Formal mathematics, dependent type theory, and the Topos Institute

Jeremy Avigad

Department of Philosophy
Department of Mathematical Sciences

Hoskinson Center for Formal Mathematics

Carnegie Mellon University

November 4, 2021

Outline and a warning

Outline:

- Formal mathematics
- Lean and the Lean community
- Why formalize mathematics?
- Dependent type theory
- Formalism and the Topos Institute

This talk overlaps:

- Kevin Buzzard's talk in this series.
- My presentation at the Hoskinson Center inauguration.
- Patrick Massot's recent talk at the New Technologies in Mathematics Seminar at Harvard.

Formalization of mathematics

"The development of mathematics toward greater precision has led, as is well known, to the formalization of large tracts of it, so one can prove any theorem using nothing but a few mechanical rules. The most comprehensive formal systems that have been set up hitherto are the system of *Principia Mathematica* on the one hand and the Zermelo-Fraenkel axiom system of set theory . . . on the other. These two systems are so comprehensive that in them all methods of proof used today in mathematics are formalized, that is, reduced to a few axioms and rules of inference. One might therefore conjecture that these axioms and rules of inference are sufficient to decide any mathematical question that can at all be formally expressed in these systems."

Formalization of mathematics

"It will be shown below that this is not the case. . . ."

(Kurt Gödel, *On formally undecidable propositions of Principia Mathematica and related systems*, 1931.)

The positive claim: most ordinary mathematics is formalizable, *in principle*.

Formalization of mathematics

"It will be shown below that this is not the case, that on the contrary there are in the two systems mentioned relatively simple problems in the theory of integers that cannot be decided on the basis of the axioms. This situation is not in any way due to the special nature of the systems that have been set up, but holds for a wide class of formal systems; among these, in particular, are all systems that result from the two just mentioned through the addition of a finite number of axioms, provided [the system is still ω-consistent]."

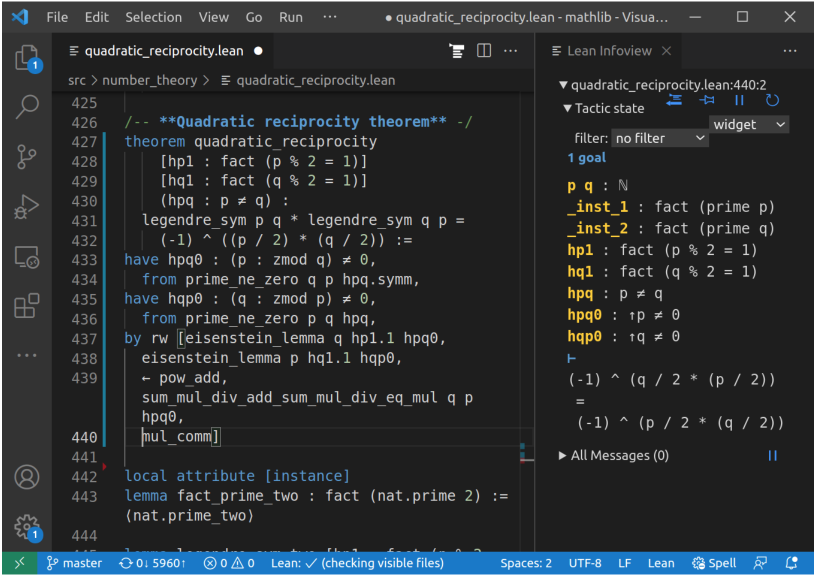
Formalization of mathematics

With the help of computational proof assistants, mathematics is formalizable *in practice*.

Working with such a proof assistant, users construct a formal axiomatic proof.

Systems with substantial mathematical libraries include Mizar, HOL, Isabelle, HOL Light, Coq, ACL2, PVS, Agda, Metamath, and Lean.

Formalization of mathematics



Formalization of mathematics

Most undergraduate mathematics, and a fair amount of graduate school mathematics, has been formalized.

A number of "big name" theorems have been formalized: the prime number theorem, the four color theorem, Dirichlet's theorem, the central limit theorem, the incompleteness theorems.

Gonthier et. al completed a verification of the Feit-Thompson theorem in 2012.

The *Flyspeck* project completed its verification of Hales' proof of the Kepler conjecture in 2014.

