Certificate-Based Verification of Derivation-Graph Structural Properties in Lean 4/Mathlib

Megan Simons[1*] and Jonathan Washburn[1]

[1*]Recognition Physics Research Institute, Austin, TX, USA.

*Corresponding author(s). E-mail(s): msimons@recognitionphysics.org;
Contributing authors: jon@recognitionphysics.org;

Abstract

We present a Lean 4/Mathlib method for checking selected development-wide audit claims as explicit compile-time certificates. In our setting, a certificate is a Lean structure whose verified field bundles named propositions, so compilation succeeds only when all bundled obligations are reproved. We apply the method to a heterogeneous scientific formalization and use it to track properties such as upstream input isolation, derivation connectivity, and qualified uniqueness claims under stated premises. The development contains 1,490 theorem/lemma declarations across 175 Lean files (32,621 lines), 36 certificates, and zero `sorry` placeholders. The d'Alembert smoothness step previously isolated as `aczel_representation_axiom` is now proved internally via an interval-average bootstrap argument, so the artifact uses no custom axioms beyond Lean 4's kernel and Mathlib. The paper's contribution is methodological: it shows how certificates can serve as reusable audit interfaces for cross-declaration invariants and regression checks. The evaluation uses one primary case study and a transferability discussion based on the Lean 4 formalization of the Polynomial Freiman–Ruzsa conjecture.

Keywords: automated reasoning, proof assistants, proof certificates, proof engineering, reflective checking, Lean 4

1 Introduction

Over the past two decades, formalized mathematics has reached a level at which large and technically demanding results can be checked inside proof assistants. Representative examples include the formalization of the four-color theorem in Coq [1], the

Flyspeck formalization of the Kepler conjecture [2], the Liquid Tensor Experiment in Lean [3], and the Lean formalization of the polynomial Freiman–Ruzsa conjecture [4]. In each case, machine verification provided strong assurance that the formalized proof was free of gaps relative to the trusted proof base.

Automated reasoning tools and interactive proof assistants are increasingly used to validate large formal developments and to provide high-assurance evidence that long derivations are gap-free. In large artifacts, however, many correctness questions are structural rather than declaration-local: not only "is theorem T proved?" but also "does this result depend on external data?", "is a constant implied under the stated premises or assumed?", and "is the advertised end-to-end chain actually connected?" Lean already offers useful declaration-local and build-level audits, such as `#print axioms` and `lake build`, but those do not by themselves package cross-declaration invariants as explicit compile-time objects. Making such structural properties machine-checkable is therefore a recurring proof-engineering challenge.

This paper contributes a certificate-based methodology for encoding and checking such structural claims inside Lean. A certificate is a Lean structure whose `verified` field is a conjunction of propositions; compilation succeeds if and only if every conjunct is proved, yielding a compact, independently checkable witness for a global property of a development graph. Several certificate obligations are discharged by reflective computation (`native_decide`), while the remaining proofs combine rewriting and arithmetic automation (e.g., `simp`, `norm_num`, `linarith`, `ring`, `nlinarith`), illustrating the blend of interactive and automated reasoning emphasized in contemporary proof assistant practice [5].

As a case study, we apply the method to a scientific formalization drawn from Recognition Science [6, 7]. We use this development because it gives a heterogeneous derivation graph spanning functional equations, combinatorics, and real analysis, which makes it a useful stress test for certificate design. We also discuss transferability through comparison with the Lean 4 formalization of the Polynomial Freiman–Ruzsa conjecture, a mainstream additive-combinatorics development with a public blueprint, dependency view, and reproducible build interface [4, 8, 9]. That comparison is meant to test whether the certificate perspective carries over to a very different Lean artifact, not to claim a second implemented benchmark in the present repository.

Some terminology in the repository comes from internal certificate names such as `InevitabilityCert`, `ExclusivityCert`, and `NonCircularityCert`. In the prose, however, we use plainer descriptions, such as "conditional selection", "sensitivity to perturbation", and "upstream input isolation", to emphasize that these are internal properties of the formalized system under stated premises. When a result depends on the internally formalized d'Alembert smoothness step or on a restricted functional family, we say so explicitly.

1.1 Claim status and assumptions

We use the following claim taxonomy throughout:

- **THEOREM (kernel-checked):** a statement proved in Lean 4's kernel from the local definitions, imported library theorems, and the explicitly declared assumptions listed in the paper.
- **CONDITIONAL THEOREM:** a THEOREM whose proof depends on one or more explicitly declared assumptions or regularity/calibration hypotheses.
- **LIBRARY-REUSED FACT:** a statement discharged primarily by an existing Mathlib theorem and included for traceability rather than as a novel mathematical contribution of this paper.
- **DEF (definition):** a definition inside the formalization. Definitions may be interpreted physically, but the interpretation is not itself a Lean theorem.
- **CERT (certificate):** a compiled Lean structure bundling multiple named propositions into a single checkable predicate.
- **EMPIRICAL COMPARISON:** any comparison to experimental datasets (e.g., PDG/CODATA/NuFIT) is outside the Lean proof and is presented only as external context.

