Introduction to Lean theorem prover

Yoh Tanimoto

University of Rome “Tor Vergata”

Tokyo operator algebras seminar
10 December 2024

(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 (~ 5) doing things 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 deduction 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 (very basic) stuff about operator algebras.

(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. E.g. conformal nets, vertex operator algebras...

I gradually learnt that, in the most advanced research, this high standard is not always kept. Cf. the UV stability of the Yang-Mills model in 4d.

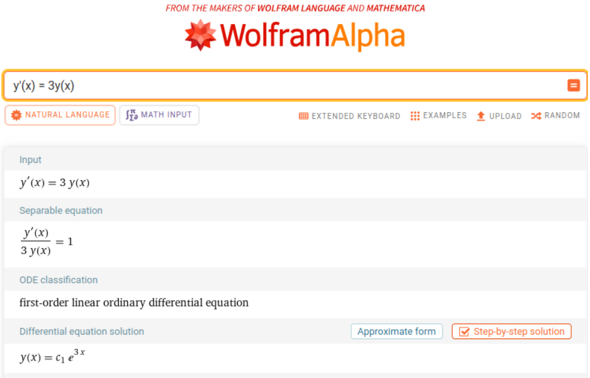
What is Lean?

Lean is a **proof assistant**, or **interactive theorem prover**. You write the statements and (some large part of) the proofs. Computer gives suggestions and **verifies the proofs**.

Not to be confused with computer algebra system (e.g. Mathematica) or automatic theorem prover (e.g. Alphaproof)

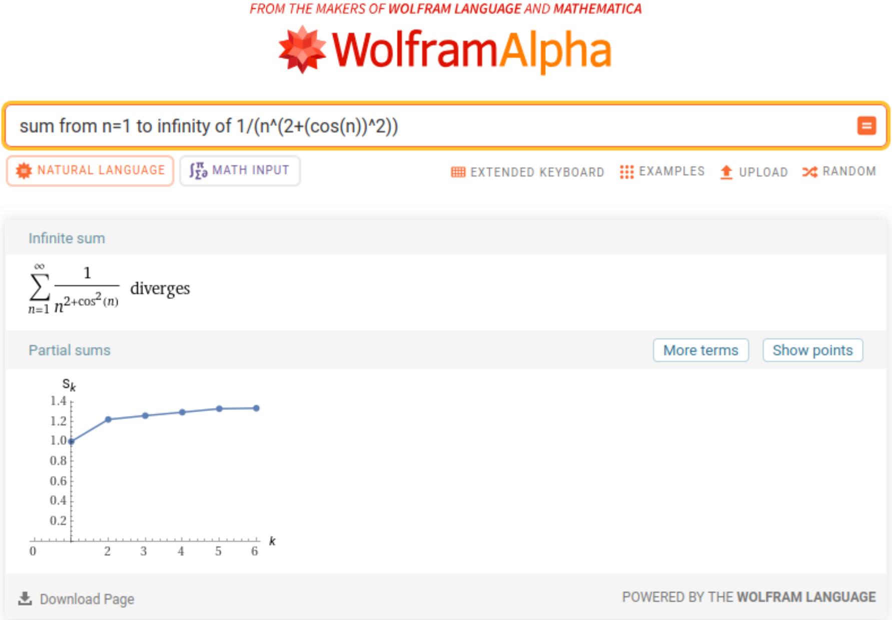
(Computer algebra system)

