Interactive theorem proving for the working logician

Jeremy Avigad

Department of Philosophy and
Department of Mathematical Sciences
Carnegie Mellon University

Prague Logic Seminar

December 2020

Prologue

The goals of this talk:

- To survey a new technology from computer science, *interactive theorem proving*.
- To explore possible topics of interest to mathematical logicians.

Prologue

Logic is fundamental to many parts of computer science:

- AI and automated reasoning
- Databases and knowledge representation
- Programming language semantics
- Formal methods in hardware and software design

It is also marginalized in mathematics.

Prologue

Looking for refuge in computer science poses constraints:

- Goals tend to be practical.
- Success = number of users (and/or citations).
- Methodology is often empirical.
- What's useful is often pretty boring.
- What's theoretically interesting and attractive is often not very useful.

An important question: is there a role for mathematical logic in computer science?

Formal methods in computer science

Formal methods 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.

Formal methods in computer science

Formal methods are used to

- say things,
- find things,
- and check things.

This is important in mathematics as well.

There is no sharp line between industrial and mathematical verification:

- Designs and specifications are expressed in mathematical terms.
- Claims rely on background mathematical knowledge.

Formal methods in computer science

- CompCert has verified the correctness of a C compiler.
- The seL4 microkernel has been verified.
- Amazon Web Services has a (really good) formal methods group.
- Facebook uses formal methods to find concurrency problems.
- Apple uses formal verification in Munich.
- Intel and AMD have used ITP to verify processors.
- Microsoft uses formal tools to verify programs and drivers.
- Airbus used formal methods to verify avionics software.
- Aesthetic Integration uses formal methods to show that trading software complies with regulations.

Formal methods in mathematics

I will focus on mathematics.

Four important domains of application:

- verified proof
- verified computation
- formal search
- digital infrastructure

I will discuss the first today.

Coordinates:

Logic in computer science
$\rightsquigarrow$ Formal methods
$\rightsquigarrow$ Interactive theorem proving
$\rightsquigarrow$ Formally verified mathematics

Table of contents

Outline:

- Interactive theorem proving
- Why mathematicians should care
- Why logicians should care
- The *Lean* theorem prover
- Openings for theory:
  - Formal foundations
  - Automation (decision procedures, search procedures)
  - Proof formats and certificates

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)

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

In the long run, formal methods can provide:

- verified proof
- verified computation
- formal search methods, to support discovery
- 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

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
- combining heterogeneous methods
- using external automation
- using computer algebra systems
- verifying numeric computation
- sharing data between proof systems
- managing large databases of knowledge
- understanding what machine learning can (and cannot) do

We need to pay attention to the mathematical details.

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
- small trusted kernel with independent type checkers
- good online documentation and tutorials
- nice syntax
- VS code editing mode with bells and whistles
- growing library
- favored by mathematicians
- lively community on Zulip
- a powerful framework for metaprogramming
- 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)
- It has been getting good press (Quanta, Notices).

Table of contents

Outline:

- Interactive theorem proving
- Why mathematicians should care
- Why logicians should care
- The *Lean* theorem prover
- Openings for theory:
  - Formal foundations
  - Automation (decision procedures, search procedures)
  - Proof formats and certificates

Formal foundations

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.

Formal foundations

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).

Formal foundations

Consider:

```
example (x : ℝ) :
  (2*x + 1)^3 = 8*x^3 + 12*x^2 + 6*x + 1 :=
by ring
```

Work is needed to infer the meaning of the symbols, interpret numerals, identify $\mathbb{R}$ as an instance of a ring.

It's also an instance of

- an additive commutative semigroup
- a multiplicative monoid
- a vector space over the reals
- a metric space
- a measure space

and lots of other things.

Formal foundations

Think of all the things that are needed to make sense of the binomial theorem:

```
theorem add_pow [comm_semiring α]
    (x y : α) (n : ℕ) :
  (x + y) ^ n =
    ∑ m in range (n + 1),
      x ^ m * y ^ (n - m) * choose n m
```

And this is pretty elementary.

Formal foundations

```
/- 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
```

Formal foundations

```
/- 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 _) :=
...
```

Formal foundations

```
/- 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
```

Automation

Automated mathematical reasoning is a new frontier.

Domain-general methods:

- Propositional theorem proving
- Equational reasoning
- First-order theorem proving
- Higher-order theorem proving

Domain-specific methods:

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

Combination methods are very important.

Automation

There is a gap between theory and practice.

- A principled search procedure or decision procedure that always times out is worthless.
- A heuristic hack that works in practice is great.

Nonetheless, theory guides the practice.

- The idealizations tell us what is possible, and where the challenges lie.
- If we start with a complete procedure, we can then seek optimizations.
- When a method fails, we want to know *why*.

