Formalising advanced mathematical definitions in Lean mathlib

Yoh Tanimoto

University of Rome "Tor Vergata"

Mathematics and AI
RIKEN Tokyo Campus
20 August 2026

What is Lean?

Lean is an interactive theorem prover. You write the mathematical **definitions, statements and proofs** in the computer language. Computer **verifies the proofs**.

```
example (a b c d : ℝ) (h : c = b*a - d) (h' : d = a*b) :
    c = 0 := by
  calc
    c = b*a - d := by rw [h]
    _ = b*a - a*b := by rw [h']
    _ = 0 := by ring
```

(from "Glimpse of Lean" Chapter 1)

What can AI do in Lean?

- LLM can produce (many) proof attempts but cannot be trusted
- Lean checks proofs
- Lean does not check **definitions** and **statements**

What can AI do in Lean?

- LLM can produce (many) proof attempts but cannot be trusted
- Lean checks proofs
- Lean does not check **definitions** and **statements**

```
theorem Give_me_1M_USD : Riemann_Hypothesis := by
   apply my_brilliant_proof
```

(link)

What can AI do in Lean?

- LLM can produce (many) proof attempts but cannot be trusted
- Lean checks proofs
- Lean does not check **definitions** and **statements**

```
theorem Give_me_1M_USD : Riemann_Hypothesis := by
   apply my_brilliant_proof
```

(link)

Can we trust mathematical **definitions** and **statements** written by AI?

- You can ask AI to write Lean proofs of some problems, or to formalise some textbooks or papers.
- "As of mid-2026, code written by an AI without the supervision of a Lean subject expert fails to meet that bar by a large margin." link
  (from `mathlib` contribution guide)

Lean's foundation

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

- `Nat : Type` ($= \mathbb{N}$) (we will see the definition in a moment)
- `zero : Nat` ($= 0$), `succ zero : Nat` ($= 1$).
- `Prop : Type` (propositions)
- $\mathbb{N} \to$ `Prop : Type` (predicates that depends on a natural number)
- `Type : Type 1`
- From one type, we can create more types (ordered pairs, functions, quotient...)

There are different foundations: first-order logic, higher-order logic, simple type theory...
Other Interactive Theorem Provers: Mizar, Methamath, Isabelle, Rocq...

Lean (and dependenty type theory) is one choice.

Human definition(s) of natural numbers

Peano axioms in first-order logic

1. $\forall x, 0 \neq S(x)$
2. $\forall x, y (S(x) = S(y) \implies x = y)$
3. $\forall x (x + 0 = x)$
4. $\forall x, y (x + S(y) = S(x + y))$
5. $\forall x (x \cdot 0 = 0)$
6. $\forall x, y (x \cdot S(y) = x \cdot y + x)$
7. (Induction "schema" for formula φ)
   $\forall \bar{y} \left((\varphi(0, \bar{y}) \wedge \forall x \left(\varphi(x, \bar{y}) \Rightarrow \varphi(S(x), \bar{y}) \right) \right) \Rightarrow \forall x \varphi(x, \bar{y}) \right)$

A model of the Peano axioms in the set theory

$0 = \emptyset, 1 = \{\emptyset, \{\emptyset\}\} = \{0, \{0\}\}, 2 = \{\emptyset, \{\emptyset, \{\emptyset\}\}\} = \{1, \{1\}\} \ldots$
$S(x) = \{x, \{x\}\}$.

Lean definition of natural numbers

In Lean, `Nat` ($= \mathbb{N}$) is defined as follows (link)

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

This means that

- `Nat` is a type
- `zero` has type `Nat`
- `succ zero` has type `Nat`
- `succ (succ zero)` has type `Nat`
- ...
- Nothing else has type `Nat`
- Because `n :  Nat` is either `zero` or `succ m` with another `m :  Nat`, if we prove a predicate `P n` for both cases, Lean accepts it as a proof of `∀ n, P n` (induction)

Which mathematics can we write in Lean?

- Lean : the formal computer language
- `mathlib` : the mathematical library in Lean

`mathlib` contains algebraic and analytic structures on `Nat`, `Int`, `Rat`..., the definition of `Real`, `Complex`.
More abstract structures as well such as

- algebraic `Mul`, `Semigroup`, `Monoid`, `Group`, `Module`, `Algebra`
- topological `TopologicalSpace`, `MetricSpace`, `CompleteSpace`
- analytic `integral`, `fderiv`, `InnerProductSpace`, `CStarAlgebra`
- geometric `Bundle.RiemannianMetric`, `ContMDiffVectorBundle`
- and many more

Design choices

- Lean defines $\mathbb{N}$ as an inductive type
- One could first define the set theory and a model of the Peano axioms in it

In the development of a mathematical library, there are **many** design choices, for each definition.

