Introduzione alla formalizzazione della matematica in Lean
Lesson 1/10 Introduction

Yoh Tanimoto

Corso di dottorato, University of Rome “Tor Vergata”

10 Dicembre 2025

(Disclaimer)

I am not an expert of Lean or logic, rather a user with a bit of experience.

I did some tutorials in Lean in 2020, then resumed in 2023, then started to contribute to `mathlib`, the main library of Lean, in early 2024.

There are very few people working on specific topics of mathematics (e.g. ~ 5 people on operator algebras). In other words, you can get to the frontline very quickly.

Why Lean, an interactive theorem prover?

In mathematics, we usually write (informal) proofs. **Formal proofs** are sequences of statements that can be derived from the axioms and inference rules.

Hilbert: "One must be able to say at all times–instead of points, straight lines, and planes–tables, chairs, and beer mugs" (link)

Mathematical statements can be written in the symbolic language and processed and verified by computer.

Some current research results are formalized in real time.

In lean we can talk about **some** advanced stuff (Riemannian manifolds, operator algebras, Lie algebra modules...).

(Why I am interested)

I studied physics in my bachelor's study. It was OK until Quantum Mechanics (operator theory).

I dropped out when I had to study (interacting) Quantum Field Theory and switched to mathematic(al physic)s.

In mathematical physics, we value **rigor**. We are supposed to define physical models and prove theorems about them.

I gradually learnt that, in the most advanced research, this high standard is not always kept, and even the experts are not sure what has been proven. e.g.

- Connes embedding conjecture
- UV stability of the Yang-Mills model in 4d
- Modular tensor category in C_2-cofinite Vertex Operator Algebras

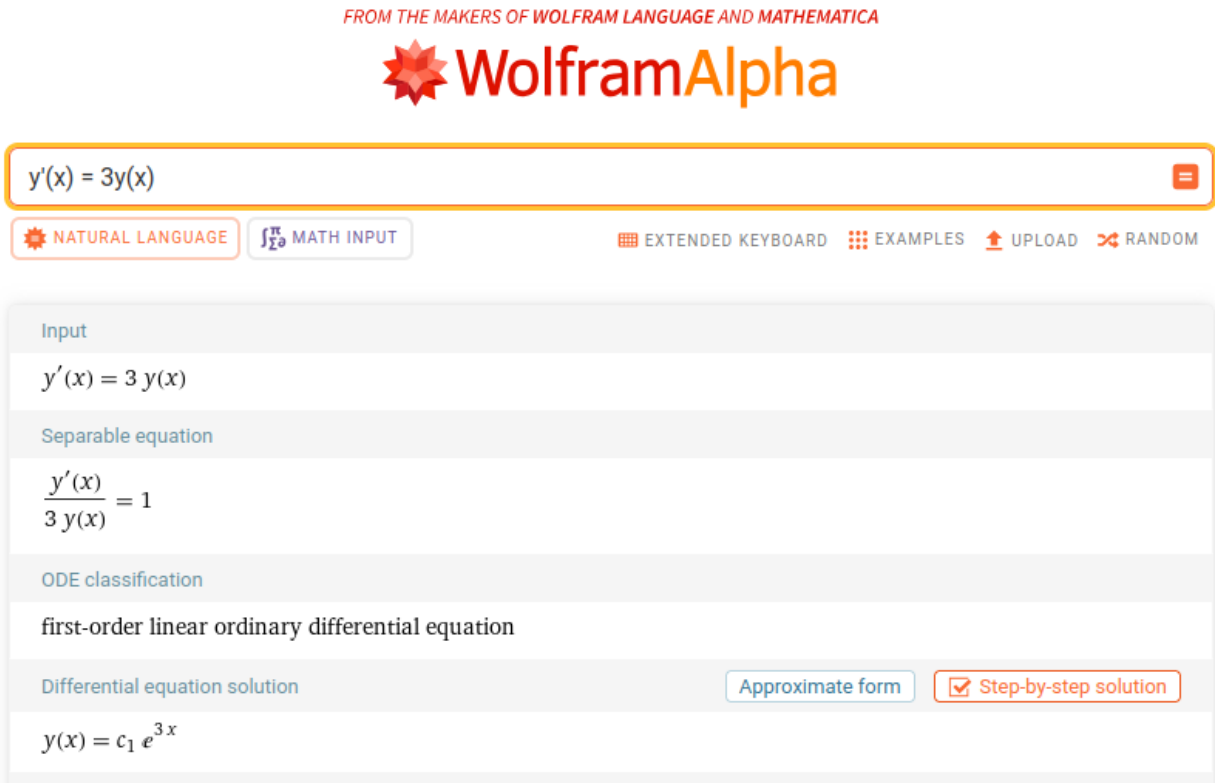
What is Lean?

