BGSMath Graduate Course: Formalizing Mathematics in Lean with AI

Session 2: From paper to Lean

Michail Karatarakis

12 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 live demonstrations in Lean. You redo them, and the exercises, at home.

Warm-up: Lean accepts all three. How many are right?

hands up

A For every x, $x - 1 < x$.

```
example : ∀ x, x - 1 < x := sorry
```

B $\frac{a}{b} \cdot b = a$.

```
example (a b : ℚ) : a / b * b = a := sorry
```

C $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$.

```
example (n : ℕ) :
    ∑ i ∈ Finset.range n, i = n * (n + 1) / 2 := sorry
```

Warm-up: Lean accepts all three. How many are right?

hands up

A For every x, $x - 1 < x$.

```
example : ∀ x, x - 1 < x := sorry
```

B $\frac{a}{b} \cdot b = a$.

```
example (a b : ℚ) : a / b * b = a := sorry
```

C $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$.

```
example (n : ℕ) :
    ∑ i ∈ Finset.range n, i = n * (n + 1) / 2 := sorry
```

None of them. All three are false. Lean only checked that they make sense.

Last time, and today

Last time: reading

> Lean checks the **proof**.
> You check the **statement**.

- Statements written by someone else.
- Surprises: in Lean $1/0 = 0$, and on $\mathbb{N}$, $2 - 3 = 0$.

Today: writing

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

- Your own definitions and statements.
- Four tests that try to break them. They catch A, B and C.
- An AI writes the draft. You test it.

Writing definitions

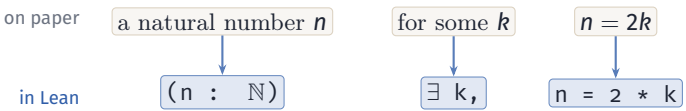
A definition, from paper

Definition

A natural number n is *even* if $n = 2k$ for some k.

In Lean

```
def IsEven (n : ℕ) : Prop :=
  ∃ k, n = 2 * k
```



- `def` gives the sentence a name. `IsEven n` is a statement, true or false: that is `Prop`.

Translate the sentence piece by piece.

Test your definition

test: small cases | live in Lean

```
-- 4 is even: take k = 2.
example : IsEven 4 := ⟨2, rfl⟩

-- 3 is not even.
example : ¬ IsEven 3 := by
  intro ⟨k, hk⟩
  omega
```

- 4 should be even. It is: $k = 2$.
- 3 should not be even. It is not.
- If you cannot prove both, the definition is wrong.
- Today, read the first line of each example. The rest is the proof.

> One example that has the property, one that does not.

Spot the problem (1)

hands up

Definition

A natural number n is *prime* if its only divisors are 1 and n.

Spot the problem (1)

hands up

Definition

A natural number n is *prime* if its only divisors are 1 and n.

```
def IsPrime (n : ℕ) : Prop :=
  ∀ d, d ∣ n → d = 1 ∨ d = n

-- 1 is "prime" by this definition.
example : IsPrime 1 := by
  intro d hd
  left
  exact Nat.dvd_one.mp hd

-- But not by Mathlib's.
example : ¬ Nat.Prime 1 := Nat.not_prime_one
```

- By this definition, 1 is prime.
- On paper, "$n > 1$" goes without saying. In Lean it must be written.

caught by: small cases

Try $n = 1$. It should not be prime, and here it is.

Use Mathlib's definition

test: compare

```
-- Mathlib has both notions already.
example : Even 4 := by decide
example : Nat.Prime 7 := by norm_num
example : ¬ Nat.Prime 1 := by norm_num

-- IsEven agrees with Mathlib's Even.
example (n : ℕ) : IsEven n ↔ Even n := by
  simp [IsEven, even_iff_two_dvd, dvd_def]
```

- Mathlib has `Even` and `Nat.Prime`, and 1 is not prime there.
- The last lines: our `IsEven` and Mathlib's `Even` say the same.
- Search Mathlib before you define.

If Mathlib has the notion, use it. Its small cases are already checked.

Writing statements

A statement has three parts

live in Lean

Theorem

The sum of two even numbers is even.

In Lean

```
theorem even_add_even (m n : ℕ)     -- 1. objects
    (hm : Even m) (hn : Even n) :   -- 2. assumptions
    Even (m + n) :=                 -- 3. claim
  hm.add hn                         -- the proof
```

1 Objects

What is each letter? Write its type.

