Where Formal Libraries Spend Their Axioms

A cross-foundation measurement, and an avoidable classical dependency in Lean's omega

Vince Gonzalez
ORCID 0009-0005-3640-014X

Abstract

A proof assistant can report which axioms a theorem rests on, but only one theorem at a time, and only whether rather than why. I measure axiom use across six libraries and two proof systems — the Metamath databases set.mm (ZFC, classical first-order), iset.mm (intuitionistic), nf.mm (Quine's New Foundations), ql.mm (quantum logic) and hol.mm (higher-order logic), and Lean 4's Mathlib (dependent type theory) — using one program for all six, so the comparison rests on identical definitions rather than analogy.

The paper has two independent contributions: a method for measuring axiom spend across foundations, and a diagnosis, produced by that method, of an avoidable source of Classical.choice in Lean. §0 states which claims are machine-checked, which rest on a stated assumption, and which are merely consistent with the evidence.

Three results. First, axiom use funnels through very few lemmas in the mature libraries: the median axiom is cited directly in 2 to 4 proofs and inherited by thousands, and entry points per theorem agree to within 13% across set.mm, iset.mm and nf.mm despite three different foundations. Two smaller databases, ql.mm and hol.mm, sit three to six times higher, so the concentration is a property of maturity rather than of formal libraries as such — a narrowing of what an earlier version of this paper claimed. A refactoring-invariance result (§3.1) settles which measurements can bear comparison at all: reach is invariant under inlining and factoring, amplification is not, and the two are reported accordingly. Second, separating each theorem's statement from its proof bounds how much classical dependence is even eligible for removal from Mathlib: 13.1% of theorems have a choice-free statement and a choice-dependent proof, and nothing outside that band can be eliminated under any argument. Third, a substitutability test that re-synthesizes each site and classifies the result by the kernel's own axiom bookkeeping finds 805 places where a choice-free instance was available and not used, of which 280 declarations would become free of Classical.choice outright.

Those 280 turn out to share a cause. 276 of them (98.6%) trace to the omega decision procedure, which supplies the Decidable arguments of six helper lemmas as a literal Classical.propDecidable and never attempts instance synthesis. On Nat and Int, where the required instances exist and are axiom-free, this makes otherwise constructive proofs depend on the axiom of choice. The behaviour reproduces in one line with no imports.

I applied the fix rather than only proposing it, and then tested whether the substitution preserves the proofs. Rewriting all 280 proof terms and submitting each to the kernel against its unchanged statement: 276 are accepted and 4 are rejected, and the four rejections are exactly the four declarations traced to a cause other than omega. 275 declarations lose their dependence on Classical.choice outright. The partition is structural — omega passes the instance to a helper lemma and never computes with it, so its terms are indifferent to which instance they receive, while the four exceptions use decidability computationally and break. The hardcoded instances are still present in Lean master as of August 2026.

Reported upstream, the issue was closed as completed with the reply that avoiding choice is a deliberate non-goal of Lean core (§6.3). No figure was disputed. The behaviour is therefore within Lean's stated design rather than a defect by its own standard, and the reply also serves as a primary source for §3's reading of why set.mm and Mathlib differ so widely in reach.

---

0. Claims, and how strongly

This paper makes two contributions, and they stand or fall independently.

A. A method for measuring axiom spend across foundations. One program measures six libraries under identical definitions, separates each theorem's statement dependencies from its proof dependencies, and decides substitutability by the kernel's own bookkeeping rather than by name. Contribution A survives regardless of what happens to B.

B. A diagnosis of an avoidable source of Classical.choice in Lean. The omega decision procedure hardcodes a classical instance where a constructive one exists, which accounts for 276 of the 280 declarations that method A flags. Contribution B is a finding produced by A, not evidence for A.

Vocabulary

These four words are not synonyms and are never used interchangeably below.

- choice-free — a term whose axiom closure, per collectAxioms, does not contain Classical.choice. A property of a completed term, checkable.
- substitutable (of a site) — instance synthesis, run again at that site with the whole library in scope, returns a choice-free instance. A property of one position in one proof.
- cleanable (of a declaration) — every route from it to Classical.choice in the dependency graph passes through a substitutable site. A property of a declaration relative to the graph, and strictly stronger than containing a substitutable site.
- removable (of a declaration) — its proof term, with every substitutable site replaced, is accepted by the kernel against the unchanged statement, and its axiom closure then omits Classical.choice. This is strictly stronger than cleanable, because Decidable is data: two instances for the same proposition are interchangeable only up to whatever definitional unfolding the

surrounding proof relies on. §6.2 establishes it for 275 declarations and refutes it for 4.