Lean is a **proof assistant**, or **interactive theorem prover**.

- You write the statement and the proof
- Lean (computer) **verifies it**
- If incomplete, Lean gives suggestions

(Computer algebra system)

Wolfram alpha



Gives correct answers to most of the questions, but sometimes give wrong answers (link)

Example

FROM THE MAKERS OF WOLFRAM LANGUAGE AND MATHEMATICA

WolframAlpha

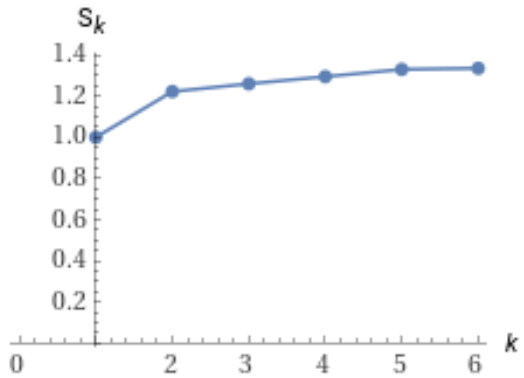
sum from n=1 to infinity of 1/(n^(2+(cos(n))^2))

NATURAL LANGUAGE | MATH INPUT | EXTENDED KEYBOARD | EXAMPLES | UPLOAD | RANDOM

Infinite sum

$$\sum_{n=1}^{\infty} \frac{1}{n^{2+\cos^2(n)}} \quad \text{diverges}$$

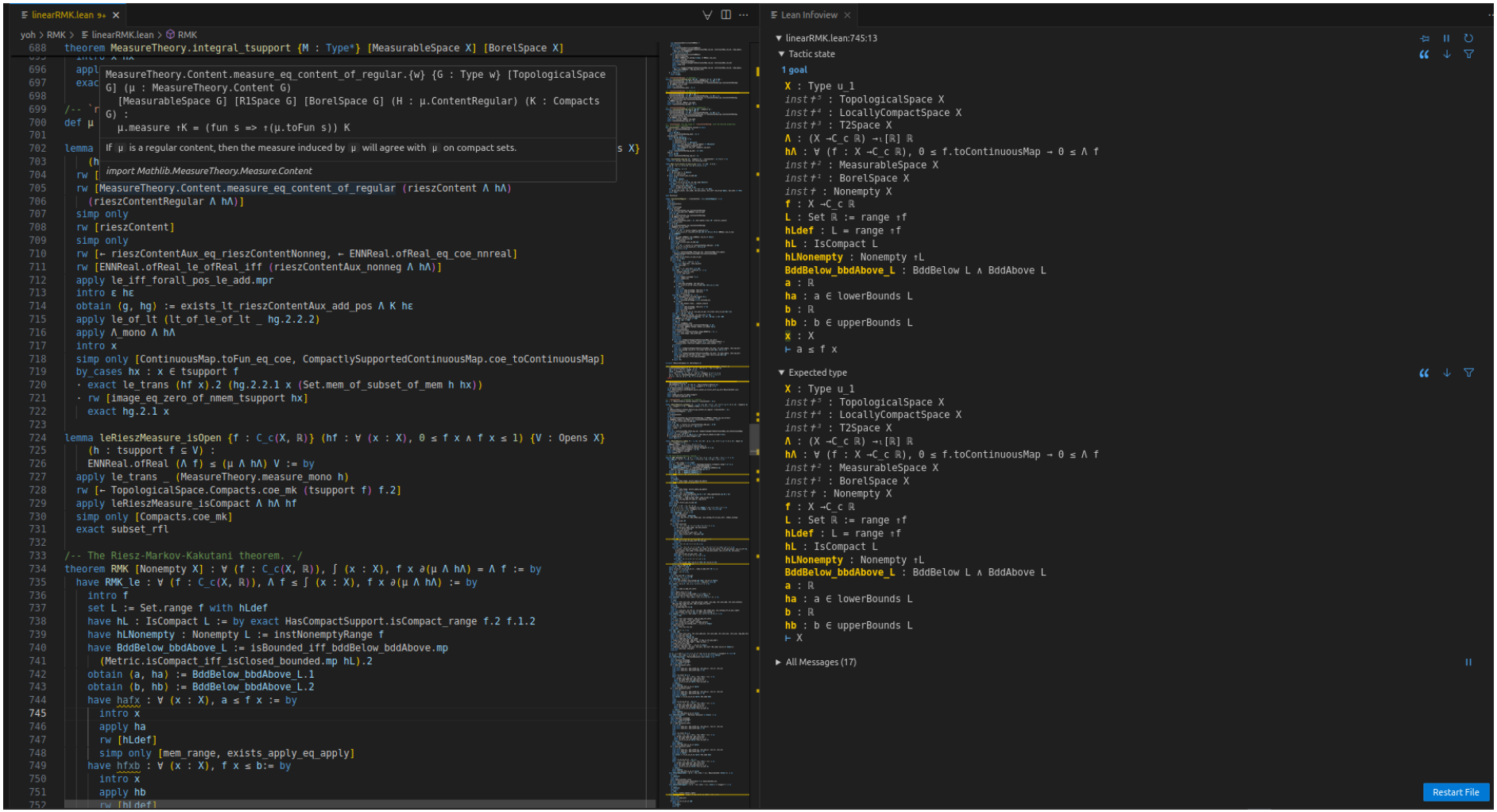
Partial sums | More terms | Show points



Download Page

POWERED BY THE WOLFRAM LANGUAGE

Lean, as a proof assistant



- You write the proof
- Lean (computer) verifies it

(Automated theorem prover)

(Lean **does not prove** theorems, but it can be **combined with AI**)

Several AI achieved gold-medal standard at the International Mathematical Olympiad (2025)
(link)

Typical workflow:

- humans formalize the problem in Lean
- AI attempts to prove it
- Lean tells whether the proof is correct or not
- If not, AI tries again

Some more recent stories:

- Vibe coding with ChatGPT5
- Autoformalisation with Aristotle

Why interacting theorem prover (in the past)?

