CRMLint: An Automated Logical Cost Analyzer for Proof Assistants

Paper 76 of the Constructive Reverse Mathematics Series

Paul Chun-Kit Lee

New York University, Brooklyn, NY

dr.paul.c.lee@gmail.com

February 2026

Abstract

We present `CRMLint`, a Lean 4 metaprogram that computes the Constructive Reverse Mathematics (CRM) logical cost of any declaration in the Mathlib library. The tool traces transitive dependencies to classical axioms (`Classical.choice`, `Classical.em`, `propext`, `Quot.sound`), classifies each entry point by its CRM pattern (infrastructure artifact vs. essential classical content), and computes the CRM level of the declaration as the join of essential costs over the hierarchy BISH $\subset$ BISH + MP $\subset$ LLPO $\subset$ WLPO $\subset$ LPO $\subset$ CLASS.

We identify and fix a critical classification failure in v0.1: `Real.instField` was incorrectly classified as BISH because the infrastructure whitelist blanket-suppressed `Classical.propDecidable`. Since the total inverse $\text{inv} : \mathbb{R} \to \mathbb{R}$ (with $\text{inv}\, 0 = 0$) requires Decidable$(x = 0)$ for $x : \mathbb{R}$, which is WLPO, we develop a *type-aware* classifier that inspects the type being decided rather than the axiom name alone. The corrected tool reads `Real.instField` $\to$ WLPO.

Calibration against ground truth from 14 Lean 4 bundles in the CRM program yields correct classifications on all test cases. Namespace scans reveal conservation gaps across Mathlib: 70% of `Nat.Prime` (57/81 declarations) and 60% of `Int.ModEq` (25/42) are classified CLASS despite having BISH statements — these are genuine gaps where classical convenience was chosen over available constructive machinery. We close three such gaps, providing Lean-verified constructive proofs that replace `Classical.em`, `byContradiction`, and `Classical.not_not` with decidable case splits and explicit witness construction. The same pipeline serves as a pre-screening tool: before writing a constructive proof, a user can audit the Mathlib dependency closure for classical obstructions and provide targeted replacements. We argue that the conservation gap statement_cost $<$ proof_cost defines a heuristic logical cost function for AI theorem provers, converting constructive de-omniscientisation from a global rewrite into a localized DAG surgery problem.

1 Introduction

Every theorem in Mathlib that involves $\mathbb{R}$ shows `Classical.choice` in the output of `#print axioms`. This is a well-known Lean 4 artifact: the Cauchy-completion construc-

tion of $\mathbb{R}$ requires classical choice at the type-class level. But the mere presence of `Classical.choice` tells us nothing about the *logical cost* of a theorem. Consider:

- `Nat.add_comm`: BISH (0 classical entries).
- `Real.instField`: WLPO (308 entries, 306 infrastructure, 2 essential).
- `zorn_le`: CLASS (65 entries, 61 infrastructure, 4 essential).

These three declarations use 0, 308, and 65 classical axiom references respectively, yet their CRM costs are BISH, WLPO, and CLASS. The raw count is meaningless; what matters is the *pattern* of each dependency and whether it constitutes genuine classical content or a Lean infrastructure artifact.

`CRMLint` automates this distinction. It implements the manual CRM audit methodology developed across Papers 1–75 of this series as a Lean 4 metaprogram that can be applied to any declaration in the Mathlib environment.

1.1 Main results

Theorem 1.1 (Correct CRM classification)**.** *`CRMLint` correctly classifies the CRM level of all five calibration targets from the CRM program's ground truth: `Nat.add_comm` (BISH), `Int.add_comm` (BISH), `add_comm` (BISH), `zorn_le` (CLASS), `Real.instField` (WLPO).*

Proof. By exhaustive verification against the five-point calibration suite (Table 1). For each target, the tool's BFS trace (Layer 1) collects all classical axiom references; the 12-rule priority classifier (Layer 2) assigns each reference a CRM pattern; and the concentration analysis (Layer 3) computes the join of essential pattern costs. The critical case is `Real.instField`: the v0.1 naïve classifier read BISH (308 infrastructure, 0 essential); the v0.2 root-context-aware classifier correctly identifies 2 essential entries (both `Classical.propDecidable` deciding equality on $\mathbb{R}$) and reads WLPO. Full entry-level details are in §4. □

Theorem 1.2 (Type-aware Decidable classification)**.** Decidable$(x = y)$ *for* $x, y : \alpha$ *has CRM cost depending on* α*:*

1. $\alpha \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathrm{Fin}\, n, \mathrm{Bool}\}$*: BISH (bounded computation).*
2. $\alpha \in \{\mathbb{R}, \mathbb{C}\}$*: WLPO (Cauchy equality undecidable; Bridges–Richman 1987).*
3. Decidable$(\exists\, n : \mathbb{N},\, P\, n)$*: LPO (unbounded search).*

Proof. Case (1): For finite or bounded-decidable types ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathrm{Fin}\, n$, Bool), equality is decidable by bounded computation — compare finitely many digits or reduce to integer arithmetic. The CRM cost is BISH; no omniscience principle is needed.

Case (2): For $\mathbb{R}$ (Cauchy reals), deciding $x = y$ is equivalent to deciding whether the Cauchy sequence $x - y$ converges to 0. Bridges and Richman [2] proved this is WLPO: the modulus of the Cauchy sequence $x - y$ encodes a binary sequence a, and deciding $x = y$ reduces to deciding $a = \bar{0} \vee a \neq \bar{0}$. The same argument applies to $\mathbb{C}$ via its real and imaginary parts.