Reach is |D(a)|/|T|, the fraction of theorems depending on axiom a. *Entry points* E(a) are theorems citing a directly. Amplification is |D(a)|/|E(a)|. §3.1 shows these three behave very differently under refactoring, which governs how much weight each can carry.

Evidence grades

grade	meaning	claims
demonstrated	machine-checked, reproducible from the shipped artifacts	all counts in §3–§5; that patched omega proves the affected shapes choice-free; that Int.natCast_eq_zero restated verbatim becomes choice-free; that the cleanup propagates along the one chain tested; that the hardcoded instances are present in current Lean master
supported	follows from verified data together with a stated assumption	the attribution of 276 declarations to omega, which assumes citing a helper lemma implies the instance came from that call site; the 13.1% ceiling, which depends on the definition of a choice-free statement
suggested	consistent with the evidence, not established	that a patched omega re-elaborates source syntax to the same terms the kernel accepted in §6.2; that the concentration observed in §3 reflects a general practice rather than four coincidences

Nothing in this paper is graded demonstrated on the strength of my own narration. §9 gives a command that recomputes every §5 and §6 figure from the artifacts and prints OK or MISMATCH against the published value.

---

1. What is being asked

#print axioms in Lean, and the equivalent bookkeeping in Metamath, answer a local question: does this theorem depend on that axiom? The questions I want are global and comparative.

- Where is an axiom spent — cited directly in a proof — as opposed to inherited through a chain of lemmas?
- Is the shape of that spending a property of one library and its community, or does it survive a change of foundation?
- When a proof rests on an axiom, could it have avoided it?

Behind these is a claim I would rather state than leave implied. A formal library's axiom record is read as a statement about what mathematics requires. If a proof draws on more than it needs, that record overstates the mathematics, and does so in a form designed to be trusted without rereading. That is the whole of the motivation here; no stronger claim about the ethics of provenance is intended or needed.

The third question is the one with consequences, and it is the one where care is most easily lost. A name-based screen is not enough. In an earlier pass over set.mm I looked for choice-dependent lemmas with choice-free siblings by name, and measured that screen at 41.5% precision. Its characteristic failure is instructive: lbsex requires ax-ac, while lbsexg reports no dependence

only because CHOICE has been moved into an explicit antecedent. Logical strength was relocated, not removed. Any method that cannot tell those apart will report progress that does not exist.

1.1 Where this sits

The question is old. Reverse mathematics, from Friedman's programme through Simpson's Subsystems of Second Order Arithmetic, asks which axioms a theorem requires, and answers it by proving equivalences over a weak base theory. That is a strictly harder question than the one here, and the distinction should not be blurred: reverse mathematics establishes necessity, whereas this paper measures use. A theorem may use choice and not need it — indeed §6 is entirely about that gap — but showing it does not need it, in the reverse-mathematical sense, requires a proof I do not give.

The machinery is old too. De Bruijn's Automath established that a proof could be checked by a program small enough to audit by hand, and that criterion shaped the kernels of HOL Light, Isabelle, Coq, Lean and Metamath. Each of those systems can already report a single theorem's axiom dependencies — Print Assumptions in Coq, #print axioms in Lean. Metamath's set.mm goes further as a matter of editorial policy, tiering full, countable and dependent choice as separate axioms so that choice-free proofs stay identifiable, which is why §3 can measure it. Kohlenbach's proof mining extracts quantitative content from classical proofs and is the closest relative in spirit, though its aim is bounds rather than bookkeeping.

What is missing from that lineage is the library-scale, cross-foundation view: per-theorem reporting exists everywhere, and nothing aggregates it. That gap is what contribution A fills.

So the test used here does not read names. For each site it asks the elaborator to synthesize the required instance afresh, then runs the kernel's own collectAxioms over the result. A site counts as substitutable only when the synthesized replacement's axiom closure omits Classical.choice. The verdict comes from the same bookkeeping that answers #print axioms.