Mathematicans have formalized some part of mathematics (Mizar, Metamath, HOL, Isabelle, Coq...), but the developments in most cases had not reached the actual research level until recently.

Theorem provers have been mainly developed by computer scientists, with real applications to verification of hardware and software. “Mistakes can be very costly, examples are the destruction of the Ariane 5 rocket (caused by a simple integer overflow problem that could have been detected by a formal verification procedure) and the error in the floating point unit of the Pentium II processor.” (Bridge, 2010. link)

“The failure resulted in a loss of more than US$370 million.” (Wikipedia entry about Ariane flight V88)

Lean is developed by people including engineers at Microsoft.

Formal verification of computer program

- Computer programs are written in computer languages.
- We want to know that programs do what we want them to do.
- Write the program as a function in Lean, prove that it does what it should do.
- (Ask Prof. Butterley for more details!)

(taken from aeneas tutorial)

```
def mul2_add1 (x : U32) : Result U32 :=
  do
  let i ← x + x
  i + 1#u32


theorem mul2_add1_spec (x : U32) (h : 2 * x.val + 1 ≤ U32.max)
  : ∃ y, mul2_add1 x = ok y ∧ ↑y = 2 * ↑x + (1 : Int)
  := by
  unfold mul2_add1
  have ⟨ x1, hEq1, hPost1 ⟩ := @U32.add_spec x x (by scalar_tac)
  simp [hEq1]
  have ⟨ x2, hEq2, hPost2 ⟩ := @U32.add_spec x1 1#u32 (by scalar_tac)
  simp [hEq2]
  scalar_tac
```

What is a proof in mathematics?