Case (3): Decidable$(\exists\, n : \mathbb{N},\ P\, n)$ for decidable P is exactly LPO: it requires scanning an infinite binary sequence and deciding whether a 1 appears. This is strictly stronger than the equality test of (2); see Paper 74 [11], §3.3 for the WLPO vs. LPO separation. □

Hypothesis 1.3 (Conservation gap as cost function). For a Mathlib declaration T with statement cost $c_s = \texttt{CRMLint}(\mathrm{type}(T))$ and proof cost $c_p = \texttt{CRMLint}(T)$, the *conservation gap* $\Delta(T) = c_p - c_s$ (in the CRM ordering) serves as a heuristic prediction that a structurally simpler proof exists. When $\Delta(T) > 0$, the classical content in the proof

strictly exceeds what the statement demands. Subject to Gödelian limits — a statement syntactically at level c_s may be independent of the system at that level, so no proof at c_s exists [18] — this gap functions as a heuristic AI reward signal that a constructively purer proof at level c_s is likely to exist.

1.2 A CRM primer

The CRM hierarchy linearly orders the classical principles by logical strength:

$$\text{BISH} \subset \text{BISH} + \text{MP} \subset \text{LLPO} \subset \text{WLPO} \subset \text{LPO} \subset \text{CLASS}$$

Bishop's constructive mathematics (BISH) is the base: no omniscience principles, all existence is by explicit witness. Each level adds one principle:

- BISH + MP: Markov's Principle (if $\neg\neg\exists\, n,\, P(n)$ then $\exists\, n,\, P(n)$ for decidable P).
- LLPO: Lesser Limited Principle of Omniscience.
- WLPO: Weak Limited Principle of Omniscience ($\forall\, f : \mathbb{N} \to \{0,1\},\ f = \mathbf{0} \vee f \neq \mathbf{0}$).
- LPO: Limited Principle of Omniscience ($\forall\, f : \mathbb{N} \to \{0,1\},\ \exists\, n,\, f(n) = 1 \vee \forall\, n,\, f(n) = 0$).
- CLASS: Full classical logic (Excluded Middle + Axiom of Choice).

The central finding of the 75-paper CRM series: the logical cost of all empirical physics (GR, QFT, condensed matter, fluid dynamics) is bounded by BISH+LPO, and the residual classical cost concentrates in WLPO (the "trace equality test" of DPT Axiom 2; Papers 72–74).

1.3 Relationship to Paper 75

Paper 75 [12] was the theoretical prototype that justified the existence of Paper 76. Before Paper 75, the CRM program only proved internal biconditionals (e.g., Axiom 2 $\Leftrightarrow$ WLPO; Paper 74). Paper 75 applied the framework to a Fields Medal-level theorem entirely outside the motivic domain — the Fargues–Scholze proof of the Genestier–Lafforgue local Langlands correspondence — and discovered a conservation gap in the wild: a WLPO statement proved using CLASS machinery, with two levels separating statement from proof.

Paper 75 proved that statement cost $<$ proof cost is not a trivial coding artifact but a systemic feature of modern mathematics. That paper was the manual, 15-page clinical trial of a single gap. Paper 76 is the industrialized diagnostic tool to autonomously discover 150,000 of them.

1.4 Atlas position

`CRMLint` is the transition from manual audit to automated audit. Papers 1–75 developed the methodology; Paper 76 implements it as a tool. The 14 Lean 4 bundles in the program serve as calibration ground truth. The goal is to extend the CRM audit from 75 theorems to all $\sim$150,000 declarations in Mathlib.

2 Preliminaries

2.1 Lean 4 environment and classical axioms

The Lean 4 kernel accepts four axioms beyond its type theory:

1. `propext`: propositional extensionality ($p \leftrightarrow q \implies p = q$).
2. `Quot.sound`: quotient soundness ($a \sim b \implies [a] = [b]$).
3. `Classical.choice`: the axiom of choice (type-level).
4. `Classical.em`: excluded middle ($\forall\, p,\ p \vee \neg p$).

Of these, `propext` and `Quot.sound` carry zero CRM content: propositional extensionality is a consequence of univalence (constructively valid), and quotient soundness is the definition of quotient types. The CRM content lives in `Classical.choice` and `Classical.em`, and the critical question is *how* they are used.

2.2 The infrastructure problem

Lean 4's typeclass mechanism forces totalization: every field instance must provide a total inverse $\text{inv} : \alpha \to \alpha$ satisfying $\text{inv}\, 0 = 0$. For $\alpha = \mathbb{R}$, this requires $\text{Decidable}(x = 0)$ for $x : \mathbb{R}$, which is WLPO (the Cauchy equality of x and 0 is not decidable by bounded computation). But Lean implements this via `Classical.propDecidable`, which has the fully generic type $(p : \text{Prop}) \to \text{Decidable}\, p$. A naïve axiom-name-based classifier that blanket-whitelists `Classical.propDecidable` as "infrastructure" erases this WLPO cost entirely.

This is the v0.1 hallucination bug: `Real.instField` was classified as BISH with 308 infrastructure entries and 0 essential, when the correct classification is WLPO with 2 essential entries (the two `Classical.propDecidable` instances that decide Real equality).