Wolfram alpha

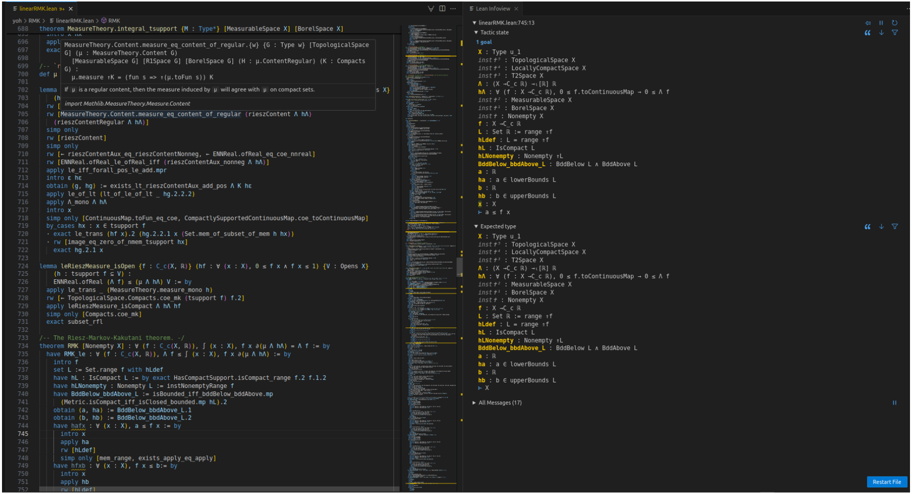


Gives correct answers to most of the questions, but sometimes give wrong answers (link)

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
$\sum_{n=1}^{\infty} \frac{1}{n^{2+\cos^2(n)}}$ diverges
Partial sums
More terms
Show points
s_k
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
2
3
4
5
6
k
Download Page
POWERED BY THE WOLFRAM LANGUAGE

Lean, as a proof assistant



You write the proof, the computer verifies it.

(Automated theorem prover)

Computer searches for proofs and finds (or fails).

Alphaproof: AI achieves silver-medal standard solving International Mathematical Olympiad problems

(link)

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.

What is a proof in mathematics?

Let P, Q be propositions, and assume that P and $P \to Q$ are correct (axioms). Then from $P, P \to Q$ we can deduce Q.

Let x a variable in some domain and $P(x)$ 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 in the domain, that is, $\forall x, P(x)$. On the other hand, if $\forall x, P(x)$ holds, then $P(x)$ holds for any x.

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
```

That is, `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 in Lean?

We can define a function `addone` by

```
def addone : ℕ → ℕ
| zero => succ zero
| succ n => succ succ n
```

We have `addone = succ`.

```
example (n : Nat) : addone n = succ n := by
induction n
case zero => rfl
case succ n ih => rfl
```

More interestingly, define

```
def myadd (m n : ℕ) : ℕ :=
match n with
| ℕ.zero => m
| ℕ.succ n => ℕ.succ (myadd m n)
```

What is a formal proof in Lean?

$\forall (n : \mathbb{N}), 0 + n = n + 0 : \texttt{Prop}$.
Proof: If $n = 0$, it is $0 + 0 = 0 + 0$ (axiom).
If $\texttt{n} = \texttt{succ m} = \texttt{m} + 1$, we have to prove $0 + (\texttt{m} + 1) = (\texttt{m} + 1) + 0$. RHS $= \texttt{m} + 1$: LHS $= (0 + \texttt{m}) + 1 = \texttt{m} + 1$.

```
def myadd (m n : ℕ) : ℕ :=
match n with
| ℕ.zero => m
| ℕ.succ n => ℕ.succ (myadd m n)

example (n : ℕ) : myadd n 0 = myadd 0 n := by
induction n
  case zero => rfl
  case succ n ih => rw [myadd, myadd, ← ih, myadd]
```

(link)

What is a formal proof in Lean?

```
def myadd (m n :  ℕ) :  ℕ :=
match n with
| ℕ.zero => m
| ℕ.succ n => ℕ.succ (myadd m n)

