# What is Formalization of Mathematics?

2025 Lean summer school @ Utrecht

*Johan Commelin*

# Welcome to Utrecht!

Wifi instructions: sms “uu63” to +31 6 3574 4774

# Demo 1

Please visit: `https://live.lean-lang.org/`

# Overview / motivation

# Formalize (verb)

1. to give a certain or definite form to
2. (a) to make formal
   (b) to give formal status or approval to

Source: Merriam–Webster

# What is “formalization of mathematics”?

Encoding mathematics in a formal language

so that it can be mechanically verified and manipulated

# What is “formalization of mathematics”?

Encoding mathematics in a formal language

so that it can be mechanically verified and manipulated

This can be done theoretically: proof theory is a subfield of maths

But also practically: using *interactive theorem provers*

# What is “formalization of mathematics”?

Encoding mathematics in a formal language

so that it can be mechanically verified and manipulated

This can be done theoretically: proof theory is a subfield of maths

But also practically: using *interactive theorem provers*

This week we will focus on the practical side, using Lean

# Practical benefits (mathematics)

Formalized mathematics is good for

- Verification
- Automation
- Education
- Search
- Preservation
- Collaboration

# Practical benefits (mathematics)

Formalized mathematics is good for

- Verification
- Automation
- Education
- Search
- Preservation
- Collaboration
- Fun, joy, pleasure

# Practical benefits (computer science)

Formalization is good for

- Verified software (compilers, cryptography, kernels)
- Verified hardware (chip design, Intel bug)
- Antidote to hallucinating AI

# Interactive theorem provers

Many ITPs exist: Automath, Mizar, Rocq, Isabelle, HOL, Agda, Lean

Active research topic since the 1960's, mostly in CS.

Since 2017, rapid growth in the maths community.

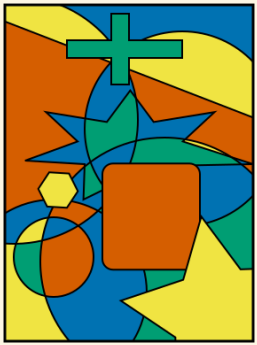
# Case studies in formalized maths

# Automath

- 1967: De Bruijn creates Automath
- 1976: Van Benthem Jutting translates
  Landau's *Foundations of Analysis* in Automath
- Long-lasting conceptual influence

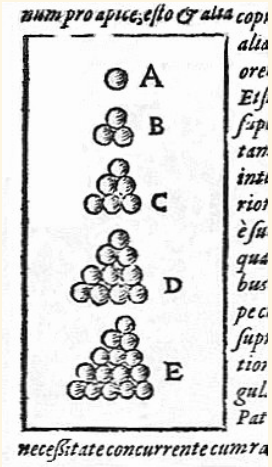
# Four colour theorem

- 1852: Guthrie conjectures
- 1976: Appel–Haken
  proof using extensive computer calculation
- 1996: Robertson–Sanders–Seymour–Thomas
  shorter proof, less cases, faster algorithm
- 2005: Gonthier–Werner formalize a proof in Rocq



# Flyspeck

- 1611: Kepler conjectures
- 19xx: Several proof attempts/claims
- 1998: Hales–Ferguson announce a proof
- 2003: *Annals* is 99% certain
- 2003: Hales launches Flyspeck
- Joint work with 21 collaborators
- 2015: Formal proof is completed
  using Isabelle and HOL Light



# Liquid Tensor Experiment

**Theorem** (Clausen–Scholze).
Let $0 < p' < p < 1$ be real numbers.
Let $S$ be a profinite set, and let $V$ be a $p$-Banach space.

Let $\mathcal{M}_{p'}(S)$ be the space of $p'$-measures on $S$.

Then

$$\mathrm{Ext}^i_{\mathrm{Cond(Ab)}}(\mathcal{M}_{p'}(S), V) = 0$$

for $i \geqslant 1$.

# Liquid Tensor Experiment

*I spent much of 2019 obsessed with the proof of this theorem, almost getting crazy over it. In the end, we were able to get an argument pinned down on paper, but I think nobody else has dared to look at the details of this, and so I still have some small lingering doubts.* — *Peter Scholze*

# Liquid Tensor Experiment

- 2020: Scholze posts formalization challenge
- 2021: Key ingredient formalized after 6 months
- Joint work of a dozen people
  (led by JMC and Adam Topaz)
- 2022: Full challenge completed in Lean

# Liquid Tensor Experiment

> *[. . .] one key problem I had when I was trying to find this proof was that I was essentially unable to keep all the objects in my 'RAM', and I think the same problem occurs when trying to read the proof*
>
> — *Peter Scholze*

Formalism helps manage cognitive load.

