Interactive Theorem Proving, Automated Reasoning, and Mathematical Computation

Jeremy Avigad

Department of Philosophy and
Department of Mathematical Sciences
Carnegie Mellon University

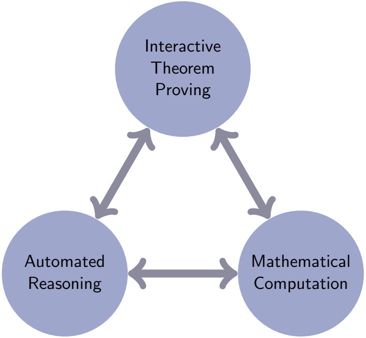
(including joint work with Grant Passmore)

December, 2012

Outline

Topics:

- Interactive theorem proving
- Automated reasoning and mathematical computation
- A logic library for Python and Sage
- Verifying the results



Certifying mathematical results

Question: how do we come to know that our mathematical claims are correct?

One answer: prove them correct.

- Use trusted axioms.
- Use valid inferences.
- Make the proof public so that others can check it.

Twentieth century logic: proofs can be verified mechanically, in principle.

Contemporary computer science: proofs can be verified mechanically, in practice.

Axiomatic foundations

The development of mathematics in the direction of greater exactness has — as is well known — led to large tracts of it becoming formalized, so that proofs can be carried out according to a few mechanical rules. The most comprehensive formal systems yet set up are, on the one hand, the system of Principia Mathematica (PM) and, on the other, the axiom system for set theory of Zermelo-Fraenkel (later extended by J. v. Neumann). These two systems are so extensive that all methods of proof used in mathematics today have been formalized in them, i.e. reduced to a few axioms and rules of inference. It may therefore be surmised that these axioms and rules of inference are also sufficient to decide all mathematical questions which can in any way at all be expressed formally in the systems concerned. It is shown below that this is not the case. . .

Kurt Gödel, "On formally undecidable propositions of *Principia Mathematica* and related systems I," 1930.

Axiomatic foundations

There are people working hard on the project of actually formalizing parts of mathematics by computer, with actually formally correct formal deductions. I think this is a very big but very worthwhile project, and I am confident we will learn a lot from it…

However, we should recognize that the humanly understandable and humanly checkable proofs that we actually do are what is most important to us, and that they are quite different from formal proofs.

William P. Thurston, “On proof and progress in mathematics,” *Bulletin of the AMS* 1994

Axiomatic foundations

How do we know that a proof is correct? By checking it, line by line. A computer might even be programmed to check it. . . .

Still, there is a nagging worry about this belief in mathematical certitude. . . .

. . . many great and important theorems don't actually have proofs. They have sketches of proofs, outlines of arguments, hints and intuitions that were obvious to the author (at least, at the time of writing) and that, hopefully, are understood and believed by some part of the mathematical community.

Melvyn B. Nathanson, "Deseparately seeking mathematical truth," *AMS Notices* 2008

Interactive theorem proving

“Interactive theorem proving” is one important approach to verifying the correctness of a mathematical proof.

Working with a “proof assistant,” the user conveys enough information to the system to confirm that there is a formal axiomatic proof.

In fact, most proof systems actually construct a formal proof object, a complex piece of data that can be verified independently.

Interactive theorem proving

Some important systems:

- Mizar (set theory)
- HOL4 (higher-order logic)
- Isabelle (higher-order logic)
- Coq (constructive dependent type theory)
- HOL light (higher-order logic)
- ACL2 ($\sim$primitive recursive arithmetic)

Interactive theorem proving

Think of an ordinary proof as a high-level description of, or recipe for constructing, a fully detailed axiomatic proof.

