Formalizing mathematics in Lean

Floris van Doorn

November 9, 2021
**Proof assistants** are programs that can check the validity of a proof if that proof is written in a language it can understand.

In CS, this is used to prove that software or hardware has no bugs.

- CompCert: A formally-verified C compiler.

In math, this is used to prove deep mathematical theorems.

- The Kepler conjecture: A formally verified proof of a four centuries old problem
  - Reviewers were unable to check the proof given on paper.
Verifying Math/CS: similarities

Verifying math is similar to verifying software.

- You can use proof assistants for either purpose.
- You have to carefully write down the statement, including all assumptions.
- Automation of routine steps is immensely helpful.
Verifying Math/CS: differences

Verifying math is different from verifying software.
Math …
  ● recursively builds on itself
  ● is very interconnected

In CS …
  ● Often definitions are much larger
  ● Proofs often need to check many different cases
Lean is a proof assistant developed by Leonardo de Moura at Microsoft Research.

It has a type theory with dependent types, very similar to Coq.

```lean
def nextPrime (n : ℕ) : { m : ℕ // prime m ∧ ∀ (k : ℕ), prime k → n < k ↔ m ≤ k }
```

This can be seen as

- a program that computes the smallest prime larger than the input;
- a proof that there are infinitely many primes.
mathlib

mathlib is the mathematical library of Lean.

It contains many areas of mathematics in a single library.

It has 440k lines of code + 100k lines of documentation/comments.

There are 180+ contributors to mathlib. All contributions are reviewed by at least one of the maintainers.

There is a highly active community on leanprover.zulipchat.com with discussions, people helping each other, people teaching newcomers.
Let’s prove a basic exercise in topology:

**Lemma**

If \( f : X \to Y \) is a map between two topological spaces that is continuous at every \( x \in X \), then \( f \) is a continuous.

Recall:

**Definition**

\( f \) is **continuous** if \( f^{-1}(U) \) is open (in \( X \)) for every open \( U \subseteq Y \).

\( A \) is a **neighborhood** of \( x \) if it is a superset of an open set containing \( x \).

Notation: \( A \in \mathcal{N}_x \).

\( f \) is **continuous at** \( x \) if \( f^{-1}(A) \in \mathcal{N}_x \) for every neighborhood \( A \in \mathcal{N}_f(x) \).
Lemma

If \( f : X \to Y \) is a map between two topological spaces that is continuous at every \( x \in X \), then \( f \) is a continuous.

Proof.

Let \( U \subseteq Y \) be open. To show that \( f^{-1}(U) \) is open, it is sufficient to show that \( f^{-1}(U) \) is a neighborhood of every point \( x \in f^{-1}(U) \). Therefore take \( x \in f^{-1}(U) \). Then \( U \) is a neighborhood of \( f(x) \), hence \( f^{-1}(U) \) is a neighborhood of \( x \). Therefore \( f^{-1}(U) \) is open.
/** Abel-Ruffini Theorem**: not every polynomial root is expressible using radicals. */

```
theorem exists_not_solvable_by_rad:
  ∃ x : ℂ, is_algebraic ℚ x ∧ ¬ is_solvable_by_rad ℚ x
```

The proof specifically shows that the roots of $x^5 - 4x + 2$ are not expressible by radicals.
Topics in mathlib: calculus

/*-- Fundamental theorem of calculus, part 2 */
theorem integral_deriv_eq_sub
  (hderiv : ∀ x ∈ interval a b, differentiable_at \( \mathbb{R} \) f x)
  (hint : interval_integrable (deriv f) volume a b):
  \( \int y \in a..b, \text{deriv f y = f b - f a} \)
There does not exist a partition of a hypercube in dimension $n \geq 3$ into finitely many smaller cubes of different sizes.

/* **Dissection of Cubes**: A cube cannot be cubed. */

```
theorem cannot_cube_a_cube :
  \forall \{n : \mathbb{N}\}, n \geq 3 \rightarrow
  \forall \{\iota : \text{Type}\} \ [\text{fintype} \ \iota\} \ \{\text{cs : }\iota \rightarrow \text{cube } n\},
  2 \leq \#\iota \rightarrow
  \text{pairwise (disjoint on (cube.to_set } \circ \text{ cs)) } \rightarrow
  (\bigcup (i : \iota), (\text{cs } i).\text{to_set}) = \text{unit_cube.to_set } \rightarrow
  \text{injective (cube.width } \circ \text{ cs) } \rightarrow
  \text{false}
```
The goal of mathlib is to be a general-purpose library for all areas of mathematics.