The Lean theorem prover

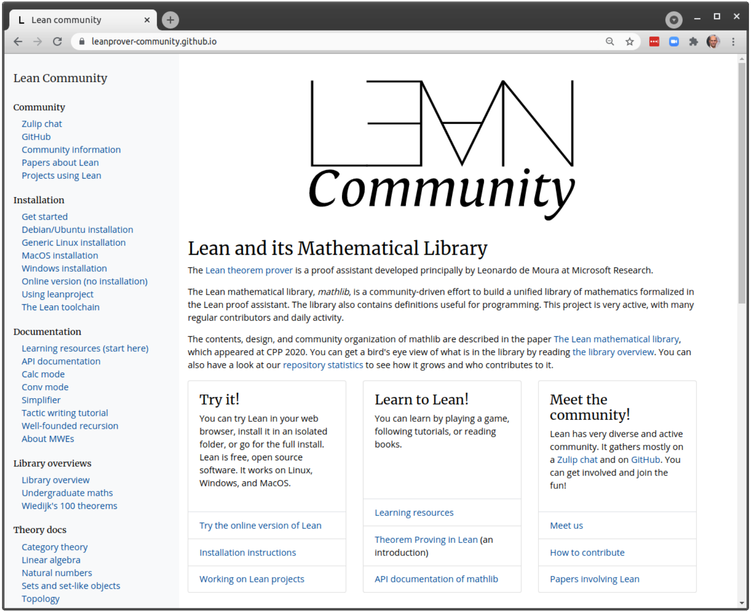
Lean is a new interactive theorem prover, developed principally by Leonardo de Moura at Microsoft Research, Redmond.

Some history:

- 2013: the project begins
- 2014: Lean 2 released
- 2016: Lean 3 released
- 2017: mathlib moved to a separate repository
- 2021: prerelease of Lean 4

Lean 3 and mathlib are maintained by a lively community of volunteers, chiefly mathematicians and computer scientists (most of them young).

The Lean community



The Lean community

There have been some notable successes.

Kevin Buzzard, Johan Commelin, and Patrick Massot formalized the notion of a *perfectoid space*.

Sander Dahmen, Johannes Hölzl, and Robert Lewis formalized a proof of the Ellenberg-Gijswijt theorem.

Jesse Han and Floris van Doorn formalized a proof of the independence of the continuum hypothesis.

Patrick Massot has launched a project to formalize sphere eversion.

The Lean community

On December 5, 2020, Peter Scholze challenged anyone to formally verify some of his recent work with Dustin Clausen.

Johan Commelin led the response from the Lean community. On June 5, 2021, Scholze acknowledged the achievement.

"Exactly half a year ago I wrote the Liquid Tensor Experiment blog post, challenging the formalization of a difficult foundational theorem from my Analytic Geometry lecture notes on joint work with Dustin Clausen. While this challenge has not been completed yet, I am excited to announce that the Experiment has verified the entire part of the argument that I was unsure about. I find it absolutely insane that interactive proof assistants are now at the level that within a very reasonable time span they can formally verify difficult original research."

The Lean community

Formal mathematics is finally getting some recognition.

- The Lean Zulip channel is lively.
- Kevin Buzzard's blog posts and talks go viral.
- Lean and mathlib have been getting good press:
  - *Quanta:* "Building the mathematical library of the future"
  - *Quanta:* "At the Math Olympiad, computers prepare to go for the gold"
  - *Nature:* "Mathematicians welcome computer-assisted proof in 'grand unification' theory"
- Lean workshops are planned at ICERM (2022), MSRI (2023), and more.

Outline

- Formal mathematics
- Lean and the Lean community
- Why formalize mathematics?
- Dependent type theory
- Formalism and the Topos Institute

Why formalize mathematics?

Reason #5: Correctness.

Mathematics is about rigor and precison.

We want our proofs to be correct.

Formalization isn't a replacement for understanding, but the mathematics that we understand isn't meaningful if it isn't correct.

Why formalize mathematics?

Reason #4: Libraries.

Digital technology allows us to develop communal repositories of knowledge.

Every definition, theorem, and proof is recorded, and can be accessed.

Readers can determine how much detail they want to see.

Libraries support exploration and search.

Why formalize mathematics?

Reason #3: Education.

Formalism is demanding, and can be frustrating at times.

But it provides instant feedback, instant gratification, and fun.

Formal tools can be designed for different audiences, from elementary school students to PhD students.

Why formalize mathematics?

Reason #2: Discovery.

Symbolic AI and machine learning have had a profound impact on hardware and software verification, AI, planning, constraint solving, optimization, knowledge representation, expert systems, databases, language processing, ...

But they have had almost no impact on pure mathematics.

We have no idea what the tools can do, and there is a lot we need to learn.

Formalization is a gateway to automation.

