# BGSMath Graduate Course: Formalizing Mathematics in Lean with AI

## Session 4: Four areas, and your own theorem

Michail Karatarakis

19 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** | Four areas, and your own theorem | Graph theory, analysis, commutative algebra and category theory, live in Lean; how to do your own theorem at home |

Today: one theorem from start to finish, then four areas of mathematics, live.

# The course in one slide

| | |
|---|---|
| **Read** | Brackets, junk values, coercions, quantifier order. The checklist. (Session 1) |
| **Write** | Definitions with tests. Statements whose hypotheses can hold, and are all needed. (Session 2) |
| **Prove** | An agent writes the proof against a pinned statement. (Session 3) |
| **Check** | No `sorry`, the three standard axioms, the statement unchanged. (Session 3) |

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

# Today

1. **One theorem, start to finish.** Everything from the course, on one small example.
2. **Four areas, live.** Most of the session: graph theory, analysis, commutative algebra and category theory, with the same steps in each.
3. **At home:** the rest of `Exercises4.lean`, and your own theorem, from `MyTheorem.lean`.

# One theorem, start to finish

# The statement

**Theorem**

Every continuous $f : [0,1] \to [0,1]$ has a fixed point.

- "$f : [0,1] \to [0,1]$" becomes two hypotheses: continuity on $[0,1]$, and *maps into* $[0,1]$.
- Pinned in a `def`, as in session 3.

In Lean

```
-- Paper: a continuous f : [0, 1] → [0, 1]
-- has a fixed point.
def Statement : Prop :=
  ∀ f : ℝ → ℝ,
    ContinuousOn f (Icc 0 1) →
    (∀ x ∈ Icc (0 : ℝ) 1, f x ∈ Icc 0 1) →
    ∃ c ∈ Icc (0 : ℝ) 1, f c = c
```

# Test it before proving it

live in Lean

```
-- The hypotheses can hold: f = 1/2.
example : ContinuousOn (fun _ : ℝ => (1 / 2 : ℝ))
      (Icc 0 1) ∧
    ∀ x ∈ Icc (0 : ℝ) 1, (1 / 2 : ℝ) ∈ Icc 0 1 :=
  ⟨continuousOn_const, fun _ _ => by norm_num⟩

-- Without "f maps into [0, 1]": f x = x + 1.
example : ¬ ∀ f : ℝ → ℝ, ContinuousOn f (Icc 0 1) →
    ∃ c ∈ Icc (0 : ℝ) 1, f c = c := by
  intro h
  obtain ⟨c, -, hc⟩ := h (fun x => x + 1) (by fun_prop)
  linarith
```

- The hypotheses can hold together: a constant function.
- The “maps into” hypothesis is needed: without it, $x \mapsto x + 1$ is a counterexample.

# The plan, on paper

**Proof**

Let $g(x) = f(x) - x$. Then $g(0) = f(0) \geq 0$ and $g(1) = f(1) - 1 \leq 0$. By the intermediate value theorem there is $c \in [0, 1]$ with $g(c) = 0$, that is, $f(c) = c$. □

- One step needs another theorem: the intermediate value theorem.
- State it precisely first, in the form the proof needs. Then look for it.

# A conditional proof

```
-- The step we need, stated precisely.
def IVT : Prop :=
  ∀ g : ℝ → ℝ, ContinuousOn g (Icc 0 1) →
    g 1 ≤ 0 → 0 ≤ g 0 →
    ∃ c ∈ Icc (0 : ℝ) 1, g c = 0

theorem main_of_ivt (h : IVT) :
    Statement := by
  intro f hf hmaps
  have h0 := (hmaps 0 ⟨le_rfl, zero_le_one⟩).1
  have h1 := (hmaps 1 ⟨zero_le_one, le_rfl⟩).2
  obtain ⟨c, hc, hgc⟩ :=
    h (fun x => f x - x)
      (hf.sub continuousOn_id)
      (by linarith) (by linarith)
  exact ⟨c, hc, by linarith⟩
```

- The missing step is a **hypothesis** of the theorem, not a `sorry` inside it.
- Anyone reading the statement sees exactly what is assumed.
- The same shape as the conditional formalization of the unit distance disproof (session 3).

# Closing the gap

```
theorem ivt : IVT := by
  intro g hg h1 h0
  exact intermediate_value_Icc' zero_le_one hg
    ⟨h1, h0⟩

theorem main : Statement :=
  main_of_ivt ivt
```

- Mathlib has the intermediate value theorem: `intermediate_value_Icc'`.
- If it did not, `main_of_ivt` would still be an honest result, with its assumption in plain view.

# The audit

live in Lean

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

- `Statement` is unchanged.
- No `sorry`; only the three standard axioms.
- The tests passed before the proof was written.

> Statement, tests, plan, proof, audit. The same steps work for your theorem.

# Four areas, live

# Four areas, the same steps

live in Lean

Graph theory, analysis, commutative algebra, category theory. In `Exercises4.lean`, each area has the same four steps:

1. **How Mathlib says it.** The `#check` lines at the top: click, and read the infoview.
2. **Search.** Try `exact?` first. It looks for a Mathlib theorem with exactly your statement.
3. **A short proof** when there is nothing to find: `simp`, `decide`, `ring`, `fun_prop`.
4. **With an AI.** Only the English is given. The AI drafts the Lean statement; you read it aloud before anything else.

> The basic facts are already in Mathlib. The work is to state them precisely enough to find them.