This taxonomy is introduced only to keep the status of individual claims clear.

Accordingly, the paper makes a conditional methodological claim: from the stated premises and definitions, Lean checks the derivation graph and the certificate-level meta-properties discussed below, using no custom axiom declarations beyond Lean 4's kernel and Mathlib.

Contributions.

This paper makes three contributions to automated reasoning:

(i) it gives a certificate schema for packaging selected artifact-level audit claims as conjunction-valued Lean structures, together with a working grammar, a compositionality account, and explicit failure modes;
(ii) it shows how this certificate layer complements Lean's built-in declaration-local and build-level audits by providing a named compile-time interface for cross-declaration invariants; and
(iii) it evaluates the method on a heterogeneous Lean development and distills proof-engineering lessons about reflective checking, modularization, and regression-sensitive artifact maintenance.

The present evaluation consists of one primary heterogeneous case study; we return to the limits of that evaluation explicitly in Section 5 and discuss transferability separately in Section 3.4.

2 The Formalization

2.1 Overview

The formalization contains 175 Lean 4 files organized into 12 substantive directories together with 3 root namespace modules (Table 1), totalling 1,490 theorem/lemma declarations, 36 verification certificate modules, and 32,621 lines of Lean source.

Artifact metric	Count
Lean files	175
Theorem/lemma declarations	1,490
Lean source lines	32,621
Verification certificate modules	36

Common proof steps use `exact`, `simp`, `norm_num`, `linarith`, `ring`, `calc`, and `nlinarith`; reflective checking via `native_decide` is used whenever a certificate conjunct reduces to a decidable finite computation.

The d'Alembert smoothness step previously isolated as `aczel_representation_axiom` is now proved internally via an interval-average bootstrap argument. The repository contains zero `sorry` placeholders and zero custom axiom declarations beyond Lean 4's kernel and Mathlib. Every theorem in the artifact is fully proved.

Assumptions registry.

For auditability, the formalization depends on the following explicit premises:

- (A0) Lean 4 kernel + Mathlib as the trusted proof base.
- (A1) Recognition Composition Law (RCL) as a premise: Eq. 1 on $\mathbb{R}_{>0}$.
- (A2) Regularity/normalization/calibration hypotheses stated in §2.2 (e.g. $F(1) = 0$ and the stated second-order calibration).

No experimental tables (PDG/CODATA) are used as inputs to the audited upstream definitions; this is tracked by the `NonCircularity` certificate family, which functions here as an upstream input-isolation check (Section 3). The trusted base is therefore Lean 4's kernel, Mathlib, and the explicitly stated modeling premises—not an additional custom axiom layer for the d'Alembert smoothness step.

The remainder of Sec. 2 summarizes enough object-level mathematics to make the certificates interpretable. In this paper, the case study is used as a benchmark for proof engineering and artifact auditing. The contribution lies in the certificate schemas, the dependency-tracing interface they induce, and the way they expose development-wide invariants as kernel-checked predicates.

2.2 Overview of the Six Links

To make the case study auditable at a glance, we organize the formalized chain into six named "links." Each link corresponds to a distinct artifact-internal target: a displayed constant, uniqueness statement, selection rule, or assembled expression whose formal status can be stated precisely and then tracked through the certificate layer. Table 2 serves as a roadmap for the rest of this section by summarizing, for each link, the informal target, the Lean-level status actually established, and the principal qualification needed for honest interpretation.

Table 1 Repository structure.

Directory	Content	Files
Cost/	$J(x)$ uniqueness, d'Alembert, derivatives, N-dimensional extension	17
Constants/	ϕ, $\hbar = \phi^{-5}$, G, $\kappa = 8\phi^5$, α^{-1}	5
Foundation/	Forcing chain, conservation, d'Alembert conditional selection	33
Masses/	Mass law, baselines, gap forcing, ladders, bridges	28
Gravity/	Framework-native quantity later interpreted as the Einstein coupling, ILG, rotation and galactic laws	19
Physics/	Higher-level physical consequence modules	14
Numerics/	Interval arithmetic and certified numeric bounds	12
Papers/	Paper-specific supporting formalizations	4
Patterns/	Gray-cycle and discrete pattern constructions	1
Derivations/	Mass-to-light bridge	1
Unification/	Master theorem, all constants from ϕ	2
Verification/	Machine-checked certificates	36
Root modules	Namespace aggregators (`Cost.lean`, etc.)	3
Total		175

The table is intentionally conservative. It does not claim that the listed statements are unconditional facts about nature, nor that every listed result is a novel mathematical contribution of this paper. Rather, it records what the present Lean artifact proves, reuses from Mathlib, or packages as a conditional artifact-level consequence of the stated definitions, premises, imported library facts, and explicit assumptions.

Table 2 High-level map of the six case-study links and their current Lean-level status.

