Formalizing invisible mathematics

Andrej Bauer
University of Ljubljana

Machine assisted proofs
Institute for pure and applied mathematics
13 February 2023

Thank you for the invitation and the chance to visit this wonderful place.
A while ago I contacted organizers asking what I should talk about, and from their response I gathered that it would be useful to have some talks that acts as bridges between math and computer science. I think I am one of the people here who may do just that.

I would like to explain a bit what it takes to create software for machine-assisted proofs, apart from just implementing a piece of code that verifies formal proofs. I hope this will give useful insights to the mathematicians in the audience. So far the design of proof assistants has mostly been in the hands of computer scientists, but I am quite certain that the problems we are facing have a strong mathematical component that requires mathematical innovation.

Overview

- What is invisible mathematics?
- How do we formalize it?
- Challenges

When we actually try to formalize mathematics it at scale, we find out that there is an “invisible” part of mathematical activity, which is rarely talked about explicitly, and is typically considered of secondary importance. Nevertheless, it is absolutely crucial that we be able to formalize also these parts of mathematics, or else there can be no cooperation between humans and machines.

Let me show you what I am talking about, what has been accomplished so far, and what challenges remain. It is easiest to explain what I mean by means of examples.

Example #1: Elaboration

Our first example is elaboration – the process of making sense of incomplete and formally incorrect mathematical communication.

Mathematicians communicate efficiently by communicating incompletely and incorrectly. They expect other mathematicians to reconstruct missing information and to rectify minor mistakes. This is not something they do for fun, but because the alternative is unworkable.

Consider the following statement. Do you understand it? Of course. There may be some questions about what precisely is going on, but you could give a proof of the statement (and it's easy) – so you must understand something about it.

Let us elaborate the statement so that we get something closer to a formally complete one.

If f is linear then $f(2 \cdot x + y) = 2 \cdot f(x) + f(y)$.

If U and V are vector spaces and $f : U \to V$ is linear then, for all $x, y \in U$, $f(2 \cdot x + y) = 2 \cdot f(x) + f(y)$.

Presumably, f is a linear map between vector spaces, so we should make that explicit. Also, we should state explicitly that x and y range over the domain. Are we done? Of course not.

If K is a field, U and V are vector spaces over K, and $f : U \to V$ is linear then, for all $x, y \in U$, $f(2 \cdot x + y) = 2 \cdot f(x) + f(y)$.

A vector space is over a field, so we need to put that in.

If K is a field, U and V are vector spaces over K, and $f : |U| \to |V|$ is linear then, for all $x, y \in |U|$, $f(2 \cdot x + y) = 2 \cdot f(x) + f(y)$.

To be quite precise, a vector space U is a structure, one component of which is its carrier set |U|. The map f acts on the carriers, not the structures themselves, and x and y range over the carrier, too.

An alternative would be to say that f is a morphism from U to V in the category of vector spaces, but then we would have to write |f| for the underlying set-theoretic map: |f|(2 · x + y) = 2 · |f|(x) + |f|(y). That's hardly better.

If K is a field, U and V are vector spaces over K, and $f : |U| \to |V|$ is linear then, for all $x, y \in |U|$, $f(2_K \cdot_U x +_U y) = 2_K \cdot_V f(x) +_V f(y)$.

There is more. The 2, is it a natural number, an integer, an element of the field K?
Also, we really should subscript operations with the vector spaces in which they happen.

So this roughly is the amount of information that we expect our colleagues and students to reconstruct. A modern proof assistant can do it, too.

If f is linear then $f(2 \cdot x + y) = 2 \cdot f(x) + f(y)$.

Elaboration

If K is a field, U and V are vector spaces over K, and $f : |U| \to |V|$ is linear then, for all $x, y \in |U|$, $f(2_K \cdot_U x +_U y) = 2_K \cdot_V f(x) +_V f(y)$.

The process we just described is called **elaboration**.

It is part of *implicit* mathematical knowledge that is passed on by observation and imitation, and rarely talked about. The 20th century logic has mostly ignored elaboration, because it preoccupied itself with other problems, that were more relevant at the time.

