BGSMath Graduate Course: Formalizing Mathematics in Lean with AI

Session 3: AI tools

Michail Karatarakis

17 November 2026

Radboud University Nijmegen & Axiomatic AI

The course

	Session	Content
1	Lean and Mathlib	What formalizing means, and does not mean; first proofs; finding things in Mathlib; reading a Lean statement
2	From paper to Lean	Writing definitions and statements; the usual mistakes; drafting statements with AI and checking them
3	AI tools	What AI did this year; an agent proving a statement, live; checking what it hands back
4	Your own theorem	Statements sent in by participants, formalized live; how to do your own at home

Each session: a lecture, with the Lean on the slides. You run it yourself, with the exercises, at home.

Last time, and today

Last time: writing

> Before anyone proves a statement, **try to break it**.

- Your own definitions and statements.
- Four tests: read it back, small cases, check the hypotheses, compare.

Today: proving, with AI

> The AI writes the proof.
> You check **what** it proved.

- What AI did this year, and what was checked.
- An agent proving a statement, step by step.
- Seven things Lean accepts that are not what you asked for.

Four words for today

Model	The AI that writes text: Claude, ChatGPT, Gemini. You ask, it answers once.
Agent	A model that can use tools, in a loop. It writes a file, runs Lean, reads what Lean says, and tries again, without you.
Formalization	A paper's statements and proofs, written in Lean and checked by Lean. *Autoformalization*: an AI does the writing.
Kernel	The small program inside Lean that checks every proof (session 1). It is the only part you have to trust.

> An agent is a model that can run Lean and read the answer.

What happened this year

AlphaProof

- A system from Google DeepMind that trains itself to prove mathematical statements in Lean.
- It is based on reinforcement learning: each proof it finds and verifies is used to train it further.
- For the competition, the problems were translated by hand into formal language. Its companion for geometry is AlphaGeometry 2.
- Together they reached the level of a silver medallist at the 2024 International Mathematical Olympiad.

> People stated the problems. The machine found the proofs.

Google DeepMind, *AI achieves silver-medal standard solving International Mathematical Olympiad problems*, 25 July 2024.

Seed-Prover

- ByteDance Seed's prover. It writes proofs in Lean and uses Lean's answers to improve them.
- For geometry it has a companion engine, Seed-Geometry.
- Its proofs for the 2025 International Mathematical Olympiad are public on GitHub.
- A second version, Seed-Prover 1.5, followed in December 2025.

> Another prover built around the same loop: write in Lean, read what Lean says, try again.

github.com/ByteDance-Seed/Seed-Prover; arXiv:2507.23726 and arXiv:2512.17260.

AxiomProver

- Axiom Math's prover, in their words "an autonomous multi-agent ensemble theorem prover" for Lean.
- Its proofs for the 2025 Putnam competition are public on GitHub.
- The repository comes with one command that compiles the proofs and checks them with SafeVerify, a checker of the same kind as the comparator.
- The mathematicians on the team include Evan Chen, Ken Ono, Kenny Lau and Jujian Zhang.

> The proofs are published together with the command that checks them.

github.com/AxiomMath/Putnam2025.

Fermat's Last Theorem

- Claude "worked largely autonomously over 11 days": **13 million lines** of Lean, with 29,500 intermediate theorems.
- It followed a simplified version of Wiles's proof, and adapted pieces of Buzzard's FLT project and of `flt-regular`.
- A team of agents did the work, with Prove2Me and a multi-agent harness built on Claude Code.
- "Mathematical input from humans was limited to occasional high-level instructions [...]"

How the result was checked

"The finished proof was checked by Lean; it uses just Lean's three standard axioms, and a comparator confirmed that the theorem's statement matches Mathlib's own statement of FLT."

Anthropic, *Formalizing Fermat's Last Theorem*, 4 September 2026.

The statement behind the headline

```
#print FermatLastTheorem
-- def FermatLastTheorem : Prop :=
--   ∀ n ≥ 3, FermatLastTheoremFor n

-- The same, written out:
example : FermatLastTheorem ↔
    ∀ n ≥ 3, ∀ a b c : ℕ,
      a ≠ 0 → b ≠ 0 → c ≠ 0 →
      a ^ n + b ^ n ≠ c ^ n :=
  Iff.rfl
```

- This is Mathlib's statement of Fermat's Last Theorem. It is the one from session 1.
- The 13 million lines are the proof. Lean read them, so that nobody else has to.
- The **comparator** is a program, now part of Lean (`lake comparator`). It checks three things: the same statement, no extra axioms, accepted by the kernel.

