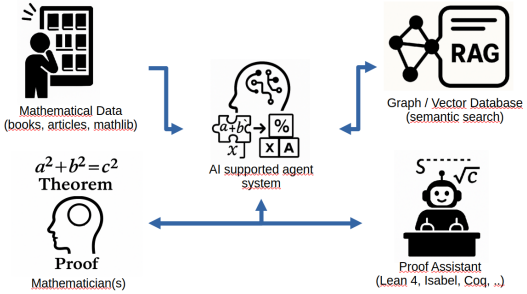
Formalising Noncommutative Geometry in Lean 4: First Steps Towards Connes' Reconstruction Theorem

Christoph Stephan

Institut für Mathematik

NCG at CIRM
April 2025

Mathematical Data, Proof Assistants, and AI



- Explosion of accessible mathematical data (arXiv, electronic books, Lean mathlib).
- Proof assistants becoming more user-friendly (e.g., Lean 4).
- AI could help organising mathematics and translating human proofs into machine-readable formats.

Overview of Proof Assistants

Proof assistants have been developed in computer science to check validity of algorithms and proofs.

What is a Proof Assistant?

- Interactive theorem prover
- Assists in developing formal proofs
- Checks correctness of logical steps
- Used in mathematics, computer science, formal verification
- Combines automated and interactive techniques

Examples of Popular Proof Assistants

- **Lean** - Calculus of Constructions (type theory), developed by Microsoft Research, user-friendly, powerful features
- **Coq** - Calculus of Constructions (type theory), developed by INRIA, formalizes mathematical proofs, verifies software
- **Isabelle** - Higher-order logic, developed by Tobias Nipkow and Lawrence Paulson, supports various logics, formal verification
- **HOL Light** - Higher-order logic, developed by Mike Gordon at Cambridge University, verifies mathematical proofs, hardware/software

Importance in Mathematics

- **Verification of complex proofs**
  - Four-color theorem (1976, verified 2005 in Coq by Gonthier)
  - Liquid Tensor Experiment by Scholze (Lean)
- **Formalizing mathematics** (focus on Lean 4)
  - Perfectoid spaces by Scholze, Buzzard, Commelin, Massot
  - Cap-set problem by Dahmen, Hölzl, Lewis
  - Continuum hypothesis by Han, van Doorn
  - PFR conjecture by Tao, Dillies, Mehta
  - Fermat's Last Theorem by Buzzard, Taylor
- **Reducing human error in proofs**
- **Facilitating collaboration**

Lean 4

Technical features of Lean 4:

- Lean 4 is a functional programming language and interactive proof assistant.
- Lean 4 is strongly typed (every variable has a fixed type, think of integers)
- Lean 4 is based on the **Calculus of Constructions** (dependent type theory)
- Fully extensible: users can modify the parser, elaborator, tactics, decision procedures, and more.

Practical features of Lean 4:

- Easily used in VS Code (via extension).
- Has a large library of formalized mathematics (mathlib).
- Has young and active community.
- Lean 4 is open source.

Lean 4 has been developed primarily by Leonardo de Moura at Microsoft Research.

Dependent Type Theory I

"Standard" foundation of mathematics:
Set theory (ZFC) and first-order logic (predicate logic).

Dependent Type Theory I

"Standard" foundation of mathematics:
Set theory (ZFC) and first-order logic (predicate logic).

Curry-Howard-Lambek correspondence: Mathematical statements and proofs can be equivalently formulated in:

- Set theory + first-order logic,
- (Dependent) type theory, or
- Category theory.

Important for Lean 4:
Curry-Howard correspondence ensures
set theory + first-order logic $\Leftrightarrow$ dependent type theory

Dependent Type Theory II

What is type theory?

Dependent Type Theory II

What is type theory?

1. There are terms and types.

Dependent Type Theory II

What is type theory?

1. There are terms and types.
2. Everything is a term. Notation: $a, A, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.

Dependent Type Theory II

What is type theory?