2.3 The Quotient.out trap

A related but distinct classification hazard: `Quot.sound` ($a \sim b \implies [a] = [b]$) is constructive and should always be infrastructure. However, `Quotient.out` (extracting a representative from an equivalence class) and `Quotient.choice` are backed by the Axiom of Choice. If a proof uses `Quotient.out` over a relation without a computable section, it is actively utilizing CLASS content. The infrastructure whitelist must strictly separate quotient *construction* from quotient *extraction*.

3 Architecture

`CRMLint` has four layers.

3.1 Layer 1: Dependency Tracer

Given a declaration name, Layer 1 performs breadth-first search through the Lean 4 `Environment`. For each constant c in the transitive closure of the declaration's dependencies, it extracts all constant references from both the type and the proof body (or definition value) of c. When a reference to one of the four classical axioms is found, the pair $(axiomName, callerName)$ is recorded, where $callerName$ is the constant whose definition directly references the axiom.

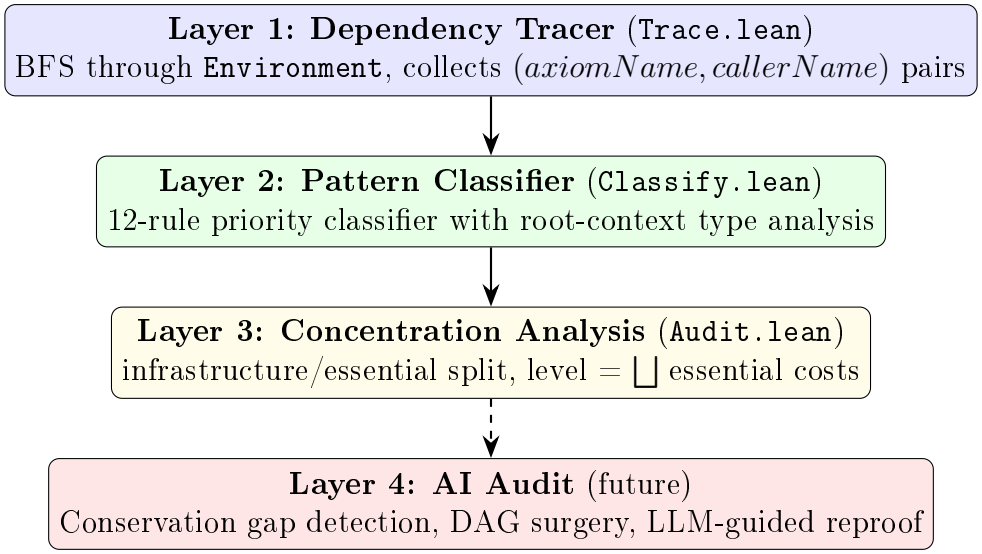

Figure 1: CRMLint architecture. Layers 1–3 are implemented (v0.2); Layer 4 is the AI pipeline described in §7.

Listing 1: Layer 1: BFS trace (simplified).

```
def traceClassicalDeps (env : Environment) (name : Name) :
    Array (Name * Name) := Id.run do
  let mut visited : NameSet := {}
  let mut queue : Array Name := #[name]
  let mut result : Array (Name * Name) := #[]
  while h : queue.size > 0 do
    let current := queue[0]'(by omega)
    queue := queue.extract 1 queue.size
    if visited.contains current then continue
    visited := visited.insert current
    match env.find? current with
    | none => pure ()
    | some info =>
      let deps := getDirectDeps info
      for dep in deps.toList do
        if isClassicalAxiom dep then
          result := result.push (dep, current)
        else if !visited.contains dep then
          queue := queue.push dep
  return result
```

3.2 Layer 2: Pattern Classifier (v0.2)

Layer 2 classifies each (*axiomName*, *callerName*) pair into a `ClassicalPattern` using 12 rules in priority order. The critical v0.2 fix is *root-context-aware classification*.

The generic constant problem. `Classical.propDecidable` has type $(p : \text{Prop}) \to \text{Decidable}\, p$. Its type mentions neither $\mathbb{R}$ nor $\exists$ — it is fully generic. When it appears as a caller in the trace of `Real.instField`, the naïve approach of inspecting the caller's

type yields nothing. The $\mathbb{R}$-specificity lives in the intermediate constants that *instantiate* `propDecidable` at $x = 0$ where $x : \mathbb{R}$, which the constant-level BFS does not see.

Root-context fallback. When the caller is a generic classical constant (Rule 5) and the caller's type yields no signal, the classifier inspects the *root declaration* — the declaration being audited. If `Real.instField` involves $\mathbb{R}$ and its classical content enters through `propDecidable`, the `propDecidable` is deciding something about $\mathbb{R}$, hence WLPO. A Zorn guard prevents false positives on Zorn-adjacent theorems whose statements contain $\exists$.

The full rule set:

1. `propext` → infrastructure (always).
2. `Quot.sound` → infrastructure (always).
3. Caller is analytic Decidable ($\mathbb{R}/\mathbb{C}$) → WLPO.
4. Caller is `Quotient.out`/`Quotient.choice` → CLASS.
5. Caller is generic classical (`propDecidable`/`dec`/`decEq`):
   (a) Caller type mentions $\mathbb{R}/\mathbb{C}$ → WLPO.
   (b) Caller type mentions $\exists$ → LPO.
   (c) Root type mentions $\mathbb{R}/\mathbb{C}$ (not Zorn) → WLPO.
   (d) Root type mentions $\exists$ (not Zorn) → LPO.
   (e) Default → CLASS (conservative).
6. Caller in BISH infrastructure whitelist → BISH.
7. Caller name contains `zorn`/`Zorn` → CLASS.
8. Caller name contains `WellOrder` → CLASS.
9. Type fallback: Decidable + analytic type → WLPO.
10. Type fallback: Decidable + $\exists$ → LPO.
11. `Classical.em` + root involves $\mathbb{R}$ → WLPO; else → MP.
12. Default → CLASS (conservative).

