Automated Reasoning for Mathematics

Jeremy Avigad

Department of Philosophy
Department of Mathematical Sciences
Hoskinson Center for Formal Mathematics

Carnegie Mellon University

July 3, 2024

Origins of automated reasoning

Some early contributions:

- Martin Davis implemented Presburger's decision procedure at the IAS in 1954.
- Allen Newell, Herbert Simon, and Cliff Shaw introduced the *Logic Theorist* in 1956.
- Henry Gelernter, J. R. Hansen, and Donald Loveland published an article on the *Geometry Machine* in 1960.
- Hao Wang implemented good provers for propositional and predicate logic in 1958.
- Davis and Hilary Putnam introduced the propositional resolution rule in 1960.
- John Alan Robinson introduced a unification algorithm in 1965.

Origins of automated reasoning

The first incompleteness theorem applies to any consistent, computably axiomatized theory containing basic arithmetic.

Gödel 1931:

The theorem "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 ..."

Origins of automated reasoning

He gave a tentative definition of computability at the IAS in 1932.

After Turing's 1936 paper, he added this footnote:

"In consequence of later advances, in particular of the fact that, due to A. M. Turing's work, a precise and unquestionably adequate definition of the general concept of a formal system can now be given, the existence of undecidable arithmetical propositions and the non-demonstrability of the consistency of a system in the same system can now be proved rigorously for *every* consistent formal system containing a certain amount of finitary number theory."

Origins of automated reasoning

More connections to automated reasoning:

- Turing presented his definition of computability with a negative solution to the *Entscheidungsproblem*.
- He also noted that incompleteness follows from undecidability, because one can computably search for proofs.
- Church gave another proof of the undecidability of arithmetic in 1936.
- Kleene was also keenly interested in logic and foundations.

Origins of automated reasoning

The origins of decision procedures are even earlier:

- In 1915, Löwenheim proved the decidability of monadic first-order logic.
- Presburger presented his decision procedure for arithmetic in 1929.
- Tarski had a decision procedure for real closed fields in 1930.

Taking stock

Where do we stand now?

- Automated reasoning has had almost no impact on mathematics.
- Few mathematicians have ever touched an automated reasoning tool.
- Automated reasoning has contributed to very few mathematical discoveries, even minor ones.

This is surprising!

Compare to numerical methods in science:

- Science is not just about calculation.
- But computers have nonetheless had a dramatic impact.

Taking stock

Mitigating factors:

- Automated reasoning has lots of other applications.
- There have been *some* notable successes.
- Mathematics is hard.

Goal of this talk:

- Review some of the successes.
- Understand the challenges.
- Think about the future.
- Convey optimism.

A personal history

My timeline:

- 2002: formalized quadratic reciprocity in Isabelle
- 2003–2004: formalized the prime number theorem (with Kevin Donnelly, David Gray, Paul Raff)
- 2005–2009: contributed to the Isabelle library
- 2009–2010: worked on the odd order theorem with Gonthier and the Mathematical Components team (Coq / SSReflect)
- 2011–2012: formalized the central limit theorem with Luke Serafin and Johannes Hölzl
- 2012–2018: did some work in homotopy type theory in Coq and then in Lean
- 2013–present: worked on Lean's libraries, documentation, automation

A personal history

In 2002, Isabelle's library was not very extensive, and there were lots of gaps.

But the automation was surprisingly mature:

- a conditional term rewriter (`simp`)
- a procedure for linear (real and integer) arithmetic (`arith`)
- a tableau prover (`blast`)
- a general reasoner (`auto`)

I used them a lot.

A personal history

The last file in the proof of the prime number theorem, PNT.thy, has about 4,000 lines.

Usage:

- `simp`: 390 times
- `auto`: 51 times
- `force`: 277 times
- `clarify`: 69 times
- `arith`: 246 times

I have yet to have a better experience with automation.

A personal history

I took a break after the PNT, and then contributed to Isabelle's libraries.

In 2009–2010, I had an opportunity to spend a sabbatical year with the Mathematical Components group in France, thanks to Georges Gonthier.

The formalization of the odd order theorem used *computational reflection* and canonical structures, but otherwise almost no automation.