- Let P, Q be propositions.
- Assume that P and $P \to Q$ are correct (axioms).
- Then we can deduce Q (modus ponens, an inference rule).

- Let x a variable in some domain.
- Let $P(x)$ be a predicate (a proposition that depends on x).
- Assume $\forall x, P(x)$ (axiom).
- Then $P(x_0)$ holds for any x_0 ($\forall$-elimination or specialization, an inference rule).

- Let x a variable in some domain.
- Let $P(x)$ be a predicate (a proposition that depends on x).
- If we can prove $P(x)$ for x without any other condition, then it should hold for all x, that is, $\forall x, P(x)$ ($\forall$-introduction, an inference rule).

What is a formal proof?

$$\frac{P \qquad P \to Q}{Q}$$

```
example (P Q : Prop) (hP : P) (hPQ : P → Q) : Q := hPQ hP
```

$$\frac{\forall x, P(x)}{P(x)}$$

```
example (X : Type) (P : X → Prop) (x : X) (h : ∀ x, P x) : P x := h x
```

What is a formal proof in Lean?

Lean uses **dependent type theory** as its basis. Everything has a **type**.

- $\mathbb{N}$
- $\mathbb{R}$
- $\mathbb{N} \to \mathbb{R}$
- `Prop` (propositions, actually there is a hierarchy inside `Prop`)
- $\mathbb{R} \to$ `Prop` (predicates that depends on a real number)

(cf. First-order logic + the set theory (Mizar, Metamath))

What is a formal proof in Lean?

In Lean, $\mathbb{N}$ is defined as

```
inductive ℕ where
| zero :  ℕ
| succ (n :  ℕ) :  ℕ
```

- `zero` $(= 0)$ has type $\mathbb{N}$
- `succ zero` $(= 1)$ has type $\mathbb{N}$
- `succ (succ zero)` $(= 2)$ has type $\mathbb{N}$
- ...
- Only these symbols have type $\mathbb{N}$

What is a formal proof good for?

- correctness
- encouraging complete proofs
- searchability
- collaboration
- education
- combination with AI?

Correctness

From the code written by human, Lean outputs “proof terms”.

The kernel checks that the proof term gives the type of the theorem.

One can write another verifier of the proof terms.

cf. a talk by F. van Doorn

Encouraging complete proofs

To write a formal proof, one needs to know a very detailed informal proof. If it becomes more common to write formal proofs, it will urge people to give details.

Terence Tao, in an attempt to formalize his proof, he noticed that he had done a division by zero.
(link)