Link	Verified statement (informal)
1	RCL + regularity + internally formalized d'Alembert smoothness step $\Rightarrow$ conditional uniqueness of $J(x) = \frac{1}{2}(x + x^{-1}) - 1$
2	Library-backed traceability facts for ϕ (algebraic identities, irrationality, interval bound)
3	An author-defined integer characterization internally selects $D = 3$
4	Within stated family/calibration choices, gap-law and mass-ladder properties
5	Positivity of framework-native constant $\kappa_{\mathrm{RS}} = 8\phi^5$ and related definitional identities
6	Certified exponential assembly of α^{-1} from $4\pi \cdot 11$ and $w_8 \ln \phi$, with interval-checked bounds and partially closed provenance of w_8

The subsections that follow expand each row of Table 2, clarifying which ingredients are definitional, which are library-reused, which are conditional on explicit assumptions, and which claims are purely artifact-level rather than physical.

2.3 Link 1: The unique cost functional

The derivation begins from the Recognition Composition Law (RCL), a functional equation on the positive reals. For all $x, y > 0$,

$$F(xy) + F(x/y) = 2F(x)F(y) + 2F(x) + 2F(y). \tag{1}$$

Under the normalization $F(1) = 0$, calibration $\lim_{t\to 0} 2F(e^t)/t^2 = 1$, and continuity, the substitution $H(t) := F(e^t) + 1$ reduces Eq. (1) to the d'Alembert equation $H(s+t) + H(s-t) = 2H(s)H(t)$. The internal d'Alembert smoothness proof then yields $H = \cosh$, hence $F = J$ where

$$J(x) = \tfrac{1}{2}(x + x^{-1}) - 1. \tag{2}$$

Accordingly, the artifact proves a conditional uniqueness statement: under the stated premises and the internally formalized d'Alembert smoothness step, J is the unique cost functional in the formalized class.

The Lean proof of $J(x) \geq 0$ is representative of the proof style throughout the formalization:

Listing 1 Non-negativity of J via AM-GM (`BaselineDerivation.lean`, lines 57–69).

```
theorem J_nonneg (x : R) (hx : 0 < x) : J x >= 0 := by
  unfold J
  have hxne : x != 0 := ne_of_gt hx
  have hxx : x * x^(-1) = 1 := mul_inv_cancel hxne
  have hsq : 0 <= (x - x^(-1)) ^ 2 := sq_nonneg _
  have hexpand : (x - x^(-1)) ^ 2
     = x ^ 2 - 2 + x^(-1) ^ 2 := by
    have : (x - x^(-1)) ^ 2
       = x ^ 2 - 2 * (x * x^(-1)) + x^(-1) ^ 2 := by ring
    rw [hxx] at this; linarith
  have hge : x ^ 2 + x^(-1) ^ 2 >= 2 := by nlinarith
  have hfactor : x + x^(-1) >= 2 := by
    nlinarith [sq_nonneg (x + x^(-1)),
               sq_nonneg (x - x^(-1))]
  linarith
```

This proof expands $(x - x^{-1})^2 \geq 0$, uses `ring` for algebraic rearrangement, and closes with `nlinarith` for the nonlinear step $x + x^{-1} \geq 2$. The same pattern — expand, rearrange, close with a linear or nonlinear arithmetic solver — recurs throughout the real-analysis proofs in the repository.

2.4 Link 2: The golden ratio

This link is included for traceability rather than novelty. The formalization defines

$$\phi = \frac{1 + \sqrt{5}}{2} \tag{3}$$

as a noncomputable constant in Lean, reuses Mathlib's theorem `Real.goldenRatio_irrational` to establish irrationality, and proves simple downstream algebraic and interval facts such as $\phi^2 = \phi + 1$ and $\phi \in (1.5, 1.62)$. The methodological point is that later derivations cite kernel-checked facts about the chosen constant. We do not present the irrationality of ϕ or the interval bound as original research contributions of this paper.

2.5 Link 3: Three dimensions

Define $\text{Wendo}(D) := D \cdot 2^{D-1} + 2D - 1$. In the current artifact, this author-specified integer characterization selects $D = 3$ uniquely among positive integers. Lean verifies the internal implication

$$\text{Wendo}(D) = 17 \;\Rightarrow\; D = 3 \tag{4}$$

using decidable evaluation for small cases and an explicit monotonicity bound for the general case. This is a property of the formalized selection rule, not an independent derivation of physical dimensionality from weaker premises. The six cube integers $\{V, E, F, A, E_{\text{passive}}, W\} = \{8, 12, 6, 1, 11, 17\}$ are then used as primitive upstream integers for the later artifact-internal constructions.

2.6 Link 4: The mass spectrum

The master mass law $m = A_s \cdot \varphi^{\,r-8+\text{gap}(Z)}$ is encoded with kernel-checked properties: positivity (`predict_mass_pos`), φ-scaling (`mass_rung_scaling`: $m(r{+}1) = \varphi \cdot m(r)$), and neutral-sector simplification (`gap_zero_neutral`: $\text{gap}(0) = 0$).

The gap function $\text{gap}(Z) = \log_\phi(1 + Z/\phi)$ is proved unique only within the stipulated affine-log family and under the stated calibration constraints. Concretely, Lean shows that the three-point conditions $g(0) = 0$, $g(1) = 1$, and $g(-1) = -2$, together with $b > 1$ and the restricted functional form under study, imply $b = \phi$ by deriving $b^2 - b - 1 = 0$ from logarithmic identities. This is therefore a conditional family-restricted uniqueness result, not a claim that ϕ would be implied outside those modeling choices.