In formal verification, it is common to refer to proofs as “code.”

```
lemma prime_factor_nat: "n ~= (1::nat) ==>
    EX p. prime p & p dvd n"
  apply (induct n rule: nat_less_induct)
  apply (case_tac "n = 0")
  using two_is_prime_nat apply blast
  apply (case_tac "prime n")
  apply blast
  apply (subgoal_tac "n > 1")
  apply (frule (1) not_prime_eq_prod_nat)
  apply (auto intro: dvd_mult dvd_mult2)
done
```

Interactive theorem proving

```
proof (induct n rule: less_induct_nat)
  fix n :: nat
  assume "n ~= 1" and
    ih: "ALL m < n. m ~= 1 --> (EX p. prime p & p dvd m)"
  then show "EX p. prime p & p dvd n"
  proof -
  { assume "n = 0"
    moreover note two_is_prime_nat
    ultimately have ?thesis by auto }
  moreover
  { assume "prime n" then have ?thesis by auto }
  moreover
  { assume "n ~= 0" and "~prime n"
    with ‘n ~= 1‘ have "n > 1" by auto
    with ‘~prime n‘ and not_prime_eq_prod_nat obtain m k where
      "n = m * k" and "1 < m" and "m < n" by blast
    with ih obtain p where "prime p" and "p dvd m" by blast
    with ‘n = m * k‘ have ?thesis by auto }
  ultimately show ?thesis by blast
```

Interactive theorem proving

Some theorems formalized to date: the prime number theorem, the four-color theorem, the Jordan curve theorem, Gödel's first incompleteness theorem, Dirichlet's theorem, Cartan fixed-point theorems

There are good libraries for elementary number theory, real and complex analysis, measure-theoretic probability, linear algebra, Galois theory, . . .

See the *Journal of Automated Reasoning*, *Journal of Formalised Reasoning*, *Journal of Formalized Mathematics*, the *Interactive Theorem Proving* conference, and Freek Wiedijk's list of 100 theorems.

Interactive theorem proving

Georges Gonthier headed a project to verify the Feit-Thompson theorem, with a group of researchers.

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

The project was completed on September 20, with roughly

- 170,000 lines of code,
- 4,200 definitions, and
- 15,000 theorems.

Interactive theorem proving

Thomas Hales' *Flyspeck* project is nearing completion (HOL light, Isabelle).

- Three essential uses of computation: enumerating tame hypermaps, proving nonlinear inequalities, showing infeasibility of linear programs.
- The formalization led to even stronger results.

Vladimir Voevodsky has launched a project to develop "univalent foundations" for algebraic topology (Coq).

- Constructive dependent type theory has natural homotopy-theoretic interpretations.
- Rules for identity types characterize homotopy theories abstractly.
- One can consistently add an axiom to the effect that "isomorphic structures are identical."

Interactive theorem proving

Interactive theorem proving is not “ready for prime time.”

- There is a steep learning curve.
- Verification can be time consuming and painful.

Short term wins:

- verifying computation
- fiddly hand calculations

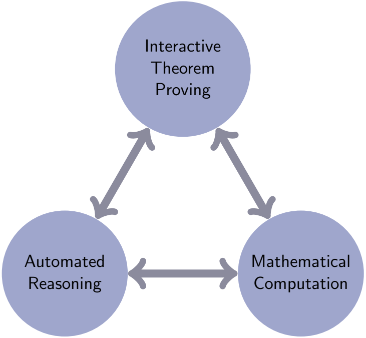
Long term:

- Need better libraries (and means to translate between them; cf. the OpenTheory project).
- Need better automated proof methods.
- Need better ways to incorporate and verify computations.

Outline

Topics:

- Interactive theorem proving
- Automated reasoning and mathematical computation
- A logic library for Python and Sage
- Verifying the results



Automated reasoning vs. mathematical computation

One distinction: the first has to do with logic, the second mathematics.

But (Boole, 1854) we can "calculate" with propositions:

$$(p \land q) \lor (r \land \neg q) = (p \lor r) \land (p \lor \neg q) \land (q \lor r)$$

just as we calculate with magnitudes:

$$(x + y)(z + y^{-1}) = xz + xy^{-1} + yz + 1$$

Automated reasoning vs. mathematical computation

The real distinction: search vs. calculation.

One can search for all kinds of mathematical objects, e.g. proofs, solutions to Diophantine equations, combinatorial objects, and so on.

The problem: infinite domains, combinatorial explosion.

Ideas:

- Exploit symmetry and choose representations carefully, to avoid duplication.
- Use heuristics.
- Use efficient data structures.

