The Axiom Profile of Computation

Larsen James Close

2026

Abstract

We report the axiom profiles of computability theory's central theorems when formalized from the equational theory of monoidal closed categories in Lean 4. Starting from a common categorical formalization, we find that core diagonal and fixed-point theorems — the Y combinator, Kleene's recursion theorem, halting undecidability, both Gödel incompleteness theorems, and Myhill's isomorphism theorem — are constructive (zero axioms); Rice's theorem sits exactly at a Markov boundary ($\neg\neg\exists \to \exists$ for halting); and full logical determination first appears at excluded middle (Post's backward direction, whose dovetail totality condition we prove equivalent to EM).

The partition tracks three regimes of the double-negation monad's counit within the fourth level of the fixed-point derivation, where self-indexing introduces extensional equivalence. The boundary is determined by whether proofs operate on values directly or must extract witnesses from double negations — not by which logical connective appears in the statement. We verify this with twenty Lean 4 files (zero sorry) — nineteen fully standalone (zero imports) plus one two-file cluster, corroborate by applying `#print axioms` across the main project's declarations, confirming that Classical.choice enters exclusively through Mathlib infrastructure, and prove that excluded middle is not derivable from the axiom systems used (all satisfiable at $\emptyset$). While the constructive character of many computability results is known — and synthetic approaches [Forster 2021] eliminate the Markov gap by assuming all functions computable — within the equational monoidal-closed formalization and the theorem formulations used here, Markov's principle is the exact boundary for Rice's theorem — a measurement of this formalization, not an absolute fact about the theorem, as the comparison with synthetic approaches (Section 6.1) confirms.

Contents

A machine-verified companion formalization in Lean 4 is archived as part of the multi-paper code repository [Close 2026e] at DOI: 10.5281/zenodo.18915083. All theorems cited in this paper are verified with zero sorry, zero Classical.choice, and zero custom axioms beyond Lean's structural axioms propext and Quot.sound.

1. Introduction

Standard mathematics uses the phrase "proof by contradiction" for two logically distinct operations:

Negation introduction. To prove ¬P, assume P and derive a contradiction. Formally: (P → False) → ¬P. This constructs a function from P to absurdity. The conclusion is negative – it says something does not hold.

Reductio ad absurdum. To prove P, assume ¬P and derive a contradiction. Formally: (¬P → False) → P, equivalently ¬¬P → P. Unlike negation introduction, this converts a negative impossibility statement into a positive conclusion.

Classical logic identifies these operations because it validates P ↔ ¬¬P. Constructive logic distinguishes them: negation introduction is a theorem of intuitionistic logic, while reductio is equivalent to the law of excluded middle. Between them sits Markov's principle, which permits reductio for computably enumerable predicates: if a Turing machine cannot not-halt, then it halts.

This paper demonstrates that the central theorems of computability theory partition according to which operation they require. The partition is not a philosophical interpretation. It is a measurement: Lean 4's kernel tracks which axioms participate in each proof term, and we report those profiles for theorems formalized from a common categorical starting point.

1.1 The measurement

We formalize computability theory starting from the equational theory of a monoidal closed category with an omega-chain converging to a Lambek fixed point – four equations that we call *identity modulation* [Close 2026a]. That paper establishes the four-level structure of the fixed-point tower (endofunctor algebra, Lambek convergence, containerization, computation), proves the specification is substrate-independent (D = 1), and verifies the formalization in Lean 4 with zero sorry. This paper discovers that Level 4 – Computation – has internal structure invisible to classical mathematics: the naming layer introduces extensional equivalence, and the axiom boundary within that equivalence decomposes into three precise layers, each corresponding to a distinct counit regime of the double-negation monad.

From this starting point, we prove the standard theorems of computability theory and record each theorem's axiom profile. Throughout, we distinguish two components: the *kernel profile* — the transitive kernel axiom dependencies reported by Lean's `#print axioms` command (propext, Quot.sound, Classical.choice) — and *theorem hypotheses* — local assumptions such as MarkovHalting or Classical.em appearing in theorem statements. The axiom profile of a theorem is the pair (kernel profile, theorem hypotheses). A theorem with kernel profile ∅ and hypothesis MarkovHalting uses no kernel axioms but assumes Markov's principle as an explicit premise.

Three distinct claims apply to formalized results, and this paper uses each precisely:

- **Machine-verified**: all proofs are checked by Lean 4's kernel. Every result in this paper is machine-verified.
- **Choice-free**: Classical.choice does not appear in the kernel profile. Files at {propext} or {Quot.sound} are choice-free but not axiom-free.
- **Constructive**: the kernel profile is ∅ (no axioms beyond Lean's core type theory). Only Layer 0 results with kernel profile ∅ are constructive in this strict sense.

Lean 4's type theory admits three axioms beyond its core:

- **propext** (propositional extensionality): equivalent propositions are equal. For purposes of this axiom-accounting analysis, we treat propext as a structural translation principle rather than a new

computational existence principle — it ensures that identity in the type theory respects logical equivalence.

- **Quot.sound**: quotients respect their equivalence relation. Like propext, this is a structural translation principle — it ensures the type theory's quotient types behave as their defining equivalence relations dictate. No new computational content enters.
- **Classical.choice**: every nonempty type has an element. This is genuine classical content – it asserts existence without construction. It implies the law of excluded middle and the axiom of choice for propositions.

The measurement distinguishes between propext/Quot.sound (structural translation principles that align type-theoretic identity with logical equivalence) and Classical.choice (genuinely new mathematical content). This accounting stance is justified by the consistency and non-derivability results of Section 5.2. The real boundary is one axiom thick.

1.2 The result

The axiom profiles partition computability theory into three layers:

Layer	Kernel axioms	Hypothesis	$\neg\neg$ counit	Theorems
0	$\emptyset$†	none	None needed	Y combinator, recursion theorem, halting, Gödel I+II, s-m-n, composition, diagonal lemma, Post forward, hierarchy separation, Myhill
1	{propext}	MarkovHalting	Partial: ε at $\exists$	Rice's theorem, Kleene normal form, Rogers totality
2	$\emptyset$	EM for P	Full: ε at all P	Post backward, hierarchy properness (full), Rogers isomorphism (full)

†Layer 0 includes Myhill's isomorphism theorem, whose kernel profile is {Quot.sound} rather than $\emptyset$. As discussed in Section 1.1, Quot.sound is a structural translation principle, not classical content. See Appendix B for the full per-theorem kernel profiles.

At Layer 0, every theorem is proved by exhibiting witnesses or by assuming a positive claim and deriving absurdity (negation introduction). No theorem at this layer needs to cross from negative to positive.

At Layer 1, one theorem – Rice's – requires extracting a positive witness (a program halts) from the impossibility of its negation (the program cannot not-halt). This is Markov's principle restricted to the halting predicate, and it is the exact point where the constructive proof breaks down: we can prove $\neg\neg$halts but not halts. The standalone file RiceBoundary.lean demonstrates the gap by proving Rice constructively up to the double-negation step, then closing it with an explicit Markov hypothesis.

At Layer 2, full excluded middle is required. Post's theorem (backward direction: RE and co-RE implies decidable) asserts that for every input, one of two parallel computations halts – a disjunction that is excluded middle for the predicate. The arithmetic hierarchy's properness and Rogers' isomorphism theorem require similar unrestricted reductio.

1.3 Three independent evidence sources

The partition is established by three independent methods:

Standalone files. For each key result, we produce a self-contained Lean 4 file with no imports – no Mathlib, no standard library, nothing. The file axiomatizes the minimum categorical or computational vocabulary needed, proves the theorem, and reports its axiom profile via #print axioms. Twenty files demonstrate this across all three layers. Layer 0 files include ConstructiveOmega (Y combinator), ConstructiveRecursionTheorem (Kleene recursion), ConstructiveHalting (halting undecidability), ConstructiveGodelI (Gödel's first

incompleteness theorem; file names follow ASCII convention), ConstructiveSmn (s-m-n theorem), ConstructiveComposition (composition, identity, constants), ConstructiveDiagonalLemma (Gödel diagonal lemma), ConstructivePostForward (Post forward direction), HierarchyProperness (arithmetic hierarchy separation), and MyhillBoundary (Myhill's isomorphism theorem) – all at $\emptyset$ or {Quot.sound}. The boundary files are ContradictionModes (negation intro vs reductio, $\emptyset$), RiceBoundary (Rice's theorem, {propext} + Markov hypothesis), KleeneNormalForm (Kleene normal form, Markov for μ-minimization), RogersBoundary (Rogers isomorphism, {propext} + Markov), and ClassicalPostBackward (Post backward, $\emptyset$ with EM hypothesis). The meta-theoretic files are Conservativity (all axiom systems consistent and EM non-derivable, $\emptyset$) and EvidenceModes (formalization of the counit characterization at $\emptyset$). The choice-analysis files are Diaconescu (choice + propext $\to$ EM, {propext}), KripkeCountermodel (EM fails constructively, $\emptyset$), and ModalityIndexedTypes (choice on stable types is redundant, $\emptyset$). Gödel I at $\emptyset$ is a headline result: the most famous limitative theorem in mathematics — often described as THE example of "proof by contradiction" — compiles with zero axioms, because both unprovability and unrefutability are negation introduction, not reductio (Section 3.4). The Myhill file is a critical test case: the back-and-forth construction branches extensively but compiles at Layer 0 because every branch is over a computed value, not a logical assertion (Section 5.4). The companion formalization builds with zero sorry, zero Classical.choice, and zero custom axioms; individual results are stratified by remaining kernel dependencies as shown in Appendix B.