A personal history

After returning from France:

- I was tired of formalization and interested in automation.
- Luke Serafin convinced me to join him in formalizing the central limit theorem.
- Chris Kapulkin talked me into formalizing limit constructions in HoTT.
- Leonardo de Moura convinced me to work on his new project, *Lean*.

A personal history

Leo convinced me automation for mathematics needs a secure, expressive foundation:

- to ensure the automation is reliable, and
- to have a specification of what the results *mean*.

The early web pages said that the aim of the project was

> *to bridge the gap between interactive and automated theorem proving, by situating automated tools and methods in a framework that supports user interaction and the construction of fully specified axiomatic proofs.*

A personal history

Lean history:

- Lean 0.1 was short-lived.
- Lean 2 (2014) introduced the tactic framework and inductive types.
- Lean 3 (2016) used Lean itself as a metaprogramming language.
- Lean 4 (2022) is (mostly) written in Lean 4, and has rich mechanisms for handling and extending syntax.

There have been recent developments in automation and AI for Lean.

Domain-general reasoning for verification

In 2019, I gave a talk at FroCoS and TABLEAUX, "Automated Reasoning for the Working Mathematician."

You can find online:

- the talk
- a repository
- notes

They include:

- surveys (hearsay)
- experiments (anecdotes)
- reflections (speculation)

Domain-general reasoning for verification

One finding: most of the best formalizers I knew used very little automation.

My explanation: even with good automation, we still have to do a lot by hand.

Power users learn the library by heart anyhow, and become very efficient at doing things manually.

Then they don't need automation.

Domain-general reasoning for verification

By “domain-general,” I mean to exclude simplifiers and term rewriters.

Equality is pretty general, but term rewriters do focused and deterministic things.

I have in mind, instead, things that require *search*.

Domain-general reasoning for verification

The gold standard is Isabelle's *Sledgehammer* (Larry Paulson, Jia Meng, Jasmin Blanchette, and many others).

We are close to having one for Lean:

- Yicheng Qian has written *LeanAuto*, for exporting problems to external tools.
- Joshua Clune, Qian, and Alex Bentkamp have written *Duper*, for proof reconstruction.
- Clune is working on a relevance filter (building on one by Piotrowski, Fernández-Mir, and Ayers).
- Clune and Haniel Barbosa, as well as Abdalrahman Mohammad and the cvc5 team, are working on proof reconstruction for cvc5.

Also, Limperg and From developed *Aesop*, inspired by Isabelle's *auto*.

Domain-general reasoning for verification

The turn of the twentieth century inaugurated structural reasoning in mathematics.

The "algebraic hierarchy" in Mathlib is huge: the library has 1,520 classes and 26,143 (direct) instances.

Whenever you write $x + y$ or say "by the commutativity of addition," Lean has the (sometimes Herculean) task of inferring the relevant structure.

This poses challenges for automation.

With *Duper*, we address these with a monomorphization procedure called *Lean-Auto* by Yicheng Qian.

Domain-specific reasoning for verification

Domain-general tools are good at reasoning about everything but not so good about reasoning about any particular thing.

At the other end of the spectrum, domain-specific tools are good for reasoning about arithmetic, algebraic identities, linear and nonlinear inequalities, etc.

Domain-specific reasoning for verification

Mathematicians know their use cases better than anyone else.

Lean's *metaprogramming facilities* make it easy to write bespoke tactics.

With help from Mario Carneiro, Heather Macbeth developed tactics, `gcongr` and `positivity`, for reasoning about inequalities.

She was immediately able to shorten hundreds of calculations in Mathlib substantially.