---

2. Method

Metamath. A .mm file must define labels before use, so a single forward pass computes exact dependency closures. Logical axioms are identified by typecode — a $a statement whose typecode is the assertion symbol |- — and not by the ax- naming convention, which differs between databases. This keeps the comparison foundation-independent: iset.mm and nf.mm name things differently and would be miscounted by a name rule.

Lean. An environment has no such ordering, so the whole declaration graph is extracted and closures are computed over it. For each constant I record the constants appearing in its type and in its value separately, which the statement-versus-proof measurement of §4 requires and which a union cannot express.

One detail here is not incidental, because it invalidated a complete round of results before I caught it. In Lean 4.32, ConstantInfo.value? returns none for theorems unless called as value? (allowOpaque := true); earlier versions returned theorem values unconditionally. An extractor written the obvious way therefore sees no proof terms at all, silently measures statements, and reports them as proofs. In my first pass every one of 532,605 theorems came back with an empty value, and the resulting figures were wrong in the direction of making the library look cleaner than it is.

What exposed it was not suspicion but arithmetic. A 2×2 classification of theorems by statement- and proof-dependence returned exactly zero in both off-diagonal cells. An exact zero across half a million cases is a logical identity or a bug; it is never a statistic. The diagnostic that localised it takes one line — count theorems whose value column is non-empty, and require that it not be zero. I recommend it to anyone analysing a Lean environment.

Scale. Lean 4.32.1 with Mathlib: 790,171 declarations — 532,605 theorems, 228,875 definitions, 15 axioms, 28,676 other — and 30,015,601 dependency edges. Without proof terms the same environment yields 14,811,838 edges, so slightly more than half of all dependency information in the library lives in proofs.

---

3. Where axioms are spent, and what that depends on

For each logical axiom I count dependents, the theorems whose closure contains it, and entry points, the theorems citing it directly. Their ratio is the amplification. Entry points per theorem normalises for library size and is the quantity to compare across databases.

	theorems	axioms used	median entries	entries/theorem	amplification
set.mm (ZFC)	47,621	1,447	2	0.3758	292×
iset.mm (intuitionistic)	16,236	480	3	0.3782	221×
nf.mm (New Foundations)	5,976	195	4	0.4259	94×
ql.mm (quantum logic)	1,140	43	9	2.4202	6×
hol.mm (higher-order)	151	43	1	1.1589	17×

In set.mm, 77% of used axioms are cited directly in three proofs or fewer. The extreme case is ax-4, cited once and inherited by 44,501 theorems: a single proof step carries an axiom into 93% of the library. Mathlib is the same shape — Classical.choice has 144 entry points among 532,605 theorems and 324,808 dependents, amplification 2,256×.

An earlier version of this paper reported the first three rows and called the concentration foundation-independent. Adding ql.mm and hol.mm shows that reading was too strong. The three large databases agree closely — 0.3758, 0.3782 and 0.4259, within 13% of each other across ZFC, intuitionistic logic and New Foundations — while the two small ones sit three to six times higher. Foundation does not separate them; size does.

ql.mm is the sharper of the two counterexamples, because 1,140 theorems is not a toy. Its median axiom is cited directly nine times against two to four

elsewhere, and only 30% of its axioms have three or fewer entry points against set.mm's 77%.

The mechanism is mundane. A young library has not yet had its shared subproofs extracted into gateway lemmas; factoring is what accumulates as a library matures, and factoring is exactly what drives entry points down. So the claim this section supports is narrower than it first appeared: **concentration of axiom use is a property of mature formal libraries, not of formal libraries as such.** That the three mature ones agree so closely across three foundations is still the striking part, and it is a fact about engineering practice rather than about logic.

This is also direct evidence for §3.1, which argues on other grounds that the metric measures presentation rather than mathematical content. Maturity is a change in presentation.

3.1 What amplification can and cannot mean

Amplification is a ratio of two quantities that behave quite differently, and before it is used to compare libraries that difference has to be settled. The result is negative, and it is the reason the rest of this section is phrased in terms of reach rather than amplification.

Consider refactorings that preserve every theorem's statement and every theorem's transitive axiom closure: inlining, replacing a citation of a lemma by that lemma's proof, and factoring, extracting a shared subproof into a new lemma. These change how the library is written, not what it proves.