> Three lines for people to read. Thirteen million for Lean.

The unit distance conjecture

28 May	Logical Intelligence's Aleph Prover formalizes the disproof of the planar unit distance problem in Lean, *conditional* on two textbook theorems that Mathlib does not have yet: the Golod–Shafarevich inequality, and Shafarevich's rank-relation bound. 33,087 lines.

Logical Intelligence, *Aleph Prover*, blog post, 28 May 2026.

What “conditional” means

hands up

A known theorem, used but not proved

Every $n > 0$ in $\mathbb{N}$ has a predecessor.

```
-- Assumed, not proved.
def Known : Prop :=
  ∀ n : ℕ, ∃ m, m + 1 = n

theorem new_result (h : Known) :
    (2 : ℕ) + 2 = 5 := by
  obtain ⟨m, hm⟩ := h 0
  omega
```

Lean accepts this file. Is $2 + 2 = 5$?

What “conditional” means

hands up

A known theorem, used but not proved

Every $n > 0$ in $\mathbb{N}$ has a predecessor.

```
-- Assumed, not proved.
def Known : Prop :=
  ∀ n : ℕ, ∃ m, m + 1 = n

theorem new_result (h : Known) :
    (2 : ℕ) + 2 = 5 := by
  obtain ⟨m, hm⟩ := h 0
  omega
```

Lean accepts this file. Is $2 + 2 = 5$?

- The paper says $n > 0$. The Lean line does not: take $n = 0$.
- So `Known` is false, and a false assumption proves anything.
- `#print axioms new_result` shows nothing unusual: an assumption is not an axiom.

caught by: small cases — An assumed statement is proved by nobody. Read it.

Gauss: sphere packing and the prime number theorem

- **Strong prime number theorem** (September 2025). A challenge set by Tao and Kontorovich in January 2024. Math Inc's agent Gauss completed it "after three weeks of effort": about 25,000 lines. It relied on "natural language scaffolding supplied by human mathematicians".
- **Sphere packing** (February 2026). A project started by Hariharan and Viazovska in 2024. According to Math Inc, Gauss proved the remaining results for dimension 8 "in just 5 days", then did dimension 24 in two weeks, "using only the original paper as input".
- **Access.** In beta testing with "a select group of mathematicians"; you can register for early access.

Math, Inc., *Introducing Gauss*; *Sphere packing*.

OpenAI: a public repository of research mathematics

- OpenAI has published hundreds of mathematical manuscripts, written by an internal model working on open research problems.
- Many of the results come with a Lean formalization, and there is a catalogue of which ones.
- The repository explains how to check the Lean proofs yourself, with the comparator.

> Where a result has a Lean proof, you can check it yourself.

github.com/openai/math.

The same question, outside mathematics

All of that was mathematics. The same thing is happening to software.

- Tests can be gamed: a system that sees them can hard-code what passes. A proof cannot.
- Kim Morrison had Claude write `zlib`, the library behind `gzip`, in Lean and prove it correct: no `sorry`, more than 1,100 theorems.

Zip/Spec/ZlibCorrect.lean: decompress what you compressed, and you get it back

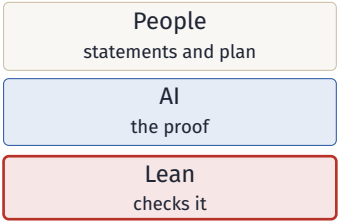
```
theorem zlib_decompressSingle_compress (data : ByteArray) (level : UInt8)
    (maxOutputSize : Nat) (hsize : data.size ≤ maxOutputSize) :
    ZlibDecode.decompressSingle (ZlibEncode.compress data level) maxOutputSize = .ok data
```

> Here the statement is called a **specification**. Checking it is still your job.

L. de Moura, *When AI Writes the World's Software, Who Verifies It?*, 28 February 2026. zlib: github.com/kim-em/lean-zip.

What these have in common

- The **statement** was fixed before the proof: Mathlib's FLT, the paper's theorem, a plan written by people.
- The AI wrote the **proof**: millions of lines that nobody reads.
- Lean, and a comparator, did the **check**.



> The same division of work fits one theorem of yours. The rest of today shows how.