Elaboration

Informal	Formal
“A field K to be specified later.”	existential variable
“Guess that x ranges over U.”	type inference
“Automatically change U to |U|.”	implicit coercion
“Read 2 as element of K.”	notational scope

Elaboration is an essential part of modern proof assistants. It is the bridge between the human and the machine. It was not developed by mathematicians or logicians, but by computer scientists who faced similar issues of human-computer interaction when designing programming languages.

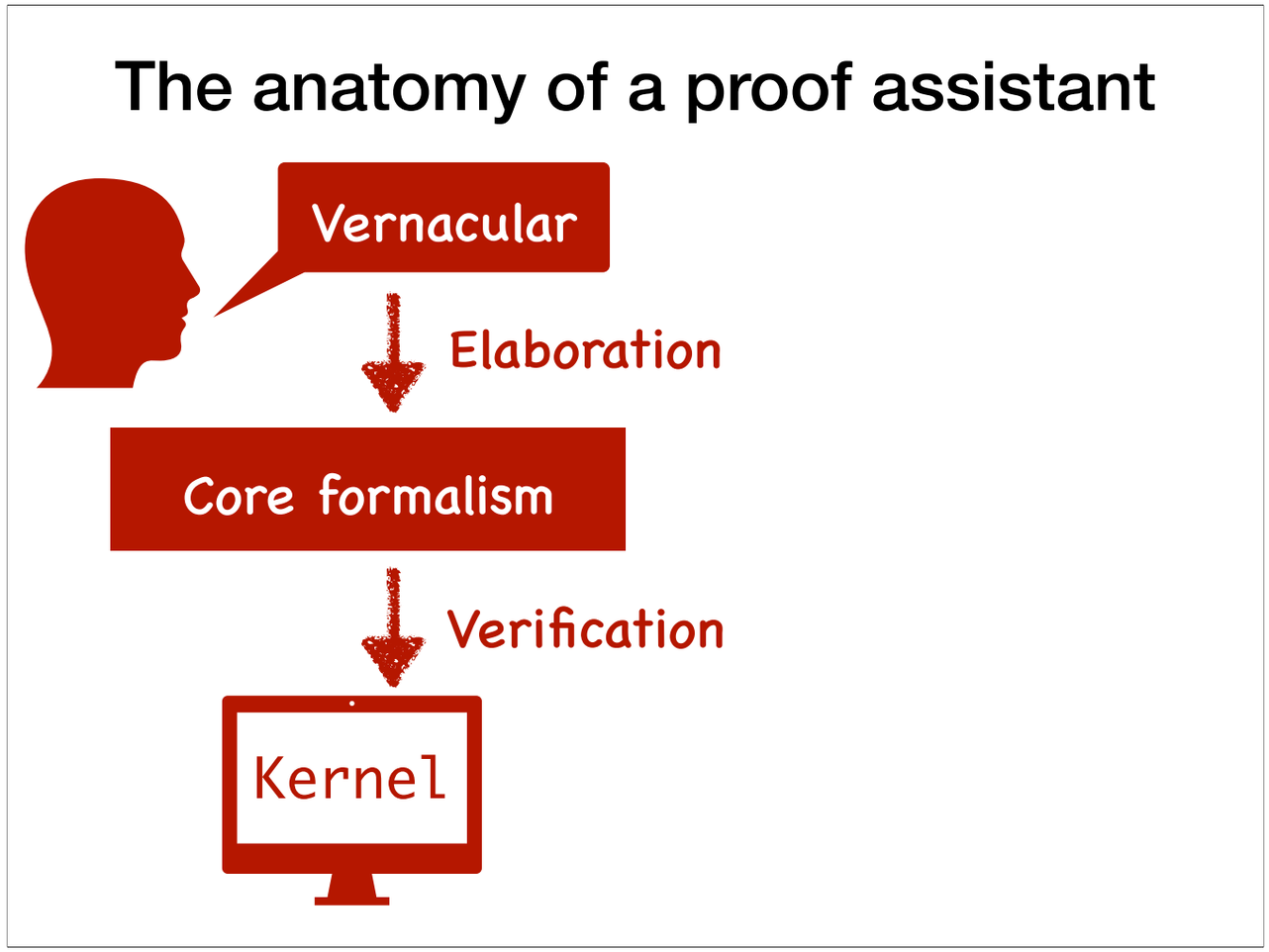
When you learn to use your first proof assistant, keep in mind that you already know the elaboration techniques, you just never gave them names or thought of them mathematically.

```
import linear_algebra

variables
  (K : Type*) [field K]
  (U : Type*) [add_comm_group U] [module K U]
  (V : Type*) [add_comm_group V] [module K V]
  (f : U →ₗ[K] V)

example: ∀ x y, f (2 • x + y) = 2 • f x + f y
  := by simp
```