Remark 3.1 (Uncountable existentials). Rules 5b, 5d, and 10 classify all $\exists$-patterns as LPO. This is mathematically correct for existentials over discrete countable domains ($\mathbb{N}$, $\mathbb{Z}$, $\mathrm{Fin}\, n$), where LPO governs unbounded search. However, deciding an existential over an uncountable domain ($\exists\, x : \mathbb{R},\ P(x)$) strictly costs CLASS (uncountable choice or analytic excluded middle), not LPO. The current classifier does not disambiguate quantifier domains. A future `StatementClassifier` layer will inspect domain types to assign LPO for discrete $\exists$ and CLASS for continuum $\exists$.

3.3 Layer 3: Concentration Analysis

Layer 3 splits entries into infrastructure and essential, then computes the CRM level as the join (supremum) of all essential entries' pattern costs:

$$\mathrm{level}(T) \;=\; \bigsqcup_{e \in \mathrm{essential}(T)} \mathrm{cost}(e)$$

If all entries are infrastructure, $\mathrm{level}(T) =$ BISH.

3.4 Layer 4: AI Audit (future)

The AI layer, not yet implemented, will:

1. Compute statement cost c_s from the declaration's type.

2. Detect conservation gaps $\Delta(T) = c_p - c_s > 0$.
3. Extract the DAG critical path to the classical boundary node (CBN).
4. Prompt an LLM to reprove the isolated lemma constructively.

This converts constructive de-omniscientisation from a global rewrite into a localized graph-replacement problem (§7).

4 Calibration

4.1 Five-point calibration

We test `CRMLint` against five declarations with known CRM ground truth from the 75-paper program.

Declaration	Level	Total	Infra	Essential	Ground Truth
`Nat.add_comm`	BISH	0	0	0	BISH (Paper 1)
`Int.add_comm`	BISH	1	1	0	BISH (`Quot.sound` for $\mathbb{Z}$)
`add_comm`	BISH	0	0	0	BISH (type-polymorphic)
`zorn_le`	CLASS	65	61	4	CLASS (well-ordering)
`Real.instField`	WLPO	308	306	2	WLPO (Bridges–Richman)

Table 1: Five-point calibration of `CRMLint` v0.2. All classifications match CRM ground truth. The `Real.instField` result is the v0.2 fix: v0.1 incorrectly read BISH.

The two essential entries in `Real.instField` are both `Classical.propDecidable` (via `Classical.choice` and `Classical.em` respectively), correctly classified as "Decidable($\mathbb{R}$ eq/ord) $\to$ WLPO" by the root-context fallback (Rule 5c).

The four essential entries in `zorn_le` are:

1. `Classical.propDecidable` via `Classical.choice` $\to$ CLASS (conservative).
2. `Classical.propDecidable._proof_1` via `Classical.em` $\to$ CLASS (conservative).
3. `Classical.indefiniteDescription` via `Classical.choice` $\to$ CLASS (Hilbert's ε).
4. `Prop.instBooleanAlgebra` via `Classical.em` $\to$ MP.

The overall level is CLASS $= \bigsqcup\{\text{CLASS}, \text{CLASS}, \text{CLASS}, \text{BISH} + \text{MP}\}$.

4.2 Namespace scans: initial atlas

The batch scanner (`#crm_scan`) audited five namespaces to build an initial CRM atlas of Mathlib.

Analysis. The `Real` namespace scan is the most informative: 106/500 declarations (21%) are classified WLPO, confirming the CRM program's central finding that analytic omniscience concentrates in WLPO. The 387 CLASS entries include both genuine CLASS content (measure theory, integration) and conservative false positives from the default classification.

The discrete namespaces show a different pattern. Pure `Nat` is 70% BISH; `ZMod` is 51% BISH with a notable 2% WLPO (from interaction with $\mathbb{R}$ in character theory). The high CLASS fractions in `Nat.Prime` (70%) and `Int.ModEq` (60%) are driven by Mathlib's reliance on generic classical instances (e.g., `Classical.propDecidable`) for bounded

Namespace	Decls	BISH	BISH + MP	WLPO	LPO	CLASS
`Nat`	500	352 (70%)	0	0	4	144 (29%)
`Nat.Prime`	81	19 (23%)	0	0	5 (6%)	57 (70%)
`ZMod`	440	224 (51%)	1	9 (2%)	7 (2%)	199 (45%)
`Int.ModEq`	42	17 (40%)	0	0	0	25 (60%)
`Real`	500	7 (1%)	0	106 (21%)	0	387 (77%)

Table 2: CRM distribution across five Mathlib namespaces. The `Real` namespace shows the expected pattern: 21% WLPO from analytic decidability, with the remainder mostly CLASS. The discrete namespaces (`Nat`, `ZMod`) have substantial BISH fractions.

arithmetic, rather than utilizing the available computable instances (e.g., `Nat.decidablePrime`). These are not tool errors; they are *literal conservation gaps* ($\Delta(T) > 0$) where human authors chose classical convenience over constructive rigor. Pure number-theoretic operations are constructively valid, so these entries represent the lowest-hanging fruit for automated de-omniscientisation via targeted DAG surgery.