Domain-specific reasoning for verification

From this:

```
calc ‖wp - wq‖ * ‖wp - wq‖
  _ = 2 * (‖a‖ * ‖a‖ + ‖b‖ * ‖b‖) - 4 * ‖u - half · (wq + wp)‖ *
        ‖u - half · (wq + wp)‖ := by rw [← this]; simp
  _ ≤ 2 * (‖a‖ * ‖a‖ + ‖b‖ * ‖b‖) - 4 * δ * δ :=
      (sub_le_sub_left eq₁ _)
  _ ≤ 2 * ((δ + div) * (δ + div) + (δ + div) * (δ + div)) -
        4 * δ * δ :=
      (sub_le_sub_right (mul_le_mul_of_nonneg_left
        (add_le_add eq₂ eq₂') (by norm_num)) _)
  _ = 8 * δ * div + 4 * div * div := by ring
exact
  add_nonneg (mul_nonneg (mul_nonneg (by norm_num) zero_le_δ)
      (le_of_lt Nat.one_div_pos_of_nat))
    (mul_nonneg (mul_nonneg (by norm_num)
      Nat.one_div_pos_of_nat.le) Nat.one_div_pos_of_nat.le)
```

Domain-specific reasoning for verification

To this:

```
calc ‖wp - wq‖ * ‖wp - wq‖
  _ = 2 * (‖a‖ * ‖a‖ + ‖b‖ * ‖b‖) - 4 * ‖u - half · (wq + wp)‖ *
        ‖u - half · (wq + wp)‖ := by simp [← this]
  _ ≤ 2 * (‖a‖ * ‖a‖ + ‖b‖ * ‖b‖) - 4 * δ * δ := by gcongr
  _ ≤ 2 * ((δ + div) * (δ + div) + (δ + div) * (δ + div)) -
        4 * δ * δ := by gcongr
  _ = 8 * δ * div + 4 * div * div := by ring
positivity
```

Domain-specific reasoning for verification

Most commonly used tactics in Mathlib:

- `apply`, `exact`, `refine`, etc. (60K instances)
- `rw` (52K)
- `simp`, `simpa`, `dsimp`, and `simp_rw` (60K)
- `obtain`, `rintro`, `rcases`, `cases`, etc. (25K)
- `induction` and variations (5K).

(Data thanks to Adam Topaz.)

Domain-specific reasoning for verification

More specialized automation:

- `linarith` (1,100 times)
- `split_ifs` (1,000)
- `ring` (1,000)
- `filter_upwards` (900)
- `norm_num` (800)
- `aesop_cat` (800)
- `positivity` (600)
- `norm_cast` (500)
- `gcongr` (500).

Giving users feedback

Wojciech Nawrocki, Edward Ayers, and Gabriel Ebner have developed widgets for Lean 4.

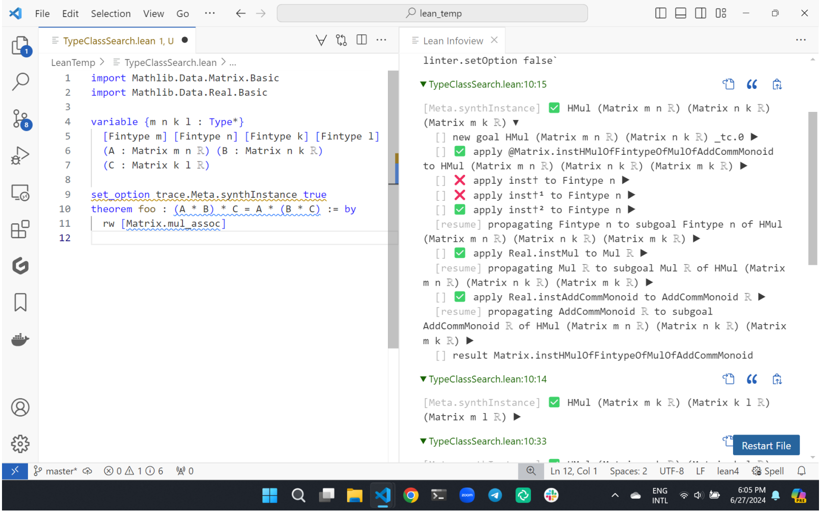
Lean automation, like type class inference, can report back structured traces of their search (successful or not).

Similarly, automation can return structured error messages.

When automation fails, it's important to be able to diagnose problems and fix them.

Search engines and relevance filters are also really important.

Giving users feedback



Automation for the discovery of new theorems

Early successes:

- McCune's resolution of the Robbins conjecture in 1996.
- Results on loops and quasigroups.

But this has nothing to do with mainstream mathematics.

Mathematicians don't use first-order reasoning from the axioms of a structure.

They prove structure classification theorems, structure theorems, representation theorems.