example (m n :  ℕ) :  myadd m n = myadd n m := by
  sorry
```

(link)

What is a formal proof good for?

- correctness
- encouraging complete proofs
- searchability
- collaboration
- education

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, moogle, leansearch

One can find statements that contain a certain combination of keywords

Loogle!

T2Space, CompactSpace

#find #lucky

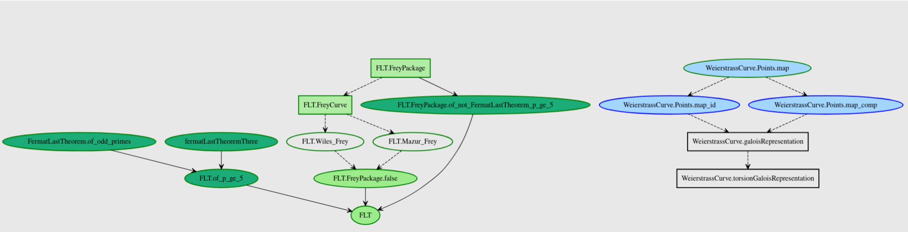
Result

Found 97 definitions mentioning CompactSpace and T2Space.

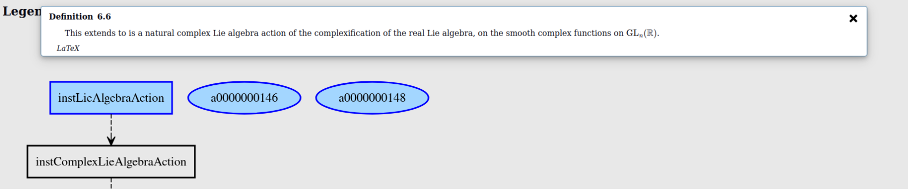
- instTotallySeparatedSpace Mathlib.Topology.Separation.Basic
```
{X : Type u_1} [TopologicalSpace X] [T2Space X] [CompactSpace X] [TotallyDisconnectedSpace X] : TotallySeparatedSpace X
```
- compact_t2_tot_disc_iff_tot_sep Mathlib.Topology.Separation.Basic
```
{X : Type u_1} [TopologicalSpace X] [T2Space X] [CompactSpace X] : TotallyDisconnectedSpace X ↔ TotallySeparatedSpace X
```
- ConnectedComponents.t2 Mathlib.Topology.Separation.Basic
```
{X : Type u_1} [TopologicalSpace X] [T2Space X] [CompactSpace X] : T2Space (ConnectedComponents X)
```
- isTopologicalBasis_isClopen Mathlib.Topology.Separation.Basic
```
{X : Type u_1} [TopologicalSpace X] [T2Space X] [CompactSpace X] [TotallyDisconnectedSpace X] : TopologicalSpace.IsTopologicalBasis {s | IsClopen s}
```
- Continuous.isClosedMap Mathlib.Topology.Separation.Basic
```
{X : Type u_1} {Y : Type u_2} [TopologicalSpace X] [TopologicalSpace Y] [CompactSpace X] [T2Space Y] {f : X → Y} (h : Continuous f) : IsClosedMap f
```

(link)

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 6)



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.

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

(There are more elements of number theory in `mathlib` because several people have formalized basic stuff)

Formalizing real-time research

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 has been formalized in Lean about operator algebras and quantum field theory?

Latest developments:

- weak operator topology
- continuous functional calculus
- Riesz-Markov-Kakutani theorem (not yet in `mathlib`)
- vertex operators (formal series)

Using Lean

Try: Lean playground

Set up: Installation instructions

Learning Lean

Natural Number Game

Mechanics of proof

Mathematics in Lean

Theorem proving in Lean

Searching in mathlib documentation

General documentation
index
foundational types
references

Library
- Aesop (file)
- Archive (file)
- Batteries (file)
- Counterexamples (file)
- ImportGraph
- Init (file)
- Lake (file)
- Lean (file)
- LeanSearchClient (file)
- Mathlib (file)
  - Algebra
  - AlgebraicGeometry
  - AlgebraicTopology
  - Analysis
    - Analytic
    - Asymptotics
    - BoxIntegral
    - CStarAlgebra
      - ContinuousFunctionalCalculus
      - Module
      - SpecialFunctions
      - ApproximateUnit

Classes of C⋆-algebras

This file defines classes for complex C⋆-algebras. These are (unital or non-unital, commutative or noncommutative) Banach algebra over `ℂ` with an antimultiplicative conjugate-linear involution (`star`) satisfying the C⋆-identity `‖star x * x‖ = ‖x‖ ^ 2`.

Notes

These classes are not defined in `Mathlib.Analysis.CStarAlgebra.Basic` because they require heavier imports.

```
class NonUnitalCStarAlgebra                                                  source
    (A : Type u_1) extends NonUnitalNormedRing A, StarRing A, CompleteSpace A, CStarRing A,
    NormedSpace ℂ A, IsScalarTower ℂ A A, SMulCommClass ℂ A A, StarModule ℂ A :
  Type u_1
```

The class of non-unital (complex) C⋆-algebras.