Which is the mathlib definition?

- $\mathbb{Z}$: quotient of $\mathbb{N} \times \mathbb{N}$ by $(a, b)\ (c, d) \Leftrightarrow a + c = b + d$
  vs positive and negative part (link)
- $\mathbb{Q}$: quotient of $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ by $(a, b)\ (c, d) \Leftrightarrow a \cdot c = b \cdot d$
  vs reduced ordered pairs in $\mathbb{Z} \times \mathbb{N} \setminus \{0\}$ (link)
- $\mathbb{R}$: quotient of Cauchy sequences vs Dedekind cut (link)

Design choices

- Lean defines $\mathbb{N}$ as an inductive type
- One could first define the set theory and a model of the Peano axioms in it

In the development of a mathematical library, there are **many** design choices, for each definition.

Which is the mathlib definition?

- $\mathbb{Z}$: quotient of $\mathbb{N} \times \mathbb{N}$ by $(a, b)\ (c, d) \Leftrightarrow a + c = b + d$
  vs positive and negative part (link)
- $\mathbb{Q}$: quotient of $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ by $(a, b)\ (c, d) \Leftrightarrow a \cdot c = b \cdot d$
  vs reduced ordered pairs in $\mathbb{Z} \times \mathbb{N} \setminus \{0\}$ (link)
- $\mathbb{R}$: quotient of Cauchy sequences vs Dedekind cut (link)

Design choices

- Lean defines $\mathbb{N}$ as an inductive type
- One could first define the set theory and a model of the Peano axioms in it

In the development of a mathematical library, there are **many** design choices, for each definition.

Which is the mathlib definition?

- $\mathbb{Z}$: quotient of $\mathbb{N} \times \mathbb{N}$ by $(a, b)\ (c, d) \Leftrightarrow a + c = b + d$
  vs positive and negative part (link)
- $\mathbb{Q}$: quotient of $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ by $(a, b)\ (c, d) \Leftrightarrow a \cdot c = b \cdot d$
  vs reduced ordered pairs in $\mathbb{Z} \times \mathbb{N} \setminus \{0\}$ (link)
- $\mathbb{R}$: quotient of Cauchy sequences vs Dedekind cut (link)

Design choices

- Lean defines $\mathbb{N}$ as an inductive type
- One could first define the set theory and a model of the Peano axioms in it

In the development of a mathematical library, there are **many** design choices, for each definition.

Which is the mathlib definition?

- $\mathbb{Z}$: quotient of $\mathbb{N} \times \mathbb{N}$ by $(a, b)\ (c, d) \Leftrightarrow a + c = b + d$
  vs positive and negative part (link)
- $\mathbb{Q}$: quotient of $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ by $(a, b)\ (c, d) \Leftrightarrow a \cdot c = b \cdot d$
  vs reduced ordered pairs in $\mathbb{Z} \times \mathbb{N} \setminus \{0\}$ (link)
- $\mathbb{R}$: quotient of Cauchy sequences vs Dedekind cut (link)

Writing Lean definitions

- If you have written the definitions and statements of your theorem in Lean and **ask an AI for a Lean proof**, it may well give one and Lean will check it.
- **If** you have written the definitions and statements of your theorem in **informal math** and **AI gives a Lean formalisation**, who checks it?

- If AI no.1 defines the real numbers in one way, say $\mathbb{R}_1$ while AI no.2 proves a theorem about the real numbers defined in another way $\mathbb{R}_2$, does this theorem apply to $\mathbb{R}_1$?
- **How do you know** that AI formalisation corresponds to your informal idea if definitions are buried in 1M LoC?

Writing Lean definitions

In `mathlib` (cf. Johan's talks), we have definitions and theorems about

- `Nat`, `Int`, `Rat`, `Real`, `Complex`.
- algebraic `Mul`, `Semigroup`, `Monoid`, `Group`, `Module`, `Algebra`
- topological `TopologicalSpace`, `MetricSpace`, `CompleteSpace`
- analytic `integral`, `fderiv`, `InnerProductSpace`, `CStarAlgebra`
- geometric `Bundle.RiemannianMetric`, `ContMDiffVectorBundle`
- and many more

which have been defined in a certain way.