> Proposition. Under inlining and factoring alone,
> (i) $D(a)$ is invariant;
> (ii) $E(a)$ is not;
> (iii) consequently $A(a)$ can be driven to any value in $[1, |D(a)|]$.
>
> Proof. (i) Neither operation alters any theorem's transitive axiom closure, so membership of $D(a)$ is untouched. (ii) Inline every intermediate lemma into each of its citers; every theorem in $D(a)$ then cites a directly, so $E(a) = D(a)$. Conversely introduce a single gateway lemma G whose statement is a's, prove G from a, and reroute every citation through it; then $E(a) = \{G\}$. (iii) $A(a) = |D(a)|/|E(a)|$, and by (ii) $|E(a)|$ ranges over the whole of $[1, |D(a)|]$.

So amplification is a property of a library's factorization, not of the mathematics it contains. The figures 292×, 221×, 94× and 2,256× describe how four communities chose to organise their proofs. That they land in the same range is a real observation about practice — it is not an observation about ZFC, intuitionistic logic, New Foundations or type theory, and this paper does not present it as one.

Reach, by contrast, is invariant by (i). This is why the comparison that follows is stated in reach, and why I regard it as the more robust of the two.

$E(a)$ does have a use, and it is operational rather than descriptive: **$E(a)$ is the edit surface.** Eliminating a from a library means changing every proof in

E(a) and no others. A small E(a) says an axiom is cheap to attempt removing; it says nothing about whether removal is possible. §6 is the extreme case — 374 declarations sit on omega's classical path, yet the edit surface is a single tactic, because all those entry points were emitted by one piece of code rather than written by 374 authors.

A nontrivial bound on A(a) would be the more satisfying result to report here, and I have not found one. By the proposition there cannot be a bound that depends only on the library's mathematical content, since A(a) is not determined by it; any bound would have to quantify over factorization conventions, which are not the kind of thing this method measures. The invariance result is what the metric actually supports.

The rates differ, and the difference is the interesting part.

	choice axiom	dependents	share of library	entry points	per theorem
set.mm	ax-ac + ax-ac2	583	1.2%	3	6.30×10^{-5}
Mathlib	Classical.choice	324,808	61.0%	144	2.70×10^{-4}

Choice is spent about four times as often per theorem in Mathlib, but it reaches fifty times as much of the library. This is not a defect. set.mm tiers its choice principles deliberately — ax-ac and ax-ac2 for full choice, ax-cc for countable choice (1,016 dependents, 3 entry points), ax-dc for dependent choice (88 dependents, 2 entry points) — because keeping track of which theorems are choice-free is one of that project's stated aims. Mathlib is classical by default and does not attempt the distinction. Each library's numbers reflect what it set out to do.

---

4. A ceiling on what is eligible

Classifying all 532,605 Mathlib theorems by whether the constants in the statement, and separately in the proof, reach Classical.choice:

	theorems	share
statement choice-dependent, proof choice-dependent	252,515	47.4%
statement choice-free, proof choice-dependent	69,571	13.1%
statement choice-dependent, proof choice-free	2,722	0.5%
neither	207,797	39.0%

The second row is the entire candidate population for avoidable classical dependence. A theorem whose statement mentions something choice-dependent — most of real analysis, ultrafilters, ordinals — cannot be made choice-free however it is proved. Only the 13.1% are even eligible.

This is a ceiling, not an estimate, and the gap between the two is wide. Many of those 69,571 need classical logic — reasoning by contradiction on a constructively meaningless statement — even though nothing in the statement mentions a choice-dependent constant. The eligible fraction is at or below 13.1%, and §5 suggests the avoidable fraction is far below that again.

---

5. Substitutability, decided by the kernel

Classical.propDecidable : (p : Prop) → Decidable p is a low-priority instance. No author writes it; the elaborator inserts it when it needs Decidable P and instance search finds nothing better. Every insertion makes the enclosing declaration depend on Classical.choice. The dependence is incidental exactly when a choice-free instance was available and simply not found.