2 Assumptions

Everything the paper assumes, said or unsaid.

3 Claim

What the paper concludes, in Mathlib's words.

Three questions, in this order: the objects, the assumptions, the claim.

Your turn: two minutes

hands up

Theorem

The product of two odd numbers is odd.

Write the statement on paper: the objects, the assumptions, the claim. Mathlib's word for odd is `Odd`.

Your turn: two minutes

hands up

Theorem

The product of two odd numbers is odd.

Write the statement on paper: the objects, the assumptions, the claim. Mathlib's word for odd is `Odd`.

```
theorem odd_mul_odd (m n : ℕ)       -- 1. objects
    (hm : Odd m) (hn : Odd n) :     -- 2. assumptions
    Odd (m * n) :=                  -- 3. claim
  hm.mul hn                         -- the proof
```

The same three parts as before. Only `Even` became `Odd`, and `+` became `*`.

Warm-up A: what went wrong?

hands up

A For every x, $x - 1 < x$.

```
example : ∀ x, x - 1 < x := sorry
```

Warm-up A: what went wrong?

hands up

A For every x, $x - 1 < x$.

```
example : ∀ x, x - 1 < x := sorry
```

```
#eval 0 - 1
-- 0

-- With the type written: true.
example : ∀ x : ℤ, x - 1 < x := by
  intro x
  omega
```

Lean infoview

```
x : ℕ
⊢ x - 1 < x
```

- No type was written, so Lean chose $\mathbb{N}$.
- On $\mathbb{N}$, $0 - 1 = 0$, and $0 < 0$ is false.

caught by: read it back — Question 1, the objects. Always write the type.

Warm-up B: what went wrong?

hands up | live in Lean

B $\frac{a}{b} \cdot b = a.$

```
example (a b : ℚ) : a / b * b = a := sorry
```

Warm-up B: what went wrong?

hands up | live in Lean

B $\frac{a}{b} \cdot b = a.$

```
example (a b : ℚ) : a / b * b = a := sorry
```

```
#eval (1 : ℚ) / 0 * 0
-- 0, and the paper says 1

theorem div_mul_self (a b : ℚ)
    (hb : b ≠ 0) :     -- was hidden
    a / b * b = a :=
  div_mul_cancel₀ a hb
```

- Take $a = 1$, $b = 0$. In Lean $1/0 = 0$, so the left side is 0.
- On paper, $b \neq 0$ hides in the notation $\frac{a}{b}$.

caught by: small cases — Question 2, the assumptions. Write the unsaid ones too.

Warm-up C: test on small cases

live in Lean

C

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

```
#eval (List.range 5).map fun n =>
  (∑ i ∈ Finset.range n, i, n * (n + 1) / 2)
-- [(0, 0), (0, 1), (1, 3), (3, 6), (6, 10)]
```

`#eval` computes both sides for $n = 0, \ldots, 4$. They differ: the statement is wrong. Why?

Warm-up C: test on small cases

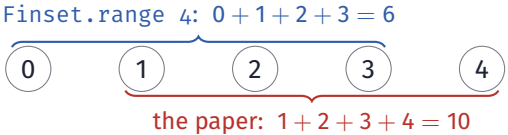
live in Lean

C

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

```
#eval (List.range 5).map fun n =>
  (∑ i ∈ Finset.range n, i, n * (n + 1) / 2)
-- [(0, 0), (0, 1), (1, 3), (3, 6), (6, 10)]
```

`#eval` computes both sides for $n = 0, \ldots, 4$. They differ: the statement is wrong. Why?



caught by: small cases

Testing costs nothing, and it works before there is a proof.

Warm-up C: the fix

```
#eval Finset.range 4
-- {0, 1, 2, 3}

example (n : ℕ) :
    ∑ i ∈ Finset.range (n + 1), i = n * (n + 1) / 2 := by
  rw [Finset.sum_range_id, Nat.add_sub_cancel, mul_comm]
```

- `Finset.range n` is $\{0, \ldots, n-1\}$, not $\{1, \ldots, n\}$.
- The fix: `range (n + 1)`. Now both sides agree for $n = 0, \ldots, 4$.

Question 3, the claim. Fix it, then run the same test again.