(From mathlib contrubition guide)
"Mathlib intentionally has very high standards (on **generality, integration with the remaining library and maintainability, including code style)**. As of mid-2026, code written by an AI without the supervision of a Lean subject expert fails to meet that bar by a large margin."

Example of Lean definition: group

- **Human definition** of a group:

Example of Lean definition: group

- **Human definition** of a group: a set G with product operation $a \cdot b \in G$ which is associative, a distinguished unit element e and an inverse a^{-1} of any element $a \in G$.

Example of Lean definition: group

- **Human definition** of a group: a set G with product operation $a \cdot b \in G$ which is associative, a distinguished unit element e and an inverse a^{-1} of any element $a \in G$.
- `mathlib` **definition** of a group (link):

Example of Lean definition: group

- **Human definition** of a group: a set G with product operation $a \cdot b \in G$ which is associative, a distinguished unit element e and an inverse a^{-1} of any element $a \in G$.
- `mathlib` **definition** of a group (link):

```
class Group (G : Type u) extends DivInvMonoid G where
  protected inv_mul_cancel : ∀ a : G, a⁻¹ * a = 1
```

`Group` extends

- `DivisionMonoid` which extends
  - `Monoid` which extends `Semigroup`, `MulOneClass`, `NPow`..., `Inv`, `Div`, `ZPow`

Example of Lean definition: group

- **Human definition** of a group: a set G with product operation $a \cdot b \in G$ which is associative, a distinguished unit element e and an inverse a^{-1} of any element $a \in G$.
- `mathlib` **definition** of a group (link):

```
class Group (G : Type u) extends DivInvMonoid G where
  protected inv_mul_cancel : ∀ a : G, a⁻¹ * a = 1
```

`Group` extends
  - `DivisionMonoid` which extends
    - `Monoid` which extends `Semigroup`, `MulOneClass`, `NPow`..., `Inv`, `Div`, `ZPow`

- Any theorem about `Semigroup` should be true for `Monoid`, any theorem about `Monoid` should be true for `Group`, and we do not want to duplicate them for each.
- If we can prove a theorem with weaker assumptions, then we should.
- Abstract sturectures (`class`) vs examples (`instance`).

Example of Lean definition: derivative

- **Human definition** of derivative: for $f : \mathbb{R} \to \mathbb{R}$,

Example of Lean definition: derivative

- **Human definition** of derivative: for $f : \mathbb{R} \to \mathbb{R}$, $f'(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$ if the limit exist, otherwise undefined.

Example of Lean definition: derivative

- **Human definition** of derivative: for $f : \mathbb{R} \to \mathbb{R}$, $f'(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$ if the limit exist, otherwise undefined. For $f : \mathbb{R}^m \to \mathbb{R}^n$, define partial derivatives, differentiability...

Example of Lean definition: derivative

- **Human definition** of derivative: for $f : \mathbb{R} \to \mathbb{R}$, $f'(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$ if the limit exist, otherwise undefined. For $f : \mathbb{R}^m \to \mathbb{R}^n$, define partial derivatives, differentiability...
- `mathlib` **definition** of `fderiv` (link): for `E, F` topological vector spaces over a nontrivially normed field $\mathbb{k}$ and `x : F`, we define `f x` as
  - one of a $\mathbb{k}$-linear map `Df : E → F` satisfying

$$\mathtt{f\ x'} = \mathtt{f\ x} + \mathtt{Df}\ (\mathtt{x'} - \mathtt{x}) + \mathtt{o}(\mathtt{x'} - \mathtt{x}).$$

  - otherwize 0.

Example of Lean definition: integral

- **Human definition** of integral: let X be measurable spaces, $f : X \to \mathbb{R}$ a measurable function, μ a measure. $f = f_+ - f_-$. Approximate f_+ by step functions, define the integral for step functions by sum. Define $\int f_+ d\mu$ by the limit. Then $\int f d\mu = \int f_+ d\mu - \int f_- d\mu$.

Example of Lean definition: integral

- **Human definition** of integral: let X be measurable spaces, $f : X \to \mathbb{R}$ a measurable function, μ a measure. $f = f_+ - f_-$. Approximate f_+ by step functions, define the integral for step functions by sum. Define $\int f_+ d\mu$ by the limit. Then $\int f d\mu = \int f_+ d\mu - \int f_- d\mu$.
- `mathlib` **definition** of $\int$ `(x : X) f x` $\partial\mu$ (link): for `X` a measurable space, `Y` a normed vector space, `f : X → Y`.
  - If `Y` is not complete, $\int$ `(x : X) f x` $\partial\mu$ `= 0`.
  - If `f` is not the almost everywhere limit of step functions, $\int$ `(x : X) f x` $\partial\mu$ `= 0`.
  - If `(x : X) ↦ ‖ f x ‖` is Lebesgue integrable, then approximate `f` by step functions, prove that their integrals converge to a unique value in `Y` and take that value.