For every declaration citing one of the five classical decidability fallbacks in its value, I walk the proof term, and at each site — inside the traversal, while the binders are live, since essentially every site sits under a binder and the instance arguments that would supply a replacement are themselves binders — ask for the instance again with the whole library in scope, then classify the result by collectAxioms.

declarations citing a fallback in their value	8,907
verdicts recorded	23,279
sites with a choice-free instance available	805
declarations containing at least one	352
declarations whose only route to choice is such a site	280
blocked by other choice use	72

Three artifact classes are excluded rather than counted, and they are large enough that omitting the distinction would inflate the result by half. Of 1,199 raw flags, 281 resolved to a bare local identifier — a Decidable hypothesis already in scope, which the elaborator arguably should have used, but which offers no library instance to substitute. A further 113 were the same case with arguments applied, detectable by the inaccessible-name marker the pretty-printer emits and invisible to a whitespace test. The unapplied head constant appears once per declaration in the traversal and is not a site at all.

The test's negatives are as informative as its positives. It returns "choice genuinely needed" for equality, ≤ and < on Real.linearOrder, and its choice-needed verdicts cluster in Analysis.Real.Hyperreal, Data.Real.Basic, Order.Filter.FilterProduct and SetTheory.Ordinal.Basic — real numbers, ultrafilters, ordinals. It is not rubber-stamping.

Verified independently of the classifier, using the assistant's own command:

```
Classical.propDecidable  depends on axioms: [propext, Classical.choice, Quot.sound]
instDecidableEqNat       does not depend on any axioms
Nat.decLt                does not depend on any axioms
Nat.decLe                does not depend on any axioms
```

The proposed replacements rest on nothing whatever. Each substitution removes a dependence rather than relocating it — the failure mode that sank the name-based screen.

---

6. Why: one cause, 276 times

#print axioms says whether. To ask why, I compute the shortest dependency path from a declaration to the axiom and label each edge as a statement or a proof edge. The distinction is what makes the answer actionable: a proof edge can often be rerouted by changing a tactic, while a statement edge cannot be touched without changing what the theorem says.

```
Int.mem_box  --proof-->  …mem_box._proof_1_5  --proof-->  Classical.propDecidable  --proof-->
Classical.choice
```

Every affected declaration has this shape — short, and entirely proof edges. The generated proof term is _proof_1_5, and the parent theorems are proved by omega.

omega does not introduce choice on plain linear-arithmetic goals, nor on disjunctions. It introduces it on goals with ↔ or ∧ structure:

```
theorem om_iff2 (a b : Nat) : a - b = 0 ↔ a ≤ b := by omega
#print axioms om_iff2   -- [propext, Classical.choice, Quot.sound]

theorem om_lin (a b : Nat) : a - b + b ≥ a := by omega
#print axioms om_lin    -- [propext, Quot.sound]
```

That the arithmetic never required it is settled by splitting the connective by hand and giving omega exactly the same work on each side:

```
theorem manual (a b : Nat) : a - b = 0 ↔ a ≤ b := by
  constructor
  · intro h; omega
  · intro h; omega
#print axioms manual    -- [propext, Quot.sound]
```

The cause is in Lean/Elab/Tactic/Omega/Frontend.lean. In pushNot and addFact, the Decidable arguments of six helper lemmas are supplied as a literal constant at eight sites:

```
| Iff P Q =>
  return some (mkApp5 (.const ``Decidable.and_not_or_not_and_of_not_iff []) P Q
    (.app (.const ``Classical.propDecidable []) P)
    (.app (.const ``Classical.propDecidable []) Q) h)
```

The helpers — for ¬(P → Q), ¬¬P, ¬(P ∧ Q), ¬(P ↔ Q), implications, and ↔ — are declared in Init/Omega/Logic.lean taking Decidable instance arguments, precisely so a constructive instance can be supplied. Nothing forces the classical one. The frontend never asks. The ¬(P ∨ Q) case, by contrast, uses a helper that takes no instance at all and is already clean.

This is not an artifact of one toolchain. The same hardcoded instances are present in Lean master as of August 2026 — nine occurrences there against eight in 4.32.1, with no call to synthInstance or trySynthInstance anywhere near them — so the behaviour is current and the upstream fix would cover nine sites.

Attributing the 280 cleanable declarations by which helper their proof terms

cite:

attributable to omega	276 (98.6%)
other cause	4

Read the other way: 374 declarations across the library sit on omega's classical path, and 276 of them — 74% — would become free of Classical.choice if omega attempted synthesis before falling back. By package: Init 223, Std 41, Mathlib 5, Batteries 3, Lean 3, Plausible 1.

The suggested fix is local: at each of the eight sites, attempt synthInstance (Decidable P) and fall back to Classical.propDecidable only on failure. Goals over Nat and Int then resolve to the axiom-free instances; goals over genuinely undecidable propositions are unaffected, since the fallback still applies.