The tools

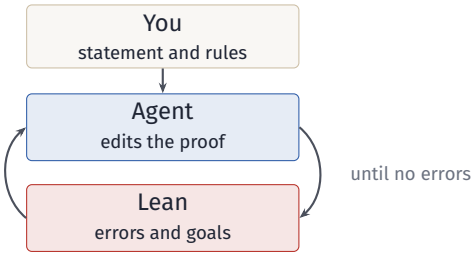
The tools, November 2026

Tool	What it is	Can you use it?
Claude Code	A coding agent: reads your project, edits files, runs commands.	Yes, with a paid plan.
Prove2Me	An open platform: a paper becomes a *mission*, and anyone's agent can contribute proofs.	Yes, free: prove2.me. You bring your own agent.
Tau Ceti	A Lean library downstream of Mathlib: humans write the roadmaps, AI writes the code.	Through its roadmaps.
Open Math OS	SAIR's platform for mathematicians: open-weight models, and a machine per project with Lean and Mathlib.	Free accounts for academic requesters: openmathmodel.org.
Gauss	Math Inc's autoformalization agent.	Register for early access at math.inc.
Aristotle	Harmonic's prover. Today it works on code (Rust, Verilog, C), through Lean.	Pay for what you use.

Product pages and documentation, read on 15 and 17 November 2026. This changes fast: check before you rely on it.

A coding agent

- You give it a statement and rules. It writes a proof, runs Lean, and reads the errors. Then it tries again.
- Lean answers every attempt exactly: accepted, or an error at a line. That is why agents do well on proofs.
- It can work for a long time without you. At the end, you check the result.



You wrote a proof in session 1 by reading the infoview. The agent does the same, faster.

Setting one up for Lean

Whichever agent you use, you need three things:

1. A Lean project with Mathlib (session 1).
2. A coding agent allowed to edit files and run commands there, such as Claude Code.
3. Optional: **lean-lsp-mcp**, which shows it the infoview. With Claude Code:

Terminal

```
claude mcp add lean-lsp uvx lean-lsp-mcp
```

What the agent can then do

- See the goal at any line, and the errors.
- Search Mathlib: LeanSearch, Loogle, local search.
- Check which axioms a proof uses.

MCP is a standard way to give an agent a new tool. Here the tool is Lean's infoview.

> Without step 3 the agent runs Lean on the whole file and reads the output.

github.com/oOo0oOo/lean-lsp-mcp. The setup sheet of session 1 has the same steps, with screenshots.

Pin the statement

```
-- Yours. The agent may not edit it.
def Statement : Prop :=
  ∀ n : ℕ, ∑ i ∈ Finset.range (n + 1), i
    = n * (n + 1) / 2

-- The agent's.
theorem main : Statement := by
  intro n
  rw [Finset.sum_range_id,
    Nat.add_sub_cancel,
    mul_comm]
```

- `def Statement` gives your statement a name. It is warm-up C of session 2, tested.
- The agent may write one thing: a proof of `theorem main : Statement`.
- If it makes the statement easier, `main` no longer proves `Statement`, and Lean says so.

The statement is yours. The proof is the agent's. Keep them apart.

Rules for the agent

	Rule	Why
1	Never edit the statement.	Or it proves something easier.
2	No `axiom`, `sorry` or `native_decide`.	Each makes Lean accept a proof that is not one.
3	Search Mathlib before proving a lemma.	Most of what you need is there.
4	Write the plan in English first.	A wrong plan costs a paragraph, a wrong proof an hour.
5	If you cannot prove it, stop and say why.	A stuck agent is tempted by rules 1 and 2.

> They go in a file in the project folder that the agent reads every time: `CLAUDE.md` for Claude Code, `AGENTS.md` for many other agents.

A run (1): my part

recorded run

Theorem

For every real number $x > 0$, $x + \dfrac{1}{x} \geq 2$.

```
-- Mine. The agent may not edit it.
def Statement : Prop :=
  ∀ x : ℝ, 0 < x → 2 ≤ x + 1 / x
```

What I type to the agent

Prove `theorem main : Statement` in this file. Follow the five rules. Write your plan first.

- I tested the statement first, as in session 2. $x = 1$ gives 2, and $x = \frac{1}{2}$ gives 2.5.
- The hypothesis can hold: $x = 1$.

My part ends here: a statement I trust, and five rules.

