Proseminar on computer-assisted mathematics

Session 7 – Introduction to Lean

Judith Ludwig and Florent Schaffhauser

Heidelberg University, Summer semester 2024

Aim for today:

- *Lean in the wild*: Why are mathematicians interested in Lean and what do they do with it?
- *Everything is a function*: Functions and types in Lean.
- *Getting started*: Play the Natural Number Game (NNG) and install Lean.

1. What is Lean and why use it?

What is Lean?

Lean is a *functional programming language and an interactive theorem prover / proof assistant*.
Lean was created by Leonardo de Moura at Microsoft Research in 2013.
Lean is an open-source development environment for formal mathematics.

The aim: "to revolutionize mathematics by empowering anyone with an interest to grow in the field using Lean as their assistant".

What can you do with Lean?
With Lean you can formalize mathematics.

What does that mean?
Formalizing a mathematical statement means you can write it down in a format that the computer can **understand**. Formalizing it in Lean means, expressing it in Lean code so that Lean can interact with it. Formalizing a proof of that statement in Lean means, writing down a proof that Lean can interact with rigorously, i.e., can **check for correctness**.

Checking means: Everything is checked back to the level of the foundations. For that Lean implements a particular logic.
This logic is called *dependent type theory*. More later.

There are other proof assistants. For example Coq (also uses dependent type theory), HOL Light and Isabelle (simple type theory) or Mizar (set theory).

What proof assistants are not: They are not computers. Asking Lean to *compute* $\sqrt{3}$ is not a sensible thing to do. The number $\sqrt{3}$ is the positive real number that solves $x^2 - 3$, and it is precisely that number and any approximation is not $\sqrt{3}$.

Why should you care?

Here are some questions:
How much mathematics does a computer understand?
Think about the most abstract concept you've encountered in your studies.
Do you think your computer could understand it, i.e., do you think one can formalize it?
Could it check your solutions of an Algebra 1 exercise sheet?

Ten years ago, the answer would have almost certainly be: NO.
Today, it might well be that you can check them yourself.
(If you know how to talk to a proof assistant!).

Another motivation:
When formalizing you think about mathematics in a different way. You might find errors in your work!

Other potential uses:

- Searchable libraries of previously formalized maths.
- Interactive textbooks, where reader can choose how much detail they want to see.
- Mechanization / automation of mathematics.
- Error free mathematics.
- AI?

Why do mathematicians care?

We should tell you about the history of formally verified theorms, about achievements like the four colour theorem, or the proof of the Kepler conjecture.
For that let us refer you to [2] in the further reading references below.

Let us focus on three (very recent!) success stories of Lean within maths (there are more).

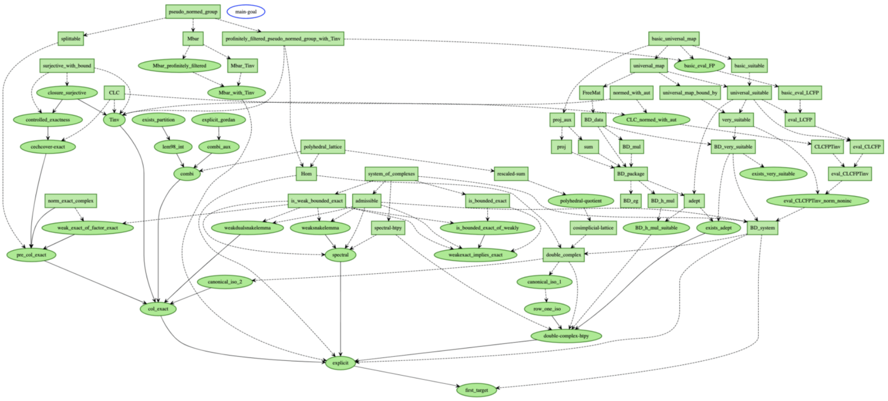
1. The Liquid Tensor Experiment

In December 2020 Peter Scholze posed a challenge to the formalization community to formally verify an important new theorem of himself and Dustin Clausen from the very new theory of *condensed mathematics*. This challenge was completed in July 2022.

It shows that *modern research* can be formalized and in particular verified.

Blueprint for the LTE.

Legend: **Boxes:** definitions **Ellipses:** theorems **Blue border:** ready **Blue bg:** proof ready **Green border:** statement done **Green bg:** proof done



2. Polynomial Freiman-Ruzsa Conjecture

Last year, mathematicians Ben Green, Timothy Gowers, Freddie Manners and Terence Tao proved an important conjecture in additive combinatorics. Within a month, a team led by Terence Tao verified the result using Lean.

It shows that formalization can happen fast and quicker than a peer-review process.

Just as with the Liquid Tensor Experiment, this was a big collaborative project. Many contributors were able to work on a small part without necessarily understanding the global proof. This is different from classical mathematical research!