Proof-trace analysis. For each theorem, we trace the standard proof step by step and identify the exact logical operation at each step. Halting undecidability uses negation introduction throughout: "assume a decider exists" (hypothesis for negation), "construct the diagonal program" (explicit construction), "derive not-halts" (negation introduction on halts), "derive halts" (explicit witness: the computation produces value yes), "contradiction" (not-P and P yield absurdity). No step crosses from negative to positive without a witness. Rice's proof, by contrast, requires "not-not-halts implies halts" at one step – the contrapositive of the reduction, where the proof has ¬¬(exists v, eval e 0 v) and needs to extract the existential. This is Markov's principle for Sigma-0-1.

Applying `#print axioms` to declarations in the 68-file main project confirms that all 655 declarations (71.3%) carrying Classical.choice inherit it through exactly three Mathlib infrastructure channels: CategoryTheory.Closed, HasInitial, and Aesop.TreeSpec — none through mathematical reasoning.

1.4 Significance

The result establishes three things:

First, **the core limitative theorems are constructive.** The Y combinator, Kleene's recursion theorem, the undecidability of the halting problem, and Gödel's first incompleteness theorem compile with zero axioms. This is not obvious: the halting problem and Gödel I are routinely described as "proofs by contradiction," and Kleene's recursion theorem is routinely proved using Gödel numbering, which appears to require classical infrastructure. The standalone files show that all three are constructive once "proof by contradiction" is disambiguated: Gödel I's unprovability (T cannot prove G) and unrefutability (T cannot prove not-G) are both negation introduction — assume the contrary and derive absurdity — not reductio.

Second, **the axiom boundary has a precise logical characterization.** It is the distinction between negation introduction and reductio ad absurdum, with Markov's principle as the intermediate step. While the constructive character of many computability results is known from the work of Bauer, Bridges, Richman, and others (Section 6.1), within the equational monoidal-closed formalization and the theorem formulations used here, Markov's principle is the exact boundary for Rice's theorem — a measurement of this formalization, not an absolute fact about the theorem, as the comparison with synthetic approaches (Section 6.1) confirms. RiceBoundary.lean establishes this by attempting the proof constructively, identifying the single step where it fails (the ¬¬halts $\to$ halts extraction), and closing the gap with an explicit Markov hypothesis that introduces no other classical content.

Third, **the methodology is replicable.** The standalone-file approach (axiomatize minimally, prove from scratch, verify axiom profile) can be applied to any theorem in any formalized theory. `#print axioms` is the ground truth at every step: it reports the transitive axiom closure of any declaration, and the standalone methodology makes that closure fully visible by eliminating library dependencies. Together they provide a

systematic method for determining the axiom content of mathematical results. We propose a classification program: determine the axiom profiles of the remaining theorems of computability theory, and extend the method to other areas of mathematics.

1.5 Outline

Section 2 presents the categorical starting point: the four levels of structure, the identity/equivalence transition, and why the starting point matters for axiom tracking. Section 3 presents the Layer 0 results. Section 4 presents Rice's theorem and the Markov boundary. Section 5 characterizes the boundary (the double-negation monad, the computed/asserted distinction refined by Myhill), presents the Layer 2 results including the Post backward equivalence, and analyzes infrastructure contamination. Section 6 discusses related work. Section 7 discusses implications.

---

2. The Starting Point: Identity Modulation

2.1 The four equations

The formalization begins from the equational theory established in [Close 2026a]: a monoidal closed category C with an endofunctor M that preserves directed colimits. The omega-chain from the initial object converges to a fixed point L satisfying the Lambek isomorphism M(L) ≅ L. When M is the internal hom functor ihom(A, -), this gives L ≅ [A, L]. Setting A = L yields the reflexive equation D ≅ [D, D]: the object D is isomorphic to its own endomorphism space.

This isomorphism — fold : [D, D] → D and unfold : D → [D, D] — generates four equations that we call the *identity loop*:

```
eval    = unfold(id)          -- evaluation IS identity unfolded
quine   = fold(selfApp)       -- self-reference IS self-application folded
fold ∘ unfold = id            -- the constructive round-trip
unfold ∘ fold = selfApp       -- the self-application equation
```

The first two equations define evaluation and self-reference in terms of fold/unfold. The third says the round-trip fold-after-unfold is the identity. The fourth says the reverse round-trip unfold-after-fold is self-application (apply a morphism to itself via the reflexive structure).

Equations 3 and 4 are NOT symmetric. Equation 3 is an isomorphism property (constructive, invertible, lossless). Equation 4 produces selfApp, which generates both the Y combinator (positive: every endomorphism has a fixed point) and the mode gap (limitative: extensional properties of programs are undecidable). The asymmetry between these equations is the structural source of the asymmetry between negation introduction and reductio, as we develop in Section 5.

2.2 The identity/equivalence transition

The four-level structure is established in [Close 2026a]. What that paper does not address — and what this paper discovers — is that the transition from Level 3 to Level 4 is not merely the addition of self-indexing but the introduction of a fundamentally new kind of mathematical object: extensional equivalence, distinct from identity. The derivation proceeds through four levels of structure, each visible only when the ambient category provides it:

1. **Dimension.** Endofunctor + initial object. The omega-chain exists.
2. **Convergence.** The colimit exists. Lambek gives M(L) ≅ L. The four equations above live here.
3. **Closure.** M = ihom(A). Self-application exists. The lambda model with eval/name and β/η lives here.
4. **Computation.** A = L. The object indexes its own endomorphisms.

Levels 1-3 are purely **identity-based**: every equation is of the form a = b where both sides are exhibited. fold ∘ unfold = id. β-reduction. Composition. Curry/uncurry. All equational, all constructive. To prove an equation is to exhibit both sides and verify they are the same — the proof IS the witness.

Level 4 introduces **equivalence**: multiple names for the same endomorphism. Two program indices a and b may satisfy φ_a = φ_b (extensional equivalence — they compute the same function on all inputs) without being identical (a ≠ b). The transition from Level 3 to Level 4 is the transition from identity to equivalence.

The axiom boundary falls within Level 4, at the interface between **constructed** equivalences (where the extensional equality is verified by tracing through equational structure) and **asserted** equivalences (where the extensional equality is known to hold but no specific computation witnesses it). The remainder of this paper measures where that boundary falls and what generates it.

2.3 Standalone files

Each key theorem is proved in a self-contained Lean 4 file with no imports. The file axiomatizes the minimum structure needed, proves the theorem, and reports its axiom profile via `#print axioms`. The dependency chain is visible in its entirety — there is no possibility of inherited contamination from libraries developed under classical assumptions. Twenty files support the classification: nineteen fully standalone (zero imports) plus ConstructiveGodelII.lean, which extends ConstructiveGodelI.lean as a two-file standalone cluster. Together they confirm axiom profiles across all three layers. Detailed discussion of the methodology and its relation to prior axiom-tracking approaches appears in Section 6.3.

---

3. The Constructive Column (Layer 0)

Every theorem in this section compiles with axiom profile ∅: no propext, no Quot.sound, no Classical.choice. Every proof uses only negation introduction and explicit witness construction — never reductio.

3.1 The Y combinator (ConstructiveOmega.lean, 168 lines)

Axiomatization. A typeclass `ReflCat` capturing the equational theory of a monoidal closed category with reflexive object: types for objects and morphisms, composition, tensor, internal hom, curry/uncurry, and the reflexive isomorphism fwd/bwd : [L, L] ≅ L.

Theorem. For all f : L → L, the morphism omega(f) = reflexiveCurry(selfApp ∘ f) satisfies:

wL(omega(f)) ∘ selfApp = selfApp ∘ f

where wL denotes left whiskering (tensoring on the left with the reflexive object L), the internal Lean name from ReflCat.wL. This is the fixed-point equation: omega(f) is a fixed point of f, and the proof constructs the witness by unwinding curry/uncurry equations.

Proof operation. Pure equational reasoning. Each step applies an axiom of the monoidal closed structure. No negation, no contradiction, no case analysis. The proof is a chain of equalities.

Axiom profile: ∅. Verified by `#print axioms omega_fixed_point`.

3.2 The recursion theorem (ConstructiveRecursionTheorem.lean, 181 lines)

Axiomatization. A typeclass `NamingSystem` capturing the minimal structure of a programming language: a type `Prog` of program indices, relational evaluation (`eval_rel e x v` means program e on input x produces value v), deterministic evaluation, pairing with projections, s-m-n (program specialization), and diagonal closure (the language can express the self-referential composition g(x,y) = φ_{f(smn(x,x))}(y)).