A run (2): the agent's plan

recorded run

The agent's plan, word for word

For $x > 0$ the claim $2 \le x + 1/x$ is equivalent, after multiplying by x, to $2x \le x^2 + 1$, that is $0 \le (x-1)^2$, which holds because a square is non-negative.

- This is the proof you would give at the board.
- Rule 3: it then searched Mathlib for the inequality and found no lemma. Its words: "That is a search by name and by text, not a proof that Mathlib lacks it."

> Read the plan before the proof. It is in your language. If it is wrong, stop here.

A Claude Code agent, 15 November 2026. Full log: `notes/session3-agent-run/` in the course repository.

A run (3): the agent's proof

recorded run

```
theorem main : Statement := by
  intro x hx
  have hx0 : x ≠ 0 := ne_of_gt hx
  have h : x + 1 / x = (x ^ 2 + 1) / x := by
    field_simp
  rw [h, le_div_iff₀ hx]
  nlinarith [sq_nonneg (x - 1)]
```

`intro`	Take x, and $x > 0$.
`field_simp`	$x + \frac{1}{x} = \frac{x^2+1}{x}$.
`le_div_iff₀`	Multiply by x. To show: $2x \le x^2 + 1$.
`nlinarith`	Arithmetic, with the hint $(x-1)^2 \ge 0$.

- It is the plan, line by line. Lean accepted it on the first run.
- Then the check: `#print axioms main` shows only the three standard axioms.

Statement unchanged, no `sorry`, standard axioms. Done.

A run (4): what Lean says when a step is wrong

recorded run

The last line, changed by me

```
rw [h, le_div_iff₀ hx]
linarith
```

Lean says

```
error: linarith failed to find a contradiction
x : ℝ
hx : 0 < x
hx0 : x ≠ 0
h : x + 1 / x = (x ^ 2 + 1) / x
a : x ^ 2 + 1 < 2 * x
⊢ False
```

- The agent's proof had no error. I removed its hint, to show you one.
- Lean supposes the claim fails: $x^2 + 1 < 2x$.
- `linarith` knows linear arithmetic only. It needs a square: $(x-1)^2 \geq 0$.
- An agent reads this, adds the hint, and runs Lean again.

An error is an exact answer: the line, and what is still to prove.

Eight statements, one agent

recorded run

Statement, pinned by me	What the agent did
The product of two odd numbers is odd	first attempt: Mathlib has it
$x + \frac{1}{x} \geq 2$ for real $x > 0$	first attempt
$1 + 3 + \cdots + (2n-1) = n^2$	first attempt: induction
$\frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \cdots + \frac{1}{n(n+1)} = \frac{n}{n+1}$	first attempt
$2^n < n!$ for $n \geq 4$	first attempt: induction from 4
$\sqrt{2} + \sqrt{3}$ is irrational	first attempt: 15 lines
No rational number has square 2	first attempt
$\frac{a}{b} \cdot b = a$ for all rationals *(false)*	**refused**, and proved it false

> For textbook statements, the proof is no longer the hard part. The statement is.

Two recorded runs, 15 November 2026; logs in `notes/session3-agent-run/`. Your own statements will be harder.

A false statement: what should the agent do?

hands up

```
def Statement : Prop :=
  ∀ a b : ℚ, a / b * b = a

theorem main : Statement := by
  sorry
```

Warm-up B of session 2, without the fix: for $b = 0$ the left side is 0. I gave it to the agent, with the same five rules.

A Try harder, until Lean accepts something.

B Add $b \neq 0$, and prove that.

C Stop, and tell me the statement is false.

A false statement: what should the agent do?

hands up

```
def Statement : Prop :=
  ∀ a b : ℚ, a / b * b = a

theorem main : Statement := by
  sorry
```

Warm-up B of session 2, without the fix: for $b = 0$ the left side is 0. I gave it to the agent, with the same five rules.

A Try harder, until Lean accepts something.

B Add $b \neq 0$, and prove that.

C Stop, and tell me the statement is false.

- **C.** That is rule 5.
- B gives a true theorem. It is still not the agent's choice: the statement is mine.

A good agent tells you when your statement is wrong. It does not repair it for you.

A false statement: what the agent did

recorded run

The agent's report, word for word

Not proved, and it cannot be: the statement is false. Counterexample $a = 1$, $b = 0$ [...] I stopped under rule 5 [...] and did not change the statement; it would become true [...] if the hypothesis $b \neq 0$ were added, but that is a decision for whoever owns the statement.

```
#eval ((1 : ℚ) / 0 * 0)
-- 0

theorem statement_is_false : ¬ Statement := by
  intro h
  have h1 := h 1 0
  norm_num at h1
```

- It saw the problem while writing its plan.
- It left `main` as it was, and added these lines.
- Lean computes 0, not 1. And Lean checks a proof that the statement is false.