Example of Lean definition: integral

- **Human definition** of integral: let X be measurable spaces, $f : X \to \mathbb{R}$ a measurable function, μ a measure. $f = f_+ - f_-$. Approximate f_+ by step functions, define the integral for step functions by sum. Define $\int f_+ d\mu$ by the limit. Then $\int f d\mu = \int f_+ d\mu - \int f_- d\mu$.
- `mathlib` **definition** of $\int$ `(x : X) f x` $\partial\mu$ (link): for `X` a measurable space, `Y` a normed vector space, `f : X → Y`.
  - If `Y` is not complete, $\int$ `(x : X) f x` $\partial\mu$ `= 0`.
  - If `f` is not the almost everywhere limit of step functions, $\int$ `(x : X) f x` $\partial\mu$ `= 0`.
  - If `(x : X) ↦ ‖ f x ‖` is Lebesgue integrable, then approximate `f` by step functions, prove that their integrals converge to a unique value in `Y` and take that value.

Exactly the same machinary `MeasureTheory.setToFun` (link) is used to define the integral of vector valued measure against a linear pairing.
$\Rightarrow$ signed measures, complex measures, projection-valued measures, positive-operator-valued measures...

Which Lean definition do we want in mathlib?

- Sobolev spaces: one may want to consider the space $W^{k,p}(\mathbb{R})$ of functions f with the norm $\|f\|_{k,p} = \left(\sum_{i=0}^{k} \|f^{(i)}\|_p^p\right)^{\frac{1}{p}}$ finite.
  - There are equivalent norms. Which one should we take?
  - There are generalizations to $k \in \mathbb{R}$. Should we first define them?
  - We may take an open subset $\Omega \subset \mathbb{R}$. Should we first define them?
- Linear map $f : X \to Y$, $f(ax) = af(x)$. The graph of f is a $\mathbb{C}$-linear subspace of $X \oplus Y$.
  - Antilinear map $f : X \to Y$, $f(ax) = \bar{a}f(x)$ The graph of f is a $\mathbb{C}$-linear subspace of $X \oplus Y$, where $\mathbb{C}$ acts antilinearly on Y. But Lean allows $X \oplus Y$ to have only one $\mathbb{C}$-multiplication structure. Need to make a copy of $X \oplus Y$.
  - Semilinear map $f : X \to Y$, X is an R-module and Y is an S-module, $\sigma : R \to S$ a ring homomorphism, $f(ax) = \sigma(a)f(x)$.
- Complex-valued function $f : X \to \mathbb{C}$. Should we define $\operatorname{Re} f, \operatorname{Im} f$ as $X \to \mathbb{R}$ (better for integration theory) or $X \to \mathbb{C}$ (has vector-space structure)?

Summary

- It is important to have a centralized library of mathematical definitions and results.
- Design matters.
- **Some** `mathlib` definitions look very different from human definitions.
- **Lean does not check** whether definitions and statements correspond to **your informal idea**.
- Once the problem is stated correctly, AI may be able to solve it and Lean can check it.
- **We** need to add more definitions to `mathlib`.
- Can AI help us do it?

Outlook: Yang-Mills existence and mass gap

Can we **state** the Yang-Mills existence and mass gap problem?

- "Prove that for any compact simple gauge group G, a non-trivial quantum Yang–Mills theory exists on $\mathbb{R}^4$ and has a mass gap $\Delta > 0$. Existence includes establishing axiomatic properties at least as strong as those cited in [45, 35]". The Millennium Problems site
- If **something** satisfies the axiomatic properties of [45, 35], how do we know that it is the **Yang-Mills theory**?
- If we take a **specific construction** of the Yang-Mills theory (cf. Bałaban's 500 pages partial results), the statement would be "this specific construction satisfies the axiomatic properties of [45, 35]"
- "This specific construction" would involve lattices, inclusion of lattices, partially reblocked lattices, Lie group-valued functions, parameter-dependent neighbourhoods of the unit element of the Lie group, "averaging" group-valued functions, random walk expansion, cluster expansion, RG transformations, UV/IR limits...
- Can AI help **us undestand** them?