Theorem. For any program index e_f and any value fn that e_f produces on input fixedPointIndex(e_f):

∀ y v, eval_rel (fixedPointIndex e_f) y v ↔ eval_rel fn y v

The fixed point index is explicitly constructed: fixedPointIndex(e_f) = smn(diag_closure(e_f), diag_closure(e_f)). The proof unwinds the s-m-n and diagonal closure specifications.

Connection to 3.1. This is the SAME fixed-point construction as omega_fixed_point, instantiated at the naming level:

Categorical (3.1)	Naming (3.2)	Role
curry	smn	Specialization
fwd/bwd	name/evaluate	Reflexive structure
selfApp	φ_x(x) (self-application)	Diagonal
omega(f)	fixedPointIndex(e_f)	Fixed point

Both compile at $\emptyset$ because both are constructions: they hand you the witness. The naming layer introduces no classical content.

Proof operation. Equational unwinding of s-m-n and diagonal closure, plus determinism of evaluation (if e produces both v and w, then v = w). No negation, no contradiction.

Axiom profile: $\emptyset$.

3.3 Halting undecidability (ConstructiveHalting.lean, 161 lines)

Axiomatization. A typeclass `DecidabilitySystem` capturing: programs, relational evaluation, determinism, two distinguished outputs yes and no (with yes $\neq$ no), a halting predicate (halts e x $\leftrightarrow \exists$ v, eval_rel e x v), pairing, and a diagonal program constructor (given a supposed decider d, construct diag_prog(d) that inspects d's answer and does the opposite).

Theorem. $\neg\exists$ d, IsHaltingDecider d.

Proof structure. This is the critical case. The theorem is a negation, so the proof is by negation introduction: assume a decider d exists, derive absurdity.

Step 1 (negation introduction): Prove ¬halts(p, p). Assume halts(p, p). Then d says "yes" on (p, p). But diag_diverges says: when d says "yes," the diagonal program produces no value. This contradicts the evaluation witness from halts(p, p). Therefore halts(p, p) $\rightarrow$ False, i.e., ¬halts(p, p).

Step 2 (witness construction): Prove halts(p, p) with explicit witness. The derivation proceeds by forward implication through the decider's specification, not by double-negation elimination: From ¬halts(p, p) and the decider's correctness, d says "no" on (p, p). Then diag_halts says: when d says "no," the diagonal program produces the value `yes`. So eval_rel p p yes holds, and halts(p, p) follows with witness `yes`.

Step 3 (contradiction): ¬halts(p, p) $\wedge$ halts(p, p) $\rightarrow$ False.

Why this is NOT reductio. Step 2 appears to derive a positive fact (halts(p, p)) from a negative premise (¬halts(p, p)). But the derivation does not use double negation elimination. It CONSTRUCTS the witness `yes` by tracing through the decider's computation: ¬halts $\rightarrow$ d outputs no $\rightarrow$ diagonal program outputs yes $\rightarrow$ halts with witness yes. The witness is exhibited, not inferred from a double negation. The logical operation at every step is forward implication, not reductio.

This distinction — between constructing a witness via a computational chain and extracting a witness from a double negation — is the structural content of the $\emptyset$/Markov boundary. Section 4 shows where the constructive chain breaks for Rice's theorem.

Axiom profile: $\emptyset$. Verified by `#print axioms halting_undecidable`.

3.4 Gödel's first incompleteness theorem (ConstructiveGodelI.lean, ∅)

Axiomatization. A typeclass `GoedelSystem` capturing: a type of sentences with a provability predicate, a Gödel sentence G satisfying the fixed-point property (G ↔ ¬Provable(G) within the theory), and consistency (no sentence is both provable and refutable).

Theorem. For any consistent GoedelSystem: - T ⊬ G (`goedel_unprovable`) - T ⊬ ¬G (`goedel_unrefutable`)

Both are negation introduction.

Why Gödel I is NOT reductio. The standard presentation says "assume G is provable, derive a contradiction; therefore G is unprovable." This is negation introduction: the conclusion is ¬Provable(G), a negation. The proof assumes Provable(G) and derives absurdity. No positive fact is asserted without a witness.

Similarly for unrefutability: assume Provable(¬G), derive a contradiction (the theory would prove both G and ¬G, contradicting consistency). The conclusion is ¬Provable(¬G), again a negation. Both halves are negation introduction — assuming the positive claim and deriving absurdity — not reductio — assuming the negation of a positive claim and deriving the positive claim.

Gödel I is often cited as the paradigmatic "proof by contradiction." The axiom profile reveals that it is constructive: the "contradiction" in both directions is assumption-for-negation, not double negation elimination. The theorem is parametric over any GoedelSystem — the proof structure is independent of the specific formalization of arithmetic.

Axiom profile: ∅. Verified by `#print axioms goedel_first_incompleteness`.

The typeclass formulation proves Gödel I at ∅ conditional on the existence of a GoedelSystem instance. This instance is constructed in ConstructiveGodelI.lean using the diagonal lemma (proved at ∅ in ConstructiveDiagonalLemma.lean), confirming that the full instantiation — not just the structural implication — compiles at profile ∅. The `#print axioms` output for both files is empty.

Gödel's second incompleteness theorem (ConstructiveGodelII.lean, ∅). Gödel II — no consistent, sufficiently expressive formal system can prove its own consistency — extends Gödel I with an implication connective, internal modus ponens, a consistency sentence, and the formalizability hypothesis (the system can internalize its own Gödel I proof — this is the main substantive assumption for Gödel II beyond what Gödel I requires, asserting the system is strong enough to formalize the meta-reasoning of Gödel I within itself). The meta-level proof is pure negation introduction: assume T ⊢ Con(T), derive T ⊢ G via the formalized Gödel I and backward diagonal, contradicting goedel_unprovable. This completes the limitative chain at ∅: Y combinator, recursion theorem, diagonal lemma, halting undecidability, and both Godel incompleteness theorems are all constructive.

Axiom profile: ∅. (ConstructiveGodelII.lean imports ConstructiveGodelI.lean, itself standalone at ∅.)

3.5 The s-m-n theorem (ConstructiveSmn.lean, ∅)

The s-m-n theorem — for any program e computing a two-argument function, smn(e, x) is a program computing y → e(x, y) — is the naming-level analogue of currying. The file proves specialization, totality preservation, divergence preservation, iterated s-m-n, determinism, and extensionality. All purely structural — s-m-n rearranges inputs without deciding any proposition.

Axiom profile: ∅.

3.6 Composition, identity, constants (ConstructiveComposition.lean, 237 lines)

Axiomatization. A typeclass `ComposableSystem` capturing: programs, relational evaluation, determinism, pairing with projections, s-m-n, composition closure (the language can express sequential evaluation), an identity program, and constant program construction.

Theorems. The file proves:

- **Composition** (`comp_spec`): given program indices e_f and e_g, the composed program comp_closure(e_f, e_g) computes f ∘ g. Both forward and converse directions are established.
- **Identity laws** (`comp_id_right`, `comp_id_left`): composing with the identity program preserves behavior in both directions.
- **Constant absorption** (`comp_const_left`, `comp_const_right`): composing with a constant program behaves as expected.
- **Associativity** (`comp_assoc_forward`): composition is associative.

Axiom profile: ∅. Verified by `#print axioms` on all eleven theorems.

3.7 Diagonal lemma (ConstructiveDiagonalLemma.lean, ∅)

The diagonal lemma — given any total computable function f, there exists a sentence σ such that $\sigma \leftrightarrow$ f($\ulcorner\sigma\urcorner$) — has the same structure as the recursion theorem (Section 3.2). The fixed-point index is explicitly constructed by the same diagonal pattern: smn applied to the diagonal closure. The proof is pure equational unwinding.

Axiom profile: ∅.

3.8 Post's theorem, forward direction (ConstructivePostForward.lean, ∅)

The forward direction of Post's theorem — decidable implies RE ∧ co-RE — is constructive: given a total decider d, the verifier runs d and checks for "yes" (deterministic output routing), and the refuter runs d and checks for "no." Both are explicit constructions.

Axiom profile: ∅.

3.9 Hierarchy separation (HierarchyProperness.lean, ∅)

The arithmetic hierarchy separates at every level: for each n, there exists a level-(n+1) predicate not equivalent to any level-n predicate. The proof is a constructive diagonal argument — at level n, construct a predicate that diagonalizes against all level-n predicates. At level 0, this IS the halting undecidability theorem. The core lemma `iff_not_self_absurd` (P ↔ ¬P is absurd) drives the diagonalization. No excluded middle is needed because the argument is purely structural: the diagonal predicate DIFFERS from every level-n predicate by construction.

This is the *separation schema* — it proves the existence of a separating predicate at each level. The full *hierarchy properness* theorem (Σ^0_n $\subsetneq$ Σ^0_{n+1} as sets over a fixed representation) additionally requires excluded middle to establish that the level-n predicates form a proper subset of the level-(n+1) predicates (Section 5.6).

Axiom profile: ∅.

3.10 Myhill's isomorphism theorem (MyhillBoundary.lean, {Quot.sound})

Theorem. If sets A and B are computably reducible to each other (A ≤_m B and B ≤_m A via computable injections), then A and B are computably isomorphic (there exists a computable bijection).