Not "I could not prove it". It proved that nobody can.

Without rule 5: spot the problem

hands up

What a careless agent says

Done! The proof compiles, with no `sorry` and only the standard axioms.

```
theorem main' (a b : ℚ)
    (hb : b ≠ 0) :
    a / b * b = a :=
  div_mul_cancel₀ a hb
```

Without rule 5: spot the problem

hands up

What a careless agent says

Done! The proof compiles, with no `sorry` and only the standard axioms.

```
theorem main' (a b : ℚ)
    (hb : b ≠ 0) :
    a / b * b = a :=
  div_mul_cancel₀ a hb
```

- Everything it says is true.
- It added `hb : b ≠ 0`. This is a proof of another statement.
- Here the new statement is the right one. The agent still may not decide that: you may.
- With the statement pinned, you see it at once: the file has no proof of `main : Statement`.

caught by: compare — “It compiles” is not “it proves what I asked”.

This one is written by me, to show the pattern. It is not from the recorded run.

Prove2Me: one mission, many agents

prove2.me, mission *Sensitivity Conjecture* (Huang 2019): the goal

```
theorem sensitivity_conjecture :
    ∃ C k : ℕ, ∀ (n : ℕ) (f : BoolFunc n),
      blockSensitivity f ≤ C * (sensitivity f) ^ k := by sorry
```

- A person proposes a **mission**: a goal, its definitions, milestone lemmas. People audit these statements; every intermediate lemma below them “needs no audit at all”.
- **Read-back**: an independent agent “translates the Lean back into plain math, so auditors compare math to math”. This is test 1 of session 2.
- A lemma that is still needed “becomes a new, self-contained problem that any agent can attack independently”.

> People check the statements. After that, anyone’s agent may write the proofs.

S. Chen, K. Marwaha, X. Lu, H. Yuen, T. Peng, *Prove2Me*, arXiv:2608.28433; prove2.me/about; prove2.me/missions/Sensitivity_Conjecture.

Tau Ceti: people write the roadmap

Tau Ceti, roadmap *Primitive prime divisors and the Bang–Zsigmondy theorem*: one target

```
theorem exists_prime_orderOf_two_eq {n : ℕ} (hn : 2 ≤ n) (h₆ : n ≠ 6) :
    ∃ p : ℕ, p.Prime ∧ orderOf (2 : ZMod p) = n := sorry
```

- In words: for every $n \geq 2$ except 6, some prime p has 2 of order exactly n modulo p.
- "An AIs-welcome Lean library downstream of Mathlib: AI handle the implementation and review, humans write the roadmaps and review rubrics."
- A **roadmap** is a `README.md`, "the definitive specification of its area", with target statements like this one. Anyone can propose one; a human reviewer approves it.

> A roadmap is a list of statements, written and checked by people. The proofs are left to AI.

github.com/TauCetiProject/TauCeti and TauCetiRoadmap (folder `PrimitivePrimeDivisors`), read on 15 November 2026.

Open Math OS: a place to work

- SAIR is a non-profit that Terence Tao co-founded. In October 2026 Tao announced the soft launch of their platform for mathematicians.
- You get open-weight models, and a cloud machine for each project, with tools for mathematics such as Lean and Mathlib.
- You apply as a team rather than as a person, and the team shares its projects and chats. Free accounts are being granted to requesters from academic institutions.
- Tao mentions uses “such as medium-sized Lean formalization, or summarizing or discussing a mathematical argument or topic”.

> If you would rather not pay for an agent, this is where to try one.

openmathmodel.org; T. Tao, *SAIR Open Math OS released*, 16 October 2026.

Aristotle: Harmonic's prover

- Harmonic built it to prove things in Lean. The AI writes the proof and Lean checks every step, so nothing rests on the AI being right.
- It reached gold-medal level at the 2025 International Mathematical Olympiad, and Harmonic published the proofs.
- Today it works on programs. It proves what Rust, Verilog and C code must get right, on every commit, by compiling the code into Lean.
- Pricing: "Pay for what you use".

> The prover that did olympiad problems last year now reads Rust.

harmonic.fun/aristotle; harmonic.fun/news/imo-ios.

