Formalising Mathematics in Lean

Christian Merten

August 15, 2025

Summer School on Geometry, Utrecht University

Demo!

Please head to:

`https://github.com/formal-methods-nl/uu-geometry-2025`

What is „Formalisation of Mathematics?"

Encoding mathematics in a formal language, so that it can be mechanically verified and manipulated.

What is „Formalisation of Mathematics?”

Encoding mathematics in a formal language, so that it can be mechanically verified and manipulated.

- Theoretically: proof theory

What is „Formalisation of Mathematics?”

Encoding mathematics in a formal language, so that it can be mechanically verified and manipulated.

- Theoretically: proof theory
- Practically: using interactive theorem provers

What is „Formalisation of Mathematics?"

Encoding mathematics in a formal language, so that it can be mechanically verified and manipulated.

- Theoretically: proof theory
- Practically: using interactive theorem provers

We will use the interactive theorem prover *Lean* and its mathematical library *mathlib*.

Why formalise mathematics?

Why formalise mathematics?

- Verification

Why formalise mathematics?

- Verification
- Automation

Why formalise mathematics?

- Verification
- Automation
- Education

Why formalise mathematics?

- Verification
- Automation
- Education
- Search

Why formalise mathematics?

- Verification
- Automation
- Education
- Search
- Collaboration

Why formalise mathematics?

- Verification
- Automation
- Education
- Search
- Collaboration

and the most important one:

Why formalise mathematics?

- Verification
- Automation
- Education
- Search
- Collaboration

and the most important one:

- It is a lot of fun!

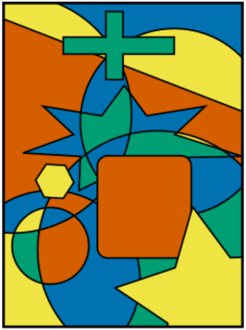
Some examples

Disclaimer: There exist many proof assistants (Automath, Mizar, Rocq, Isabelle, HOL, Agda, Lean, etc.) and they all come with important and successful formalisation projects.

What follows is not a comprehensive list!

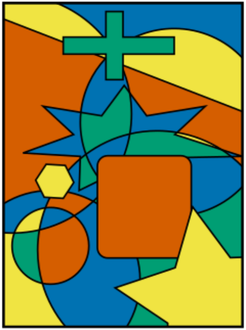
Four colour theorem

- 1852: Guthrie conjectures



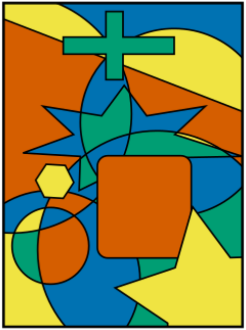
Four colour theorem

- 1852: Guthrie conjectures
- 1976: Appel-Haken proof using extensive computer calculation



Four colour theorem

- 1852: Guthrie conjectures
- 1976: Appel-Haken proof using extensive computer calculation
- 2005: Gonthier-Werner formalised proof in Roq



Liquid Tensor Experiment

- 2019: Clauen-Scholze prove fundamental theorem on liquid vector spaces

Liquid Tensor Experiment

- 2019: Clauen-Scholze prove fundamental theorem on liquid vector spaces

  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. — Scholze

Liquid Tensor Experiment

- 2019: Clauen-Scholze prove fundamental theorem on liquid vector spaces

  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. — Scholze

- 2020: Scholze posts formalisation challenge

Liquid Tensor Experiment

- 2019: Clauen-Scholze prove fundamental theorem on liquid vector spaces

  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. — Scholze

- 2020: Scholze posts formalisation challenge
- 2022: Full challenge completed in Lean

Liquid Tensor Experiment

- 2019: Clauen-Scholze prove fundamental theorem on liquid vector spaces

  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. — Scholze

- 2020: Scholze posts formalisation challenge
- 2022: Full challenge completed in Lean
- Joint work of a dozen people, led by Johan Commelin and Adam Topaz

Foundations

- Formalising mathematics requires a logical foundation.

Foundations

- Formalising mathematics requires a logical foundation.
- In the past century, ZFC has dominated mathematics.

Foundations

- Formalising mathematics requires a logical foundation.
- In the past century, ZFC has dominated mathematics.
- *Type theory* is an alternative foundation.

Foundations

- Formalising mathematics requires a logical foundation.
- In the past century, ZFC has dominated mathematics.
- *Type theory* is an alternative foundation.
- It has the same logical strengh, but some practical advantages.

Naive type theory

Type theory	Set theory
Type `X`	Set X
Term `x : X`	Element $x \in X$
Function `X → Y`	Function $X \to Y$
Product `X × Y`	Product $X \times Y$
Sum `X ⊕ Y`	Disjoint union $X \sqcup Y$

Naive type theory

Type theory	Set theory
Type `X`	Set X
Term `x : X`	Element $x \in X$
Function `X → Y`	Function $X \to Y$
Product `X × Y`	Product $X \times Y$
Sum `X ⊕ Y`	Disjoint union $X \sqcup Y$

- Everything has a type.

Naive type theory

Type theory	Set theory
Type `X`	Set X
Term `x : X`	Element $x \in X$
Function `X → Y`	Function $X \to Y$
Product `X × Y`	Product $X \times Y$
Sum `X ⊕ Y`	Disjoint union $X \sqcup Y$

- Everything has a type.
- In set theory you *can* prove $x \in X$.

Naive type theory

Type theory	Set theory
Type `X`	Set X
Term `x : X`	Element $x \in X$
Function `X → Y`	Function $X \to Y$
Product `X × Y`	Product $X \times Y$
Sum `X ⊕ Y`	Disjoint union $X \sqcup Y$

- Everything has a type.
- In set theory you *can* prove $x \in X$.
- In type theory you *cannot* prove `x : X`.

Naive type theory

Type theory	Set theory
Type `X`	Set X
Term `x : X`	Element $x \in X$
Function `X → Y`	Function $X \to Y$
Product `X × Y`	Product $X \times Y$
Sum `X ⊕ Y`	Disjoint union $X \sqcup Y$

- Everything has a type.
- In set theory you *can* prove $x \in X$.
- In type theory you *cannot* prove `x : X`.
- The type checker can verify `x : X`.

Propositions as types

- In Lean propositions are a special case of types.

Propositions as types

- In Lean propositions are a special case of types.
- `ℕ : Type` and `Odd n : Prop`.

Propositions as types

- In Lean propositions are a special case of types.
- `ℕ : Type` and `Odd n : Prop`.

Type `P`	Proposition P
Term `hp : P`	Proof of P
Function `P → Q`	Implication $P \implies Q$
Product `P × Q`	Implication $P \wedge Q$
Sum `P ⊕ Q`	Implication $P \vee Q$

Propositions as types

- In Lean propositions are a special case of types.
- `ℕ : Type` and `Odd n : Prop`.

Type `P`	Proposition P
Term `hp : P`	Proof of P
Function `P → Q`	Implication $P \implies Q$
Product `P × Q`	Implication $P \wedge Q$
Sum `P ⊕ Q`	Implication $P \vee Q$

- Question: `Prop : ?`

Propositions as types

- In Lean propositions are a special case of types.
- `ℕ : Type` and `Odd n : Prop`.

Type `P`	Proposition P
Term `hp : P`	Proof of P
Function `P → Q`	Implication $P \implies Q$
Product `P × Q`	Implication $P \wedge Q$
Sum `P ⊕ Q`	Implication $P \vee Q$

- Question: `Prop : ?`
- Question: `Type : ?`

Enough theory! Let's go back to Lean.