They also simplify and modularize past theorems and look for new applications.

Automation for the discovery of new theorems

Results from SAT solvers fare better.

In 1912, Schur proved that for any finite coloring of the positive integers, there is a monochromatic solution to $x + y = z$.

Let $S(k)$ be the largest value such that there is a k-coloring of $\{1, 2, \ldots, S(k)\}$ with no monochromatic solution.

Easy: $S(1) = 1$, $S(2) = 4$, $S(3) = 13$.

In 1965, Golomb and Baumert showed $S(4) = 44$.

In 2017, Heule used a SAT solver to establish $S(5) = 160$.

Automation for the Discovery of New Theorems

The *Happy Ending Theorem* (E̋rdos, Szekeres, and Klein) says that for every positive *n*, any sufficiently large finite set of points in general position contains a convex *n*-gon.

One can also ask about *empty* *n*-gons.

There are infinite sets of points with no empty convex 7-gon.

In 2024, Heule and Scheucher showed that 30 points guarantee the existence of an empty hexagon, but not 29.

Automation for the Discovery of New Theorems

Many mathematicians will dismiss these as "finite problems," i.e. recreational mathematics or computer science.

Questions:

- Will mathematicians and computer scientists find ways to reduce "real" (infinitary) problems to SAT?
- Will attitudes toward "finite problems" change?
- What can be done with other technologies, like model finders?

Interesting things happen when mathematicians begin to use new technologies.

Machine Learning and Symbolic AI

Machine learning opens up new frontiers:

- copilots and automated support for interactive theorem proving
- discovery of new proofs (formal or informal)
- discovery of new results.

Neuro-symbolic approaches, which combine the strengths of machine learning with symbolic AI, are promising.

For example, Jiang et al., "Draft, sketch, and prove" uses an LLM to write the skeleton of an Isabelle proof and sledgehammer to fill in the details.

Similarly, Trinh and Luong's *AlphaGeometry* uses neural methods to learn from explorations with symbolic engines.

Recap

I began with the observation that automated reasoning currently plays almost no role in mathematics.

I discussed four domains where we are making progress:

- domain general automation for verification
- domain specific automation for verification
- automation for discovery of new theorems
- syntheses machine learning and symbolic AI.

I believe we'll see exciting developments over the next decade.

Optimism

In 2017, very few mathematicians were using proof assistants.

The situation has changed dramatically since then.

- There have been celebrated successes:
  - the Liquid Tensor Experiment
  - the Sphere Eversion Project
  - Mehta's formalization of work by Campos, Griffiths, Morris, and Sahasrabudhe
  - the formalization of the Gowers, Green, Manners, and Tao proof of the Polynomial Freiman-Ruzsa conjecture.
- There have been articles in Quanta, Nature, The New York Times, Scientific American, ...
- The *Bulletin of the American Mathematical Society* just ran two consecutive special issues on new technologies for mathematics.

Optimism

With ITP, we *used* to think the most effective strategy was to show mathematicians how to formalize mathematics.

We were wrong.

Optimism

Lean managed to meet the mathematicians where they were.

- We wrote documentation and tutorials with mathematicians in mind.
- We interacted with early adopters.
- We answered questions.
- We built parts of the library that mathematicians needed.
- We wrote tactics that they needed.
- We found points of collaboration.

Mathematicians knew better than we did what to do with the technology.

We just had to listen and help them do what they wanted to do.

Optimism

The results were astounding.

They also had a snowball effect:

- Enthusiasm leads to successes.
- Successes generate enthusiasm.

The time is ripe for automated reasoning for mathematics:

- Formalization is a gateway to the use of automation.
- Mathematicians are warming to the use of automation and formal methods.
- There is general interest in neural and symbolic AI (tempered with caution).

Optimism

Communication with mathematicians can be difficult.

The two communities think and act differently:

- Computer science is a young science; mathematicians acknowledge Pythagoras, Euclid, and Archimedes.
- Computer scientists publish in conferences; mathematicians publish in journals.
- Computer scientists use citations to measure impact; mathematicians look to experts to assess importance and depth.

If you are a computer scientist, you might enjoy adopting a mathematical outlook every once in a while.