Why formalize mathematics?

Reason #1: Collaboration.

The Lean Zulip channel is remarkable. People ask questions, explain things, pose challenges, share results, discuss plans.

Newcomers are welcome. Each successive generation helps the next.

Of course, we can collaborate without formalism.

But contributing to a formal library is *transcendental*, and provides focus.

Outline

- Formal mathematics
- Lean and the Lean community
- Why formalize mathematics?
- Dependent type theory
- Formalism and the Topos Institute

Dependent type theory

There are three main classes of logical foundations used by proof assistants today:

- set theory
- simple type theory
- dependent type theory

Debates over which is the "best" or "true" foundation leads to inane arguments.

Dependent type theory has advantages and disadvantages.

I'll try to explain some of the advantages.

Dependent type theory

In set theory, everything in a set: numbers, functions, tuples, triangles, groups, measures, ...

But it helps to keep track of what *types* of objects we are dealing with:

- The meaning of $x \cdot y$ can be inferred from the types of x and y.
- The meaning of $\sum_{x \in A} f(x)$ can be inferred from the codomain of f.
- The expression $gcd(5, f)$ is probably a typographical error.

Dependent type theory

In a typed framework, every expression has a *type*.

```
variables m n : ℤ
variables x y : ℝ
variables w z : ℂ
variable  R   : Ring
variables a b : R.carrier

#check m * 7 + n
#check x * 7 + y
#check w * 7 + z
#check a * 7 + b
```

Dependent type theory

In simple type theory, *type constructors* build compound types:

```
variables m n : ℕ
variable  b   : bool
variable  f   : ℕ → ℕ
variable  g   : ℕ × ℕ → bool
variable  F   : (ℕ → ℕ) → ℕ

#check f m
#check g (m, n)
#check F f
```

Dependent type theory

Systems generally allow *polymorphic* constructions over types:

```
variable  α     : Type
variables a b   : α
variables s t   : list α
variables m n   : ℕ
variables l₁ l₂ : list ℕ

#check a :: s ++ t
#check m :: l₁ ++ l₂
```

Dependent type theory

In dependent type theory, types can depend on parameters:

```
variables m n : ℕ
variables v   : vector ℝ 2
variable  w   : vector ℝ 3
variable  u   : vector ℝ (m^2 + 1)
variable  M   : matrix ℝ 3 2
variable  K   : matrix ℝ (m + 1) (n + 2)
```

This is useful with structures:

```
variable    R : Ring
variables a b : Ring.carrier R
```

Dependent type theory

Dependent types complicate things:

- A type checker needs to determine whether an expression has a given type.
- A type can depend on *any* expression.

```
def m := if goldbach_conjecture then 3 else 4

variable s : vector ℝ 3
variable t : vector ℝ m

#check s + t
```

Moral: dependent types can be useful, but they should be used wisely and sparingly.

Dependent type theory

In dependent type theory, everything is an expression.

- Data types: `T : Type`
- Objects: `t : T`
- Mathematical statements: `P : Prop`
- Proofs: `p : P`

Two fundamental operations:

- `#eval t`
- `#check p`

Dependent type theory

In dependent type theory, everything is an expression.

- Data types: `T : Type`
- Objects: `t : T`
- Mathematical statements: `P : Prop`
- Proofs: `p : P`

Two fundamental operations:

- `#eval t` (all of computer programming)
- `#check p`

Dependent type theory

In dependent type theory, everything is an expression.

- Data types: `T : Type`
- Objects: `t : T`
- Mathematical statements: `P : Prop`
- Proofs: `p : P`

Two fundamental operations:

- `#eval t` (all of computer programming)
- `#check p` (all of mathematics)

Dependent type theory

```
theorem quadratic_reciprocity (p q : ℕ)
    (primep : prime p) (primeq : prime q)
    (hp : p % 2 = 1) (hq : q % 2 = 1)
    (hpq : p ≠ q) :
  legendre_sym p q * legendre_sym q p =
    (-1) ^ (p / 2) * (q / 2) :=
...
```

