Formal Mathematics and the Lean Theorem Prover

Jeremy Avigad

Department of Philosophy and
Department of Mathematical Sciences
Carnegie Mellon University

Logicians in Quarantine

August 2020

Interactive theorem proving

"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."

Interactive theorem proving

"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*.

Interactive theorem proving

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

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

In many systems, this proof object can be extracted and verified independently.

Interactive theorem proving

Some systems with substantial mathematical libraries:

- Mizar (set theory)
- HOL (simple type theory)
- Isabelle (simple type theory)
- HOL Light (simple type theory)
- Coq (constructive dependent type theory)
- ACL2 (primitive recursive arithmetic)
- PVS (classical dependent type theory)
- Agda (constructive dependent type theory)
- Metamath (set theory)
- Lean (dependent type theory)

Interactive theorem proving

Some theorems formalized to date:

- the prime number theorem (2004, and via complex analysis, 2009)
- the four-color theorem (2004)
- the Jordan curve theorem (2005)
- Gödel's first and second incompleteness theorems (1986 and 2013, respectively)
- Dirichlet's theorem on primes in an arithmetic progression (2009)
- the central limit theorem (2014)
- the independence of the continuum hypothesis (consistency, 2008, unprovability, 2019)

Interactive theorem proving

There are good libraries for

- elementary number theory
- real and complex analysis
- point-set topology
- measure-theoretic probability
- linear algebra
- group theory
- category theory
- dynamical systems

. . . and lots more.

Interactive theorem proving

Georges Gonthier and coworkers verified the Feit-Thompson Odd Order Theorem in Coq.

- The original 1963 journal publication ran 255 pages.
- The formal proof is constructive.
- The development includes libraries for finite group theory, linear algebra, and representation theory.

The project was completed on September 20, 2012, with roughly

- 150,000 lines of code,
- 4,000 definitions, and
- 13,000 lemmas and theorems.

Interactive theorem proving

Thomas Hales announced the completion of the formal verification of the Kepler conjecture (*Flyspeck*) in August 2014.

- Most of the proof was verified in HOL light.
- The classification of tame graphs was verified in Isabelle.
- Verifying several hundred nonlinear inequalities required roughly 5000 processor hours on the Microsoft Azure cloud.

Interactive theorem proving

"It is not in heaven, that thou shouldest say: 'Who shall go up for us to heaven, and bring it unto us, and make us to hear it, that we may do it?' " (*Deuteronomy* 30:12)

You can download these systems and get started right away.

- Isabelle: `https://isabelle.in.tum.de/`
- Coq with Mathematical Components: `https://math-comp.github.io/`
- Metamath: `http://us.metamath.org/`

There are online documentation, tutorials, user mailing lists, online chat groups, and more.

Interactive theorem proving

Later, I will talk about the Lean theorem prover.

You can find resources for learning about Lean at the Lean community web pages:

`https://leanprover-community.github.io/`.

In particular:

- *Theorem Proving in Lean*
- *Mathematics in Lean*
- *Lean for the Curious Mathematician* (a workshop, with recorded tutorials)

Table of contents

Outline:

- Interactive theorem proving
- Why mathematicians should care
- Why logicians should care
- The *Lean* theorem prover

Why mathematicians should care

Mathematics strives for

- *rigor* and
- *understanding*.

In 1993, Arthur Jaffe and Frank Quinn published an article in *Bulletin of the AMS* which distinguished between

- rigorous mathematics and
- theoretical mathematics,

with a proposal to mediate between the two.

Prominent mathematicians, physicists, computer scientists, and historians weighed in with responses.

Why mathematicians should care

In the late 70s and early 80s, Miller Brewing Company ran a series of advertisements for its light beer.

Hordes of riled-up beer drinkers argued whether Miller Lite

- *tastes great* or is
- *less filling*.

Mathematical rigor and understanding are not at odds.

The rules of mathematics

- allow us to share understanding,
- provide a scaffolding for understanding, and
- make our understanding meaningful.

Why mathematicians should care

Formal proof assistants are *tools* that can help us do mathematics better.

Compare to Latex:

- Mathematics is not just about communicating results.
- But typesetting is important.
- There is a learning curve.
- But it's worth the effort if it helps us communicate better.