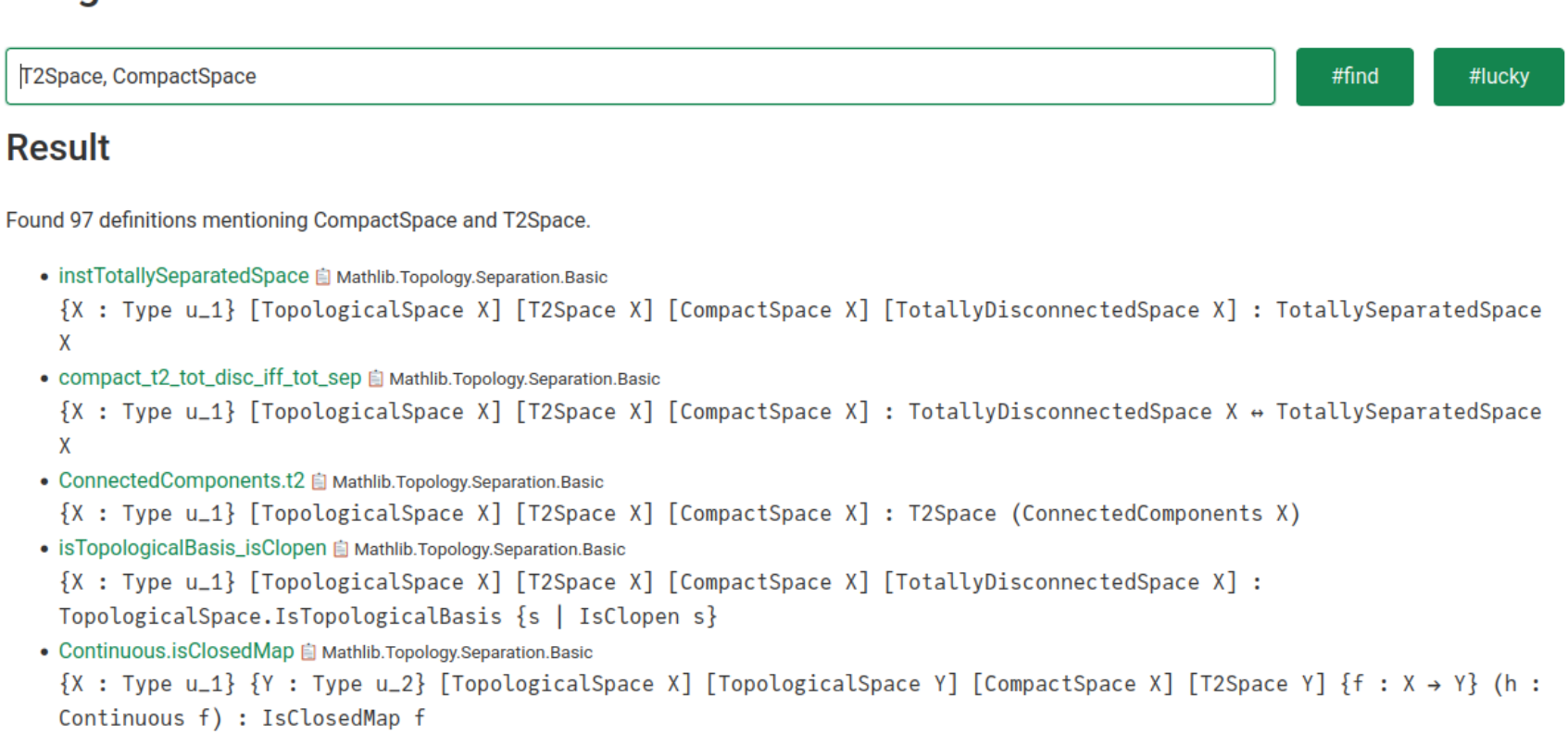
Gouëzel–Shchur found a reversed inequality in a paper, corrected it and check the proof (in Isabelle/HOL).
(link)

Searchability

`mathlib` is accompanied with various search engines.

loogle, leansearch

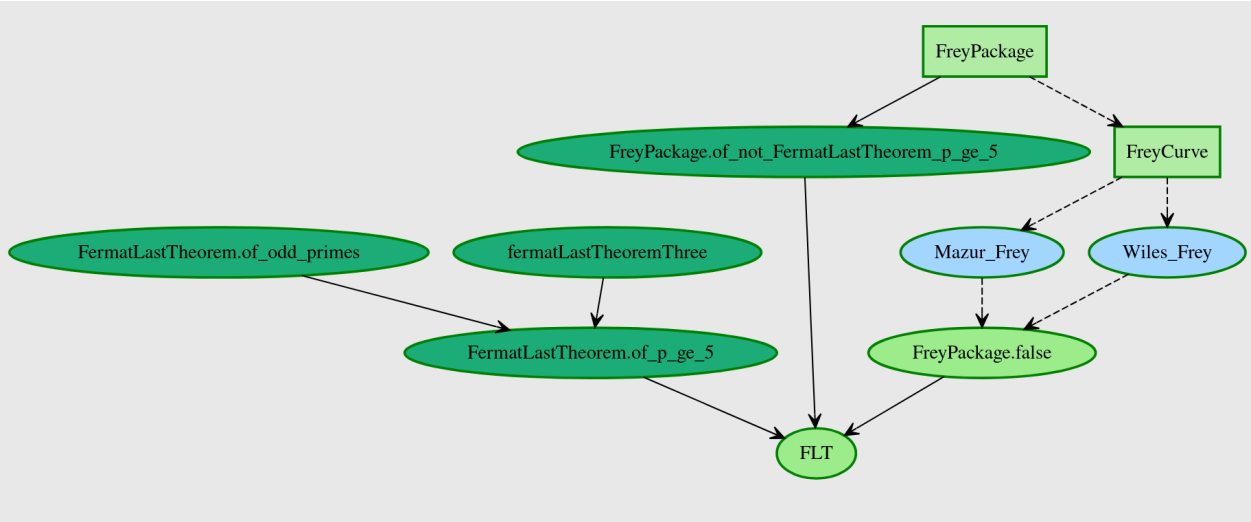
One can find statements that contain a certain combination of keywords