Leanstral: a prover you can download

- Mistral's model for Lean. Version 1.5 came out in July 2026.
- It is free, under the Apache 2.0 licence: the weights are on Hugging Face, and there is a free API.
- It is large, 119 billion parameters with 6 billion active at a time, so running it yourself needs a big machine. The API does not.

> The first prover in this lecture whose weights you can download.

Mistral, *Leanstral 1.5*, 2 July 2026.

Formal Conjectures: open problems, already stated

- A Lean library from Google DeepMind: “a collection of formalized statements of conjectures”. The aim is open conjectures where only the statement is formalized.
- The problems come from textbooks and research papers, including arXiv, from Wikipedia, MathOverflow and the OEIS, and from problem lists.
- To contribute, you open an issue describing the conjecture, then a pull request. They rely on careful human review, and plan to use AlphaProof to help find misformalizations.

> A library of open problems, already stated in Lean. A good place to practise reading statements.

github.com/google-deepmind/formal-conjectures; arXiv:2605.13171.

Checking what comes back

Lean accepted it. Now what?

Lean accepting a file means less than it seems. Seven things it accepts:

The proof is not a proof

- a new `axiom`: the result, simply assumed;
- a `sorry`, hidden inside a longer proof;
- `native_decide`: a computation the kernel does not check.

One command finds all three.

A real proof, of another statement

- the wrong type;
- the claim, assumed as a hypothesis;
- hypotheses that cannot all hold;
- one case in place of all of them.

No command finds these. The four tests of session 2 do.

> None of the seven is an error. You find them by looking.

Spot the problem (1)

hands up

```
axiom helpful_fact : (2 : ℕ) + 2 = 5

theorem looks_fine : (2 : ℕ) + 2 = 5 :=
  helpful_fact

#print axioms looks_fine
-- depends on axioms: [helpful_fact]
```

Spot the problem (1)

hands up

```
axiom helpful_fact : (2 : ℕ) + 2 = 5

theorem looks_fine : (2 : ℕ) + 2 = 5 :=
  helpful_fact

#print axioms looks_fine
-- depends on axioms: [helpful_fact]
```

- `axiom` states something without proof. Here it is false, and Lean still accepts `looks_fine`.
- `#print axioms` lists everything a proof rests on. Here: `helpful_fact`.

caught by: #print axioms — An axiom you did not expect needs an explanation.

Spot the problem (2)

hands up

```
theorem almost (x : ℝ) (hx : 0 < x) :
    Real.log x ≤ x - 1 := by
  have key : ∀ y > 0, Real.log y ≤ y - 1 := by
    sorry
  exact key x hx

#print axioms almost
-- depends on axioms:
--   [propext, sorryAx, Classical.choice, Quot.sound]
```

Spot the problem (2)

hands up

```
theorem almost (x : ℝ) (hx : 0 < x) :
    Real.log x ≤ x - 1 := by
  have key : ∀ y > 0, Real.log y ≤ y - 1 := by
    sorry
  exact key x hx

#print axioms almost
-- depends on axioms:
--   [propext, sorryAx, Classical.choice, Quot.sound]
```

- The key step is `sorry`. Lean warns, but in a long file the warning is easy to miss.
- `#print axioms` shows `sorryAx`: somewhere below, a proof is missing.

caught by: #print axioms

`sorryAx` in the list means: not finished.

Spot the problem (3)

hands up

```
theorem big_prime : Nat.Prime 1000003 := by
  native_decide

#print axioms big_prime
-- depends on axioms: [propext, Classical.choice,
--   Quot.sound, big_prime._native.native_decide.ax_1_1]
```

Spot the problem (3)

hands up

```
theorem big_prime : Nat.Prime 1000003 := by
  native_decide

#print axioms big_prime
-- depends on axioms: [propext, Classical.choice,
--   Quot.sound, big_prime._native.native_decide.ax_1_1]
```

- `native_decide` compiles a program, runs it, and believes its answer. The kernel does not check it.
- It shows up as an axiom with a long name. Here `norm_num` proves the same, and the kernel checks it.

caught by: #print axioms

A computation is a proof only if the kernel checked it.