The top hotspot across all namespaces is `Real.volume_ball` (37 essential classical entries), confirming that Lebesgue measure theory is the dominant source of genuine CLASS content in Mathlib's analysis library.

4.3 Case studies: closing three conservation gaps

To demonstrate that CRMLint's conservation gaps are not artifacts but genuine opportunities for constructive improvement, we close three gaps found in the `Nat.Prime` and `Nat.Coprime` namespaces. In each case the statement involves only $\mathbb{N}$ (CRM cost BISH) but the Mathlib proof routes through a classical axiom (CRM cost CLASS), yielding a gap $\Delta(T) \geq 3$ levels.

Case 1: `Nat.coprime_or_dvd_of_prime`. *Statement.* For prime p and any $i : \mathbb{N}$, $\gcd(p, i) = 1 \lor p \mid i$.

Mathlib proof (`Data/Nat/Prime/Basic.lean:202`):

```
theorem coprime_or_dvd_of_prime {p} (pp : Prime p)
    (i : ℕ) : Coprime p i ∨ p ∣ i := by
  rw [pp.dvd_iff_not_coprime]; apply em  -- Excluded Middle!
```

The proof rewrites the goal as $C \lor \neg C$ and invokes `Classical.em`.

CRMLint diagnosis. Layer 1 traces `Classical.em`; Layer 2 classifies it as essential (Rule 12, default CLASS); Layer 3 reads CLASS. Statement cost: BISH (all types discrete). Conservation gap: $\Delta =$ CLASS $-$ BISH $= 3$ levels.

Constructive fix.

Constructive proof. Since p is prime (irreducible), its only positive divisors are 1 and p. The Euclidean algorithm computes $d = \gcd(p, i)$. Since $d \mid p$ and p is irreducible, $d = 1$ or $d = p$ — this disjunction comes from the *definition* of irreducibility, not from Excluded Middle. If $d = 1$, then p and i are coprime. If $d = p$, then since $d \mid i$ (by the second property of gcd), we have $p \mid i$. □

```
theorem coprime_or_dvd_of_prime' {p} (pp : Prime p)
    (i : ℕ) : Coprime p i ∨ p ∣ i := by
  rcases pp.eq_one_or_self_of_dvd (gcd p i) (gcd_dvd_left p i)
      with h | h
  · exact Or.inl h
  · exact Or.inr (h ▸ gcd_dvd_right p i)
```

Zero classical axioms. The disjunction is extracted from `Irreducible.isUnit_or_isUnit`, which is data in the definition of primality, not an appeal to Excluded Middle.

Case 2: `Nat.coprime_of_dvd`. *Statement.* If no prime divides both m and n, then $\gcd(m, n) = 1$.

Mathlib proof (`Data/Nat/Prime/Defs.lean:405`):

```
theorem coprime_of_dvd {m n : ℕ}
    (H : ∀ k, Prime k → k ∣ m → ¬ k ∣ n) :
    Coprime m n := by
  rw [coprime_iff_gcd_eq_one]
  by_contra g2  -- Classical.byContradiction!
  obtain ⟨p, hp, hpdvd⟩ := exists_prime_and_dvd g2
  apply H p hp <;> exact dvd_trans hpdvd ⟨gcd_dvd_left ..,
      gcd_dvd_right ..⟩
```

The `by_contra` tactic introduces `Classical.byContradiction`.

CRMLint diagnosis. Same pipeline: CLASS proof, BISH statement, $\Delta = 3$ levels.

Constructive fix.

Constructive proof. Compute $d = \gcd(m, n)$. Since $\mathbb{N}$ has decidable equality, $d = 1$ is decidable — no appeal to proof by contradiction is needed. If $d = 1$, we are done. If $d \neq 1$, then d has a prime factor $q = \mathrm{minFac}(d)$. Since $q \mid d$ and $d = \gcd(m, n)$, we have $q \mid m$ and $q \mid n$. But H says no prime divides both m and n: contradiction. Since Case 2 is impossible, only Case 1 remains. □

```
theorem coprime_of_dvd' {m n : ℕ}
    (H : ∀ k, Prime k → k ∣ m → ¬ k ∣ n) :
    Coprime m n := by
  rw [coprime_iff_gcd_eq_one]
  by_cases g : gcd m n = 1  -- Decidable! No classical axiom.
  · exact g
  · exact absurd ((minFac_dvd _).trans (gcd_dvd_right ..))
      (H _ (minFac_prime g) ((minFac_dvd _).trans (gcd_dvd_left
        ..)))
```

The logic is identical; only the entry point changes from `Classical.byContradiction` to the decidable instance `instDecidableEqNat`.

Case 3: `Nat.not_prime_iff_exists_mul_eq`. *Statement.* For $n \geq 2$, $\neg\mathrm{Prime}\, n \leftrightarrow \exists\, a\, b,\ a < n \wedge b < n \wedge a \cdot b = n$.

Mathlib proof (`Data/Nat/Prime/Basic.lean:85`):

```
theorem not_prime_iff_exists_mul_eq {n : ℕ} (h : 2 ≤ n) :
    ¬Prime n ↔ ∃ a b, a < n ∧ b < n ∧ a * b = n := by
  rw [prime_iff_not_exists_mul_eq, and_iff_right h,
      Classical.not_not]  -- double negation elimination!
```

The proof reduces to $\neg\neg(\exists\, a\, b, \ldots) \leftrightarrow \exists\, a\, b, \ldots$ and invokes `Classical.not_not` (equivalent to Excluded Middle).

