# THE CANON

Strategy for a Formal Library of Physical Reality

Recognition Science Initiative

January 17, 2026

**Abstract**

This document describes a plan to build a unified, machine-verified library of physics. The goal is straightforward: derive the laws of nature from first principles and prove those derivations correct using formal methods. We call this library **The Canon**. It functions like a software library for mathematics (such as Mathlib), but for physical reality. The key difference from traditional physics is that we do not fit models to data—we derive constants algebraically and verify the derivations with a proof assistant. This document outlines the architecture, the workflow for contributors, and the long-term integration with existing mathematical infrastructure.

# Contents

# 1 Background: The Problem with Current Physics

Modern physics is remarkably successful at prediction but struggles with explanation. The Standard Model of particle physics, for example, requires approximately 26 free parameters—masses, coupling constants, mixing angles—that must be measured experimentally. The theory cannot tell you *why* the electron has the mass it does; it can only tell you that if you plug in the measured value, the equations work.

This creates several practical problems:

- **Fragmentation.** We have separate theories for different domains (quantum mechanics, general relativity, thermodynamics) that do not always fit together cleanly.
- **Arbitrariness.** The constants feel like accidents. There is no derivation explaining why $\alpha \approx 1/137$ rather than some other value.
- **Reproducibility.** "Discovery" is statistical. A result is published if $p < 0.05$, but that threshold is arbitrary and has led to a replication crisis across the sciences.

Recognition Science (RS) takes a different approach. It posits that physical reality is an information-processing system—a "Ledger" of recognition events—governed by a cost-minimization principle. In this framework, the constants are not inputs; they are outputs. The fine-structure constant, the speed of light, and particle masses are all derived algebraically from a single seed: the Golden Ratio $\phi$.

Whether or not you accept that claim, the *method* is valuable: formalize the derivations, prove them in a proof assistant, and let anyone verify the logic by compiling the code.

# 2 What is The Canon?

The Canon is a software repository containing formal proofs of physical laws. It is written in Lean 4 and depends on Mathlib for standard mathematical machinery (real numbers, analysis, topology, etc.).

The structure is hierarchical:

## 2.1 The Core

The Core contains the foundational axioms and the "forcing chain"—the sequence of theorems that bootstrap existence from minimal assumptions. This includes:

- The definition of the cost functional $J$ and its properties.
- The derivation of logic from cost (consistency is cheap, contradiction is expensive).
- The proof that 3 spatial dimensions are forced by ledger stability.
- The derivation of the Golden Ratio $\phi$ as the fundamental self-similarity ratio.
- The proof that the minimal update cycle is 8 ticks.

The Core is intended to be stable. Changes here are rare and require careful review, since everything else depends on it.

## 2.2 The Surface

The Surface contains derived physics—the things that follow from the Core. This includes:

- **Constants:** Derivations of $c$, $\hbar$, $G$, and $\alpha$ from $\phi$.
- **Particles:** Definitions of leptons and quarks as stable ledger configurations, with mass formulas.

- **Interactions:** Derivations of electromagnetic, gravitational, and (eventually) weak and strong forces.
- **Chemistry:** The periodic table, ionization energies, bond angles.
- **Astrophysics:** Galaxy rotation curves, cosmological parameters.

### 2.3 Certificates

A Certificate is a formal proof that a specific physical prediction matches the derived value. For example, we might have a certificate stating:

> "The inverse fine-structure constant $\alpha^{-1}$ lies in the interval $(137.030, 137.039)$, derived without free parameters."

Certificates are the "unit tests" of physics. They link abstract derivations to concrete, falsifiable predictions.

## 3 How Contributions Work

The Canon uses a standard open-source workflow: Git, GitHub, and Continuous Integration (CI). But the "review" process is different from traditional software development. Here, the CI system acts as a referee.

### 3.1 The Certificate Protocol

When a contributor wants to add a new derivation (say, the mass of the proton), they do not write a PDF paper. They write a Lean module containing:

1. **Definitions:** The mathematical objects involved (e.g., what "proton mass" means in the RS framework).
2. **Theorems:** The claims being made (e.g., "the proton mass is $X$ in RS-native units").
3. **Proofs:** The Lean terms that verify the theorems. If the proof compiles, it is valid.

The key constraint is that the proof must derive from the authorized Core axioms. If the contributor introduces a new assumption (a "magic number"), the CI will flag it.

### 3.2 The Pull Request Process

**Step 1: Clone and Branch.** The contributor clones the Canon and creates a feature branch.

**Step 2: Write the Derivation.** They write Lean code that defines the physical system and proves the relevant theorems. This often involves importing modules from the Core and building on existing lemmas.

**Step 3: Local Testing.** They run `lake build` locally to check that the code compiles. Compilation failures usually mean a logical error in the proof.