Two plausible hypotheses were tested and are false, and I record them because each cost time. omega on plain linear or disjunctive goals is clean. And the classical tactic is clean — it registers propDecidable at low priority, so a real Nat.decLe still wins instance resolution. The blunt instrument I expected to find is not blunt.

6.0 Prior art: this is not a new category

The phenomenon is known, and was reported for a different tactic three years before this work. In August 2023, F. G. Dorais filed leanprover/lean4 issue #2414, observing that the split tactic invoked Classical.em even where the ite expression already carried a decidability instance, so that

```
theorem ifthisthenthat (n : Nat) : (if n = 0 then 0 else 0) = 0 := by split <;> rfl
```

depended on Classical.choice with no need. His framing is the one I would use: that Lean's occasional use of classical axioms to avoid handling decidability instances is well known and mostly harmless, but that some cases are not necessary. The issue was accepted and fixed.

What follows is therefore not the discovery of a category but the observation that omega was not covered by that sweep, together with three things the earlier report did not attempt: a measurement of how far the problem extends (§5), an attribution of the affected declarations to a single cause (§6), and a kernel-level test of whether the substitution preserves the proofs (§6.2). The per-site fix I propose in §6.1 is the same move Dorais proposed for split — use the instance that is already available — applied where it was missed.

6.1 The fix, compiled and run

The claim above is not left as an argument from source reading. I copied the frontend into an ordinary Lake module, replaced each of the eight literal instances with a call that attempts synthesis and falls back only on failure, compiled it, and exposed the result as a tactic omega_fix:

```
def decInst (P : Expr) : MetaM Expr := do
```

```
let goal ← mkAppM ``Decidable #[P]
match ← trySynthInstance goal with
| .some inst => return inst
| _          => return .app (.const ``Classical.propDecidable []) P
```

Every goal shape that makes stock omega classical — ↔, ∧, implication, over both Nat and Int — is proved by omega_fix with axioms [propext, Quot.sound]. The plain linear goal, already clean, is unchanged.

The test that matters is on a shipped theorem rather than a synthetic one. Int.natCast_eq_zero, at Init/Data/Int/DivMod/Lemmas.lean:30, is stated (n : Int) = 0 ↔ n = 0 and proved by omega. Restating it verbatim and changing only the tactic:

```
Int.natCast_eq_zero      shipped:  [propext, Classical.choice, Quot.sound]
my_natCast_eq_zero       patched:  [propext, Quot.sound]
```

One qualification, which cuts against me and is the reason this is worth reporting carefully. A theorem whose proof cites another affected lemma does not clean up on its own, because the lemma it cites still comes from the unpatched toolchain. Int.le_mul_iff_le_left is proved rw [Int.ediv_le_iff_le_mul hz]; omega, and re-proving it with omega_fix alone leaves it classical, since Int.ediv_le_iff_le_mul is itself in the affected set. Rebuilding the cited lemma too — which is what a genuine core rebuild does — propagates the cleanup:

```
Int.ediv_le_iff_le_mul   shipped:  [propext, Classical.choice, Quot.sound]
Int.le_mul_iff_le_left   shipped:  [propext, Classical.choice, Quot.sound]
my_ediv_le_iff_le_mul    rebuilt:  [propext, Quot.sound]
my_le_mul_iff_le_left    rebuilt:  [propext, Quot.sound]
```

So a single-file test can only clean leaf theorems, and therefore understates what the fix achieves rather than overstating it.

What this does not do is rebuild Lean core, which requires a C++ toolchain I do not have here. The mechanism is verified end to end, including propagation through a dependency chain; what remains unverified is that all 276 affected proofs still elaborate under the patch. That is a compilation question, and only a full rebuild answers it.

6.2 Does the substitution preserve the proof?

Everything to this point established that a choice-free instance exists at each site. That is a weaker claim than the proof surviving the swap, and the gap is real: Decidable is data, so a proof closing a goal by rfl on something that computes through propDecidable will break when the instance changes.