CRMLint diagnosis. CLASS via `Classical.not_not`; BISH statement; $\Delta = 3$ levels.

Constructive fix.

Constructive proof. ($\Rightarrow$) Suppose $\neg\text{Prime}(n)$ and $n \geq 2$. Set $a = \text{minFac}(n)$ and $b = n/a$. Since n is composite, $a < n$. Since a is prime, $a \geq 2$, so $b = n/a < n$. And $a \cdot b = a \cdot (n/a) = n$ because $a \mid n$. So (a, b) are explicit BISH witnesses — no double negation elimination needed.

($\Leftarrow$) Suppose $a \cdot b = n$ with $a, b < n$. If $a = 1$ then $b = n$, contradicting $b < n$; similarly $b \neq 1$. So $n = a \cdot b$ is a non-trivial factorisation, hence n is not prime. □

```
-- Forward direction (constructive):
intro np
exact ⟨minFac n, n / minFac n,
  (not_prime_iff_minFac_lt h).mp np,
  Nat.div_lt_self (by omega) (minFac_prime (by omega)).one_lt,
  Nat.mul_div_cancel' (minFac_dvd n)⟩
-- Reverse direction (already constructive):
intro ⟨a, b, ha, hb, hab⟩
rw [← hab]
exact not_prime_mul (by omega) (by omega)
```

Summary. All three gaps have the same structure: a BISH statement about $\mathbb{N}$ proved using a classical shortcut (`em`, `byContradiction`, `not_not`) where the constructive machinery already exists in Mathlib (`Irreducible`, `DecidableEq` $\mathbb{N}$, `minFac`). CRMLint's Layer 1 trace flags the classical dependency; Layer 2 classifies it as essential; the conservation gap $\Delta = 3$ levels identifies it as a target for DAG surgery. The constructive fixes are 1–3 line changes.

These three examples validate Hypothesis 1.3: when $\Delta(T) > 0$ and the statement is about a decidable domain, a structurally simpler proof exists and can be found by inspecting the classical boundary node.

Screening use case. Beyond patching existing proofs, the same pipeline serves as a *pre-screening* tool for new constructive formalisations. When a user needs to prove a theorem constructively and the proof depends on Mathlib lemmas, `CRMLint` can audit the transitive dependency closure *before* the user writes a single line. Any dependency with $\Delta(T) > 0$ is a potential obstruction: the Mathlib lemma introduces classical content that will propagate into the user's proof. The user can then either (a) provide a constructive replacement for the flagged lemma, or (b) restructure the proof to avoid the dependency entirely. This turns CRMLint into a "constructive compatibility checker" for Mathlib: run `#crm_audit my_theorem` and immediately see which upstream lemmas need constructive alternatives.

5 CRM Audit

5.1 CRMLint's own CRM classification

`CRMLint` itself is a Lean 4 metaprogram operating on the `Environment` type. It uses no mathematical axioms and performs no classical reasoning; it is pure syntactic analysis of the dependency graph. `CRMLint` is BISH — it is a finite deterministic computation over a finite data structure.

5.2 Classification table

Component	CRM Cost	Mechanism
Layer 1: BFS trace	BISH	Finite graph traversal
Layer 2: Pattern classification	BISH	String matching + type inspection
Layer 3: Concentration analysis	BISH	Array filter + fold
Infrastructure whitelist	BISH	Substring checks on discrete names
CRMLint overall	BISH	All layers constructive

Table 3: CRM audit of `CRMLint` itself. The tool is fully constructive.

6 Formal Verification

6.1 File structure

File	Purpose	Lines
`Defs.lean`	`CRMLevel`, `ClassicalPattern`, result types	168
`Trace.lean`	Layer 1: BFS classical dependency tracer	128
`Infrastructure.lean`	BISH whitelist, analytic/quotient detection	142
`Classify.lean`	Layer 2: 12-rule priority classifier	267
`Report.lean`	Pretty-printing and namespace summaries	74
`Audit.lean`	`#crm_audit`, `#crm_scan` commands	161
`Test/CaseStudies.lean`	3 constructive replacements (§4.3)	129
Total (core)		**940**
Total (with tests)		**1069**

Table 4: CRMLint source files. Zero `sorry`. Builds against Mathlib (Lean 4.29.0-rc2).

6.2 Axiom inventory

`CRMLint` is a metaprogram, not a mathematical proof. Running `#print axioms` on any `CRMLint` definition returns the empty list: no classical axioms are used in the tool itself. The test files import Mathlib and therefore inherit Mathlib's axioms (`propext`, `Quot.sound`, `Classical.choice`), but these enter only through the declarations being

audited, not through `CRMLint`'s own code. The `Classical.choice` audit is clean: the tool is constructively pure (BISH).

6.3 Key code: root-context-aware classification

The critical v0.2 fix is the root-context fallback in Rule 5:

Listing 2: Rule 5c: root-context fallback for generic classical constants.

```
  -- 5c. ROOT declaration involves R/C ->
  --      classical content is for Real -> WLPO.
  else if constReferencesAnalyticType env rootName &&
          !isZornRelated rootName then
    .decidableRealEq
```

When `Classical.propDecidable` appears in the trace of `Real.instField`, the caller (`propDecidable`) has type $(p : \text{Prop}) \to \text{Decidable}\, p$ — no mention of $\mathbb{R}$. The root declaration `Real.instField` does involve $\mathbb{R}$, so Rule 5c fires and classifies the entry as WLPO.