Baseline rungs are derived from cube geometry: $r_e = A + 1 = 2$, $r_q = 2^{D-1} = 4$, $r_\nu = -(V{+}E{+}F{+}E_{\text{passive}}{+}W) = -54$.

2.7 Link 5: Gravity

Within the artifact, a framework-native quantity κ_{RS} is defined by

$$\kappa_{\text{RS}} = 8\phi^5, \tag{5}$$

and Lean proves the internal mathematical statement $\kappa_{\text{RS}} > 0$ together with selected definitional identities in framework-native units. Any interpretation of κ_{RS} as the Einstein gravitational coupling is external to Lean and is not certified by the present artifact. Accordingly, this link should be read as a traceability result about a formalized constant, not as a machine-checked physical validation claim.

2.8 Link 6: The fine-structure constant

Within the current formalization, the inverse fine-structure constant is assembled in canonical form as

$$\alpha^{-1} = \alpha_{seed} \exp(-f_{gap}/\alpha_{seed}), \qquad \alpha_{seed} = 4\pi \cdot 11, \tag{6}$$

where $f_{gap} := w_8 \ln \phi$ and $w_8 := w8_{from\_eight\_tick}$ is a parameter-free closed form on the canonical 8-tick basis.

Lean verifies the exact assembly formula together with the interval bounds $137.030 < \alpha^{-1} < 137.039$. The provenance of the seed factor 11 is fully traced to cube geometry, and the interval proofs bound the canonical gap term appearing in the exponential. What remains incomplete is a stronger closure statement identifying the raw DFT-based candidate gap weight with the canonical closed-form projection weight used in the certified α pipeline. Accordingly, this link should be read as a certified symbolic assembly with interval-checked bounds and only a partially closed provenance story for w_8.

3 Automated Reasoning Contributions and Certificates

3.1 Certificate schemas as artifact-level audit interfaces

Beyond proving individual theorems, the formalization includes 36 certificates: Lean structures whose `verified` field is a conjunction of separately proved propositions. A certificate compiles only when all of its conjuncts are reproved, so it acts as an explicit audit boundary for a development-wide claim. The underlying construction is straightforward. What matters here is its disciplined use as a stable interface for cross-declaration invariants: instead of leaving global claims in prose or README files, the development packages them as named predicates that can break visibly under later revisions.

Formal schema.

Let Γ be a Lean environment. A certificate schema in Γ consists of a finite family of propositions $P_1, \ldots, P_k$ together with a structure declaration whose field `verified` has type

$$P_1 \wedge \cdots \wedge P_k. \tag{7}$$

An inhabitant of that structure is therefore a kernel-checked witness that all bundled obligations hold simultaneously. In the present paper, the bundled obligations are drawn from the following article-level grammar:

$$\begin{aligned} P ::= T \mid \mathrm{DependsOnlyOn}(s, S) \mid \mathrm{NoExternalData}(s) \mid \\ \mathrm{Connected}(s, t) \mid \mathrm{UniqueWithin}(s, \mathcal{F}, C) \mid P \wedge P, \end{aligned} \tag{8}$$

where T ranges over previously proved theorems, $\mathrm{DependsOnlyOn}(s, S)$ asserts that a named artifact-level claim about symbol s is supported only by an audited upstream

set S, NoExternalData(s) asserts that specified external numerals do not occur in the audited upstream definitions for s, Connected(s, t) expresses end-to-end derivation connectivity between named endpoints, and UniqueWithin($s, \mathcal{F}, C$) records uniqueness only within a stipulated family $\mathcal{F}$ under constraints C.

The schema is compositional because each conjunct may itself be a previously proved theorem, a definitional equality, or a decidable fact discharged by reflection. Its failure modes are explicit: a certificate ceases to compile if an upstream theorem is weakened, a dependency claim becomes false after refactoring, a displayed closed form no longer reduces as advertised, or an audited input-isolation claim is violated.

Compared with theorem-by-theorem reporting or informal repository-level dependency claims, certificate schemas provide a kernel-checked interface for development-wide invariants. In practice, this yields (i) explicit summaries of dependency structure for reviewers and artifact auditors, (ii) regression checks that fail when an advertised global invariant breaks, and (iii) a clear separation between object-level mathematics and artifact-level claims. Section 3.4 returns to transferability through a brief comparison with the Lean 4 formalization of the Polynomial Freiman-Ruzsa conjecture.

Built-in declaration-local audits versus certificate-level audits.