Why this result matters for the boundary. Myhill's back-and-forth construction branches at every stage: is this element already in the domain? Already in the range? The naive prediction — branching requires excluded middle — is wrong. Every branch in Myhill's proof is over a **computed value**: at each stage, the construction checks a concrete condition on a finite list and takes a concrete action. The branching is over Bool (a Type-level value produced by running a check), not over a Prop-level disjunction (a logical assertion without computational content).

This is the critical test case for the boundary characterization. The statement of Myhill's theorem involves equivalence (A ≅ B), but the proof constructs the equivalence by identity-level operations at each step. The bijection is BUILT, not asserted. The construction never needs to determine "P(x) or not-P(x)" for an

undecidable P — it only needs to check "is x in this finite list?" which is decidable by inspection. Section 5.4 develops the implications for the formal characterization of the boundary.

Axiom profile: {Quot.sound}. The Quot.sound enters through list membership operations — it is quotient functoriality (Section 1.1), not classical content.

---

4. The Markov Boundary (Layer 1)

Rice's theorem — no nontrivial extensional property of programs is decidable — requires exactly Markov's principle: ¬¬(∃v, eval e x v) → ∃v, eval e x v. Not more (excluded middle is overkill), not less (constructive logic cannot close the gap). This section presents the experimental evidence from RiceBoundary.lean (471 lines), which attempts the proof at progressively stronger axiom levels to identify the exact boundary within this formalization.

4.1 The constructive attempt

The standard proof of Rice reduces to halting undecidability. Given a supposed decider d for a nontrivial extensional property P, construct a program rice_prog(d, e) that behaves like witness_in (a P-member) if e halts on zero, and behaves like empty_prog (a non-P-member) if e diverges on zero. Then d composed with rice_prog decides halting.

The forward direction is constructive:

- halts(e, 0) → rice_prog(d, e) is extensionally equal to witness_in → P(rice_prog(d, e)) → d(rice_prog(d, e)) = yes.
- ¬halts(e, 0) → rice_prog(d, e) is extensionally equal to empty_prog → ¬P(rice_prog(d, e)) → d(rice_prog(d, e)) = no.

Both implications are proved by forward chaining through extensionality and the decider's specification. No classical reasoning. The theorem `rice_constructive_reduction` verifies this at axiom profile {propext}.

4.2 The gap

The backward direction breaks. We need: d(rice_prog(d, e)) = yes → halts(e, 0). The proof attempt:

Suppose d says yes on rice_prog(d, e). Suppose ¬halts(e, 0). Then d says no on rice_prog(d, e) (by the forward direction). But d is deterministic and yes ≠ no. Contradiction. Therefore ¬¬halts(e, 0).

This is valid constructive reasoning — it's negation introduction applied to ¬halts(e, 0). But the conclusion is ¬¬halts(e, 0), not halts(e, 0). The step from ¬¬halts to halts is double negation elimination — reductio — and it is the ONLY non-constructive step in the entire proof.

The theorem `rice_constructive_gap` documents this precisely: the constructive proof produces a "double-negation decider" for halting. It satisfies: - halts(e, 0) → d(rice_prog(d, e)) = yes [constructive] - ¬halts(e, 0) → d(rice_prog(d, e)) = no [constructive] - d(rice_prog(d, e)) = yes → ¬¬halts(e, 0) [constructive] - d(rice_prog(d, e)) = no → ¬halts(e, 0) [constructive]

Without Markov, this does not yield a genuine halting decider, and the contradiction with halting undecidability does not go through.

4.3 Markov closes the gap

Adding Markov's principle for the halting predicate as an explicit hypothesis:

MarkovHalting := ∀ e x, ¬¬halts(e, x) → halts(e, x)

converts ¬¬halts(e, 0) to halts(e, 0), making the double-negation decider a genuine decider. The contradiction with halting undecidability (which is itself constructive, per Section 3.3) then goes through.

The theorem `rice_theorem_markov` proves Rice with Lean axiom profile {propext} and explicit hypothesis MarkovHalting. The propext enters through the Iff (↔) in the extensional equality definition — it is a structural translation principle (Section 1.1), not classical content. All classical content is isolated in the Markov hypothesis.

4.4 Markov is necessary

The gap in 4.2 is not an artifact of the proof strategy. In the effective topos (Hyland 1982), Markov's principle holds and Rice holds. In modified realizability models where Markov fails, the double-negation gap is genuine: there exist "deciders" that satisfy the double-negation specification without satisfying the genuine specification. Rice's theorem (as stated) fails in these models.

This confirms: Markov's principle is both necessary and sufficient for Rice's theorem. The boundary is not approximate — it is exact.

4.5 The logical mechanism

The ¬¬halts → halts step is reductio ad absurdum restricted to a Σ^0_1 predicate. The halting predicate — ∃v, eval_rel e x v — is an existential over a decidable relation. Markov's principle says: for such predicates, if the existential cannot be false, it is true.

Contrast with the halting proof (Section 3.3), where halts(p, p) is proved by CONSTRUCTING the witness yes. In the halting proof, the positive fact is obtained by tracing a computation. In Rice's proof, the positive fact (halts(e, 0)) is obtained by eliminating a double negation — the computation exists but is not traced. The witness is recovered from the impossibility of its nonexistence, not exhibited directly.

This is the boundary between negation introduction and reductio, made visible by a single theorem (Rice) that needs exactly one step of reductio (for a single Σ^0_1 predicate) to close. This analysis is specific to the halting-reduction proof strategy formalized in RiceBoundary.lean. Alternative proof strategies for Rice's theorem may distribute the non-constructive step differently; the standalone file makes the single Markov use in this particular strategy machine-verified.

4.6 Markov at other results

Two additional standalone files confirm the same Markov-level boundary:

Kleene normal form (KleeneNormalForm.lean). The T-predicate and output extraction are constructive (∅). The μ-minimization step — unbounded search for a computation trace satisfying T(e, x, y) — requires Markov's principle: we can prove the search cannot fail (¬¬∃y, T(e,x,y)) but extracting the witness y requires ¬¬∃ → ∃. The file uses Bool-valued T_decide rather than Prop-level decidability, keeping the constructive parts at ∅.

Rogers isomorphism (RogersBoundary.lean). The translation index between acceptable numberings is constructive (∅). The extensional correctness is constructive (∅). But totality of the translation — every program index in one numbering maps to a halting program in the other — requires Markov, again for the same shape: the translation cannot fail to halt (¬¬halts), so Markov extracts the halting witness. Three results at Markov, all sharing the same structure: ¬¬∃ → ∃ for search termination.

---

5. The Boundary Characterized

We present the boundary's formal characterization and its theoretical foundation. Three standalone files formalize this section: ContradictionModes.lean (axiom profile ∅), Conservativity.lean (axiom profile ∅), and

EvidenceModes.lean (axiom profile ∅ on core theorems, {propext} on derived ordering).

5.1 Theorem: Reductio is the counit of the double-negation monad

Double negation is a monad on Prop. Its unit, join, and functorial action are all constructive (proved at axiom profile ∅ in ContradictionModes.lean):

- **Unit** (dne_pure): $P \to \neg\neg P$.
- **Join** (dne_join): $\neg\neg(\neg\neg P) \to \neg\neg P$.
- **Map** (dne_map): $(P \to Q) \to (\neg\neg P \to \neg\neg Q)$.

This monad has no counit in general. The counit would be $\varepsilon\_P : \neg\neg P \to P$ — double negation elimination — which is exactly reductio ad absurdum.

> **Theorem** (reductio_iff_em, axiom profile ∅). *The existence of a counit $\varepsilon\_P : \neg\neg P \to P$ for all P is equivalent to the law of excluded middle.*

The three-layer boundary in computability theory is the counit's availability:

> **Counit Characterization.**
>
∅	`no counit`	`negation introduction only`
`Markov`	`partial counit`	ε `restricted to ∃-statements (`Σ^0_1`)`
`Classical`	`full counit`	ε `for all P`
>
> The chain is strict: $\emptyset \subsetneq$ Markov $\subsetneq$ Classical.

The strict inclusions are established by: - EM → Markov: proved constructively (em_implies_markov, axiom profile ∅) - Markov $\not\to$ EM: the effective topos [Hyland 1982] validates Markov but not EM - $\emptyset \not\to$ Markov: modified realizability models invalidate Markov

Each layer of the computability-theoretic partition (Sections 3-4) uses exactly the counit available at that layer and no more. Layer 0 theorems operate entirely within the ¬¬ monad. Layer 1 (Rice) applies the partial counit once, to extract a halting witness from a double negation. Layer 2 theorems apply the full counit to arbitrary propositions.

5.2 Consistency and non-derivability (Conservativity.lean, ∅)

The Layer 0 results are not merely empirical observations — they follow from a structural fact: the axiom systems axiomatized in Sections 3-4 are consistent with intuitionistic logic, and excluded middle is not derivable from them. Conservativity.lean proves this by constructing concrete models of all four axiom systems (ReflCat, NamingSystem, DecidabilitySystem, DiagonalSystem) at axiom profile ∅:

- ReflCat on Unit: all morphisms are the unique element of Unit. Every equation holds by reflexivity.
- NamingSystem on Unit: eval_rel is the trivially true relation. Determinism holds because Unit is a subsingleton.
- DecidabilitySystem on Bool: yes = true, no = false, yes_ne_no is provable constructively. The diagonal program axioms are satisfiable with Bool's decidable equality.
- DiagonalSystem on Unit: same subsingleton argument.