Lean

You might be curious how the statement would be written in a proof assistant. Here it is in the syntax of the Lean proof assistant. There are some mysterious bits, but for the most part it is comprehensible, and it certainly is not worse than LaTeX. The last line, “by simp”, is how you tell Lean to prove the statement using the tactic “simp” (which stand for “simple”).

I shall say more about Lean and its rapidly growing community at the end of the talk.



Having explained elaboration, let us look at how a modern proof assistant is structured.
There are *two* formal languages: the vernacular and the core formalism.

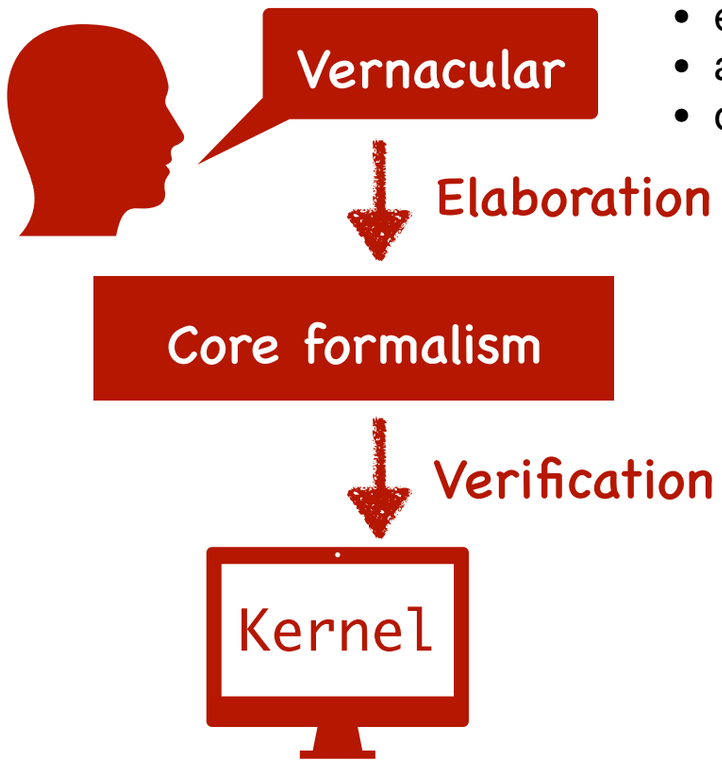
The vernacular is designed for humans, while still formally precise so that computers can process it. It supports common mathematical notations, provides support for automation, and generally attempts to accommodate the user. This is what you need to learn to master a proof assistant.

The elaborator translates the vernacular to a core formalism. This is the underlying formal mathematical foundation as commonly understood by mathematicians and logicians. It should be mathematically well-understood and free of unnecessary complexity.

The verification is carried out by a special component of the proof assistant, the kernel. This is the **only** critical component: a bug in the elaborator is just annoying, whereas a bug in the kernel may lead to faulty mathematics. The kernel is designed to be small so that it can be more easily audited and trusted.