Interactive theorem proving is not there yet.

Why mathematicians should care

ITP is an instance of *formal methods*, which are used for

- specifying,
- developing, and
- verifying

complex hardware and software systems.

They rely on:

- *formal languages* to make assertions and express constraints,
- *formal semantics* to specify intended meaning, and
- *formal rules of inference* to verify claims and carry out search.

Why mathematicians should care

Applied to mathematics, they can provide:

- verified proof
- verified computation
- formal search methods, to support new results
- digital infrastructure for storing, sharing, and communicating results

See my article in the *Notices of the AMS*, *The mechanization of mathematics*, and an associated talk.

See also a recent article in *Quanta* on the resolution of the Keller conjecture.

ITP is a gateway to formal methods in general.

Why logicians should care

Formal methods rely on classical results in logic:

- formal languages and axiomatic systems
- semantics, definability, and completeness proofs
- structural proof theory
- computability theory
- normalization and the lambda calculus
- Skolemization and properties
- decision procedures
- model theory of algebraic structures
- Craig's interpolation lemma

Computer scientists have added many important insights, but the theory guides the enterprise.

Why logicians should care

But what have we done lately?

In the twentieth century, mathematical logic made important contributions, clarifying

- basic concepts of mathematics
- methods of proof and rules of inference
- notions of language, meaning, expressivity, and definability
- the notion of computation

These had bearing on all aspects of mathematics, as well as computer science, linguistics, philosophy, and beyond.

Formal methods are the modern embodiment of this tradition.

Why logicians should care

The three main families of foundations:

- Set theory (Mizar, Metamath)
- Simple type theory (HOL4, Isabelle, HOL Light)
- Dependent type theory (Coq, Agda, PVS, Lean)

For an overview, see my draft chapter, Foundations, for an upcoming *Handbook for Proof Assistants and their Applications in Mathematics and Computer Science*.

In set theory, everything is a set. In type theory, every object has its own type.

Why logicians should care

Theoretically, the differences are irrelevant:

- We can interpret types as sets.
- We can construct (or posit) types whose elements satisfy axioms of set theory.

The differences have to do with user interaction:

- Types allow for overloading.
- Types provide syntactic error checking.
- Types can be used to infer information (like associated algebraic structure).

Why logicians should care

```
/- Author: Chris Hughes -/

def legendre_sym (a p : ℕ) (hp : prime p) : ℤ :=
  if (a : zmodp p hp) = 0 then 0 else
    if ∃ b : zmodp p hp, b ^ 2 = a then 1 else -1

theorem quadratic_reciprocity (hp1 : p % 2 = 1)
    (hq1 : q % 2 = 1) (hpq : p ≠ q) :
  legendre_sym p q hq * legendre_sym q p hp =
    (-1) ^ ((p / 2) * (q / 2))

...

lemma exists_subgroup_card_pow_prime
    {G : Type _} [group G] [fintype G] (p : ℕ) :
  ∀ {n : ℕ} [hp : p.prime] (hdvd : p ^ n ∣ card G),
    ∃ H : subgroup G, fintype.card H = p ^ n
```

Why logicians should care

```
/- Author: Scott Morrison -/

variables {C : Type u} [category.{v} C]

def yoneda : C ⟹ (C^op ⟹ Type v) :=
{ obj := λ X,
  { obj := λ Y, unop Y ⟶ X,
    map := λ Y Y' f g, f.unop ≫ g,
    map_comp' := ...,
    map_id' := ...},
  map := λ X X' f, { app := λ Y g, g ≫ f } }

instance yoneda_full : full (@yoneda C _) :=
...

instance yoneda_faithful : faithful (@yoneda C _) :=
...
```

Why logicians should care

```
/- Author: Sebastien Gouezel -/

/-- Typeclass defining smooth manifolds with corners with
    respect to a model with corners, over a field `𝕂` and
    with infinite smoothness to simplify typeclass search
    and statements later on. -/

class smooth_manifold_with_corners
  {𝕂 : Type _} [nondiscrete_normed_field 𝕂]
  {E : Type _} [normed_group E] [normed_space 𝕂 E]
  {H : Type _} [topological_space H]
  (I : model_with_corners 𝕂 E H)
  (M : Type _) [topological_space M] [charted_space H M]
extends
  has_groupoid M (times_cont_diff_groupoid ∞ I) : Prop
```

Why logicians should care

Automated mathematical reasoning is a new frontier.