The joint satisfiability theorem `all_systems_satisfiable` witnesses that all four systems have models in Lean's constructive core (CIC without axioms). Since CIC without axioms does not prove excluded middle [Coquand-Werner 1997], excluded middle is not derivable from the axiom systems. The Layer 0 results are therefore not artifacts of hidden classical content — the axiom systems themselves contain no classical content to hide.

This transforms the Layer 0 results from individual measurements ("we checked these theorems and they compiled at ∅") into consequences of a structural property ("these axiom systems are satisfiable at ∅ and EM is not derivable from them, so everything derived from them constructively must compile at ∅"). The

standalone files for individual theorems remain valuable as confirmations and as exhibits of proof technique, but the non-derivability result provides the theoretical guarantee.

Axiom profile: $\emptyset$.

5.3 Counit characterization formalized (EvidenceModes.lean, $\emptyset$)

EvidenceModes.lean defines three propositional types corresponding to the counit regimes:

- **P** (direct): the proposition holds with an exhibited witness.
- $\neg\neg$**P** (double-negated): the proposition cannot be refuted, but no witness is exhibited — the counit would extract one.
- **P** $\vee$ $\neg$**P** (decided): a definite determination is available — the full counit resolves arbitrary propositions.

The file proves that the $\neg\neg$ monad structure (map, join, and) operates entirely on $\neg\neg$P at $\emptyset$, confirming the monad characterization of Section 5.1. The key theorem `irrefutable_constructed_iff_all_determined` establishes: the ability to convert $\neg\neg$P to P for all P (reductio, the full counit) is equivalent to P $\vee$ $\neg$P for all P (excluded middle). This is the formalized version of reductio_iff_em from ContradictionModes.lean, restated in terms of the three propositional types.

The file also proves that any predicate on Bool is decidable — Bool always gives P $\vee$ $\neg$P without EM. This is the formal reason Myhill's back-and-forth compiles at $\emptyset$: its branches are over Bool (decidable by computation), not over arbitrary propositions (decidable only with EM).

Axiom profile: $\emptyset$ on core theorems; {propext} on the AxiomLayer ordering (from simp on the inductive type).

5.4 The computed/asserted boundary

The counit characterization (Section 5.1) describes the formal mechanism: Markov provides the counit at existentials, EM provides it at all propositions. But a naive reading — "existentials need Markov, disjunctions need EM" — is refuted by Myhill's theorem (Section 3.10), which branches extensively at Layer 0. The refinement reveals a more fundamental principle.

> **Boundary principle.** *The axiom layer of a theorem is determined not by which logical connective appears in its statement, but by whether evidence is computed or asserted:*
>
> `Computed evidence (identity-level)` $\rightarrow$ $\emptyset$
> `Asserted existence behind ¬¬` $\rightarrow$ `Markov`
> `Asserted disjunction behind ¬¬` $\rightarrow$ `EM`

Myhill as exhibit. Myhill's back-and-forth construction branches at every stage: is this element already mapped? The answer is determined by checking a concrete, finite list — a Bool computation in Type, not a logical assertion in Prop. The construction never encounters a double negation to eliminate. Every branch is identity-level: run a check, get a definite answer, act on it. Myhill compiles at Layer 0 ({Quot.sound}) despite its extensive branching.

Rice as contrast. Rice's proof derives $\neg\neg$halts(e, 0) — the program "cannot not halt" — via contrapositive reasoning through an extensional property. The witness exists somewhere but has not been traced. The evidence is asserted (the impossibility of nonexistence) rather than computed (a specific halting trace). Extracting the witness requires Markov's counit at $\exists$ in Prop.

Post as further contrast. Post's backward direction needs halts(v, x) $\vee$ halts(r, x) — which of two parallel computations halts. No computation determines this in general. The evidence would need to be asserted as a Prop-level disjunction, and extracting it from $\neg\neg$(A $\vee$ B) requires full EM because the disjunction carries branch information (which side) that double negation destroys.

The pattern: Myhill's branching is over computed Bool values (Type-level, identity). Rice's extraction is from a Prop-level existential behind $\neg\neg$. Post's determination is of a Prop-level disjunction behind $\neg\neg$.

The boundary tracks whether the proof operates on values directly or must extract witnesses from double negations — not the logical form of the statement.

This connects to the identity/equivalence transition (Section 2.2): direct computation stays at the identity level (the container's fold/unfold on values), while double-negated propositions engage equivalence — the witness or branch is not exhibited, only its irrefutability. The three counit regimes correspond to three positions relative to the container boundary at Level 4: operating directly on values (no counit), extracting existential witnesses from ¬¬ (partial counit), and determining arbitrary propositions (full counit).

5.5 Conjecture: the fold/unfold asymmetry generates the boundary

The identity loop's four equations (Section 2.1) contain an asymmetry:

fold ∘ unfold = id (equation 3) unfold ∘ fold = selfApp (equation 4)

Equation 3 is invertible — the round-trip is lossless. Equation 4 produces selfApp, which generates the diagonal constructions that create both fixed points and impossibility results.

This asymmetry suggests an informal mapping onto the contradiction modes — the content of the conjecture stated below, not a proved correspondence:

Negation introduction corresponds to the fold direction. To prove ¬P (= P → False), you construct a morphism from P to the initial object (⊥). This is fold: take a value and embed it in the function space pointing to absurdity. The Lambek iso's fold direction is constructive, always available.

Reductio corresponds to extracting a witness through a double negation in the unfold direction. The proof state contains ¬¬(∃v, …) — the existential is irrefutable. To extract ∃v requires unfolding the double negation. But unfold through ¬¬ is not the same as unfold through the Lambek iso. The Lambek iso unfolds values constructively. Double negation unfolds propositions only with a counit — Markov or EM.

Conjecture 5.1 (Fold/unfold source of the axiom boundary). *The mode gap — the fact that equation 4 gives selfApp rather than id — is the structural source of the gap between negation introduction and reductio. Self-application applied to a decision procedure generates the diagonal program whose halting status is stuck behind a double negation. The fold direction (equation 3) resolves it constructively when a computational witness exists (halting proof). The unfold direction through ¬¬ (equation 4's shadow on propositions) requires Markov when the witness must be recovered from a double negation (Rice).*

Formalizing this connection as a theorem — rather than an observation about proof structure — is open.

5.6 The classical column (Layer 2)

Layer 2 theorems require full excluded middle. The distinguishing feature is the need for a logical determination — which of two things holds — where no computation produces the answer (Section 5.4).

Post's theorem, backward direction (ClassicalPostBackward.lean, 473 lines). RE ∧ co-RE → decidable. If a predicate P is both RE (verifier v: P(x) → halts(v, x)) and co-RE (refuter r: ¬P(x) → halts(r, x)), then P is decidable. The proof dovetails (interleaves) v and r on input x; since P(x) ∨ ¬P(x), one of them halts.

The file factors the proof into a constructive core and a classical step:

1. **Constructive core** (`post_constructive_core`, axiom profile ∅): the dovetail construction is correct. If P(x), the dovetail outputs yes. If ¬P(x), the dovetail outputs no. No EM needed.
2. **Classical step** (`em_gives_dovetail_total`): EM for P gives dovetail totality — ∀x, halts(v, x) ∨ halts(r, x). This is the only non-constructive step.
3. **Equivalence** (`dovetail_total_iff_em`, axiom profile ∅): with sound and complete verifier/refuter, dovetail totality IS excluded middle for the predicate P. Not merely implied by EM — equivalent to it.

The equivalence is the key structural finding. The backward direction (dovetail totality implies EM for P) holds because: if halts(v, x), the verifier is sound, so P(x); if halts(r, x), the refuter is sound, so ¬P(x). The

branch determination gives P(x) ∨ ¬P(x). Markov is insufficient because its partial counit extracts witnesses from ¬¬∃, while Post needs the full counit to resolve P ∨ ¬P — a determination that no computation produces for undecidable P.

The file also proves `dovetail_sound`: the dovetail's yes-output gives only ¬¬P constructively (not P), exhibiting the same double-negation gap as Rice. With EM, `dovetail_sound_em` strengthens this to P. Every theorem in the file has Lean axiom profile ∅; all classical content is carried as explicit hypotheses.

Hierarchy properness (full). $\Sigma^0\_n \subsetneq \Sigma^0\_\{n+1\}$ as a proper subset relation. The separation schema (Section 3.9) constructs a separating predicate at ∅. The full properness — that the level-n predicates form a proper subset of the level-(n+1) predicates over a fixed representation — requires excluded middle to establish subset containment across the entire representation space.

Rogers isomorphism. Any two acceptable numberings are computably isomorphic. The back-and-forth construction requires case analysis at each step — decidability that accumulates over infinitely many steps.