Further reading: Robert Pollack, “How to believe a Machine-Checked Proof” (1996)

The anatomy of a proof assistant



- expressive & exact
- automates work
- designed for humans

The anatomy of a proof assistant

Vernacular

- expressive & exact
- automates work
- designed for humans

Elaboration

Core formalism

- simple & exact
- foundation of math
- designed for computer

Verification

Kernel

The anatomy of a proof assistant

Vernacular

- expressive & exact
- automates work
- designed for humans

Elaboration

Core formalism

- simple & exact
- foundation of math
- designed for computer

Verification

Kernel

- checks formal proofs
- small piece of code
- the **only** critical component

Example #2:
Identifying (canonically) isomorphic objects

We just saw how computer science and software design provided a solution to elaboration.

Our next example was given a solution by mathematicians. Not all problems in proof automation are in the domain of computer science and human-computer interaction.

We identify $\mathbb{Q}$ with the image of the inclusion $i : \mathbb{Q} \to \mathbb{R}$.

We identify R[1/f][1/g] with R[1/fg]
as they are canonically isomorphic.

Sh(CompHaus) = Sh(Profinite) = Sh(ExtrDisc)

Mathematicians sometimes pretend that isomorphic objects are equal, even though formally they are not.

For instance, we think of rationals as reals. Nobody frets about the difference between “1/2 as a rational” and “1/2 as a real”.

While the first example may be considered somewhat superficial, the second one is more serious. Certain constructions in algebra are identified as equal on the grounds of them being “canonically isomorphic”. What is a formal definition of “canonical isomorphism” and why should we think it makes sense to pretend that unequal objects are equal? Kevin Buzzard has disscussed these issues very nicely on several occasions.

We saw the third example in the first talk.

We identify $\mathbb{Q}$ with the image of the inclusion $i : \mathbb{Q} \to \mathbb{R}$.

We identify R[1/f][1/g] with R[1/fg]
as they are canonically isomorphic.

Vladimir Voevodsky proposed a mathematical solution to the problem. His univalence axiom makes the idea of identifying equivalent objects precise, and leads to a new foundation of mathematics. Benedikt Ahrens just gave a talk about it.

Even if you do not subscribe to the univalent foundations, this example makes an important point: when a seemingly trivial matter, a wrinkle ignored for decades, is paid proper mathematical attention, it can lead to new mathematical discoveries.

I am absolutely certain that formalized mathematics will uncover more opportunities for original mathematical ideas. Even if not all of them will lead to grand new discoveries, mathematicians should attend to them, if for not other reason to have better computer tools.

We identify $\mathbb{Q}$ with the image o… $\mathbb{Q} \to \mathbb{R}$.

…][1/g] with R[1/fg]
…re canonically isomorphic.

Voevodsky's univalence axiom

Example #3: Interoperability

It does not take much to find an opportunity for improvement. Here is one that does not have a good solution at present, and will become increasingly more important as we develop larger bodies of formalized mathematics.

If you know how to solve the problem for mathematics, you will also revolutionize software development.

Paper A: a monoid is a structure (M, ○) with associative ○ : M × M → M such that there exists a neutral element for ○.

Paper B: a monoid is a structure (M, ○, e) with associative ○ : M × M → M and a neutral element e ∈ M for ○.

Mathematics is not a cathedral in which everything fits together perfectly. Quite the contrary, it is a chaotic bazaar. Different authors set up their definitions and constructions in slightly different ways which do not fit together from a strictly formal point of view. This of course is a problem for machine assisted proofs.

For example, we see here two slightly different definitions of a monoid. Can results from paper A be used in paper B? You will say “yes of course, the difference is inessential”. You are correct, but what is the underlying mathematical reason for your claim, precisely? And how do we explain the discrepancy to the machine – without having to do a great deal of plumbing work that is mathematically trivial?

Univalence axiom offers a partial solution in case mismatched concepts are equivalent. I am not sure that it is a good solution, as inserting appeals to univalence axiom seems to treat the symptoms and not the cause.

Current solution

Current formalization efforts attack the problem by careful design. They create large libraries of mathematical structures in which everything fits together. This part of formalized mathematics resembles software engineering more than mathematics. Unfortunately, the problem remains when we ask about interoperability between different libraries of formal mathematics – even for a single proof assistant.