It is decentralized: every contributor comes with their own plans and goals.

One unified goal is to have a full undergraduate math curriculum by next year.
Projects

- Definition of perfectoid spaces (Buzzard, Commelin, Massot)
- Independence of the Continuum hypothesis (Han, van Doorn)
- Witt vectors (Commelin, Lewis)
- Huang’s sensitivity theorem (Barton, Commelin, Han, Hughes, Lewis, Massot)
- Finiteness of the class group of a global field (Baanen, Dahmen, Narayanan, Nuccio)
- Liquid Tensor Experiment (Commelin et al)
- Sphere Eversion and convex integration (Massot, Nash, van Doorn)
Tools

We have various tools to help develop and maintain mathlib.

- Automatically generated documentation pages;
- Tactics for general-purpose or domain-specific automation
  - suggest searches the library for an applicable lemma.
  - simp: general-purpose simplifier.
  - abel, ring, linarith, omega, continuity: domain-specific automation.
  - tidy, finish, solve_by_elim: general purpose automation.
Consider:

```lean
def yoneda C ⇒ (C^op ⇒ Type v_1) :=
{ obj := λ X,
  { obj := λ Y, unop Y ---> X,
    map := λ Y Y' f g, f.unop ▶ g },
  map := λ X X' f, { app := λ Y g, g ▶ f } }

@[simp] lemma obj_obj (X : C) (Y : C^op) :
  (yoneda.obj X).obj Y = (unop Y ---> X) := rfl

@[simp] lemma obj_map (X : C) {Y Y' : C^op} (f : Y ---> Y') :
  (yoneda.obj X).map f = λ g, f.unop ▶ g := rfl

@[simp] lemma map_app {X X' : C} (f : X ---> X') (Y : C^op) :
  (yoneda.map f).app Y = λ g, g ▶ f := rfl
```

These three lemmas can be automatically generated by the @[simp] attribute.
Tools: Semantic Linters

We have a suite of semantic linters: they look through mathlib for common mistakes.

They run on every new commit to mathlib.

Some mistakes that it catches:

- A lemma has a hypothesis that is never used;
- A definition is incorrectly marked as a lemma;
- A definition has no documentation string;
- You created a loop in the simplification lemmas;
- You created a loop in the type-class search.
- ...

Design Decisions

In mathlib we have made some design decisions to make it convenient to formalize mathematics.

- Most of the library uses classical logic.
- Definitional equality is used sparingly.
- Dependent types and quotients are used to define general types. Example: $L^1(X,Y;\mu)$.
- We have a single repository where all components can work together.
Refactoring mathematics

One important feature of mathlib is to reduce duplication of proofs.

- Definitions and proofs should be in the greatest generality possible (within reason).
- This regularly requires refactoring of mathematics.
- There are also regular large-scale refactors on the library to make basic definitions more convenient or more general.
Refactoring mathematics: limits

\[
\begin{align*}
\lim_{x \to x_0} f(x) &= y_0 \\
\lim_{x \to x_0^-} f(x) &= -\infty \\
\lim_{x \to x_0^+} f(x) &= y_0^+ \\
\lim_{x \to -\infty} f(x) &= y_0
\end{align*}
\]

There are many different versions of limits.
Refactoring mathematics: limits

\[
\begin{align*}
\lim_{x \to x_0} f(x) &= y_0 & \lim_{x \to x_0^-} f(x) &= -\infty \\
\lim_{x \to x_0} f(x) &= y_0 & \lim_{x \to -\infty} f(x) &= y_0^+
\end{align*}
\]

There are many different versions of limits.