The question is settled directly, without rebuilding anything. For each of the 280 cleanable declarations: take the stored proof term, replace every substitutable site with the synthesized instance, and hand the result to addDecl against the original, unchanged type. addDecl invokes the kernel, so acceptance is the same standard a rebuild would apply to the resulting term. Then run collectAxioms on what the kernel accepted.

outcome	declarations
kernel accepted, axiom closure now free of Classical.choice	275
kernel accepted, choice retained (2 of 14 sites not substitutable)	1
kernel rejected the rewrite	4
total	280

885 individual sites were rewritten across the corpus.

The four rejections are worth more than the 275 successes, because they are not a random sample. They are exactly the four declarations §7 traces to a cause other than omega:

declaration	kernel objection
Equiv.propEquivBool._proof_2	congrArg Eq Bool.decide_eq_true no longer applies — the proof computes through the instance
CategoryTheory….BinaryBicone.toBiconeFunctor._proof_1	declaration type mismatch
MeasureTheory.VectorMeasure.dirac._proof_1	declaration type mismatch
SkewPolynomial.coeff_erase	declaration type mismatch

So the partition is clean and was not designed: **every declaration attributed to omega survives the substitution, and every declaration attributed to something else does not.** The reason is structural rather than lucky. omega passes the instance as an argument to a helper lemma and never computes with it, so the term is indifferent to which instance it receives. The four exceptions use decidability computationally — deciding a Bool equality, indexing a Finset difference, casing on a two-element inductive — and there the choice of instance is load-bearing.

This also converts the status of Equiv.propEquivBool. §7 argues on the strength of reading the source that it is correct as written and a false positive of the test. The kernel now says so independently: the substitution does not type-check.

What remains untested is narrower than before. The rewritten term is accepted; whether the source syntax would re-elaborate to this term under a patched omega is a separate question that only a full rebuild answers. Since patched omega constructs its term by the same route with a different instance argument, and that argument is exactly what was substituted here, I expect agreement — but expectation is graded suggested, not demonstrated.

6.3 What the Lean project said

I reported this upstream, with the reproduction, the source locations, the fix and the kernel results of §6.2. The issue was closed as completed by a Lean FRO maintainer, with the reply:

> Avoiding choice is a deliberate non-goal.

That is a statement of policy, and it settles the matter for Lean core. No figure in this paper was disputed; the 276 declarations do rest on Classical.choice without needing to, and the kernel accepts the substitution for every one of them. What the reply establishes is that Lean core does not treat axiom minimality as an objective, so the behaviour is within its stated design rather

than a defect by its own standard. The issue is correctly closed.

Two things follow, and the first is a correction to my own framing. §6.0 cites issue #2414 — the same category of problem in split, accepted and fixed in 2023 — and I read that acceptance as evidence of a general policy on axiom hygiene. One accepted issue is not a policy. The inference was mine and it was wrong, and it shaped how the report was written.

The second is that the reply is a primary source for a claim §3 makes on inference alone. §3 argues that the gap between set.mm, which reaches 1.2% of its library with full choice, and Mathlib, which reaches 61.0%, reflects a difference in design intent rather than a difference in care: set.mm tiers its choice principles because tracking them is a stated aim of that project, and Lean does not attempt the distinction. A maintainer has now stated that position directly. The measurement stands on its own; the interpretation no longer rests on my reading of it.

None of this changes what the tooling is for. Whether a project wants to minimise its axiom use is a decision for that project. Whether it can tell what it is using is a separate question, and the answer for every library measured here was previously no.

---

7. The remaining four

Each was traced individually.

- MeasureTheory.VectorMeasure.dirac — the definition is preceded by open scoped Classical in, which supplies Classical.propDecidable (x ∈ ∅) where Set.decidableEmptyset applies.
- SkewPolynomial.coeff_erase — inherits Classical.decEq on Multiplicative ℕ from the underlying erase, where instDecidableEqMultiplicative applies.
- CategoryTheory.Limits.BinaryBicone.toBiconeFunctor — decidable equality on WalkingPair, a two-element inductive with instDecidableEqWalkingPair available.
- Equiv.propEquivBool — **correct as written, and a false positive of my test.** The definition is noncomputable and reads toFun p := @decide p (Classical.propDecidable _); its entire content is that Prop ≃ Bool requires classical logic. My scan flagged the auto-generated right_inv proof, whose proposition happens to be decidable. Substituting there would contradict the definition it belongs to.

One false positive in 280 is the honest precision figure for the cleanable classification, and it was found by reading the source rather than by any automatic check. I do not claim the method would have caught it.