Challenges

In conclusion, let us

Better software

- Better proof assistants
- Larger libraries of mathematics
- Improve interoperability
- Incorporate automation and AI
- Incorporate symbolic computation
- Incorporate large math datasets

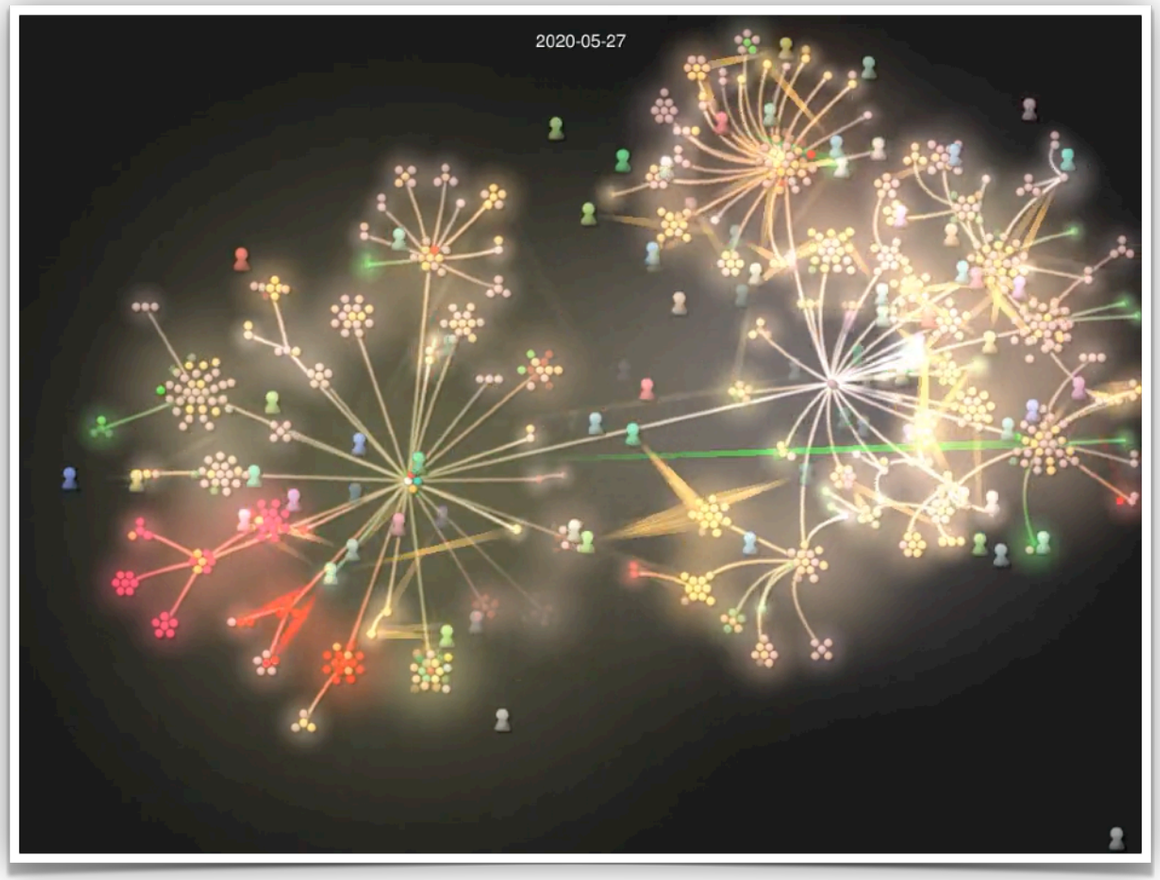
It is relatively easy to come up with a wish list for better software.

We are already working on all of these topics.

New mathematics

- Is type theory your final answer?
- Explain mathematically the vernacular and elaboration.
- What is a canonical isomorphism?
- Why do mismatching pieces of mathematics fit together?