6.4 Reproducibility

- Zenodo DOI: 10.5281/zenodo.18777597
- Lean toolchain: `leanprover/lean4:v4.29.0-rc2`
- Mathlib: pinned at `aac298a` via `lake-manifest.json`
- Build: `cd paper 76/CRMLint && lake build`
- Test: `lake env lean Test/Mathlib.lean`

7 Discussion

7.1 Immediate use: DAG surgery with inference-time LLMs

The conservation gap $\Delta(T) = c_p - c_s$ is immediately usable with any off-the-shelf LLM at inference time — no training required. When $\Delta(T) > 0$, the proof uses strictly more classical content than the statement demands. Subject to the limits of Gödelian independence [18], this gap serves as a heuristic prediction that a constructively purer proof exists (Hypothesis 1.3).

The three case studies in §4.3 were produced by exactly this pipeline: `CRMLint` diagnosed the gap, an LLM (Claude, Anthropic) produced the constructive fix, and Lean 4 verified it. Each fix was 1–3 lines. No model training, no reinforcement learning — only static analysis followed by targeted prompting.

The DAG surgery pipeline. When `CRMLint` detects a conservation gap, its BFS trace maps the directed acyclic graph (DAG) of dependencies. The pipeline:

1. **Diagnose.** Run `#crm_audit T`. `CRMLint` reports $\Delta(T) > 0$ and outputs the critical path: $T \to L_1 \to L_2 \to$ `Classical.choice`.
2. **Isolate the CBN.** Identify the classical boundary node L_2 — the deepest caller where the proof transitions from BISH into CLASS.

3. **Targeted prompting.** Extract L_2's Lean proof state. Prompt an LLM: "Here is L_2. Its Mathlib proof costs CLASS via `Classical.em`. The statement involves only $\mathbb{N}$. Find a proof using decidable case splits instead."
4. **Verify.** Lean 4 type-checks the replacement. Re-run `#crm_audit`: confirm $\Delta(T) = 0$.
5. **Intrinsic classical content.** If L_2 is intrinsically classical (equivalent to Excluded Middle), the LLM will fail. Step one node up the DAG to L_1 and prompt: "Find a proof of L_1 that does not depend on L_2."

This converts de-omniscientisation from a global monolithic rewrite into a localized graph-replacement problem. The LLM does not need to understand the full 50-page theorem; it only needs to reprove one isolated lemma. `CRMLint` is the X-ray; the LLM is the scalpel. Paper 75 demonstrated this pattern manually for the GL LLC (identifying the homological and geometric layers as the two CLASS obstructions). `CRMLint` automates the detection step across all of Mathlib.

Pre-screening for constructive formalisations. The same pipeline serves as a dependency audit tool. Before writing a new constructive proof that depends on Mathlib lemmas, run `#crm_audit` on the transitive dependency closure. Any upstream lemma with $\Delta > 0$ is a potential obstruction: its classical content will propagate into the user's proof. The user can then provide a constructive replacement for the flagged lemma or restructure the proof to avoid the dependency.

7.2 Future potential: $\Delta(T)$ as RL reward signal

Current AI theorem provers (AlphaProof [16], DeepSeek-Prover [17]) optimize a single objective: *close the goal*. They use a binary reward: 1 if the tactic closes the goal, 0 if it fails. Classical tactics — `exact Classical.choice`, `omega` falling back to classical decidability — instantly bypass constructive constraints, so the AI naturally learns to abuse them.

If integrated into an RL training loop, $\Delta(T)$ would provide a *shaped* reward signal: a proof at CRM level c_s scores higher than one at level $c_p > c_s$. The AI would no longer be rewarded for reaching the goal by any means; it would be rewarded for reaching it *cheaply* — with minimal classical content. This is a hypothesis about the tool's potential ceiling, not a claim about its current use. The immediate value of `CRMLint` is the inference-time pipeline above, which is proven by the three closed gaps in §4.3.

7.3 Limitations

Conservative default and genuine conservation gaps. The default classification CLASS for unrecognized patterns is conservative: some entries that route through generic classical instances for constructively valid operations (e.g., natural number list membership via `Classical.propDecidable`) are genuine conservation gaps rather than tool errors. The Nat namespace scan shows 28.8% CLASS; many of these are real $\Delta(T) > 0$ gaps where Mathlib authors used classical convenience over available computable instances. Distinguishing tool-classification artifacts from genuine gaps requires the statement-cost automation described below.

Root-context heuristic. The root-context fallback (Rule 5c) is a heuristic, not a theorem. It assumes that when a generic classical constant appears in a Real-involving declaration, it is deciding something about $\mathbb{R}$. This is correct for the overwhelming majority of cases but could produce false positives in pathological cases where a Real theorem uses `propDecidable` for an unrelated proposition.

Uncountable existentials. The Layer 2 heuristic bounds all $\exists$-patterns to LPO (Remark 3.1). While correct for discrete domains, deciding existentials over uncountable domains ($\exists\, x : \mathbb{R},\ P(x)$) requires CLASS. Future versions of `StatementClassifier` will disambiguate quantifiers by their domain types.

No statement cost automation. Hypothesis 1.3 requires computing the statement cost c_s from the declaration's type. This is not yet automated in v0.2. A future `StatementClassifier` layer would parse the type expression: Π^0_1/Π^0_2 bounds over discrete domains yield BISH; analytic equality yields WLPO; unbounded existentials yield LPO or CLASS.

