mechanics and Quine's New Foundations set theory.

References

Allison, Lloyd. 2024. “Circular Programs and Self-Referential Structures.” *arXiv*.

Backus, John. 1978. “Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs.” *Communications of the ACM* 21, no. 8: 613–41.

Buss, Samuel R. 1985. “Bounded Arithmetic.” PhD diss., Princeton University.

Crabbé, Marcel. 1991. “Stratification and Cut-Elimination.” *The Journal of Symbolic Logic* 56, no. 1: 213–26.

Crabbé, Marcel. 1992. “On NFU.” *Notre Dame Journal of Formal Logic* 33, no. 1: 112–19.

Crabbé, Marcel. 1994. “The Hauptsatz for Stratified Comprehension: A Semantic Proof.” *Mathematical Logic Quarterly* 40, no. 4: 481–87.

de Bruijn, Nicolaas Govert. 1972. “Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem.” *Indagationes Mathematicae (Proceedings)* 75, no. 5: 381–92.

Diestel, Reinhard. 2025. *Graph Theory*. 6th ed. Graduate Texts in Mathematics. Berlin: Springer.

Forster, Thomas. 2025a. “Scrapbook on Set Theory with a Universal Set.” Available at: https://www.dpmms.cam.ac.uk/~tef10/NFnotesredux.pdf

Forster, Thomas. 2025b. *T-weaking Typed Λ-terms: a Typing Scheme for Λ-Calculus that Respects the Stratification Constraints for Set Existence… Or Curry-Howard for NF*. Cambridge: University of Cambridge Department of Pure Mathematics and Mathematical Statistics.

Forster, Thomas, Pawel Lewicki, and Joshua Vidrine. 2019. “Category Theory with Stratified Set Theory.” Cambridge: University of Cambridge Department of Pure Mathematics and Mathematical Statistics.

Hailperin, Theodore. 1944. “A Set of Axioms for Logic.” *The Journal of Symbolic Logic* 9, no. 1: 1–19.

Haynes, Houston. 2026a. “Dimensional Type Systems and Deterministic Memory Management: Design-Time Semantic Preservation in Native Compilation.” *arXiv*.

Haynes, Houston. 2026b. “The Program Hypergraph: Multi-Way Relational Structure for Geometric Algebra, Spatial Compute, and Physics-Aware Compilation.” *arXiv*.

Holmes, M. Randall. 1995. “Untyped λ-Calculus with Relative Typing.” Technical report, Boise State University Department of Mathematics.

Holmes, M. Randall, and Jim Alves-Foss. 2020. *The Watson Theorem Prover*. Boise: Boise State University Department of Mathematics.

Lafont, Yves. 1997. “Interaction Combinators.” *Information and Computation* 137, no. 1: 69–101.

Landauer, Rolf. 1991. “Information Is Physical.” *Physics Today* 44, no. 5: 23–29.

McLarty, Colin. 1992. “Failure of Cartesian Closedness in NF.” *The Journal of Symbolic Logic* 57, no. 2: 555–56.

Negri, Sara, and Jan von Plato. 2001. *Structural Proof Theory*. Cambridge: Cambridge University Press.

Quine, W. V. 1937. "New Foundations for Mathematical Logic." *The American Mathematical Monthly* 44, no. 2: 70–80.

Ryan-Smith, Calliope. 2025. "Stratified Formulas Are Not Context-Free." *Formal Logic* 66(3): 301–311.

Schönfinkel, Moses. 1924. "Über die Bausteine der mathematischen Logik." *Mathematische Annalen* 92, no. 3–4: 305–16.

Taelin, Victor. 2024. "HVM2: A Parallel Evaluator for Interaction Combinators." *Github*.

Tarjan, Robert. 1972. "Depth-First Search and Linear Graph Algorithms." *SIAM Journal on Computing* 1, no. 2: 146–60.

Wilshaw, C. 2025. "New Foundations Is Consistent." *Github*.