These can all be unified by defining limits in terms of filters.

\[
\mathcal{N}_x = \text{neighborhoods of } x \\
\mathcal{N}_x^+ = \{ A \mid A \cap [x, \infty) \text{ is a neighborhood of } x \text{ in } [x, \infty) \} \\
\mathcal{N}_{+\infty} = \{ A \mid \exists y, [y, \infty) \subseteq A \}
\]

\(f : X \to Y\) converges to a filter \(G\) on \(Y\) along filter \(F\) on \(X\) if

\[\forall A \in G, \ f^{-1}(A) \in F.\]
Type Classes

A major component of reusability of definitions is the use of type-classes.

This allows us to define concepts and prove theorems about general types with some structure.

Applications:
- Common operations: like $a \times b$, $\|x\|$, $\square_i A_i$.
- General theory: group, normed_space, complete_lattice
- Decidable propositions for implementing decision procedures.

They play a similar role as canonical structures in Coq.
class has_mul (A : Type) := (mul : A → A → A)

class semigroup (A : Type) extends has_mul A :=
  (mul_assoc : ∀ a b c, (a * b) * c = a * (b * c))

class monoid (A : Type) extends semigroup A, has_one A :=
  (one_mul : ∀ a, 1 * a = a) (mul_one : ∀ a, a * 1 = a)

variables {A : Type} [monoid A]
def pow (a : A) : ℕ → A
| 0 := 1
| (n+1) := a * pow n

theorem pow_add (a : A) (m : ℕ) : ∀ n, a ^ (m + n) = a ^ m * a ^ n
| 0 := by rw [add_zero, pow_zero, mul_one]
| (n+1) := by rw [add_succ, pow_succ, pow_add, mul.assoc]

instance : linear_ordered_comm_ring ℤ := ...
A part of the type-class hierarchy in October 2019.
Type Classes

In total there are 760 classes with more than 11000 instances between the classes.

They are used essentially everywhere.

Classes typically have a type as argument, like `topological_space X`.

There are also many “mixin” type-classes that depend on another type class, like `second_countable_topology X`, `compact_space X`, `t2_space X`, `connected_space X`, ...

In fact, there are 25 classes that depend on `topological_space` (and no other classes).

There are also many classes that depend on multiple other classes: `[ring R] [topological_space R] [topological_ring R]`
Example: intermediate fields

In algebra, you often work with field extensions. \( L/K \) (\( L \) is an extension of \( K \)) means that \( K \) is a subfield of \( L \). Example: \( \mathbb{C}/\mathbb{R} \).

One often has to work with multiple extensions: \( F \) is an intermediate field if \( L/F \) and \( F/K \).
Example: intermediate fields

In algebra, you often work with field extensions. \( L/K \) (\( L \) is an extension of \( K \)) means that \( K \) is a subfield of \( L \). Example: \( \mathbb{C}/\mathbb{R} \).

One often has to work with multiple extensions: \( F \) is an intermediate field if \( L/F \) and \( F/K \).

One option: \( F \) and \( K \) are subsets of the type \( L \).
Problem: inconvenient in practice: the type \( \mathbb{R} \) is not the same as the subset \( \mathbb{R} \subseteq \mathbb{C} \).
Example: intermediate fields

In algebra, you often work with field extensions. $L/K$ ($L$ is an extension of $K$) means that $K$ is a subfield of $L$. Example: $\mathbb{C}/\mathbb{R}$.

One often has to work with multiple extensions: $F$ is an intermediate field if $L/F$ and $F/K$.

One option: $F$ and $K$ are subsets of the type $L$.
Problem: inconvenient in practice: the type $\mathbb{R}$ is not the same as the subset $\mathbb{R} \subseteq \mathbb{C}$.

```
variables {K F L : Type*}
variables [field K] [field F] [field L]
variables [algebra K F] [algebra F L] [algebra K L]
variables [is_scalar_tower K F L]
```
Lean 4

A new version of Lean was released this year.

It is a full-fledged dependently-typed programming language, with a compiler to C.

It is highly extensible, with a flexible macro system.

It is not backwards compatible, but mathlib will be ported to Lean 4 next year:

- We can already port compiled Lean 3 files to compiled Lean 4 files (binport).
- There is an experimental method to port source Lean 3 to source Lean 4 files (synport).
- We need to manually implement the Lean 3 tactics in Lean 4: the meta-language has changed significantly.
Thank You