# Graph theory

live in Lean

- A graph on vertices `V` is `G : SimpleGraph V`.
- Its one field: `G.Adj u v`, "*u* and *v* are adjacent". Symmetric, no loops.
- To count, `V` must be finite: `[Fintype V]`.

```
variable {V : Type} (G : SimpleGraph V)

-- No vertex is adjacent to itself.
example (v : V) : ¬ G.Adj v v :=
  G.irrefl

-- The handshake lemma: found by exact?
example [Fintype V] [DecidableRel G.Adj] :
    ∑ v, G.degree v = 2 * #G.edgeFinset :=
  G.sum_degrees_eq_twice_card_edges
```

`exact?` finds the handshake lemma by its statement. You never need its name.

# Analysis

live in Lean

- Limits are written with filters. Read `Tendsto u atTop (nhds a)` as "$u_n \to a$".
- In the exercises, `nhds a` is written $\mathcal{N}\, a$, as in Mathlib.
- `fun_prop` proves continuity of formulas.

```
-- 1/n → 0: found by exact?
example : Tendsto (fun n : ℕ => 1 / (n : ℝ))
    atTop (nhds 0) :=
  tendsto_one_div_atTop_nhds_zero_nat

-- Continuity: one tactic.
example : Continuous
    (fun x : ℝ => 2 * x + 1) := by
  fun_prop
```

You can use a filter without knowing what one is. Read it as the sentence it stands for.

# Commutative algebra

live in Lean

- A commutative ring: `R` with `[CommRing R]`.
- An ideal: `I : Ideal R`. The ideal $(a)$ is `Ideal.span {a}`.
- Prime and maximal: `I.IsPrime`, `I.IsMaximal`.

```
variable {R : Type} [CommRing R]

-- 6 ∈ (2) in ℤ: membership is divisibility.
example : (6 : ℤ) ∈ Ideal.span {2} := by
  simp [Ideal.mem_span_singleton]

-- Maximal ideals are prime: found by exact?
example (I : Ideal R) (h : I.IsMaximal) :
    I.IsPrime :=
  h.isPrime
```

Tell `simp` what membership in $(a)$ means, and the question becomes arithmetic.

# When the search finds nothing

hands up

```
-- The sum of two nilpotents is nilpotent.
theorem nil_add (x y : R)
    (hx : IsNilpotent x) (hy : IsNilpotent y) :
    IsNilpotent (x + y) :=
```

`exact?` finds nothing. Is the statement false?

# When the search finds nothing

hands up

```
-- The sum of two nilpotents is nilpotent.
theorem nil_add (x y : R)
    (hx : IsNilpotent x) (hy : IsNilpotent y) :
    IsNilpotent (x + y) :=
```

`exact?` finds nothing. Is the statement false?

- No. Mathlib states it for *any* ring, where it needs $xy = yx$.
- In a commutative ring, `Commute.all x y` gives that.

```
Commute.isNilpotent_add
  (Commute.all x y) hx hy
```

Mathlib states a theorem in its most general form. Your case may need one extra step.

# Category theory

live in Lean

- A category: objects `C` with `[Category C]`. Morphisms `f : X ⟶ Y`.
- Composition in diagram order: `f ≫ g` is "first $f$, then $g$", that is $g \circ f$.
- Functors: `F : C ⥤ D`.

```
variable {C : Type} [Category C]

-- f ≫ g is "first f, then g".
example {W X Y Z : C} (f : W ⟶ X)
    (g : X ⟶ Y) (h : Y ⟶ Z) :
    (f ≫ g) ≫ h = f ≫ g ≫ h := by
  simp

-- Monos in sets are injective: exact?
example {X Y : Type} (f : X ⟶ Y) :
    Mono f ↔ Function.Injective f :=
  mono_iff_injective f
```

In the category of sets, the abstract notions become the familiar ones.

# At home

# Your own theorem

Open `MyTheorem.lean`. It has one section per step:

| | | |
|---|---|---|
| **1** | Paper statement | Every hypothesis, including the implicit ones. |
| **2** | Definitions | Only if Mathlib has none. Test each one. |
| **3** | Statement | Pinned. The agent may not edit it. |
| **4** | Tests | Hypotheses can hold; each one is needed; a case you know. |
| **5** | Steps | The plan, as precise statements. |
| **6** | Proof | You, an agent, or both. |
| **7** | Audit | `#print axioms main`. |

Steps 1–4 matter most. A well-tested statement with an open proof is a good outcome.

# When you are stuck

- **A definition is missing.** Search Mathlib first (Loogle, LeanSearch, the documentation). If it really is missing, write it and test it (session 2).
- **A step will not go through.** State it as its own lemma, and give the agent only that lemma.
- **The agent keeps failing.** Ask it to try the negation, or to look for a counterexample. The statement may be false.
- **The gap is real.** Make it a hypothesis, as in `main_of_ivt`. A conditional result with a precise assumption is progress.
- **You are lost.** Read the infoview, then ask on the Lean Zulip.

# After the course

# Where to go next

- **The Lean Zulip chat**, stream *#new members*: questions of every level are welcome.
- **Mathematics in Lean**, for the parts of Mathlib you did not see here.
- **Contributing to Mathlib**: a result in the right generality, with Mathlib's names and style, reviewed by maintainers. Start with something small.
- **Larger projects**: Tau Ceti roadmaps, Prove2Me missions, or a blueprint of your own.

# The one idea of the course

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

- AI now writes a large part of the proofs. That makes the statement the part that needs you.
- Read it, test it, pin it, audit what comes back.

Thank you.