Domain-general methods:

- Propositional theorem proving
- First-order theorem proving
- Higher-order theorem proving
- Equality reasoning
- Combination methods

Domain-specific methods:

- Linear arithmetic (integer, real, or mixed)
- Nonlinear real arithmetic (real closed fields, transcendental functions)
- Algebraic methods (such as Gröbner bases)

Why logicians should care

Here are some major challenges:

- developing languages that are expressive in practice
- developing foundations that are adequate in practice
- developing practical search methods and decision procedures
- managing large databases of knowledge
- combining heterogeneous methods
- using external automation
- using computer algebra systems
- sharing data between proof systems
- verifying numeric computation
- understanding what machine learning can (and cannot) do

We need to pay attention to the mathematics!

Table of contents

Outline:

- Interactive theorem proving
- Why mathematicians should care
- Why logicians should care
- The *Lean* theorem prover

The Lean Theorem Prover

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

It is open source, released under a permissive license, Apache 2.0.
See http://leanprover.github.io.

The project began in 2013.

In 2017, the Lean community split off the library.

- Lean's developers can focus on Lean 4 without distraction.
- The community has been maintaining Lean 3 and building the library.

See http://leanprover-community.github.io.

The Lean Theorem Prover

Notable features:

- based on a powerful dependent type theory
- written in C++, with multi-core support
- small trusted kernel with independent type checkers
- elegant syntax and a powerful elaborator
- well-integrated type class inference
- flexible means of writing declarative proofs and tactic-style proofs
- server support for editors, with proof-checking and live information

The Lean Theorem Prover

- editor modes for Emacs and VSCode
- a javascript version runs in a browser
- a fast bytecode interpreter for evaluating computable definitions
- a powerful framework for metaprogramming via a monadic interface to Lean internals
- simplifier with conditional rewriting, arithmetic simplification
- good online documentation and tutorials
- enthusiastic, talented people involved

The Lean theorem prover

A number of mathematicians have begun using Lean, including:

- Reid Barton (algebraic topology)
- Kevin Buzzard (algebraic number theory)
- Bryan Gin-ge Chen (physics, mathematical physics)
- Johan Commelin (algebraic geometry and algebraic number theory)
- Sander Dahmen (number theory)
- Sébastien Gouëzel (dynamical systems and ergodic theory)
- Yury Kudryashov (dynamical systems)
- Patrick Massot (differential topology and geometry)
- Scott Morrison (higher category theory and topological quantum field theories)
- Neil Strickland (stable homotopy theory)

The Lean theorem prover

- Hundreds of messages are posted every day on the Lean chat forum on Zulip.
- Buzzard has been training very talented undergraduate students, like Chris Hughes, Kenny Lau, Amelia Livingston, and Jean Lo.
- Lean's library, *mathlib*, is growing quickly.
- Dahmen, Hölzl, and Lewis have formalized a proof of the Ellenberg-Gijswijt theorem (*Annals of Mathematics* 2017)
- Buzzard, Commelin, and Massot have formalized Peter Scholze's notion of a *perfectoid space* (and published a nice paper about it)

The Lean theorem prover

Why has Lean drawn such an audience?

- It's a nice system.
- The language (dependent type theory) is good for algebraic constructions.
- There is good introductory documentation.
- Early on, experts (like Mario Carneiro, Johannes Hölzl, and Robert Lewis) answered questions and provided support.
- The community is energetic, welcoming, and fun.

Social aspects are important.

Table of contents

Outline:

- Interactive theorem proving
- Why mathematicians should care
- Why logicians should care
- The *Lean* theorem prover

Demos

That is all I wanted to tell you about.

Some things I can show you if there is time:

- Lean community web pages
- Zulip chat group
- Lean web editor
- Theorem Proving in Lean
- Mathematics in Lean
- Lean perfectoid spaces
- The sphere eversion project

I can also give you a short demo.