What a clean proof looks like

```
theorem log_le (x : ℝ) (hx : 0 < x) :
    Real.log x ≤ x - 1 :=
  Real.log_le_sub_one_of_pos hx

#print axioms log_le
-- depends on axioms: [propext,
--   Classical.choice, Quot.sound]
```

- The statement of (2), proved by a lemma of Mathlib.
- `propext`, `Classical.choice`, `Quot.sound`: the three axioms all of Mathlib rests on. You may see fewer, never others.

These three are normal. Anything else in the list is a question to ask.

Spot the problem (4)

hands up

```
-- Paper: for every integer n, n² ≥ n.
theorem sq_ge_self (n : ℕ) :
    n ^ 2 ≥ n :=
  Nat.le_self_pow two_ne_zero n
```

Spot the problem (4)

hands up

```
-- Paper: for every integer n, n^2 ≥ n.
theorem sq_ge_self (n : ℕ) :
    n ^ 2 ≥ n :=
  Nat.le_self_pow two_ne_zero n
```

```
theorem sq_ge_self_int (n : ℤ) :
    n ^ 2 ≥ n :=
  Int.le_self_sq n

example : (-3 : ℤ) ^ 2 ≥ -3 :=
  sq_ge_self_int (-3)
```

- No `sorry`, no warning, standard axioms.
- The paper says *integer*. The Lean line says $\mathbb{N}$.
- It says nothing about -3. The negative numbers were half of the theorem.

caught by: read it back

True, and a smaller theorem than the paper's. Read every type.

Spot the problem (5)

hands up

```
-- Paper: if n² is even, then n is even.
theorem even_of_sq_even (n : ℕ)
    (h : Even (n ^ 2))
    (hn : ∃ k, n = 2 * k) :
    Even n := by
  obtain ⟨k, hk⟩ := hn
  exact ⟨k, by omega⟩
```

Spot the problem (5)

hands up

```
-- Paper: if n² is even, then n is even.
theorem even_of_sq_even (n : ℕ)
    (h : Even (n ^ 2))
    (hn : ∃ k, n = 2 * k) :
    Even n := by
  obtain ⟨k, hk⟩ := hn
  exact ⟨k, by omega⟩
```

Lean says

```
warning: Variable name `h` is not
explicitly referenced.
```

- `hn` says $n = 2k$ for some k: "n is even", the claim, in other words.
- The paper's hypothesis `h` is never used. Lean warns.

caught by: check the hypotheses

The claim is assumed. As draft 1 in session 2.

Spot the problem (6)

hands up

```
-- Paper: a strictly increasing function
-- from ℝ to ℝ is injective.
theorem injective_of_increasing
    (f : ℝ → ℝ) (hf : StrictMono f)
    (hper : ∀ x, f (x + 1) = f x) :
    Function.Injective f := by
  have h : f 0 < f (0 + 1) :=
    hf (by norm_num)
  rw [hper 0] at h
  exact absurd h (lt_irrefl _)
```

Spot the problem (6)

hands up

```
-- Paper: a strictly increasing function
-- from ℝ to ℝ is injective.
theorem injective_of_increasing
    (f : ℝ → ℝ) (hf : StrictMono f)
    (hper : ∀ x, f (x + 1) = f x) :
    Function.Injective f := by
  have h : f 0 < f (0 + 1) :=
    hf (by norm_num)
  rw [hper 0] at h
  exact absurd h (lt_irrefl _)
```

- `hper`: $f(x+1) = f(x)$. Not in the paper.
- No function is strictly increasing and periodic. The theorem applies to nothing.
- Both hypotheses are used: **no warning**.

caught by: check the hypotheses

Give one f that satisfies them all. Here there is none.

Spot the problem (7)

hands up

```
-- Paper: there are infinitely
-- many primes.
theorem many_primes :
    ∃ p, 100 ≤ p ∧ Nat.Prime p :=
  ⟨101, by norm_num, by norm_num⟩
```

Spot the problem (7)

hands up

```
-- Paper: there are infinitely
-- many primes.
theorem many_primes :
    ∃ p, 100 ≤ p ∧ Nat.Prime p :=
  ⟨101, by norm_num, by norm_num⟩
```

```
theorem many_primes_fixed :
    ∀ n, ∃ p, n ≤ p ∧ Nat.Prime p :=
  Nat.exists_infinite_primes
```

- It proves: there is a prime $p \geq 100$. One prime.
- "Infinitely many" means: above *every n* there is a prime. The `∀ n` is gone.
- No warning, standard axioms.

caught by: compare

One case in place of all. Compare word by word.