Automated reasoning vs. mathematical computation

Domain-general methods:

- Propositional theorem proving ("CDCL")
- First-order theorem proving
- Higher-order theorem proving
- Equality reasoning
- "Combination" methods ("SMT")

Domain-specific methods:

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

Automated reasoning vs. mathematical computation

In practice, there is no sharp line between “search” and “computation.”

For example, the “theory of real closed fields” is decidable (Tarski, 1948).

Verifying hybrid systems:

- Gao, Avigad, Clarke: combine interval methods with SMT search
- Platzer, Paulson, Passmore: use symbolic methods, with heuristics

Even when problems are decidable in principle, may still need search methods.

Automated reasoning vs. mathematical computation

How to bring the two domains together?

Bring computation into automated reasoners:

- Combination methods (SMT provers)
- Integrate computation into resolution provers (MetiTarski)

Bring proof search to computer algebra systems

- Theorema (Buchberger et al.)
- Analytica (Clarke et al.)

A logic library for Python and Sage

Grant Passmore, Leo de Moura, and I are working on a library:

- Have Python classes for languages, terms, formulas, models, goals, proofs
- Include interfaces to automated reasoners
  - SMT provers: Z3, CVC3, SMT format
  - Resolution provers: Vampire, E, Spass, Prover 9, TPTP format
  - Computer algebra: Sage, Mathematica, Metitarski
  - Model finders: Mace, Kodkod
  - Interactive theorem provers: Isabelle, HOL-light, Coq
- Develop a reasoning toolbox
  - Users can explore hypotheses and conjectures
  - Users can write special-purpose reasoning procedures

A logic library for Python and Sage

Examples:

- Plot polynomials in Sage, ask Z3 about the roots.
- Ask Prover 9 to verify that any group of exponent 2 is abelian.
- Have Mace find a nonabelian group of exponent 3.
- Ask Z3 to find kissing configurations.
- Interactively work through a proof, look for counterexamples.

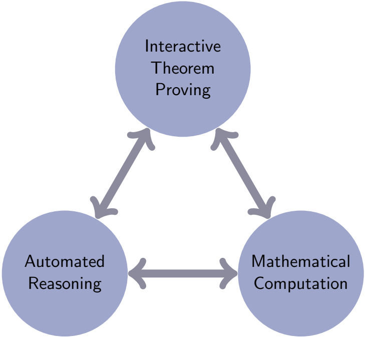
Related projects:

- A proof checker for Euclidean geometry.
- An open-source version of *Tarski's World*.
- A heuristic procedure for proving inequalities.

Outline

Topics:

- Interactive theorem proving
- Automated reasoning and mathematical computation
- A logic library for Python and Sage
- Verifying the results



Verifying the results

Challenge: bring automated reasoners and mathematical computation the kind of assurances one gets from interactive theorem provers.

One solution: verify the automated reasoners and systems of computation, or have these systems verify their results.

But this is very hard, and pulls in the wrong direction.

Verifying the results

Interactive theorem provers:

- maintain a high standard of correctness
- emphasis on rigor and precision

Automated reasoning systems:

- deal with vast search spaces
- emphasis on speed, efficiency, and heuristics

Computer algebra systems:

- abundance of mathematical concepts and structures
- emphasis on ease of use and flexibility

Verifying the results

We need more subtle ways of verifying correctness:

- Reconstruct proofs after the fact (Sledgehammer, Isabelle and Z3).
- Use certificates (semidefinite programming, algebraic computations).
- Verify facts and procedures selectively.

Goals:

- Bring more ease and flexibility to interactive theorem proving.
- Make automated reasoning and mathematical computation more trustworthy.

Conclusions

- Interactive theorem proving, automated reasoning, and mathematical computation provide important ways of extending mathematical knowledge.
- Their strengths are complementary.
- A flexible logic library will help integrate automated reasoning and mathematical computation, and support experimentation and exploration.
- Formal methods help to ensure correctness.
- Interactive theorem proving meets a very high standard.
- The central challenge: verify results, while maintaining flexibility and efficiency.