Lean already provides useful declaration-local and repository-level audit tools. In particular, `#print axioms` reports the axiom footprint of a specific declaration, import/module inspection exposes coarse dependency structure, and `lake build`—typically run under CI—confirms that the repository still compiles. In the present repository, for example, one may inspect a purely local arithmetic proof such as `RecognitionScience.Masses.BaselineDerivation.J_nonneg`, or a declaration in the internally formalized d'Alembert module such as `RecognitionScience.Cost.FunctionalEquation.aczel_dAlembert_smooth`. At the command line, the comparison is as follows:

```
#print axioms RecognitionScience.Masses.BaselineDerivation.J_nonneg
#print axioms RecognitionScience.Cost.FunctionalEquation.aczel_dAlembert_smooth
```

These tools are necessary but not sufficient for the artifact-level claims studied here. They are either declaration-local or coarse-grained: they can tell us the dependency footprint of a particular theorem, or whether the repository still builds, but they do not by themselves package a cross-declaration claim such as "the audited upstream ingredients for the mass and α derivations contain no PDG/CODATA numerals" or "the displayed end-to-end chain from early premises to later closed forms is still intact after refactoring" into a single named predicate.

That additional layer is what the certificate schema contributes. In the present repository, `NonCircularityCert.verified` bundles upstream input-isolation obligations for sector constants, baseline rungs, and the α-seed, while `ForcingChainCert.verified` bundles the end-to-end chain from local facts about J and ϕ through baseline rungs, the mass law, and $\kappa = 8\phi^5$ into a single compile-time predicate. Schematically, the contrast is:

```
-- declaration-local audit
#print axioms RecognitionScience.Masses.BaselineDerivation.J_nonneg

-- artifact-level audit interface
@[simp] def NonCircularityCert.verified (_c : NonCircularityCert) : Prop := ...
@[simp] def ForcingChainCert.verified (_c : ForcingChainCert) : Prop := ...
```

Accordingly, the certificate layer should be understood not as a replacement for Lean's native auditing tools, but as a reusable artifact-level interface built on top of them: the native tools expose local proof dependencies, while the certificate schemas package selected global invariants so that they fail at a single visible boundary when later revisions break them.

Four certificates encode structural meta-properties in this narrower sense:

InevitabilityCert (10 conjuncts; better read as a conditional-selection certificate). Packages uniqueness and selection results that hold only relative to the stated premises, imported library facts, and explicit assumptions.

ExclusivityCert (6 conjuncts; better read as a sensitivity-to-perturbation certificate). Packages claims that selected modifications of the current formal setup violate specified artifact-internal properties.

NonCircularityCert (16 conjuncts; better read as an upstream-input-isolation certificate). Records that the audited upstream definitions for selected mass-formula inputs reduce to cube-derived ingredients and do not import PDG/CODATA numerals.

ForcingChainCert (22 conjuncts; better read as a derivation-connectivity certificate). Bundles the named chain of artifact-internal implications from early premises to later displayed closed forms.

The certificate architecture is intended as a reusable automated-reasoning pattern for large formalizations: it provides machine-checkable guarantees about selected artifact-level invariants, not a claim that the object-level scientific content is thereby validated.

Figure 1 summarizes the workflow from stated premises and local theorem proving to compiled certificate-level guarantees, together with the trusted checking base on which the artifact depends.

3.2 Decidable certificate checking and reflective computation

Some certificate conjuncts reduce to finite arithmetic facts about explicitly defined integers (e.g., identities among cube-derived constants). These are discharged using Lean's decidable evaluation and reflection mechanisms; reflective checking via `native_decide` is used whenever a certificate conjunct reduces to a decidable finite computation. This separation is deliberate: proof search and proof checking are disentangled, and the certificate boundary exposes only a small, kernel-checkable witness that the required decidable facts hold.

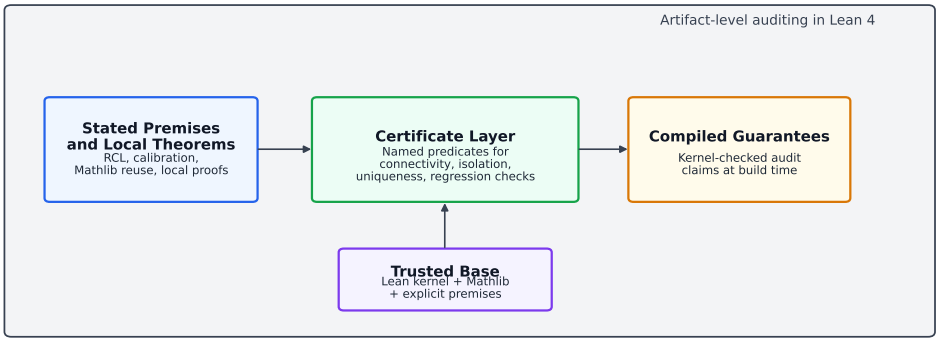

Fig. 1 Certificate-based workflow. Local theorem-level proofs feed a certificate layer that packages selected artifact-level claims as explicit predicates. Those predicates compile only when all conjuncts are re-proved, yielding a compact kernel-checked audit layer relative to the stated premises, imported library results, and an internally formalized d'Alembert smoothness step.

3.3 Automation profile and proof-engineering lessons

The remaining obligations are proved using standard tactic-based automation over Mathlib's analysis and algebra libraries. Across the artifact, rewriting and simplification (`simp`) and arithmetic normalization (`norm_num`, `linarith`, `ring`, `nlinarith`) dominate the proof script (Section 2). A recurring engineering tactic is to isolate "mathematical seams" where classical results or heavy library infrastructure is needed: for example, the d'Alembert smoothness step is confined to a single module (`AczelTheorem.lean`), keeping the downstream certificate interface independent of the proof technique used there.