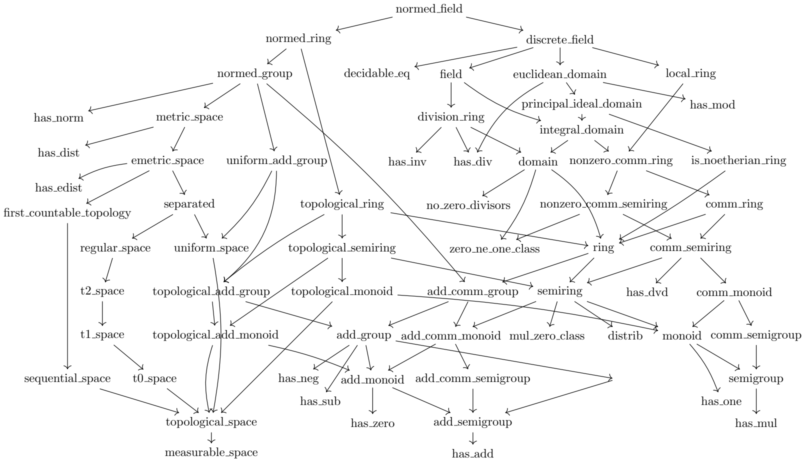
Dependent type theory

```
structure group (G : Type) : Type :=
(mul : G → G → G)
(mul_assoc : ∀ (a b c : G), (a * b) * c = a * b * c)
(one : G)
(one_mul : ∀ (a : G), 1 * a = a)
...

def zmod : ℕ → Type
| 0     := ℤ
| (n+1) := fin (n+1)

instance comm_ring :
  Π (n : ℕ), comm_ring (zmod n) :=
...
```

Dependent type theory



Dependent type theory

For programmers, it's a framework in which one can:

- write computer programs
- write specifications
- prove that the programs meet their specifications
- statically enforce runtime guarantees
- use powerful abstractions (type classes, monads)

. . . all in the same language.

Dependent type theory

For mathematicians, it's a framework in which one can:

- make mathematical statements
- prove mathematical theorems
- compute with mathematical objects
- develop algebraic abstractions

. . . all in the same language, and in a verified way.

Dependent type theory

Metaprogramming makes it possible to

- define new syntax and domain-specific languages
- develop small-scale automation and reasoning procedures
- develop large-scale automation and tools
- extend the capabilities of the system

. . . all in the same language, and all in the same system.

Formalism and the Topos Institute

Lean has an impressive category theory library.

Authors: Scott Morrison, Bhavik Mehta, and many others.

The Liquid Tensor Experiment should also be of interest.

Formalism and the Topos Institute

Projects:

- Connected intelligence: Foundational mathematics for understanding computation and intelligence
- Model-Based Computing: Theory and software for computing with scientific models
- Networked Mathematics: Organizing mathematics into a coherent, searchable whole

Can formal methods contribute to these projects?

Formalism and the Topos Institute

Formalization can also play a role in applied mathematics, like engineering and finance.

Formal verification down to axiomatic primitives is usually more than most users want or need.

Interactive proof assistants can offer:

- formal specification of a complex problem or model
- a gateway to the use of external tools like computer algebra systems, numeric computation packages, and automated reasoning systems
- selective verification of most critical or sensitive results.

Even the first alone is very important.

Model-Based Computing

"Scientific models are traditionally programmed by hand, accreting bells and whistles over years or decades. Even moderately complex models can be difficult to specify in conventional natural and mathematical language, leading some to adopt the slogan that 'the code is the model.' This conflation, a response to inadequate conceptual and computational tools, severely impacts both the productivity of scientists and the reliability of their science. Models-as-code are laborious and error prone to create, modify, or extend.

"We are developing the theory and software that will enable scientific and statistical models to be treated as first-class entities, which may be created, transformed, compared, and executed with the same ease as conventional data structures. Mathematically, we draw on ideas from category theory, especially categorical logic. . . "

Connected Intelligence

“In this theme we create new, fundamental mathematical languages for computation and intelligence.

“For example, our current model of computation is based on the Turing machine. Just as Roman numerals do indeed specify numbers, Turing machines do specify computations. However, the model is clunky and ad hoc, and it does not take into account anything other than a single disk and a single processor: no keyboard, monitor, or printer, no user, no internet. But like Arabic numerals, there's a mathematical formalism called polynomial functors that is much more versatile. In particular, it allows for machines (multiple disks, processors, keyboards, monitors, even routine mental procedures) to connect or disconnect, to send information to each other, and to thereby organize to solve larger problems.”

Networked Mathematics

“How can it be so hard for even experts to find widely published, basic results?”

“To solve this problem, we build MathFoldr, a search tool for mathematics. MathFoldr leverages both statistical, similarity-based methods from natural language processing and logical, semantic methods from proof assistants to integrate mathematical knowledge into a coherent, searchable whole.”

Digital representations can / should complement natural language representations.

Parting thoughts

Shankar: “We are in the golden age of metamathematics.”

It’s an exciting time to be involved.