5.7 The circularity of choice (Diaconescu analysis)

Three standalone files provide a complete formal analysis of what Classical.choice actually does:

Diaconescu.lean (276 lines, axiom profile {propext}). Proves Diaconescu's theorem: choice + propext → excluded middle. Given any proposition P, define two predicates on Bool — pred_U(b) := (b = true ∨ P) and pred_V(b) := (b = false ∨ P). Both are nonempty. Choice picks an element from each. If P holds, propext makes the predicates equal, so choice picks the same value, and Bool decidability resolves the disjunction. The mechanism is entirely explicit: choice enters as an axiom-hypothesis (GlobalChoice class), and propext enters through the step that equates the two predicates.

KripkeCountermodel.lean (328 lines, axiom profile ∅). Constructs a two-world Kripke model where excluded middle fails. The base world w_0 has an undetermined proposition P (P.at_w_0 = False, P.at_w_1 = True). The theorem `em_not_derivable` proves ∃ p, ¬(p ∨ ¬p).at_w_0 — EM fails at w_0 for this proposition. Crucially, it also shows where Diaconescu's proof breaks: `diaconescu_global_step_fails` proves the global equality step (pred_U = pred_V when P holds locally) fails without propext, because truth-at-w_0 and truth-at-w_1 are independent.

ModalityIndexedTypes.lean (161 lines, axiom profile ∅). The key result: `stable_diaconescu_gives_em` proves that if Diaconescu's predicates (b = true ∨ P and b = false ∨ P) are decidable, then P ∨ ¬P already holds — choice is redundant. The converse `em_gives_stable_diaconescu` proves that if P is decidable, the predicates are decidable. Together they show: Diaconescu's types are stable exactly when P is already decided.

The circularity: choice produces EM only by operating on types whose structure depends on the proposition being decided. If the types are stable (their structure is settled), choice is unnecessary — the answer is already available by computation. If the types are unstable (their structure depends on P), choice treats the undetermined structure as determined and extracts an answer from that assumption. The non-constructive content of choice is not selection from a nonempty type but treating undetermined structure as determined — assuming the answer in the mechanism that is supposed to produce it.

5.8 Infrastructure contamination

The 68-file formalization project (0 sorry) uses Mathlib for categorical infrastructure. Applying `#print axioms` across the main project's declarations, 655 (71.3%) carry Classical.choice in their reported axiom profile. Tracing `#print axioms` output on targeted declarations identifies exactly three entry channels:

1. **CategoryTheory.Closed** (215 declarations). Mathlib's `Closed` class bundles the right adjoint with an `Adjunction` proof whose kernel- elaborated recursor uses Classical.choice. This is not a mathematical use of choice — the adjunction in our setting is explicitly constructed via the Lambek isomorphism. The classical content is in Mathlib's generic existence proof, not in our specific construction.

2. **HasInitial** (141 declarations). Mathlib asserts initial object existence classically. Our omega-chain construction produces the initial algebra explicitly.

3. **Aesop.TreeSpec** (47+ declarations). Lean's Aesop tactic framework uses classical axioms internally. This is pure tactic contamination with zero mathematical content.

The standalone files validate these cut predictions: ConstructiveOmega.lean replaces CategoryTheory.Closed with a direct axiomatization of curry/uncurry and compiles at ∅. ConstructiveRecursionTheorem.lean replaces the Mathlib computability infrastructure with a direct NamingSystem axiomatization and compiles at ∅. Each standalone file demonstrates that the corresponding Mathlib infrastructure channel introduces no mathematical content — only generic existence proofs that are unnecessary when specific constructions are available.

Summary. The main project's Classical.choice profile is infrastructure contamination, not mathematical content. The standalone files prove the content is constructive. `#print axioms` on the standalone replacements confirms the axiom profile drops to the predicted level, identifying exactly which definitions to replace. The three-layer boundary (∅ / Markov / Classical.choice) describes the mathematics, not the engineering of the formalization. During subsequent development, `#print axioms` detected a contamination path through Lean's core library (`Nat.pow_lt_pow_right` carries {propext, Classical.choice, Quot.sound} despite being pure natural-number arithmetic); replacement with direct inductive proofs confirmed the constructive profile, demonstrating that `#print axioms` on targeted declarations detects contamination paths that would otherwise be invisible, and that replacement with direct inductive proofs restores the constructive profile.

---

6. Related Work

6.1 Constructive and synthetic computability theory

Bauer's "First Steps in Synthetic Computability Theory" [Bauer 2006] develops computability theory in a constructive setting, proving many results including a version of Rice's theorem. Bridges and Richman [Bridges-Richman 1987] develop constructive analysis including computability-adjacent results. Troelstra and van Dalen [Troelstra-van Dalen 1988] provide a comprehensive treatment of constructive logic including Markov's principle.

Forster's synthetic program. Forster [Forster 2021, Forster-Kirst-Smolka 2019] formalizes Rice's theorem, Myhill's isomorphism theorem, and Post's simple sets in Coq without Markov's principle or choice, using a synthetic parametric Church-Turing thesis — the axiom that all functions are computable: $\forall\ f : N \to N, \exists\ e, \forall\ n, \varphi\_e(n) = f(n)$. In this setting, Markov's principle is derivable: if $\neg\neg\exists n, P(n)$ and P is decidable, the search function exists by synthetic CT, so the search terminates. Forster obtains Rice at ∅ and Myhill at ∅.

Our categorical starting point makes no such assumption. We axiomatize a reflexive object in a monoidal closed category — explicit Lambek fixed-point equations, not "all functions are computable." In this setting, Rice requires Markov and Myhill requires {Quot.sound}. The contrast is a feature, not a bug: it shows the axiom boundary is foundation-dependent. The synthetic CT axiom eliminates the Markov gap by making every search constructively terminating. Our equational setup preserves the gap because it does not assume unbounded search always terminates — it makes the termination assumption visible as an explicit hypothesis.

The two approaches are complementary. The synthetic program shows what is provable when all functions are assumed computable. Our equational program shows what is provable from the structural equations alone, without assuming any closure property about which functions exist. The Markov boundary in our setting is the precise point where "the search cannot not terminate" ($\neg\neg\exists$) fails to yield "the search terminates" ($\exists$) without an additional assumption — an assumption that synthetic CT provides for free and that our equational setup exposes.

The mechanism is precisely the identity/equivalence transition (Section 2.2): synthetic CT forces the equivalence layer to collapse into the identity layer by postulating that every function has a name. In our equational setup, the naming layer introduces extensional equivalence — multiple names for the same function

— and the gap between identity and equivalence is where Markov enters. Synthetic CT eliminates this gap by making the naming layer total. Consistent with Forster's results [Forster 2021], we predict Post's backward direction and hierarchy properness remain at EM even under synthetic CT, since the dovetail totality condition requires a determination ($P \vee \neg P$) that no amount of naming-layer totality provides. If confirmed, this would establish that Layer 2 is genuinely distinct from the Markov gap — about logical determination, not search termination.

Novelty claim. Within the equational monoidal-closed formalization and the theorem formulations used here, Markov's principle is the exact boundary for Rice's theorem — neither more nor less. This is established by machine-verified standalone proof. Synthetic approaches shift this boundary by assuming different axioms; the boundary is a property of the foundation, not an absolute fact about the theorems.

6.2 Realizability

Kleene realizability [Kleene 1945], function realizability [Kleene-Vesley 1965], and the effective topos [Hyland 1982] provide semantic models where constructive computability holds. Longley and Normann [Longley-Normann 2015] give a comprehensive treatment of higher-order computability in a realizability setting.

Our work is complementary: realizability provides the MODELS that separate the three layers (the effective topos validates Markov but not EM; modified realizability invalidates Markov). We provide the SYNTACTIC boundary in a specific proof assistant, identifying the exact step in each proof where the model separation manifests.

6.3 Axiom tracking in proof assistants

Lean 4's `#print axioms` command reports the transitive axiom closure of any definition. Similar facilities exist in Coq (Print Assumptions) and Agda (postulate tracking). Prior work has used these tools to verify specific results' classical content.

A subtlety: `#print axioms` reports which axioms appear in the proof term's dependency graph, not which axioms are "needed" in any mathematical sense. A theorem whose proof references a Mathlib definition that was built using Classical.choice will show Classical.choice in its profile, even if the theorem's mathematical content is constructive. This is the infrastructure contamination problem (Section 5.8).

Two complementary approaches address this. The **standalone methodology** produces import-free files where the entire dependency chain is visible: if `#print axioms` reports $\emptyset$, the result is genuinely axiom-free, because there is nothing to contaminate it. This provides the strongest guarantee but requires re-axiomatizing the needed vocabulary for each theorem. The systematic application of `#print axioms` to targeted declarations in library-dependent proofs traces contamination paths — the intermediate definitions through which a given axiom enters — and identifies which definitions to replace.

The two approaches are self-validating: `#print axioms` on targeted main-project declarations identifies contamination paths; standalone files realize the replacements; and `#print axioms` on the standalone files confirms the predicted axiom profile. All checks agree in every case tested.