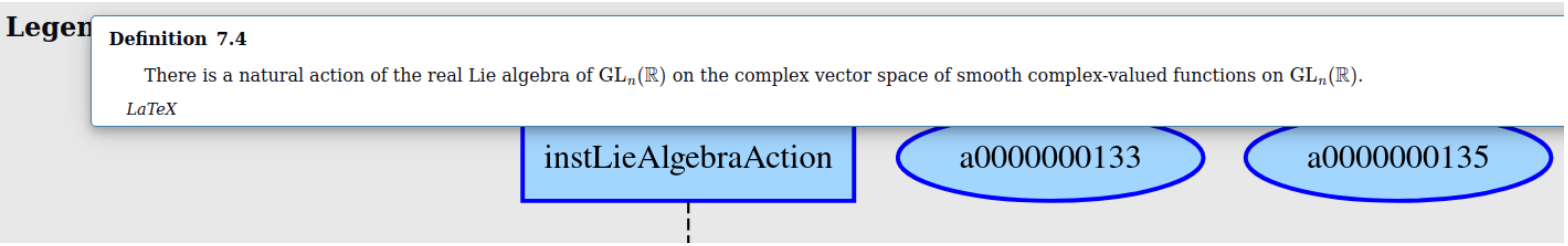
(link)

Collaboration

There are some theorems that require very different fields of mathematics. Anyone can help by filling in auxilliary results needed in a bigger project.
(Fermat Last Theorem blueprint Chapter 2)



(Fermat Last Theorem blueprint Chapter 7)



Education

Even with informal proofs, students often get confused with an implication and its inverse, $\forall$ and $\exists$, P and $P \to Q$...
After learning informal proofs, by using an interactive theorem prover, a student may learn which opereations are accepted.
cf. a talk by G. Marasingha

Graduate students can learn advanced materials and try to formalize it, and in the course obtain more detailed, complete comprehension.
If done well, that can be added to the library to help the current research.

Combination with AI?

- **Humans write** the definitions and theorem statements
- Either humans or AI try to prove them
- Lean tells whether the proof is complete or not
- AI can try and make errors, hopefully self-correct?

cf. Formal conjectures project collects conjectures written in Lean.

Formalizing recent research

Definition of a perfectoid space of Peter Scholze, formalized by Buzzard, Commelin, Massot in 2018
cf. a talk by Buzzard

Liquid tensor experiment (fundamental theorem of "liquid vector spaces", Clausen and Scholze 2019), formalized by 18 people in 2022

The polynomial Freiman–Ruzsa conjecture, proved by Gowers, Green, Manners and Tao in November 2023.

Formalized in Lean by 25 people **three weeks later**.
(link)

If the library contains enough prerequisits, one can formalize the current research in a reasonable time.

What are you interested in and what have been formalized in Lean?

- Mathlib documentation
- Leansearch

Using Lean

Try: Lean playground

Set up: Installation instructions

Learning Lean

- Mechanics of proof (for learning logic and mathematics in Lean at the same time)
- Logic and proof (for learning logic and its implementation in Lean at the same time)
- Theorem proving in Lean (for understanding the foundation)
- Mathematics in Lean (for understanding how to write advanced mathematics)

- The Lean Language Reference (for comprehensive information)
- Type checking in Lean 4 (for understanding how Lean checks proofs)

Explore

- Some theorems
- number theory game and other games
- Tips: in the game, Lean playground and VS code, by typing typical LaTeXcode, you get a unicode character (such as `\forall` gives $\forall$).

Plan of the course

1. Introduction
2. Logic of Lean (dependent type theory)
3. Basic tactics
4. Real numbers
5. Sets and functions
6. Structures
7. Hierarchy of structures
8. Magmas, monoids, semigroups and groups
9. More advanced structures
10. Advanced topics (creating own projects, contributing to `mathlib`, some automation...)