From the perspective of automated reasoning, the case study illustrates a scalable workflow: (i) define the derivation graph in small modules, (ii) prove local lemmas with domain automation, (iii) expose global claims as certificate conjuncts, and (iv) discharge decidable conjuncts reflectively whenever possible.

Operationally, the certificates act as artifact-level regression checks. Because each certificate compiles only when all conjuncts are re-proved, changes to upstream definitions or dependencies immediately surface at the boundary where a global invariant is advertised. This makes the certificate layer useful not only for exposition but also for maintenance of the development.

3.4 Transferability discussion via a mainstream Lean artifact

To reduce domain-dependence concerns, we use the Lean 4 formalization of the Polynomial Freiman-Ruzsa (PFR) conjecture as a comparison point for transferability [4, 8, 9]. We selected PFR for three reasons. First, it is a mainstream mathematical formalization in additive combinatorics rather than a scientifically controversial framework. Second, it is implemented in Lean 4, matching the technical setting of the present

paper. Third, the project already exposes artifact-facing structure through a public blueprint, file-dependency view, and reproducible build interface, making it a natural comparison point for the certificate perspective developed here [8, 9].

This comparison is modest by design. We do not claim that the present repository implements certificate modules inside the PFR artifact, and we do not present a second empirical benchmark. The narrower question is whether the certificate grammar from Section 3.1 carries over to a mainstream Lean development with very different subject matter and engineering history. The comparison suggests that it does. PFR already exposes natural audit targets: end-to-end theorem reachability, dependency-graph visibility, and build stability in a large evolving repository [8]. Those are the same kinds of development-wide properties that our certificate layer is meant to make explicit and regression-sensitive.

The main methodological lesson is the separation of concerns enforced by the certificate architecture: local proofs can evolve independently, reflective computation can discharge finite obligations cheaply, and only the advertised artifact-level invariants must remain stable at the certificate boundary. This separation, rather than the scientific interpretation of the case-study symbols, is the paper's transferable contribution.

4 Related Work

Automated reasoning, proof evidence, and certificates. Proof certificates separate proof search from proof checking by packaging evidence in a form that a small kernel or proof checker can validate independently. Miller advocates a broad, technology-agnostic notion of proof certificates as a lingua franca for communicating and archiving proof evidence [10]. Avigad surveys the role of automated reasoning and proof assistants in large-scale mathematics and argues for artifact-centered evaluation and scalable combinations of automation with interaction [5]. Our certificates are complementary: rather than encoding object-level proof evidence for each theorem, they encode checkable predicates about the dependency structure and invariants of a large Lean development.

This positioning also differs from two adjacent traditions that should be acknowledged explicitly. First, LCF-style systems ground soundness in a small trusted kernel and theorem-producing abstract data types [11]; for a JAR-centered perspective on trustworthy proof checking and small trusted kernels, see also Appel [12]. Second, framework-style independent proof checkers such as Dedukti emphasize portable proof representations and re-checking across systems [13]. Our use of "certificate" is narrower and artifact-centered: we are not proposing an interchange format for theorem-level proof objects, but a way to package cross-declaration audit claims as named compile-time predicates inside a single Lean development.

Formalizations supporting scientific mathematics and verification.

Harrison [14] studies self-verification of HOL Light, directly relevant to trusted-kernel arguments and artifact assurance. Immler [15] develops a verified ODE solver in

Isabelle/HOL and applies it to the Lorenz attractor, illustrating end-to-end verification of nontrivial continuous mathematics. Boldo et al. [16] introduce Coquelicot, a user-friendly Coq library for real analysis, and Affeldt et al. [17] provide a Coq library formalizing core results in information theory. These works complement ours by strengthening the ecosystem of formally verified mathematics and tooling that large developments (including the present case study) rely on.

Recent Lean-based efforts in formalized science include mechanized treatments of scientific theories in chemical physics [18] and work toward reusable libraries for physics notation and high-energy physics calculations [19, 20]. These projects differ in scope and goals from the present work, which emphasizes a single end-to-end derivation chain in a heterogeneous scientific case study together with machine-checked certificates for its structural properties.

Formalizations of mathematical physics provide useful comparison points for scope and methodology. Fleuriot [21] formalized Newton's Principia in Isabelle, focusing on geometric reasoning, and the QBF project explored Coq formalizations of quantum-mechanical foundations. Our point of departure is different: we focus on how to package development-wide audit claims—such as upstream input isolation, conditional selection, sensitivity to perturbation, and forcing-chain connectivity—as explicit certificates inside a Lean artifact. The intended novelty claim is therefore methodological rather than a comprehensive priority claim about all prior formalizations of physics.

We also include a brief transferability discussion using the Lean 4 formalization of the Polynomial Freiman-Ruzsa conjecture as a comparison point (Section 3.4). Because PFR is a mainstream additive-combinatorics development with public blueprint, file-dependency, and build interfaces, it provides a cleaner comparison target for the certificate schema than the primary case study alone [4, 8, 9].