6.4 Lean 4's axiom hierarchy

Carneiro [Carneiro 2019] analyzes Lean's axioms, distinguishing between the structural axioms (propext, Quot.sound) and the classical axiom (Classical.choice). Our interpretation — propext and Quot.sound as structural translation principles, Classical.choice as genuine non-constructive content — extends this analysis by connecting it to the categorical starting point and showing that the distinction is load-bearing for computability theory's structure. The accounting stance (treating propext/Quot.sound as non-classical) is justified by the consistency and non-derivability results of Section 5.2.

---

7. Discussion

7.1 The Church-Turing thesis decomposes

The Church-Turing thesis is usually stated as a single claim: every effectively computable function is computable by a Turing machine (or equivalently, by lambda calculus, recursive functions, etc.). The axiom profiles reveal that this thesis conflates two components with different logical content:

Component A (Layer 2, classical): All acceptable numberings are computably isomorphic. This is the Rogers isomorphism theorem. It says that the NAMING SYSTEMS are equivalent — every way of assigning indices to programs gives the same class of computable functions. This is a statement about the naming layer and requires full classical reasoning.

Component B (Layer 0, constructive): Computation is identity modulation. The Y combinator, the recursion theorem, and the core of self-referential computation compile with zero axioms. This is a statement about what computation IS — the fixed-point phenomenon in the equational theory of monoidal closed categories — prior to any choice of naming system.

The axiom profiles suggest a natural decomposition: the naming systems are equivalent (A), and what they name is identity modulation (B). A requires Classical.choice because it spans all naming systems. B requires nothing because it is the structural phenomenon itself.

7.2 The naming layer is constructive

The recursion theorem at axiom profile $\emptyset$ (Section 3.2) proves that the naming layer — programs, indices, Gödel numbering — does NOT introduce classical content. Names are constructive. Evaluation is constructive. Self-reference through names is constructive. Myhill's isomorphism theorem (Section 3.10) extends this further: even the back-and-forth construction of computable bijections between named sets is constructive, because every branch is over a computed value.

What introduces classical content is the transition from identity to equivalence (Section 2.2) when the equivalence is ASSERTED rather than CONSTRUCTED. Rice's theorem says: for ALL nontrivial extensional properties P, P is undecidable. The extensional property asserts equivalence across all inputs simultaneously. The contrapositive reasoning produces $\neg\neg$halts — an asserted existence — requiring Markov to extract the witness. Post's backward direction asserts a determination ($P \vee \neg P$) that no computation produces, requiring full EM.

The boundary is not between "computation" (constructive) and "naming" (classical). It is between constructed equivalences (where the equality is traced through equational structure) and asserted equivalences (where the equality is known to hold but not witnessed by a specific computation).

7.3 The classification program

We propose a systematic classification of computability-theoretic results by axiom profile. The verified results:

Theorem	Layer	$\neg\neg$ counit
Y combinator	0 ($\emptyset$)	None: equational fixed point
Kleene recursion theorem	0 ($\emptyset$)	None: smn diagonal
s-m-n theorem	0 ($\emptyset$)	None: input rearrangement
Halting undecidability	0 ($\emptyset$)	None: witness `yes` exhibited
Gödel I	0 ($\emptyset$)	None: negation introduction
Gödel II	0 ($\emptyset$)	None: negation introduction
Composition, identity, constants	0 ($\emptyset$)	None: equational
Diagonal lemma	0 ($\emptyset$)	None: same as recursion theorem
Post forward (decidable $\to$ RE $\wedge$ co-RE)	0 ($\emptyset$)	None: deterministic output routing

Theorem	Layer	¬¬ counit
Hierarchy separation (diagonal schema)	0 (∅)	None: diagonal construction
Myhill isomorphism	0 ({Quot.sound})	None: branching over Bool (decidable)
Reductio ↔ EM equivalence	0 (∅)	Meta-theorem: counit ↔ EM
Diaconescu circularity	0 (∅)	Meta-theorem: choice on stable types is redundant
Rice's theorem	1 (Markov)	Partial: ¬¬halts → halts
Kleene normal form (μ-minimization)	1 (Markov)	Partial: ¬¬∃y, T(e,x,y) → ∃y
Rogers isomorphism (totality)	1 (Markov)	Partial: ¬¬halts → halts
Dovetail totality ↔ EM	2 (EM)	Full: P(x) ∨ ¬P(x)
Post backward (RE ∧ co-RE → decidable)	2 (EM)	Full: which branch halts
Hierarchy properness (full)	2 (EM)	Full: subset over representation space
Rogers isomorphism (full)	2 (Classical)	Full: main project (classical stack)

Twenty results verified across all three layers by standalone file or main project formalization. Each layer has multiple verified exhibits. The Myhill result is particularly informative: it was initially predicted to require EM (back-and-forth branching appears to need excluded middle), but compiles at Layer 0 because every branch is over a computed value. The structural analysis (Section 2.2) correctly predicted this — the construction builds the bijection by identity-level operations — while the naive type-former prediction was wrong.

Open problems include: Friedberg-Muchnik (predicted Layer 2) and Church-Rosser (predicted Layer 0). Each prediction is falsifiable by the same standalone methodology.

7.4 Future directions

The constructive replacement program. Systematic `#print axioms` surveying of the main project identifies exactly three Mathlib definitions whose replacement with constructive alternatives would eliminate all infrastructure contamination: CategoryTheory.Closed, HasInitial, and Aesop.TreeSpec (the three channels identified in Section 5.8). This is an engineering task, not a mathematical one: the standalone files demonstrate that constructive alternatives exist and work.

Synthetic variants and the relativized boundary. Adding a parametric synthetic Church-Turing thesis ($\forall$ f : N → N, $\exists$ e, $\forall$ n, φ_e(n) = f(n)) as an explicit hypothesis in the standalone files would produce a second column in the classification table — the axiom profile under synthetic CT. Based on Forster's Coq results, the prediction: Rice, Kleene normal form (μ-minimization), and Rogers totality drop from Layer 1 (Markov) to Layer 0. Based on Forster's synthetic program [Forster 2021], we expect Post's backward direction and hierarchy properness to remain at EM even under synthetic CT, since dovetail totality requires logical determination rather than search termination. This would confirm that Markov is the boundary for search termination specifically, while EM is the boundary for logical determination — two structurally distinct gaps. The existing standalone infrastructure (RiceBoundary.lean, KleeneNormalForm.lean, RogersBoundary.lean) is the natural baseline: adding a `SyntheticCT` typeclass and re-running `#print axioms` would produce the comparison directly.

A deeper question: does the reflexive object D ≅ [D, D] with the four identity-loop equations already imply a form of synthetic universality? The reflexive object gives a universal self-applicator and fixed-point combinator, but not the full synthetic CT claim that ALL partial functions have indices. If it does, some Markov-level results might descend to ∅ without an additional axiom. If it does not, the gap between categorical reflexivity and synthetic CT is itself a measurable boundary.

Compression boundaries. ComplexityBoundary.lean (standalone, axiom profile ∅) adds a cost measure to the reflexive object and traces costs through the Y combinator construction. The result: the morphism

cost of omega(f) is linear in cost(f), bounded by universal constants from the reflexive structure (cost(fwd) + cost(bwd) + K_wL + K_unc + K_cur). The fixed-point construction is cheap. The complexity blowup lives in EVALUATION — iterating selfApp on inputs — where cost depends on the semantics of f, not on the morphism's construction cost. This formally separates construction cost (bounded, axiom profile $\emptyset$) from evaluation cost (potentially unbounded, input-dependent), and the separation tracks fold/unfold: fold builds the fixed point cheaply, unfold $\circ$ fold = selfApp evaluates it expensively. The immediate formal target is the bounded evaluator impossibility theorem: no uniform polynomial-time evaluator can realize arbitrary omega(f) behavior, which would bridge the axiom-profile framework to complexity-theoretic separations.

Extension to other theories. The standalone methodology and dependency- graph analysis apply to any formalized mathematical theory. Preliminary application to abstract algebra confirms replicability and reveals domain-sensitivity: core algebraic theorems (Bezout, CRT, Lagrange, Sylow) compile at Layer 0, while existence theorems requiring Zorn's lemma (maximal ideals, bases for infinite-dimensional spaces) require Layer 2. Strikingly, the Markov layer — the boundary between $\neg\neg\exists$ and $\exists$ that is central to computability — appears to be empty in algebra. This suggests the Markov gap is a feature of computation specifically, not of mathematical reasoning in general.

Polynomial Markov and complexity. The polynomial enrichment of the Layer 1 boundary has been shown to be equivalent to P = NP [Close 2026c]. Markov's principle restricted to polynomial predicates — irrefutable bounded existence implies polynomial findability — is the search form of P = NP. The axiom profile methodology thus connects directly to the central question of computational complexity.

---

8. Conclusion

The axiom profiles of computability theory's central theorems, when formalized from the equational theory of monoidal closed categories, partition into three layers that track the identity/equivalence transition within the fixed-point derivation. The first three levels of categorical structure (endofunctor algebra, Lambek convergence, containerization) are purely identity-based and entirely constructive. The fourth level (computation, where the object indexes its own endomorphisms) introduces equivalence, and the axiom boundary falls within it.