Automation

Let's distinguish between two clusters of automated methods.

Small scale:

- used to perform small or domain-specific tasks.
- deterministic, or at least predictable
- examples: linear arithmetic, tactics for ring calculations, a `contradiction` tactic, a `continuity` tactic

Large scale:

- domain general
- often involves open-ended search using large parts of the library.
- examples: Sledgehammers, tableaux, resolution, SMT

A term rewriter / simplifier sits between the two.

Automation

Of the systems I have used (chiefly Isabelle, Coq, Lean), Isabelle has the best automation, by far.

I was surprised the Lean has become so popular, despite a lack of automation, especially at first.

A saving grace: Isabelle has a good *metaprogramming* language, making it easy to add new tactics.

Automation

A lot of the pain of formalization comes from searching the library for basic facts.

A *Sledgehammer* tool uses a heuristic relevance filter and external provers (typically resolution provers or SMT solvers), to dispell proof obligations.

Isabelle has a very good one.

Experts generally learn the library well and do things by hand.

This is a current target for machine learning.

Automation

In 2019 I did a little study, *Automated reasoning for the working mathematician* (talk, github).

Messages to the ATP community:

- Be wary of benchmarks.
- Algebraic and structural reasoning is important.
- It would be great if we could combine strengths of Sledgehammer and other tools.
- Second-order reasoning is unavoidable (in restricted, focused forms).
- Arithmetic is important.
- Combination methods are important.
- Users need feedback.
- It would help to give users and library designers more control.
- Certificates are nice.

Proof formats and certificates

In automated reasoning, calculi facilitate search.

First-order methods:

- resolution
- tableaux

Combination methods (SMT)

Propositional methods:

- unit propagation
- redundant clauses
- symmetry-breaking rules

Proof formats and certificates

In verification, we want to:

- store proofs for later checking or independent checking
- exchange proofs (or certificates) for proof (re)construction

One can optimize for different parameters:

- size of proof
- time / space needed to check the proof
- simplicity / complexity of the checker
- difficulty of manufacturing the proof

Proof formats and certificates

At one extreme, there is *Metamath*:

- Like proving in assembly language.
- The library has over 23,000 proofs.
- Checkers can check the library, from ASCII source, in a few seconds.

In contrast, for dependent type theory:

- One has to *elaborate* source code to obtain much more verbose expressions.
- The kernel checker has to normalize terms.

Proof formats and certificates

Mario Carneiro is working on verifying a checker for a variant of Metamath, MM0, down to x86 code.

Start with:

- A specification of Peano arithmetic with schematic formula variables.
- A specification of the semantics subset of the x86 processor.
- A specification of the specification language itself.

Goal: obtain a sequence of bytes (ELF format), and a formal theorem:

> *If you put this in memory, followed by an input string, and run it, then if it terminates with success, the input is a correct MM0 theorem.*

He will then run it on the theorem itself.

Proof formats and certificates

In competitions, propositional SAT solvers are required to output DRAT proofs.

First-order provers and SMT solvers aren't:

- there are no standard proof formats
- for performance, automated tools don't produce detailed proofs
- they can often be asked to output sketches
- other automated reasoners can check the sketches
- interactive theorem provers can try to use them

Isabelle's *Sledgehammer* usually ignores the proof sketch, and simply redoes the search with an internal tool, using only the relevant lemmas.

Proof formats and certificates

Seul Baek is working on a standard format for first-order proofs, TESC (Theory Extensible Sequent Calculus).

- The format is very explicit.
- Proof checkers are fast.
- He has verified a checker in Agda.

The idea is to compile resolution prover output down to that format.

He has developed a tool that can reconstruct proofs from sketches in Vampire with over 98% success.

Proof formats and certificates

Theoretical questions:

- What proof systems are good for search?
- What proof systems admit fast checking?
- What is the complexity of checking a proof?
- What is the speed improvement when producing a more explicit format?
- What is the space increase in producing a more explicit format?
- What are the complexity requirements to produce the more explicit format?
- What is the minimal amount of information that a prover needs to emit to facilitate efficient proof reconstruction?

Proof formats and certificates

In the end, we care about systems that work well on the kinds of problems that come up in practice.

So tests, benchmarks, and usability will determine success.

But theory can help us make better sense of what we are doing.

Table of contents

Outline:

- Interactive theorem proving
- Why mathematicians should care
- Why logicians should care
- The *Lean* theorem prover
- Openings for theory:
  - Formal foundations
  - Automation (decision procedures, search procedures)
  - Proof formats and certificates

Conclusions

There are interesting things going on computer science, and in formal methods and interactive theorem proving in particular.

Theoreticians and mathematical logicians should take a look.