3. Mathlib4

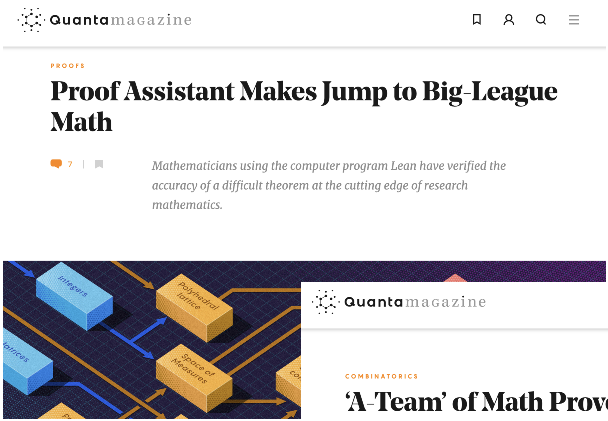
The Lean mathematical library, mathlib, is a community-driven effort to build a unified library of mathematics formalized in the Lean proof assistant.
The project is very active. It has many regular contributors and it grows every daily.
It already contains a lot of mathematics.

Checkout
https://leanprover-community.github.io/mathlib-overview.html

But there is lots to do.
You could become a contributor!

In the last ten years, the *formalization of mathematics* has grown immensely as a subject bridging maths and computer science.

Beyond mathematics and computer science, proof assistants are now being noticed.



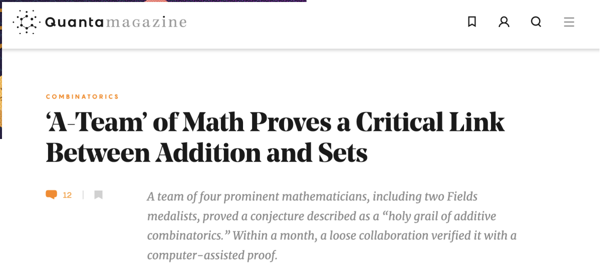
Quanta magazine

PROOFS

Proof Assistant Makes Jump to Big-League Math

7

Mathematicians using the computer program Lean have verified the accuracy of a difficult theorem at the cutting edge of research mathematics.



Quanta magazine

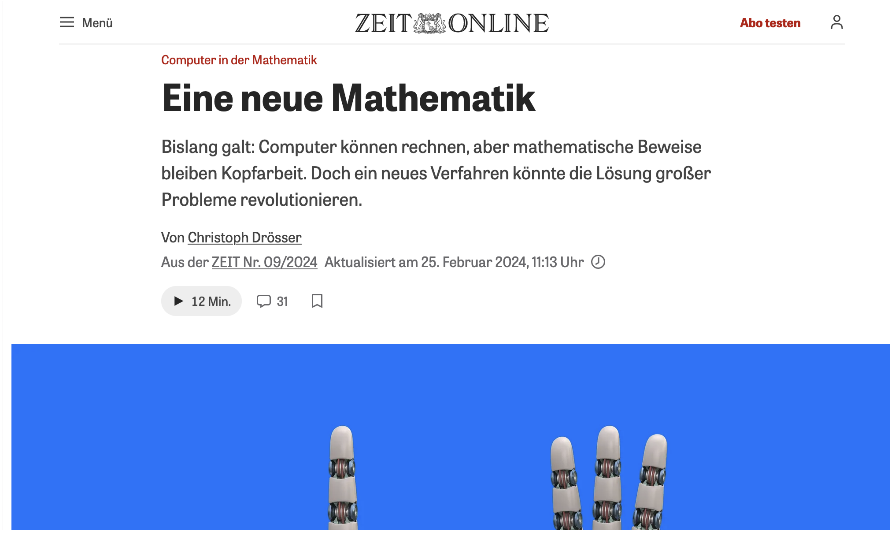
COMBINATORICS

‘A-Team’ of Math Proves a Critical Link Between Addition and Sets

12

A team of four prominent mathematicians, including two Fields medalists, proved a conjecture described as a “holy grail of additive combinatorics.” Within a month, a loose collaboration verified it with a computer-assisted proof.

https://www.quantamagazine.org/lean-computer-program-confirms-peter-scholze-proof-20210728
https://www.quantamagazine.org/a-team-of-math-proves-a-critical-link-between-addition-and-sets-20231206/



Computer in der Mathematik

Eine neue Mathematik

Bislang galt: Computer können rechnen, aber mathematische Beweise bleiben Kopfarbeit. Doch ein neues Verfahren könnte die Lösung großer Probleme revolutionieren.

Von Christoph Drösser

Aus der ZEIT Nr. 09/2024 Aktualisiert am 25. Februar 2024, 11:13 Uhr

https://www.zeit.de/2024/09/computer-mathematik-beweise-ueberpruefung-vertrauen

2. Functions and types in Lean

In contemporary university courses, mathematics is taught in the language of **set theory**: sets are the primary objects and everything else is built out of them.