**Step 4: Open a Pull Request.** They push the branch to GitHub and open a PR. This triggers the CI pipeline.

**Step 5: Automated Review.** The CI system runs several checks:

- **Compilation.** Does the code typecheck? (This is the primary "proof" step.)
- **Axiom Audit.** Does the module introduce any new axioms? (Only whitelisted axioms are allowed.)
- **Sorry Check.** Are there any `sorry` placeholders? (A `sorry` is an incomplete proof; it is allowed in development branches but not in the main branch.)
- **Regression Testing.** Does this PR break any existing proofs?

**Step 6: Merge or Revise.** If the CI passes, a maintainer reviews the PR for clarity and style, then merges it. If the CI fails, the contributor revises their code.

### 3.3 Example: Adding a New Particle Mass

Suppose a physicist wants to derive the tau neutrino mass. They would:

1. Create a branch: `feature/tau-neutrino-mass`.
2. Import the lepton mass framework from the Core.
3. Define the tau neutrino as a specific ledger pattern.
4. Prove that the mass of this pattern equals the RS-derived value.
5. Open a PR and wait for CI.

If the derivation is correct, the theorem compiles, and the mass becomes part of the Canon. If the derivation is wrong, the proof fails to compile, and the contributor knows exactly where the logic broke.

## 4 Integration with Mathlib

Currently, the Canon depends on Mathlib for foundational mathematics: real numbers, limits, derivatives, linear algebra, and so on. This is practical—Mathlib has thousands of lemmas that would take years to reprove.

However, the long-term vision is more ambitious.

### 4.1 Phase 1: Dependency (Current State)

The Canon imports Mathlib:

```
require mathlib from git "https://github.com/leanprover-community/mathlib4"
```

This gives us access to standard analysis without reinventing the wheel.

### 4.2 Phase 2: Mutual Contribution

As the Canon matures, we contribute lemmas back to Mathlib where appropriate. For example, if we develop new results about the Golden Ratio or continued fractions, those might be useful to the broader Lean community.

### 4.3 Phase 3: Unification

The RS framework claims that mathematics and physics share a common foundation: the cost functional. If this is true, then eventually the axioms of Mathlib (type theory, ZFC equivalents) should be derivable from the RS Core.

This is speculative and far in the future, but the architecture allows for it. If we ever prove that the natural numbers are a consequence of ledger dynamics, we could replace the Mathlib import with an RS-native derivation.

## 5 Practical Considerations

### 5.1 Build Times

Compiling Mathlib from scratch takes 2–4 hours. To avoid this, we use cached builds. Mathlib provides pre-compiled binaries via `lake exe cache get`. Contributors should always download the cache before building.

### 5.2 Style Guidelines

Lean code in the Canon should follow these conventions:

- Use `simp only [...]` with explicit lemma lists, not bare `simp`.
- Prefer `linarith` for linear arithmetic, `nlinarith` for nonlinear.
- Use `norm_num` for concrete numeric computations.
- Keep proofs readable; avoid golf-style one-liners for complex theorems.

### 5.3 Documentation

Every module should have a header docstring explaining:

- What physical system it models.
- What theorems it proves.
- What assumptions (if any) it makes beyond the Core.

### 5.4 Versioning

The Canon uses semantic versioning:

- **Major version** changes when the Core axioms change.
- **Minor version** changes when new modules are added.
- **Patch version** changes for bug fixes and documentation updates.

## 6 Roadmap

### 6.1 Near Term (30 Days)

- Migrate repository to a public GitHub organization.
- Set up CI with axiom and sorry auditing.
- Document the existing Core and Surface modules.
- Reduce the sorry count in the certified surface to zero.

### 6.2 Medium Term (90 Days)

- Complete derivations for all lepton masses.
- Add the periodic table (noble gas closures, ionization energies).
- Expand the empirical comparison matrix to 20+ verified predictions.
- Accept the first external contribution.

### 6.3 Long Term (1 Year)

- Full Standard Model particle content (quarks, bosons).
- Cosmological predictions (Hubble constant, dark energy fraction).
- Interactive documentation site with searchable API.
- Recognition as an alternative physics framework in academic literature.

# 7 Conclusion

The Canon is an attempt to do physics the way we do mathematics: with machine-verified proofs. The goal is not to replace experiment—empirical data is still the ultimate arbiter—but to make theoretical derivations unambiguous and verifiable.

If a claim is in the Canon, you can check it yourself by compiling the code. If it compiles, the logic is correct. If it does not compile, the error message tells you exactly where the argument fails.

This is a long-term project. We do not expect to derive all of physics overnight. But by building incrementally, one theorem at a time, we can create a resource that future scientists will use as a foundation.