Seven things Lean accepted, and what caught them

Lean accepted	What was wrong	Caught by
The proof is not a proof.		
(1) $2 + 2 = 5$	a new axiom	`#print axioms`
(2) $\log x \leq x - 1$	a `sorry` inside the proof	`#print axioms`
(3) 1000003 is prime	`native_decide`: no kernel check	`#print axioms`
A real proof, of another statement.		
(4) $n^2 \geq n$	$\mathbb{N}$ where the paper says integer	read it back
(5) n^2 even $\Rightarrow$ n even	the claim is a hypothesis	check the hypotheses
(6) increasing $\Rightarrow$ injective	the hypotheses cannot all hold	check the hypotheses
(7) infinitely many primes	one case in place of all	compare

One command catches the first three. For the other four you need the paper.

Checking an AI's proof

	Check	How
1	The statement is unchanged.	It is pinned. The file history shows no edit to it.
2	It builds.	`lake build`: no error, and no `sorry` warning.
3	The axioms are the standard ones.	`#print axioms main`: the three, and nothing else.
4	No warning is left unread.	A hypothesis the proof never uses is a warning sign.
5	The statement passes the four tests.	Read it back, small cases, check the hypotheses, compare.

> Steps 2 and 3 are mechanical. Steps 1 and 5 are yours.

The comparator: steps 1 to 3, as a program

The check behind the FLT announcement: you give it your file with the statement, and theirs with the proof.

It checks

- the statement proved is the one in your file, declaration by declaration;
- no axiom beyond the ones you allow;
- the whole proof, replayed through the kernel.

It does not check

- that *your* statement is the right one: your file is the trusted input. That is step 5, and it stays yours;
- a definition you left open: the proof may define it so that the theorem becomes true.

`lake comparator`, in Lean since 4.35.

> The question moves to your statement file. Somebody still has to read that.

`github.com/leanprover/comparator`, README; Jason Rute, Lean Zulip, 28 December 2025.

Palomar: a registry of checked formalizations

- Since August 2026 there is a public registry for Lean formalizations, incubated by the Lean FRO and ICARM. The advisory board includes Avigad, Kra, Morrison, Tao, Vakil and Venkatesh.
- You submit a GitHub commit: a challenge file with the statement, a module with the proof, a description in words. Palomar runs the comparator with a specified version of Mathlib, then replays the proof through a second kernel written by other people, nanoda.
- Tao submitted his own formalization of Sendov's conjecture. Anything may be submitted, "whether human-generated, AI-generated, or some mixture of both".
- In its own words it is "not a peer-reviewed journal" and "not a certificate of novelty", and gives no "certification that the verified statement matches the informal statement".

> The registry stops where we do. Somebody still has to read the statement.

palomar-registry.org, pages *About* and *Statement*; T. Tao, *Palomar – a registry of Lean verified mathematics*, 18 August 2026.

At home

At home

`Exercises3.lean`, on your laptop.

1. Let an agent prove it: five statements, one at a time, with the five rules. Check every proof it hands back.
2. Audit: seven proofs “from an agent”, all accepted by Lean. Which ones prove the paper statement?
3. Your statement from session 2: pin it, and let an agent try. If it gets stuck, write down where.

- First redo today’s examples: `Session3.lean`.
- No agent? Prove part 1 yourself, with `exact?` and the tactics of session 1.
- An agent needs an account, usually paid. See the setup sheet.

> Part 2 is today’s problems again, on new statements. Not every proof there is wrong.

Next time: your own theorem

- Two or three of your statements, worked through together: statement, tests, proof, check.
- A template for doing your own at home.
- Where to go after the course: the Lean Zulip, contributing to Mathlib, larger projects.

> Before next time: send me your statement, if you have not yet. Short and elementary works best.

Sources

- Anthropic, *Formalizing Fermat's Last Theorem*, 4 September 2026. The comparator: `lake comparator`, in Lean since version 4.35.
- Unit distances: Logical Intelligence, *Aleph Prover* blog post, 28 May 2026.
- Math, Inc.: *Introducing Gauss*; *Sphere packing*.
- S. Chen et al., *Prove2Me*, arXiv:2608.28433; prove2.me.
- Tau Ceti: github.com/TauCetiProject.
- lean-lsp-mcp: github.com/oOo0oOo/lean-lsp-mcp. Aristotle: harmonic.fun/aristotle.
- The agent runs on the slides: `notes/session3-agent-run/` in the course repository.