The partition is machine-verified by twenty Lean 4 files (nineteen standalone, one two-file cluster), corroborated by dependency-graph analysis, and tracks three regimes of the double-negation monad's counit — a refinement of the classical distinction between negation introduction and reductio ad absurdum.

The Y combinator, Kleene's recursion theorem, both Gödel incompleteness theorems, the s-m-n theorem, program composition, the diagonal lemma, hierarchy separation, and Myhill's isomorphism theorem compile with no counit — proofs operate directly on values. Rice's theorem, the Kleene normal form's μ-minimization, and Rogers' isomorphism totality require exactly Markov's principle — the partial counit that extracts a halting witness from $\neg\neg\exists$. Post's backward direction requires excluded middle — the full counit that resolves arbitrary propositions — and we prove the dovetail totality condition IS excluded middle, an equivalence, not merely an implication. Myhill's theorem, which branches extensively but compiles at Layer 0, confirms that the boundary tracks whether proofs operate on values or must extract witnesses from double negations, not which logical connective appears in the statement.

A formal analysis of Diaconescu's theorem reveals the circularity of choice: choice produces excluded middle only by operating on types whose structure depends on the proposition being decided. When those types are stable (decidable), choice is redundant — the answer is already available by computation. The non-constructive content of choice is not selection but treating undetermined structure as determined.

This is a measurement, not an interpretation.

Appendices

A. Formalization

The formalization relevant to this paper comprises 68 Lean 4 files (0 sorry): nineteen fully standalone files (zero imports), one two-file standalone cluster (ConstructiveGodelII extends ConstructiveGodelI), plus 48 Mathlib-dependent files implementing the categorical tower. The repository additionally contains nine standalone files for extensions beyond this paper's scope, bringing the total to 77 files. The project is available at:

- **Repository:** https://github.com/LarsenClose/lean4-fixed-point
- **Zenodo archive:** [Close 2026e] DOI: 10.5281/zenodo.18915083

The twenty files discussed in this paper (nineteen standalone, one two-file cluster):

File	Lines	Axiom profile	Section
`ConstructiveOmega.lean`	168	∅	3.1
`ConstructiveRecursionTheorem.lean`	181	∅	3.2
`ConstructiveHalting.lean`	161	∅	3.3
`ConstructiveGodelI.lean`	279	∅	3.4
`ConstructiveGodelII.lean`	—	∅	3.4
`ConstructiveSmn.lean`	229	∅	3.5
`ConstructiveComposition.lean`	237	∅	3.6
`ConstructiveDiagonalLemma.lean`	202	∅	3.7
`ConstructivePostForward.lean`	—	∅	3.8
`HierarchyProperness.lean`	146	∅	3.9
`MyhillBoundary.lean`	916	{Quot.sound}	3.10
`ContradictionModes.lean`	393	∅	5.1
`Conservativity.lean`	—	∅	5.2
`EvidenceModes.lean`	352	∅/{propext}	5.3
`RiceBoundary.lean`	471	{propext}+Markov	4
`KleeneNormalForm.lean`	153	Markov	4.6
`RogersBoundary.lean`	—	{propext}+Markov	4.6
`ClassicalPostBackward.lean`	473	∅+EM hyp	5.6
`Diaconescu.lean`	276	{propext}	5.7
`KripkeCountermodel.lean`	328	∅	5.7
`ModalityIndexedTypes.lean`	161	∅	5.7

Note: ConstructiveGodelII.lean imports ConstructiveGodelI.lean (itself standalone at ∅) and is not strictly standalone; all others have zero imports. Line counts marked "—" are omitted for brevity; all files are in the repository. The twenty-file count treats the ConstructiveGodelI/II cluster as a single unit; both are listed separately above for completeness.

The project additionally contains nine standalone files for work beyond this paper's scope (ComplexityBoundary, ComplexityReflexive, InvariantSubspace, InvariantSubspaceDeep, InvariantSubspaceOrbits, AntiCompression, EpiReflexive, OmegaAlgebra, ToyInstantiation), plus 48 Mathlib-dependent files implementing the categorical tower.

B. Complete axiom profile table

#	Result	Layer	Lean axioms	¬¬ counit	File
1	Y combinator	0	∅	None	ConstructiveOmega
2	Kleene recursion thm	0	∅	None	ConstructiveRecursionThm
3	s-m-n theorem	0	∅	None	ConstructiveSmn
4	Halting undecidability	0	∅	None	ConstructiveHalting

#	Result	Layer	Lean axioms	¬¬ counit	File
5	Gödel I	0	∅	None	ConstructiveGodelI
6	Gödel II	0	∅	None	ConstructiveGodelII
7	Composition, identity, constants	0	∅	None	ConstructiveComposition
8	Diagonal lemma	0	∅	None	ConstructiveDiagonalLemma
9	Post forward	0	∅	None	ConstructivePostForward
10	Hierarchy separation (diagonal schema)	0	∅	None	HierarchyProperness
11	Myhill isomorphism	0	{Quot.sound}	None	MyhillBoundary
12	Reductio ↔ EM	0	∅	Meta	ContradictionModes
13	Diaconescu circularity	0	∅	Meta	ModalityIndexedTypes
14	Rice's theorem	1	{propext}+Markov	Partial: $\neg\neg\exists \to \exists$	RiceBoundary
15	Kleene normal form (μ-min)	1	Markov	Partial: $\neg\neg\exists \to \exists$	KleeneNormalForm
16	Rogers isomorphism (totality)	1	{propext}+Markov	Partial: ¬¬halts→halts	RogersBoundary
17	Dovetail totality ↔ EM	2	∅+EM hyp	Full: $P \vee \neg P$	ClassicalPostBackward
18	Post backward	2	∅+EM hyp	Full: $P \vee \neg P$	ClassicalPostBackward
19	Hierarchy properness (full)	2	EM hyp	Full: ∀ representation	—
20	Rogers isomorphism (full)	2	Classical	Full	Main project

All Layer 0 and Layer 1 profiles verified by `#print axioms` in standalone files. Layer 2 profiles verified by explicit EM hypothesis in standalone files or by `#print axioms` in the main project.

C. Contamination tracing methodology

For declarations in the main project, `#print axioms` reports the transitive axiom closure including all Mathlib dependencies. Contamination is traced by applying `#print axioms` to intermediate definitions along suspected inheritance paths until the entry point of Classical.choice is isolated. The standalone files validate each identified entry point: replacing the Mathlib definition with a direct axiomatization and re-running `#print axioms` confirms the profile drops to the predicted level.

D. The categorical starting point

The formalization begins from the equational theory established in [Close 2026a], which proves the following in Lean 4 with zero sorry:

Four-level tower.

Level	Structure	Key equation
1. Dimension	Endofunctor M + initial object ⊥	Omega-chain $\bot \to M(\bot) \to M^2(\bot) \to \cdots$
2. Convergence	Colimit L of the chain	Lambek: $M(L) \cong L$
3. Closure	$M = \text{ihom}(A, -)$	$L \cong [A, L]$ with eval/name, β/η
4. Computation	$A = L$	$D \cong [D, D]$ — self-indexing

Substrate independence. The specification's dimension D = 1: any monoidal closed category with the requisite colimit produces the same equational theory. The Y combinator, recursion theorem, and all Layer 0 results are consequences of this equational theory, independent of the choice of ambient category.

Identity loop. Setting A = L in Level 4 yields four equations:

```
eval    = unfold(id)          identity unfolded
quine   = fold(selfApp)       self-application folded
```

```
fold ∘ unfold = id              constructive round-trip
unfold ∘ fold = selfApp         the self-application equation
```

Equations 1-3 are identity-based. Equation 4 produces selfApp, which generates both the Y combinator (every endomorphism has a fixed point) and the mode gap (extensional properties are undecidable). This asymmetry is the structural source of the axiom boundary measured in Sections 3-5.

For the full development, proofs, and Lean 4 verification, see [Close 2026a].

References

Bauer, A. (2006). First steps in synthetic computability theory. In *Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI)*, Electronic Notes in Theoretical Computer Science 155, 5–31.

Carneiro, M. (2019). The type theory of Lean. MSc thesis, Carnegie Mellon University.

Close, L. J. (2026a). The Point. DOI:10.5281/zenodo.18878239. Lean 4 formalization, 42 files.

Close, L. J. (2026c). Reflexive Compression Boundaries in Graded Categories. Manuscript.

Close, L. J. (2026e). Witness Extraction Asymmetry Across Logic, Complexity, and Fixed-Point Mathematics. Lean 4 formalization, 93 files. DOI:10.5281/zenodo.18915083.

Coquand, T. and Werner, B. (1997). Constructive type theory does not prove excluded middle. Unpublished note.

Forster, Y. (2021). Computability in constructive type theory. PhD thesis, Saarland University.

Forster, Y., Kirst, D., and Smolka, G. (2019). On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '19)*, 38–51.

Hyland, J. M. E. (1982). The effective topos. In *The L.E.J. Brouwer Centenary Symposium*, Studies in Logic and the Foundations of Mathematics 110, 165–216.

Kleene, S. C. (1945). On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic* 10(4), 109–124.

Kleene, S. C. and Vesley, R. E. (1965). *The Foundations of Intuitionistic Mathematics*. North-Holland.