Large-scale theorem proving. The most relevant comparisons are projects that verify extended mathematical arguments: Flyspeck [2] (175k lines, one theorem), the Liquid Tensor Experiment [3] (60k lines, one theorem), and the odd order theorem [22] (170k lines, one theorem). Our formalization is smaller (32,621 lines) but spans a heterogeneous chain of reasoning—from functional equations through combinatorics and real analysis to derived closed forms collected in the case study.

Lean 4 and Mathlib. Lean 4 [23] provides a dependent type theory with a small trusted kernel. Mathlib [24] supplies the analysis infrastructure (real numbers, logarithms, power functions, irrationality proofs) that this formalization relies on. Our use of Mathlib's `Real.goldenRatio_irrational` for the irrationality of φ is an example of effective library reuse.

5 Scope and Limitations

1. **The RCL is a premise, not a conclusion.** The Lean proofs establish conditional consequences of the Recognition Composition Law together with the stated regularity, normalization, and calibration assumptions. Whether those premises describe nature is outside the scope of formal verification.
2. **The repository is globally sorry-free and uses no custom axioms beyond Lean 4 and Mathlib.** Every theorem in the current artifact is fully

proved. The trusted base is therefore Lean 4's kernel, Mathlib, and the explicitly stated modeling premises.

3. **Some object-level facts are library reuse, not new formal contributions.** In particular, the irrationality of ϕ is obtained from Mathlib's `Real.goldenRatio_irrational`. The paper's contribution is not a new proof of such facts, but the way they are exposed, classified, and reused inside an artifact-level audit framework.
4. **Several uniqueness claims are family-relative.** Results such as the identification of the gap-law base parameter are proved only within explicitly stipulated functional families and calibration conditions. We do not claim a stronger unrestricted uniqueness or selection theorem.
5. **Physical interpretation is external to Lean.** Lean verifies type-correct mathematical propositions about the formalized constants and definitions. It does not verify that a framework-native symbol is the Einstein gravitational coupling, that a chosen integer characterization discovers the dimensionality of nature, or that the resulting formulas are empirically adequate.
6. **Link 6 is only partially closed at the provenance level.** The canonical α^{-1} pipeline is certified symbolically and numerically: Lean checks the exponential assembly formula and the interval bounds reported in Section 2. What remains incomplete is a stronger proof identifying the raw DFT-based candidate gap weight with the canonical closed-form projection weight used for w_8 in the certified pipeline.
7. **A transferability comparison reduces, but does not eliminate, case-study risk.** The Recognition Science artifact remains the primary heterogeneous benchmark for stress-testing certificate design, and we supplement it with a brief transferability discussion using the Lean 4 formalization of the Polynomial Freiman-Ruzsa conjecture as a comparison point (Section 3.4). This supports the claim that the certificate grammar is not domain-specific. Even so, the evaluation remains modest: we do not report a second implemented benchmark or a full re-implementation of the certificate layer across multiple external repositories.
8. **The object-level mathematics is modest.** The main difficulty of the artifact lies in packaging cross-cutting audit claims across many declarations, not in formalizing a single theorem of Flyspeck-scale mathematical depth. Our methodological claims should be read accordingly.
9. **Reproducibility metadata must be complete at submission.** A stable public repository location, commit hash, Lean toolchain version, and Mathlib revision are part of the minimum information needed to support the published claims. Placeholder text is not acceptable in a submission version.

6 Conclusion

We have presented a Lean 4/Mathlib method for packaging selected development-wide audit claims as explicit certificates checked at compile time. In the artifact studied here, 36 certificates summarize conditional selection results, upstream input-isolation claims, sensitivity-to-perturbation claims, and derivation-connectivity claims across

1,490 theorem/lemma declarations in 175 Lean files. The repository is globally `sorry`-free and uses no custom axioms beyond Lean 4's kernel and Mathlib; in particular, the d'Alembert smoothness step is now proved internally rather than imported as a separate axiom. The main contribution is therefore a practical one: a way to make selected global structural claims about a proof development explicit, checkable, and regression-sensitive.

For automated reasoning, the main lesson is that certificates provide a useful complement to declaration-local tools such as `#print axioms` and ordinary CI builds. They do not replace those tools; they sit one level above them, at the level of advertised development-wide invariants. The scientific case study is included as a demanding proof-engineering benchmark, while the comparison with PFR supports the narrower claim that the certificate grammar is portable to a mainstream Lean 4 artifact.

Declarations

Data availability.

No experimental or observational datasets were generated or analyzed in this study. The formal proof artifact, source code, and build metadata are publicly available in the repository identified below.

Code and artifact availability.

The source repository for the artifact analyzed in this paper is available at https://github.com/jonwashburn/recognition-science. The snapshot analyzed in this paper corresponds to Git commit `0f9fcbe8`, Lean toolchain `leanprover/lean4:v4.29.0-rc6`, and Mathlib commit `d7ea5678`. To reproduce the verification result, run `lake exe cache get` followed by `lake build` from the artifact root. A successful build checks all files, including the certificate structures in `Verification/`, with zero `sorry` placeholders. The repository includes the `lean-toolchain` file and build instructions.

