AI-Assisted Formal Mathematics with Lean 4

A potential paradigm shift in mathematical research

Xingyu Zhong

Beijing Institute of Technology

2026-06-03

Table of Contents

Table of Contents

Why this talk, why now

Two things changed rapidly in 2025–2026.

- LLMs became much better at code and symbolic reasoning. Companies doubled down on proving abilities.
- MCP, skills, memories, and tool access. Coding agents became able to read projects, edit files, run commands, and react to errors.

This is especially good news for functional programming:

- it tends to provide unusually rich compiler feedback than traditional languages.
- Lean does not merely say "try again". It shows goals, assumptions, expected types, elaboration errors, and library names.

That feedback is exactly what an agent can use.

A motivating example

In a recent project[1] on nilpotent orbits in classical Lie algebras, we used Lean 4 to verify a small combinatorial core:

- admissible partitions
- dominance order
- cover relations

The point of the example is not the Lie theory. The point is that some proof fragments are:

- annoying but routine
- full of cases and boundary conditions
- easy for humans to leave gaps in
- of light infrastructure boilerplate

Nobody wants to do them by hand. No reviewer wants to check them by hand. But they are necessary for the overall proof.

[1] https://github.com/sun123zxy/nilpotent-orbit-classical-formalization/

The heavy lifting

- A formalization confirms the correctness of the statements and proofs. It serves as an ultimate guarantee that the result is true.
- During the formalization process, you may discover gaps in your proof, find mistakes, and clarify your ideas. It can be a valuable learning experience.
- However, formalization is often tedious and time-consuming. It requires writing detailed code, handling edge cases, and dealing with the proof assistant's feedback.
- Now coding agents can do the heavy lifting for you.

Strikingly, in the end we were able to formalize the core:

- in merely 2 days
- cost about 3 dollars of Codex subscription
- 5000+ lines of Lean code

Quick, good, and cheap — all at once!

A formal anecdote

On June 2, 2026, an arXiv paper[2] appeared claiming a Lean 4 machine-verified proof of `P = NP`.
The repository looked reassuring at first glance.

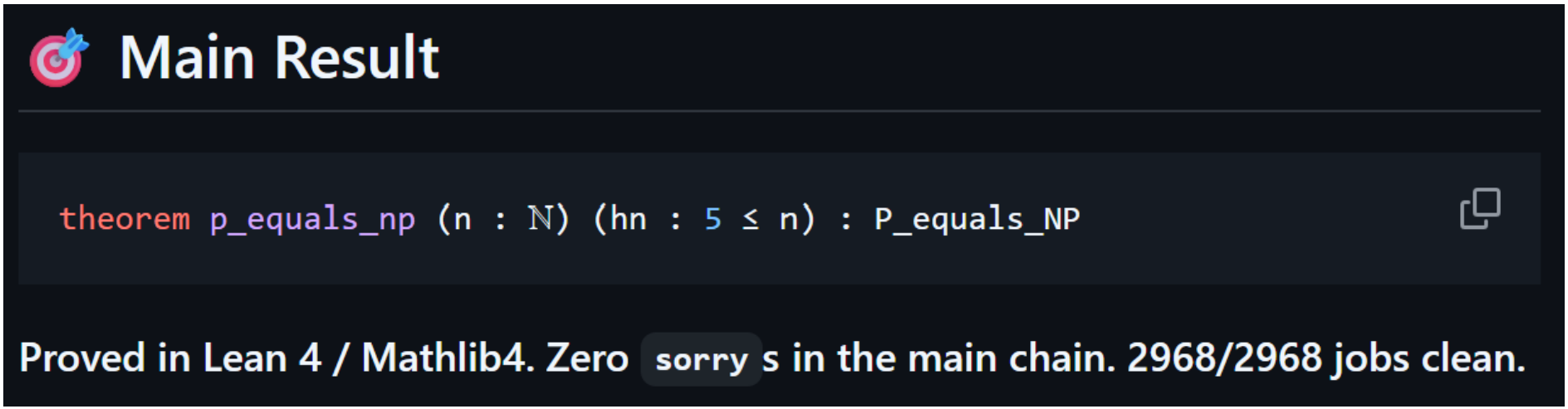

Figure: No sorries, build succeeded, zero errors

[2] https://arxiv.org/abs/2606.03194

Proof by define it as True

Then people looked into the details and things went witty.

Figure: Issue #1

Lean did not prove `P = NP`. Lean proved `True`.

No silver bullet

The above formal anecdote vividly illustrates what we should not expect from AI-assisted formalization:

- Zero `sorry`s is not enough.
- Zero `axiom`s is not enough.
- AI-assisted formalization is no silver bullet.
- It still relies on mathematicians to steer the direction, review the meaning, and certify the result.

