Mathematics, automation, theorem proving

Damiano Testa

University of Warwick

Colloquio, Università di Pisa

13 dicembre 2023

Mathematics, automation, theorem proving

Mechanical computations and mathematics have always been close.

Nevertheless, I usually find abstract arguments that avoid computations more appealing – and computers share this feeling.

Recently, there has been a surge of interest and development in formalization of mathematics.

My talk is an introduction to computer-verified mathematics:

- past successes,
- current projects and
- future challenges.

Past formalizations

Theorem	Year	Formalization
Prime Number Theorem	1896	2004
Jordan Curve Theorem	1893	2005
Four Color Theorem	1976	2005
Odd Order Theorem	1962-63	2012
Kepler Conjecture	1998	2014
Continuum Hypothesis	1963	2019
Perfectoid spaces	2012	2019
Poincaré-Bendixson Theorem	1892-1901	2020
Liquid Tensor Experiment	2019	2021-22
Sphere Eversion (movie)	1957	2022

Real-time formalizations

- Gardam's disproof of Kaplansky's Unit Conjecture (formalization by Gadgil);
- Bloom's proof of a conjecture of Erdős and Graham (formalization by Bloom-Mehta);
- Campos, Griffiths, Morris and Sahasrabudhe's recent breakthrough on exponential bounds in Ramsey theory (formalization by Mehta).

Current and future projects

- Fermat's Last Theorem for regular primes (Kummer's Theorem)[1];
- Class Field Theory;
- Shimura varieties;
- Fermat's Last Theorem (Wiles et al);
- Connectedness of the Mandelbrot set and of its complement;
- Polynomial Freiman–Ruzsa conjecture[1];
- and so on.

[1] Formalized on Dec 5th, 2023, both!

Big lists

An inspiration to formalization comes from "big lists" of theorems.

Currently, the only result in Freek Wiedijk's list of 100 theorems that has not yet been formalized is Fermat's Last Theorem.

Oliver Knill also produced an extensive list of theorems.

Big lists like the ones above are a great source of interesting formalization projects.

Undergraduate curriculum

Nowadays, most of the results that are usually taught in undergraduate programs are formalized.

Or, at least, they are close to formalized results!

This is something that has changed in the last few years.

Why formalize mathematics?

Mistakes

To find mistakes – not really!

At least, not only and certainly not as a motivation!

Every formalization project uncovers some mistake.

These mistakes are typically inconsequential:

- a missing $a \neq 0$ inequality;
- a forgotten assumption that is "expected" in the given context;
- applying a result that does not exactly work in the given situation, but can be tweaked to work.

I would be very surprised if a fatal flaw was found during the formalization of a well-known result.

Why formalize mathematics? – Proof development

- Complex or subtle proofs in an unfamiliar area
- Relying on theorems that you personally have not checked
- Generalize results, remove assumptions, weaken hypotheses
- Generalize and uniformize definitions

Mathematical knowledge grows very quickly and in several directions.

More and more results rely on so much background that it is becoming unreasonable to expect any single mathematician to be an expert at all the pre-requisites.

For instance, Scholze's challenge and the Liquid Tensor Experiment.

Why formalize mathematics? – Proof assistance

Formalization and proof assistants mitigate these issues.

"Importing" several formalized mathematical theories and using them in your project is certainly simpler and quicker than

- digesting the theories;
- figuring out what the common pitfalls are;
- learning how to comfortably overcome the "simple" mistakes.

Why formalize mathematics? – Documentation

- Systematic retrieval of digitized results
- Simpler cross-referencing
- No need for background sections
- Inspection of each definition/theorem to clarify all details

Why formalize mathematics? – Automation

- Automated proof inspection
- De-duplicate repetitive arguments
- Discover patterns and parallels across areas
- Automated proof generation

Artificial Intelligence and Machine Learning may (will!) play an increasingly bigger future role in solving mathematical questions.

Why formalize mathematics? – Really!

Personally, formalizing is

- a learning experience;
- highly stimulating;
- fun!

Formalizing a vast library of mathematics means really thinking about the "best" possible definitions.

I find the whole process incredibly rewarding.

How does it work

The initial goal is to feed "mathematics" to a computer:

- choose a set of axioms as foundation for mathematics;
- convince ourselves that this is consistent(!) and workable;
- produce a model of this theory in a computer.

Bonus points if the resulting model rejects quickly the syntactically correct set-theoretic non-sense

$$\left(\begin{array}{rcl} f : \mathbb{Q} & \to & \mathbb{R} \\ t & \mapsto & \frac{\cos t}{2+\sin(t^2)} \end{array} \right) \in \pi^2.$$

Models of mathematics

I learned mathematics using set-theory as foundations.

Seriously thinking about a different model changed deeply

- my *perception* of mathematics,
- my *understanding* of mathematics.

Of course, a new model also affects

- the results that we can *prove*,
- the results that we can *state*!

Foundations of mathematics

Different proof assistants use different foundations of mathematics.

Some are based on set theory.

Others (most?) on some form of Type theory.

The actual foundations used should be "barely visible" to a user.

There is a lot of potential for improvement on this front.

The natural numbers in Lean

My experience is mostly based on Lean.

Let's see the definition of the natural numbers in Lean.

```
inductive Nat
  | zero : Nat
  | succ : Nat → Nat
```

The underlying rules make the newly defined type `Nat` a "universal solution" of the implied categorical diagram.

If this is not helpful, another analogy is defining a group by generators and relations:

the freest possible object subject to the given constraints.

```
inductive Nat
  | zero : Nat
  | succ : Nat → Nat
```

In the case of `Nat`, this is a structure that contains `0` and a function

$$\mathrm{succ}\colon \mathrm{Nat} \longrightarrow \mathrm{Nat}.$$

The only ways of obtaining `Nat`s are

- starting with `0` and
- applying `succ` to a previously defined `Nat`.

Different numbers of `succ` applications result in different `Nat`s.

Nat in Lean

Here is an example of a proof in Lean.

```
example {n : ℕ} :
    Σ i in range (n + 1), (i : ℚ) = n * (n + 1) / 2 := by
  induction n with
    | zero =>
      simp
      done
    | succ n hn =>
      rw [sum_range_succ]
      rw [hn]
      field_simp
      ring
      done
```

Sums in Lean

As it happens, someone comes along and says:

"I just learned a cool fact! A polynomial with coefficients in $\mathbb{N}$ is monotone!"

First – what do they really means?

Surely they intended to say that viewing a polynomial with coefficients in $\mathbb{N}$ as a function $\mathbb{N} \to \mathbb{N}$, we obtain a monotone function.

Next...

Let's formalize this result!

Conclusion: formalizing mathematics helps with

Bookkeeping:	Proof assistance:
maintain rigour uniformize conventions universal repository referencing	tracking hypothesis and assumptions parallelize proofs retrieval of useful results

Proof automation:	
take over repetitive proofs suggest possible next steps generalize known results	. . . and more!

The data obtained from formalization is also the current first step to guide computers into self-discovery of mathematics.

Artificial Intelligence and Machine Learning may occupy a more promininent role in the future.

Thank you!

Questions?