Competing interests.

The authors declare no competing interests.

Funding.

No funding was received for conducting this study.

Author contributions.

Jonathan Washburn: conceptualization, methodology, formalization, writing—original draft, and writing—review and editing. Megan Simons: writing—review and editing, project administration.

References

[1] Gonthier, G.: A computer-checked proof of the four colour theorem. Unpublished manuscript, Microsoft Research, Cham (2005)

[2] Hales, T., *et al.*: A formal proof of the Kepler conjecture. Forum of Mathematics, Pi **5**, 2 (2017)

[3] Scholze, P.: Liquid tensor experiment. Experimental Mathematics **31**(2), 349–354 (2022)

[4] Tao, T., et al.: A formalization of the Polynomial Freiman–Ruzsa Conjecture in Lean. Preprint (2023)

[5] Avigad, J.: Automated reasoning for mathematics. In: Benzmüller, C., Heule, M.J.H., Schmidt, R.A. (eds.) Automated Reasoning (IJCAR 2024). Lecture Notes in Computer Science (LNAI), vol. 14739, pp. 3–20. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-63498-7_1

[6] Washburn, J., Zlatanović, M., Allahyarov, E.: Recognition geometry. Axioms **15**(2), 90 (2026) https://doi.org/10.3390/axioms15020090

[7] Washburn, J., Allahyarov, E.: Particle Masses Spectrum from Harmonic Cascade Principles (2025). https://arxiv.org/abs/2506.12859

[8] The Polynomial Freiman–Ruzsa Project: The Polynomial Freiman–Ruzsa Conjecture. https://teorth.github.io/pfr/. Project website and build documentation; accessed 2026-03-13 (2026)

[9] Tao, T.: Formalizing the proof of PFR in Lean4 using Blueprint: a short tour. https://terrytao.wordpress.com/2023/11/18/formalizing-the-proof-of-pfr-in-lean4-using-blueprint-a-short-tour/. Blog post; accessed 2026-03-13 (2023)

[10] Miller, D.: Communicating and trusting proofs: The case for foundational proof certificates. Preprint, HAL hal-00772727 (2013). https://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/clmps2011.pdf

[11] Gordon, M.J., Milner, A.J., Wadsworth, C.P.: Edinburgh LCF: A Mechanized Logic of Computation. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg (1979). https://doi.org/10.1007/3-540-09724-4

[12] Appel, A.W.: A trustworthy proof checker. Journal of Automated Reasoning **31**(3–4), 231–260 (2003) https://doi.org/10.1023/B:JARS.0000021013.61329.58

[13] Boespflug, M., Carbonneaux, Q., Hermant, O.: The $\lambda\Pi$-calculus modulo as a universal proof language. In: Pichardie, D., Weber, T. (eds.) Proceedings

of the 1st International Workshop on Proof Exchange for Theorem Proving (PxTP 2012). CEUR Workshop Proceedings, vol. 878, pp. 28–43 (2012). https://ceur-ws.org/Vol-878/paper2.pdf

[14] Harrison, J.: Towards self-verification of HOL Light. In: International Joint Conference on Automated Reasoning (IJCAR 2006). Lecture Notes in Computer Science, vol. 4130, pp. 177–191. Springer, Cham (2006)

[15] Immler, F.: A verified ODE solver and the Lorenz attractor. Journal of Automated Reasoning **61**, 73–111 (2019)

[16] Boldo, S., Lelay, C., Melquiond, G.: Coquelicot: A user-friendly library of real analysis for Coq. Mathematics in Computer Science **9**(1), 41–62 (2015)

[17] Affeldt, R., Gaberón, M., Saikawa, T.: A library for formalization of information theory in Coq. Journal of Automated Reasoning **64**, 527–563 (2020)

[18] Bobbin, M.P., *et al.*: Formalizing chemical physics using the Lean theorem prover. Digital Discovery (2024) https://doi.org/10.1039/D3DD00077J

[19] Tooby-Smith, J.: Heplean: Digitalising high energy physics. Computer Physics Communication **308**, 109457 (2025) https://doi.org/10.1016/j.cpc.2024.109457

[20] Tooby-Smith, J.: Formalization of physics index notation in Lean 4. arXiv:2411.07667 (2024). https://arxiv.org/abs/2411.07667

[21] Fleuriot, J.: A Combination of Geometry Theorem Proving and Nonstandard Analysis with Application to Newton's Principia. Springer, Cham (2001)

[22] Gonthier, G., *et al.*: A machine-checked proof of the odd order theorem. In: Interactive Theorem Proving (ITP 2013). Lecture Notes in Computer Science, vol. 7998, pp. 163–179. Springer, Cham (2013)

[23] Moura, L., Ullrich, S.: The Lean 4 theorem prover and programming language. In: CADE-28. Lecture Notes in Computer Science, vol. 12699, pp. 625–635. Springer, Cham (2021)

[24] mathlib Community: The Lean mathematical library. In: Certified Programs and Proofs (CPP 2020), pp. 367–381 (2020)