Introduction to Formal Mathematics with Lean 4

For newcomers to formalization methods

Xingyu Zhong

Beijing Institute of Technology

2025-09-17

Table of Contents

Table of Contents

Natural language vs. formal language

- ambiguity in natural language
  - implicit assumptions
  - skipping details: "It's clear that we have..."
  - "viewed as" arguments: $V^* = V$, $(A \times B) \times C = A \times (B \times C)$ [1]
  - abuses of notation: $3 \in \mathbb{Z}/5\mathbb{Z}$, $\mathbb{C} \subseteq \mathbb{C}[x]$
- precision in formal language
  - computer programs are formal languages

[1] Knowledgable audience may recognize them as examples of natural isomorphisms in category theory.

Mathematical proofs vs. Computer programs [2]

Logic	Programming
proposition	type
proof	term
proposition is true	type has a term
proposition is false	type doesn't have a term
logical constant `TRUE`	unit type
logical constant `FALSE`	empty type
implication $\to$	function type
conjunction $\wedge$	product type $\prod$
disjunction $\vee$	sum type $\sum$
universal quantification $\forall$	dependent product type $\prod$
existential quantification $\exists$	dependent sum type $\sum$

Table: Curry–Howard correspondence

[2] see also Computational Trilogy, with category theory as the third vertex

Set theory vs. Type theory

- Mathematicians choose axiomatic set theory (with first-order logic) as the foundation of mathematics.
  - naive set theory fits human's intuition well
- Type theory is an alternative foundation that is equally expressive, but more suitable for computer formalization.

Set Theory	Type Theory
everything is a set	everything has a type
$3 \in \mathbb{R}$ is a proposition	$(3 : \mathbb{R})$ is a typing judgment
$\mathbb{Q} \subseteq \mathbb{R}$ is an inclusion	$\mathbb{Q} \to \mathbb{R}$ is a type conversion

What is Lean 4

- A mordern functional programming language designed for theorem proving

  "Lean is based on a version of dependent type theory known as the Calculus of Constructions, with a countable hierarchy of non-cumulative universes and inductive types." — *Theorem Proving in Lean 4*

Lean's dependent type theory

- Dependent type theory is a powerful extension of type theory where
  - types may depend on terms "given before" them
  - first-order logic can be implemented in dependent type theory
- functions, inductive types and quotient types[3] are the basic methods to construct new types.

Set Theory	Lean's dependent type theory
$\forall x \in \mathbb{R},\ x^2 \geq 0$	has type $(x : \mathbb{R}) \to (x^2 \geq 0)$
$(n \in \mathbb{N}) \mapsto (1, 0, \ldots, 0) \in \mathbb{R}^n$	has type $(n : \mathbb{N}) \to \mathbb{R}^n$
$\{0, 1\} = 2$ is a set equality	make no sense
cardinality is an equivalence class	is a quotient type
Russell's paradox	Girard's paradox

[3] Though seemingly redundant, there is a reason for making quotient types as a fundamental constructing method.

An example Lean 4 code

- FLT
- TendsTo

Table of Contents

The rise of AI

AI excels in Python. Why not Lean?

- Automated theorem proving
  - especially those "abstract nonsense"
- Natural language to formal language
  - automatically transplanting textbooks and papers into Lean
  - Converse? Already happening!
- Proposing conjectures
  - on which facts should we care about

Rigor matters

- It's the foundation of mathematics
- Inprecise natural language often leads to misunderstandings and glitches
  - Especially when proofs get longer and longer
- formalization fully confirms the correctness of a theorem
  - things that are too "technical" (boring) or simply impossible to verify by oneself
    - e.g. classification of finite simple groups
    - e.g. "technical"

"I spent much of 2019 obsessed with the proof of this theorem, almost getting crazy over it. In the end, we were able to get an argument pinned down on paper, but I think nobody else has dared to look at the details of this, and so I still have some small lingering doubts." — *Peter Scholze*

“Mathematical engineering”

- manipulating tons of theorems and proofs with mature software engineering techniques
- referencing existing theorems as dependencies
- collaborative work across the globe

The beauty of the system: you do not have to understand the whole proof of FLT in order to contribute. The blueprint breaks down the proof into many many small lemmas, and if you can formalise a proof of just one of those lemmas then I am eagerly awaiting your pull request. — *Kevin Buzzard on the FLT Project*

Formalization as learning

- proofs with infinite detail
  - intuitive textbooks, rigorous formalization
- makes us understand things better
  - Global: How to build natural numbers from scratch?
    - natural number game
    - A journey to the world of numbers, by Riccardo Brasca
  - Local: reducing the cognitive load
    - (With good organization at the beginning) you can focus on small parts of the proof at a time

Table of Contents

Mathematician-friendly community

- mathematician-friendly languages, interfaces, tools and community emerges

Mathlib 4 is expanding explosively

- undergraduate may contribute: some low fruits

Table of Contents

What you will learn

A typical Lean file

- mathematical analysis

Drawbacks

- Formalization is tedious in its nature, Lean 4 is no exception
- Type conversions can be an extra burden (exclusive for type-theory-based systems)
- Knowledge needs to be re-learned before being referenced

References

blah