# Equational theories

- 2024: Terence Tao starts project, many contributors
- If a binary operation satisfies

$$\forall x, y, z: \quad x \cdot y \cdot x \cdot z = z \cdot y$$

does it also satisfy commutativity?

# Equational theories

- 2024: Terence Tao starts project, many contributors
- If a binary operation satisfies

$$\forall x, y, z : \quad x \cdot y \cdot x \cdot z = z \cdot y$$

  does it also satisfy commutativity?
- There are $\approx 4000$ small "laws":
  what about all $\approx 4000^2$ implications?

# Equational theories

- Computers settled $> 99.99\%$ of cases, human ingenuity settled the rest.
- Formalized in Lean.
- Modern AI only played a very small role.

# Equational theories

- Computers settled $> 99.99\%$ of cases, human ingenuity settled the rest.
- Formalized in Lean.
- Modern AI only played a very small role.
- A handful of these axioms are “Martian” and deserve more study.

# DeepMind: AlphaProof

- 2024: AlphaProof solves 4 out of 6 IMO problems
- Equivalent of a silver medal

# DeepMind: AlphaProof

- 2024: AlphaProof solves 4 out of 6 IMO problems
- Equivalent of a silver medal
- We do not know many details about the implementation
  - Lean to verify formal correctness
  - Large language models to translate to Lean
  - Reinforcement learning to learn proving
- The generated proofs are *weird*!

# AI @ IMO 2025

- IMO 2025 happened last week @ Australia
- Organization asked to first celebrate human IMO

# What is Lean?

# History of Lean

- 2013: Leonardo de Moura launches Lean at Microsoft Research
- 2015: Lean 2 is released (HoTT mode)
- 2017: Lean 3 (start of Mathlib)
- 2021: Lean 4

# Lean 4

- Full-fledged functional programming language
- Almost self-hosted: Lean 4 is written in Lean 4
- Dependent type theory
- Extensible syntax
- Extensible tactics (automation)

# Mathlib

- $\approx$ 1.9 million lines of code
- $\approx$ 300 000 results
- $\approx$ 500 contributors
- covering the foundations of linear algebra, topology, analysis, algebra, number theory, geometry, category theory, ...

# Tactics

A *tactic* is a program (often written in Lean)

that can create a partial proof.

Many proof techniques have a corresponding tactic:

- `induction`
- `by_cases`
- `contrapose`

# Proof terms

Under the hood, tactics build *proof terms*.

The *kernel* verifies proof terms for correctness.

# Proof terms

Under the hood, tactics build *proof terms*.

The *kernel* verifies proof terms for correctness.

You will sometimes write proof terms by hand.

For example using the `exact` tactic:

```
exact [proof term goes here]
```

# Declarations (def/theorem/lemma)

The rough anatomy of a declaration in Lean:

```
decl_cmd someName (an :  assumption) (one :  more assumption) :
    some Lean statement := some proof/value
```

- `decl_cmd` is `def` or `theorem` or `lemma`
- `someName` is the name of your def/theorem (like a LaTeX label)
- followed by a list of assumptions
- `:` signals that the statement/conclusion follows
- `:=` signals that the proof/value follows

# Superfast theory crash course

# Foundations

Formalizing mathematics requires a logical foundation.

In the past century, ZFC has dominated mathematics.

*Type theory* is an alternative foundation.

It has the same logical strength, but some practical advantages.

# Naive type theory

| | |
|---|---|
| Type $X$ | — Set $X$ |
| Term $x : X$ | — Element $x \in X$ |
| Function $X \to Y$ | — Function $X \to Y$ |
| Product $X \times Y$ | — Product $X \times Y$ |
| Sum $X \oplus Y$ | — Disjoint union $X \sqcup Y$ |

# Curry–Howard correspondence

| | |
|---|---|
| Type $P$ | — Proposition $P$ |
| Term $p : P$ | — Proof $p$ of $P$ |
| Function $P \to Q$ | — Implication $P \Rightarrow Q$ |
| Product $P \times Q$ | — Conjunction $P \wedge Q$ |
| Sum $P \oplus Q$ | — Disjunction $P \vee Q$ |

# Typing judgement

- In set theory you *can* prove $x \in X$
- In type theory you *cannot* prove $x : X$
- The type checker can verify $x : X$

# Typing judgement

- In set theory you *can* prove $x \in X$
- In type theory you *cannot* prove $x : X$
- The type checker can verify $x : X$
- By the Curry–Howard correspondence,
  proof checking is a special case of type checking

# No better time to act then now

*The only way to learn mathematics*
*is to do mathematics*

*— Paul Halmos*

# Demo 2

Please visit: `https://live.lean-lang.org/`