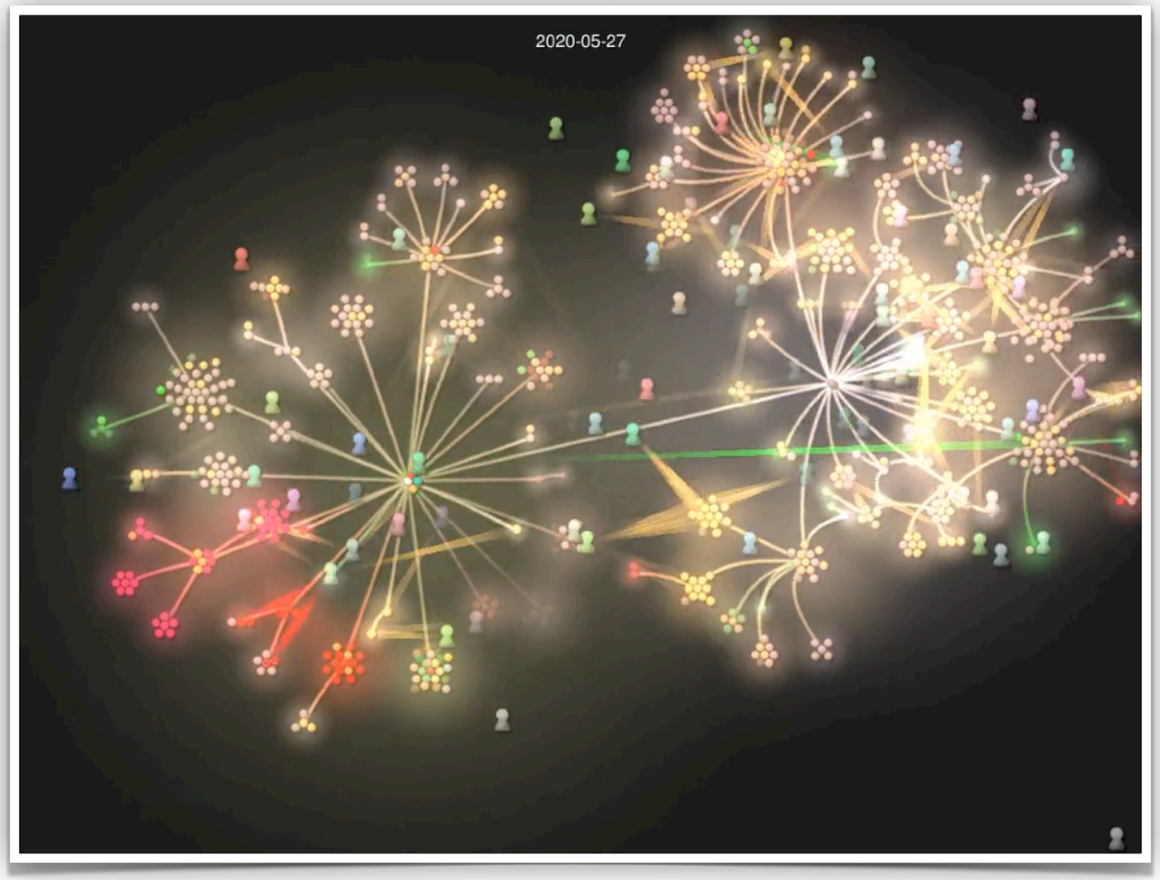
As a logician, I am personally more drawn to those aspects of machine assisted proofs that expose blind spots in (foundations of) mathematics.



The biggest challenge, in my opinion, is neither technical nor mathematical, but deeply human.

Mathematicians have always been lonely creatures. The average number of authors on a math paper has been creeping up slowly, but is still under 3. We are just not used to working in tightly connected large groups.

Here is a video showing the activities of mathematicians working on the mathlib library. This is unlike anything mathematicians have been doing in the 20th century. We are entering a new era that will change the nature of mathematicians' work deeply and fundamentally. We must design not only better machines and new mathematics, but also transform our own community.


2020-05-27