Table of Contents

The interactive loop

It's not a single prompt shot of "give me the proof of ...". It's a loop:

```
human chooses the target
  -> agent writes or edits Lean
  -> Lean reports goals and errors
  -> agent revises the code
  -> human steers the high-level direction
```

More over, it's highly interactive:

- Human interacts with agent
- Agent interacts with Lean

Three roles

Mathematician	AI agent	Lean
chooses the target	comprehense libraries	checks proof terms
provides proof sketches	drafts code	reports goals and errors
controls abstraction	handles boilerplate	rejects invalid proofs
reviews meaning	iterates quickly	certifies the result

Table: Three roles

- Human provides insights
- AI does the heavy lifting
- Lean guarantees correctness
- Every component plays a crucial role

Agent tooling

A useful Lean agent should be able to:

- interact with OS: read and edit Lean files
- interact with Lean: query goals, run commands, inspect diagnostics, try tactics
- RAG: look up documentation and search the library

Lean LSP and MCP-style tooling make this loop practical.

- Without tool access, the model is mostly guessing.
- With tool access, it knows the evolving library. It can work against the real proof state.

Table of Contents

Natural language vs. formal language

- ambiguity in natural language
  - implicit assumptions
  - skipping details: "It's clear that we have..."
  - "viewed as" arguments: $V^{**} = V$, $(A \times B) \times C = A \times (B \times C)$ [3]
  - abuses of notation: $3 \in \mathbb{Z}/5\mathbb{Z}$, $\mathbb{C} \subseteq \mathbb{C}[x]$
- precision in formal language
  - computer programs are formal languages

[3] Knowledgable audience may recognize them as examples of natural isomorphisms in category theory.

Mathematical proofs vs. Computer programs[4]

Logic	Programming
proposition	type
proof	term
proposition is true	type has a term
proposition is false	`p → False` has a term
logical constant `True`	unit type
logical constant `False`	empty type
implication $\to$	function type
conjunction $\wedge$	product type $\prod$
disjunction $\vee$	sum type $\sum$
universal quantification $\forall$	dependent product type $\prod$
existential quantification $\exists$	dependent sum type $\sum$

Table: Curry–Howard correspondence

[4] see also Computational Trilogy, with category theory as the third vertex

Set theory vs. Type theory

- Mathematicians choose axiomatic set theory (with first-order logic) as the foundation of mathematics.
  - naive set theory fits human's intuition well
- Type theory is an alternative foundation that is equally expressive, but more suitable for computer formalization.

Set Theory	Type Theory
everything is a set	everything has a type
$3 \in \mathbb{R}$ is a proposition	$(3 : \mathbb{R})$ is a typing judgment
$\mathbb{Q} \subseteq \mathbb{R}$ is an inclusion	$\mathbb{Q} \to \mathbb{R}$ is a type conversion

What is Lean 4

- A modern functional programming language designed for theorem proving

  "Lean is based on a version of dependent type theory known as the Calculus of Constructions, with a countable hierarchy of non-cumulative universes and inductive types." — *Theorem Proving in Lean 4*

Lean's dependent type theory

- Dependent type theory is a powerful extension of type theory where
  - types may depend on terms "given before" them
  - first-order logic can be implemented in dependent type theory
- functions, inductive types and quotient types[5] are the basic methods to construct new types.

Set Theory	Lean's dependent type theory
$\forall x \in \mathbb{R},\ x^2 \geq 0$	has type $(x : \mathbb{R}) \to (x^2 \geq 0)$
$(n \in \mathbb{N}) \mapsto (1, 0, \ldots, 0) \in \mathbb{R}^n$	has type $(n : \mathbb{N}) \to \mathbb{R}^n$
$\{0, 1\} = 2$ is a set equality	make no sense
cardinality is an equivalence class	is a quotient type
Russell's paradox	Girard's paradox

[5] Though seemingly redundant, there are reasons for making quotient types as a fundamental constructing method. funext thesis

An example Lean 4 code

```
theorem FLT (n : ℕ) (hn : n > 2) (a b c : ℕ) :
    a ≠ 0 → b ≠ 0 → c ≠ 0 → a^n + b^n ≠ c^n := by
  sorry

def TendsTo (a : ℕ → ℝ) (t : ℝ) : Prop :=
  ∀ ε > 0, ∃ n₀ : ℕ, ∀ n, n₀ ≤ n → |a n - t| < ε

theorem tendsTo_sandwich {a b c : ℕ → ℝ} {L : ℝ}
    (ha : TendsTo a L) (hc : TendsTo c L)
    (hab : ∀ n, a n ≤ b n) (hbc : ∀ n, b n ≤ c n) :
    TendsTo b L := by
  sorry
```

Closing

AI-assisted formalization is not:

- AI replacing mathematicians
- Lean replacing natural language mathematical writing
- fully automatic theorem proving for everything

It is:

- Lean as a rigorous proof checker
- LLMs and coding agents exploiting rich compiler feedback
- humans steering the mathematical goal

This combination may become the new paradigm for mathematical research.

Resources

- Lean: https://lean-lang.org/
  Our proof assistant of choice.
- Mathlib: https://github.com/leanprover-community/mathlib4
  The mathematics library for Lean 4.
- Mathematics in Lean:
  https://leanprover-community.github.io/mathematics_in_lean/
  A tutorial focusing on mathematical examples and the design of `mathlib`.
- Introduction to Formal Mathematics with Lean 4:
  https://github.com/sun123zxy/lean4-formal-math-intro
  Bottom-up, steady-paced, "Illustrate the theory"-flavored tutorial for mathematicians.
- Lean LSP MCP: https://github.com/oOo0oOo/lean-lsp-mcp
  The glue between Lean and agents.