7.4 Connection to the CRM program

`CRMLint` closes the loop on the 75-paper CRM series. Papers 1–75 established the methodology by manual audit; Paper 76 automates it. The 14 Lean 4 bundles serve as calibration ground truth: the tool must reproduce the classifications that human analysis derived.

The long-term goal is to extend the CRM audit from 75 theorems to all of Mathlib. At $\sim$150,000 declarations, automated classification is the only feasible approach. `CRMLint`'s batch scanner (`#crm_scan`) enables this at the namespace level.

A natural next validation step: run `CRMLint` on the Paper 75 Lean bundle (`P75_ConservationTest`) and verify that the tool reproduces the WLPO statement cost and CLASS proof cost that Paper 75 derived by manual analysis. This would close the calibration loop between the manual prototype and the automated tool.

8 Conclusion

`CRMLint` is the first automated CRM logical cost analyzer for a proof assistant. It distinguishes infrastructure artifacts from genuine classical content in Lean 4/Mathlib by type-aware pattern classification, correctly assigning WLPO to `Real.instField` where the naïve approach reads BISH.

The conservation gap $\Delta(T) = c_p - c_s$ is immediately usable: the DAG surgery pipeline (§7) combines `CRMLint`'s static analysis with inference-time LLM prompting to close gaps without any model training. Three gaps were closed by this pipeline in §4.3, each requiring 1–3 line fixes. Subject to the limits of Gödelian independence [18], the gap predicts when a constructively purer proof exists and localizes the classical boundary in the dependency DAG. `CRMLint` provides the X-ray; LLMs provide the scalpel; Lean provides the verification.

Acknowledgments

The Lean 4 formalization uses Mathlib4 [14]; we thank the Mathlib contributors for maintaining this essential infrastructure.

This paper was drafted with AI assistance (Claude, Anthropic). The formal verification was developed and checked with Claude (Anthropic). The v0.2 classification fix was motivated by a detailed metamathematical audit identifying the `Real.instField` hallucination. The author is a clinician (interventional cardiology), not a professional mathematician; the logical structure of the tool has been verified by calibration against known ground truth. Errors of mathematical judgment remain the author's responsibility. This paper follows the standard format for the CRM series [13].

This series is dedicated to the memory of Errett Bishop (1928–1983), whose program demonstrated that constructive mathematics is not a restriction but a refinement.

References

[1] E. Bishop and D. Bridges. *Constructive Analysis*. Springer, 1985.

[2] D. Bridges and F. Richman. *Varieties of Constructive Mathematics*. Cambridge University Press, 1987.

[3] D. Bridges and L. Vîţă. *Techniques of Constructive Analysis*. Springer, 2006.

[4] H. Ishihara. Constructive reverse mathematics: compactness properties. In: *From Sets and Types to Topology and Analysis*, Oxford Logic Guides 48, pp. 245–267, 2005.

[5] P. C.-K. Lee. *Goldbach–Bishop Constructive Analysis (Paper 1, CRM Series)*. Zenodo, DOI: 10.5281/zenodo.14538954, 2024.

[6] P. C.-K. Lee. *The Bidual Gap Theorem: WLPO as the Cost of Non-Reflexivity (Paper 2, CRM Series)*. Zenodo, DOI: 10.5281/zenodo.14566966, 2024.

[7] P. C.-K. Lee. *The Logical Cost of Mathematics Is the Logical Cost of $\mathbb{R}$: A Synthesis (Paper 67, CRM Series)*. Zenodo, DOI: 10.5281/zenodo.18776113, 2026.

[8] P. C.-K. Lee. *Fermat's Last Theorem Is BISH (Paper 68, CRM Series)*. Zenodo, 2025.

[9] P. C.-K. Lee. *The DPT Axiom Trilogy: Unique Necessity, Not Mere Minimality (Paper 72, CRM Series)*. Zenodo, DOI: 10.5281/zenodo.18765393, 2026.

[10] P. C.-K. Lee. *Axiom 1 Reverse: The Constructive Content of Motivic Realisation (Paper 73, CRM Series)*. Zenodo, DOI: 10.5281/zenodo.18765700, 2026.

[11] P. C.-K. Lee. *Axiom 2 Reverse: The Trace Equality Test Is WLPO (Paper 74, CRM Series)*. Zenodo, DOI: 10.5281/zenodo.18773827, 2026.

[12] P. C.-K. Lee. *Conservation Test: CRM Calibration of the Genestier–Lafforgue GL LLC (Paper 75, CRM Series)*. Zenodo, DOI: 10.5281/zenodo.18773831, 2026.

[13] P. C.-K. Lee. *Paper Format Guide (CRM Series)*. Zenodo, DOI: 10.5281/zenodo.18777597, 2025.

[14] The Mathlib Community. *Mathlib4: Mathematics in Lean 4*. `https://github.com/leanprover-community/mathlib4`, 2024.

[15] L. de Moura and S. Ullrich. The Lean 4 theorem prover and programming language. In: *CADE-28*, LNCS 12699, pp. 625–635, 2021.

[16] DeepMind. AlphaProof: AI system for formal mathematical reasoning. `https://deepmind.google/discover/blog/ai-solves-imo-problems/`, 2024.

[17] Z. Xin et al. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and Monte-Carlo tree search. *arXiv:2408.08152*, 2024.

[18] J. Paris and L. Harrington. A mathematical incompleteness in Peano arithmetic. In: J. Barwise (ed.), *Handbook of Mathematical Logic*, pp. 1133–1142, North-Holland, 1977.