For instance, a *function* $f : X \to Y$ is (supposed to be) defined as a subset

$$\Gamma \subset X \times Y \text{ such that } \forall\, x \in X, \exists! y \in Y, (x, y) \in \Gamma.$$

Similarly, an *equivalence relation* R on a set X is defined as a subset

$$R \subset X \times X$$

such that:

- $\forall\ x \in X, (x, x) \in R$ (*reflexivity*).
- $\forall\ x \in X$, $\forall\ y \in X$, $(x, y) \in R \Rightarrow (y, x) \in R$ (*symmetry*).
- $\forall\ x \in X$, $\forall\ y \in X$, $\forall\ z \in X$, $(x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$ (*transitivity*).

Another possible approach is to express mathematics in type-theoretic language, as opposed to set-theoretic. **Type theory** dates back to Bertrand Russell, who developed it to avoid certain paradoxes in set theory.

But more precisely, the basis for this alternate approach to mathematics is the simply-typed λ-calculus developed by Alonzo Church in the 1930s. There, the primary notion is the notion of *function*, and everything else is defined from that.

Consider for instance the following two approaches to natural numbers:

Von Neumann's encoding

0	$\emptyset$
1	$\{\emptyset\}$
2	$\{\emptyset, \{\emptyset\}\}$
3	$\{\emptyset, \{\emptyset, \{\emptyset\}\}\}$

The successor function is $n \mapsto n \cup \{n\}$.

Church's encoding

0	$f \mapsto (x \mapsto x)$
1	$f \mapsto (x \mapsto f(x))$
2	$f \mapsto (x \mapsto f(f(x)))$
3	$f \mapsto (x \mapsto f(f(f(x))))$

The successor function is $f^{\circ n} \mapsto f \circ f^{\circ n}$.

Note that the statement $0 \in 1$ makes sense in Von Neumann's encoding!

The reason why the type-theoretic approach is well-adapted to programming is because of the **Curry-Howard correspondence** between propositions and types.

In the interpretation of propositions-as-types, a function $f : P \to Q$ represents an *implication* between the proposition P and Q. More precisely, the function f is seen as a *proof* of the implication $P \to Q$.

The set-theoretic notation $P \Rightarrow Q$ is usually no longer used in the type-theoretic approach.

When a proposition P is viewed as a type, a **term** of type P is a proof of that proposition. So proving an implication $P \to Q$ means defining a function that sends a proof of P to a proof of Q.

In the type-theoretic approach, a *subset* P of a type X is a special type of function, called a **predicate**:

$$P : X \to \mathbf{Prop}.$$

Here, a proposition is a formal statement (provable or not).

Example:

- Of a proposition: "42 is divisible by 2" or "42 is divisible by 3".
- Of a predicate: the function $P : \mathbb{N} \to \mathbf{Prop}$ that sends $n : \mathbb{N}$ to the proposition "n is even".

Similarly, an equivalence relation R on a type X is defined as a function

$$R : X \to X \to \textbf{Prop}$$

such that:

- $\forall\ x : X$, the proposition $R\ x\ x$ has a proof (*reflexivity*).
- $\forall\ x : X$, $\forall\ y : X$, if the proposition $R\ x\ y$ has a proof, then the proposition $R\ y\ x$ has a proof (*symmetry*):

$$\forall\ x : X, \forall\ y : X, R\ x\ y \to R\ y\ x$$

- $\forall\ x : X$, $\forall\ y : X$, $\forall\ z : X$, if the proposition $R\ x\ y$ has a proof and the proposition $R\ y\ z$ has a proof, then the proposition $R\ x\ z$ has a proof (*transitivity*):

$$\forall\ x : X, \forall\ y : X, \forall\ z : X, R\ x\ y \to R\ y\ z \to R\ x\ z$$

Example: $X = \mathbb{N}$ and $R\ m\ n := (m - n \text{ is an even number})$.

3. Getting started

You can start by choosing an option from the following list:

- Play the Natural Number Game.
- Practice online with the following intro file.
- Install Lean 4 on your computer (see also the install instructions on the Leanprover-community website).
- Check out other resources and references on the next slide.

Resources:

- https://leanprover-community.github.io/
- Online Lean Server: https://live.lean-lang.org/
- The ADAM Project: https://adam.math.hhu.de/

Further reading:

1. Jeremy Avigad, *The Mechanization of Mathematics*, http://dx.doi.org/10.1090/noti1688, 2018.
2. Kevin Buzzard, *What is the point of computers? A question for pure mathematicians*, https://arxiv.org/abs/2112.11598, 2022.
3. Johan Commelin, *Liquid Tensor Experiment*, https://www.degruyter.com/document/doi/10.1515/dmvm-2022-0058/html?lang=en, 2022.