Checking a statement

Is this theorem true?

hands up

```
theorem silly (n : ℕ) (h1 : n < 3) (h2 : 5 < n) :
    n = 42 := by
  omega
```

Is this theorem true?

hands up

```
theorem silly (n : ℕ) (h1 : n < 3) (h2 : 5 < n) :
    n = 42 := by
  omega
```

- Yes. Lean accepts the proof.
- No number is below 3 and above 5. The theorem applies to nothing.
- Last time: `everything_is_42`.

caught by: check the hypotheses Give one example where all of them hold. If there is none, the theorem says nothing.

The four tests

	Test	How	Seen today
1	**Read it back**	hover over each letter; say it in English	warm-up A
2	**Small cases**	try 0, 1, 2 with `#eval`	prime, warm-ups B and C
3	**Check the hypotheses**	can they all hold? is each one used?	$n = 42$
4	**Compare**	with Mathlib, and with the paper's words	`IsEven`, `Nat.Prime`

> Run them before the proof, not after.

Drafting statements with AI

What an AI can do here

Useful

- Finding Mathlib's names: `Even`, `Nat.Prime`, `Finset.range`.
- Getting a draft to compile.
- Translating a Lean statement back into English.

Check for

- Today's mistakes: the wrong type, a missing hypothesis, off by one.
- A statement changed until it is easy to prove.

> The AI writes the draft. You are responsible for what it says.

A prompt that helps

Prompt

Formalize the following statement in Lean 4 with Mathlib. Use Mathlib's definitions; do not define your own. Do not prove it: end with `sorry`. Then:

1. translate your Lean statement back into English;
2. list each hypothesis and say why it is needed;
3. give one example where all the hypotheses hold.

Statement: ...

- Asking for `sorry` keeps the statement and the proof apart.
- Step 1 is test 1, steps 2 and 3 are test 3. The AI runs them; you check its answers.

Spot the problem (2)

hands up

```
-- Paper: if a ≤ b then a² ≤ b².
theorem draft1 (a b : ℕ) (h : a ≤ b)
    (h2 : a ^ 2 ≤ b ^ 2) :
    a ^ 2 ≤ b ^ 2 :=
  h2
```

Spot the problem (2)

hands up

```
-- Paper: if a ≤ b then a² ≤ b².
theorem draft1 (a b : ℕ) (h : a ≤ b)
    (h2 : a ^ 2 ≤ b ^ 2) :
    a ^ 2 ≤ b ^ 2 :=
  h2
```

Lean says

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

caught by: check the hypotheses

The claim is already assumed, as `h2`. True, and it says nothing.

Spot the problem (3)

hands up

```
-- Paper: if a ≤ b then a² ≤ b².
theorem draft2 (a b : ℕ) (h : a < b) :
    a ^ 2 ≤ b ^ 2 :=
  Nat.pow_le_pow_left h.le 2
```

Spot the problem (3)

hands up

```
-- Paper: if a ≤ b then a² ≤ b².
theorem draft2 (a b : ℕ) (h : a < b) :
    a ^ 2 ≤ b ^ 2 :=
  Nat.pow_le_pow_left h.le 2
```

- A proof, no `sorry`, no warning.
- The paper says $a \leq b$. The draft says $a < b$.

caught by: compare — True, but weaker: the case $a = b$ is not covered.

The fix

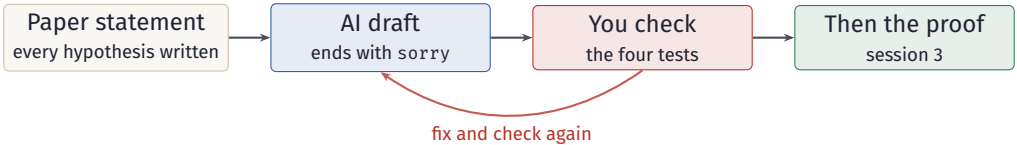
```
theorem sq_le_sq_of_le (a b : ℕ) (h : a ≤ b) :
    a ^ 2 ≤ b ^ 2 :=
  Nat.pow_le_pow_left h 2

-- A case we know: a = b = 3.
example : 3 ^ 2 ≤ 3 ^ 2 :=
  sq_le_sq_of_le 3 3 (le_refl 3)
```

- No `h2`: the claim is no longer assumed.
- $\leq$, as in the paper: the case $a = b = 3$ is covered again.

Two warning signs: a hypothesis the proof never uses, and a hypothesis stronger than the paper's.

The workflow



> Do not ask for a proof until you trust the statement.