---

8. Limitations

What §6.2 does not cover. The kernel accepted 276 rewritten proof terms and

rejected 4, so the concern that motivated reserving the word removable — that a proof might depend on unfolding particular to propDecidable — has been tested rather than assumed, and in 4 cases it was vindicated. What remains is narrower: a patched omega builds its own term rather than rewriting the stored one, and whether source syntax re-elaborates to the terms checked here is a question only a full core rebuild answers. The route is the same and the substituted argument is the same, so I expect agreement; that expectation is graded suggested and is not relied on by any other claim.

Coverage is sound, not complete. 12 of 861 sites in the first pass, and 12 of the full scan, remained undecided under the synthesis budget and are reported as undecided rather than assigned a verdict. The reported counts undercount.

The 13.1% ceiling depends on its definition. "Statement choice-free" means no constant appearing in the statement has Classical.choice in its axiom closure. A different reading of what it means for a statement to be constructively meaningful would move the number.

The comparison of §3 measures spending, not virtue. A library that reaches 61% of its theorems with choice has made a design decision, not an error.

---

9. Reproduction

Everything is deterministic and re-runnable. The environment is Lean 4.32.1 with Mathlib; the Metamath databases are the public set.mm, iset.mm and nf.mm.

Every quantitative claim in §5 and §6 is recomputed from the shipped artifacts, diffed against its published value, and reported OK or MISMATCH by

```
python verify.py
```

which needs no Lean, no Mathlib and no network, and runs in seconds. At the time of deposit it reports all 11 claims reproduce. Two derived inputs are shipped for it — the constants whose axiom closure contains Classical.choice, and the value dependencies of the 8,907 scanned declarations — because the 640 MB environment dump they come from is too large to deposit and they are exactly what the cleanable and attribution computations consume.

The §3 and §4 figures are library-wide and cannot be checked from a subset; they require regenerating the dump, for which the commands are below.

file	what it does
axioms/mmclosure.py	Metamath parser and closure engine
axioms/crosslib.py	amplification across the three databases (§3)
axioms/lean/Extract.lean	declaration-graph dump
axioms/lean/Split.lean	dump with statement and proof deps separated (§4)
axioms/lean/Substitute.lean	kernel-verified substitutability test (§5)
axioms/lean/report.py	sites → declarations → cleanable (§5)
axioms/lean/why.py	shortest dependency path to an axiom, edges labelled (§6)
axioms/lean/OmegaFix.lean	patched frontend + self-contained verification (§6.1)
axioms/lean/verify.py	recomputes every §5/§6 claim and diffs it against the paper

axioms/lean/Rewrite.lean	rewrites all 280 proof terms and kernel-checks them (§6.2)
axioms/lean/rewrite.tsv	per-declaration outcome, 885 sites rewritten
axioms/lean/sites2.tsv	all 23,279 verdicts
axioms/lean/cleanable.tsv	the 280 declarations

Four practical notes for anyone repeating this in Lean. Theorem values require allowOpaque := true (§2). Heartbeat exhaustion is a runtime exception that ordinary catch re-throws, so bounding instance search needs tryCatchRuntimeEx, and a cap needs both maxHeartbeats in the reader context and Core.withCurrHeartbeats to reset the baseline. Sites must be tested inside the traversal while binders are live — a closed-subterm filter finds zero sites in the entire library. And import Mathlib costs minutes per run, so output should be appended and flushed per declaration, which makes a long pass resumable.

References

Metamath databases: set.mm, iset.mm, nf.mm, ql.mm, hol.mm,
Metamath project.
Lean 4 and Mathlib: the Lean FRO and the Mathlib community.
F. G. Dorais, "RFC: avoid using Classical.em to split ifs",
leanprover/lean4 issue #2414, August 2023 (fixed).
omega frontend: src/lean/Lean/Elab/Tactic/Omega/Frontend.lean, Lean 4.32.1.
Helper lemmas: src/lean/Init/Omega/Logic.lean, Lean 4.32.1.

The measurements are maintained as a standing table, updated as libraries change: The Kernel Index, https://f-keys.com/gonzalgo/kernel-index/ (CC-BY-4.0), with kernel-index.json and kernel-index.csv alongside.

The tooling is released as gonzalgo (Apache-2.0), https://pypi.org/project/gonzalgo/, source at https://github.com/zengineco/gonzalgo.