```
norm : A → ℝ
add : A → A → A
add_assoc : ∀ (a b c : A), a + b + c = a + (b + c)
zero : A
zero_add : ∀ (a : A), 0 + a = a
add_zero : ∀ (a : A), a + 0 = a
nsmul : ℕ → A → A
nsmul_zero : ∀ (x : A), AddMonoid.nsmul 0 x = 0
nsmul_succ : ∀ (n : ℕ) (x : A), AddMonoid.nsmul (n + 1) x = AddMonoid.nsmul n x + x
neg : A → A
sub : A → A → A
sub_eq_add_neg : ∀ (a b : A), a - b = a + -b
```

return to top

source

- Imports
- Imported by

NonUnitalCStarAlgebra
NonUnitalCommCStarAlgebra
CStarAlgebra
CommCStarAlgebra
CStarAlgebra.toNonUnitalCStarAlgebra
CommCStarAlgebra.toNonUnitalCommCStarAlgebra
StarSubalgebra.cstarAlgebra
StarSubalgebra.commCStarAlgebra
NonUnitalStarSubalgebra.nonUnitalCStarAlgebra
NonUnitalStarSubalgebra.nonUnitalCommCStarAlgebra
instCommCStarAlgebraComplex
instNonUnitalCStarAlgebraForall
instNonUnitalCommCStarAlgebraForall
instCStarAlgebraForall
instCommCStarAlgebraForall
instNonUnitalCStarAlgebraProd
instNonUnitalCommCStarAlgebraProd
instCStarAlgebraProd
instCommCStarAlgebraProd

Search entries using loogle, moogle, leansearch.

From Blueprint of FLT project

2.2 Reduction to $n \geq 5$ and prime

Lemma 2.1. ✓

If there is a counterexample to Fermat's Last Theorem, then there is a counterexample $a^p + b^p = c^p$ with p an odd prime.

Proof ▾

Note: this proof is in mathlib already; we run through it for completeness' sake.

Say $a^n + b^n = c^n$ is a counterexample to Fermat's Last Theorem. Every positive integer is either a power of 2 or has an odd prime factor. If $n = kp$ has an odd prime factor p then $(a^k)^p + (b^k)^p = (c^k)^p$ is the counterexample we seek. It remains to deal with the case where n is a power of 2, so let's assume this. We have $3 \leq n$ by assumption, so $n = 4k$ must be a multiple of 4, and thus $(a^k)^4 = (b^k)^4 = (c^k)^4$, giving us a counterexample to Fermat's Last Theorem for $n = 4$. However an old result of Fermat himself (proved as `fermatLastTheoremFour` in `mathlib`) says that $x^4 + y^4 = z^4$ has no nontrivial solutions.

Euler proved Fermat's Last Theorem for $p = 3$; at the time of writing this is not in mathlib.

Lemma 2.2. ✓

There are no solutions in positive integers to $a^3 + b^3 = c^3$.

Proof ▾

A proof has been formalised in Lean in the FLT-regular project here. Another proof has been formalised in Lean in the FLT3 project here by a team from the Lean For the Curious Mathematician conference held in Luminy in March 2024 (its dependency graph can be visualised here).

Corollary 2.3. ✓

If there is a counterexample to Fermat's Last Theorem, then there is a counterexample $a^p + b^p = c^p$ with p prime and $p \geq 5$.

Proof ▾

Follows from the previous two lemmas.

2.3 Frey packages

For convenience we make the following definition.

Definition 2.4. ✓

Using Lean

Lean community

Zulip chat

(an introductory talk by F. Nuccio)

(Personal) outlook

Spectral theorem for bounded self-adjoint operators on a Hilber space.
What is done:

- Hilbert spaces
- Bounded operators, adjoint
- Definition of C^*-algebras
- Continuous functional calculus in a C^*-algebra
- Weak and strong operator topologies
- Measure theory
- Riesz-Markov-Kakutani representation theorem

Todo:

- Definition of resolution of unity (projection-valued measure)
- Continuity results of functional calculus
- Riesz-Markov-Kakutani theorem for bounded complex functionals
- (Correspondence between sesquilinear forms and operators)