For fun (1): Lean proves this

hands up

```
-- "There is a number greater than 5 that is equal to 3."
example : ∃ n : ℕ, n > 5 → n = 3 :=
  ⟨0, by omega⟩
```

For fun (1): Lean proves this

hands up

```
-- "There is a number greater than 5 that is equal to 3."
example : ∃ n : ℕ, n > 5 → n = 3 :=
  ⟨0, by omega⟩
```

```
-- What was meant: ∧, not →. This one is false.
example : ¬ ∃ n : ℕ, n > 5 ∧ n = 3 := by
  omega
```

- It says: there is n such that *if* $n > 5$ *then* $n = 3$.
- Take $n = 0$. It is not greater than 5, so nothing is claimed.

caught by: read it back

After “there is”, write $\wedge$, not $\to$.

For fun (2): Lean computes this

hands up

```
-- "1 + 1/2 + 1/3 + ··· + 1/100"
#eval ∑ i ∈ Finset.range 100, 1 / (i + 1)
-- 1
```

For fun (2): Lean computes this

hands up

```
-- "1 + 1/2 + 1/3 + ··· + 1/100"
#eval ∑ i ∈ Finset.range 100, 1 / (i + 1)
-- 1
```

```
-- With the type written: the first four terms.
#eval ∑ i ∈ Finset.range 4, (1 : ℚ) / (i + 1)
-- 25 / 12
```

- No type was written, so every term is a natural number.
- On $\mathbb{N}$, $1/2 = 0$, $1/3 = 0$, … Only the first term is left.

caught by: read it back — The mistake of warm-up A again. Always write the type.

For fun (3): Lean proves this

hands up

```
-- "ℕ and ℝ have the same number of elements."
example : Nat.card ℕ = Nat.card ℝ := by
  simp
```

For fun (3): Lean proves this

hands up

```
-- "ℕ and ℝ have the same number of elements."
example : Nat.card ℕ = Nat.card ℝ := by
  simp
```

```
-- Nat.card of an infinite set is 0.
example : Nat.card ℕ = 0 := Nat.card_eq_zero_of_infinite
example : Nat.card ℝ = 0 := Nat.card_eq_zero_of_infinite
```

- `Nat.card` is the number of elements, as a natural number.
- An infinite set has no such number, so Lean returns 0. The line says $0 = 0$.

caught by: read it back

Lean always returns a value, also when there is no answer.

Today's six bugs, and what caught them

Lean accepted	What was wrong	Caught by
They say something false.		
Definition of prime	1 is prime: "$n > 1$" was not written	small cases
A. $x - 1 < x$	no type written, so Lean chose $\mathbb{N}$	read it back
B. $\frac{a}{b} \cdot b = a$	$b \neq 0$ was hidden in the notation	small cases
C. $1 + 2 + \cdots + n$	`Finset.range n` stops at $n - 1$	small cases
True, with a proof, and still not the paper's theorem.		
AI draft 1	assumes its claim: it says nothing	check the hypotheses
AI draft 2	$a < b$ for $a \leq b$: it says less	compare

> Lean accepted all six. A short test caught each one.

At home

At home

`Exercises2.lean`, in the browser or on your laptop.

1. Definitions: write the `def`; its tests are below it.
2. Statements: from the paper sentence. The proof is in Mathlib.
3. Find the bug: true, false, or true for a silly reason? Fix the false ones.
4. Your own statement, from session 1. Draft it with an AI, then check it.

- First redo today's examples. They are in `Session2.lean`.
- That file ends with harder examples: functions, intervals, limits. Read them before parts 2 and 3.
- Send me your statement: in session 4 I formalize some of them live.

> Today the stating numbers are higher than the proving numbers. That is the point.

Next time: AI tools

- AI agents that write Lean proofs, in practice.
- Case studies: how the large formalizations of the last year were done.
- An agent proving a statement, live. At home, you try it on yours.

> Before next time: Lean and Mathlib working on your laptop (see the setup sheet), and your statement from part 4 compiling with `sorry`.

Resources

- J. Avigad and P. Massot, *Mathematics in Lean*. Its chapters on sets, functions and structures show how Mathlib states things.
- K. Buzzard, *Formalising Mathematics* (Imperial College). Notes on reading and writing statements.
- The Mathlib documentation, Loogle and LeanSearch: find Mathlib's version before you write your own.
- The Lean Zulip chat, stream *#new members*.
- A. Bauer, *Formalizing invisible mathematics* (IPAM, 13 February 2023). A talk on what mathematicians leave unsaid, and how a proof assistant fills it in. Video: `youtu.be/wZSvuCJBaFU`