1. There are terms and types.
2. Everything is a term. Notation: $a, A, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
3. Terms can be types. Notation: $A, B, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.

Dependent Type Theory II

What is type theory?

1. There are terms and types.
2. Everything is a term. Notation: $a, A, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
3. Terms can be types. Notation: $A, B, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
4. Every term has a type. Notation $a : A$ (a has type A).

Dependent Type Theory II

What is type theory?

1. There are terms and types.
2. Everything is a term. Notation: $a, A, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
3. Terms can be types. Notation: $A, B, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
4. Every term has a type. Notation $a : A$ (a has type A).
5. Rules how to construct new types from given types.

Dependent Type Theory II

What is type theory?

1. There are terms and types.
2. Everything is a term. Notation: $a, A, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
3. Terms can be types. Notation: $A, B, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
4. Every term has a type. Notation $a : A$ (a has type A).
5. Rules how to construct new types from given types.

Some basics:

- If there is a term a of type A, i.e. $a : A$, we say that A is inhabited.

Dependent Type Theory II

What is type theory?

1. There are terms and types.
2. Everything is a term. Notation: $a, A, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
3. Terms can be types. Notation: $A, B, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
4. Every term has a type. Notation $a : A$ (a has type A).
5. Rules how to construct new types from given types.

Some basics:

- If there is a term a of type A, i.e. $a : A$, we say that A is inhabited.
- A type inhabited by types is called a universe.
  Notation: $\mathcal{U}, \mathcal{U}_k, \mathsf{Prop}$, etc.

Dependent Type Theory II

What is type theory?

1. There are terms and types.
2. Everything is a term. Notation: $a, A, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
3. Terms can be types. Notation: $A, B, \mathcal{U}, \mathbb{N}, \mathsf{Prop}$, etc.
4. Every term has a type. Notation $a : A$ (a has type A).
5. Rules how to construct new types from given types.

Some basics:

- If there is a term a of type A, i.e. $a : A$, we say that A is inhabited.
- A type inhabited by types is called a universe.
  Notation: $\mathcal{U}, \mathcal{U}_k, \mathsf{Prop}$, etc.
- $\mathbf{0}/\mathbf{1}$ denote the generic type with no/one element.
  Note: there are more types with no/one element

Dependent Type Theory III

Constructing types from types (a few examples)

- Function types:
  Given types A, B, the function type is denoted $A \to B$.
  Construction: if $f : A \to B$ and $a : A$ then $f(a) : B$.
  Note: $A \to B$ is not inhabited iff A is inhabited and B is not inhabited.
- Product types:
  Given types A, B, the procduct type is denoted by $A \times B$.
  Construction: if $a : A$ and $b : B$ then $(a, b) : A \times B$.
- Dependent function types:
  Given a type A and a universe $\mathcal{U}$ and $B : A \to \mathcal{U}$.
  The dependent function type is denoted by $\Pi_{(x:A)} B(x)$.
  Construction: if $f : \Pi_{(x:A)} B(x)$ and $a : A$ then $f(a) : B(a)$.

Propositions and Proofs in Type Theory

Propositions as types

- Prop is the universe of all propositions (true or false).
- Propositions are types $P : \text{Prop}$ with no or one representative.
  Example: $P = \text{Fermat's last theorem}$
- $P : \text{Prop}$ is true if it is inhabited, otherwise it's false.
- Proofs are terms $\text{proof} : P$ but not types.
  Example: $P = \text{Fermat's last theorem}$ is inhabited, i.e. there exists a proof.

Overview of Dependent Type Theory

Types	Logic	Sets
A	proposition	set
$a : A$	proof	element
$B(x)$	predicate	family of sets
$b(x) : B(x)$	conditional proof	family of elements
$\mathbf{0}, \mathbf{1}$	$\bot, \top$	$\emptyset, \{\emptyset\}$
$A + B$	$A \vee B$	disjoint union
$A \times B$	$A \wedge B$	set of pairs
$A \to B$	$A \Rightarrow B$	set of functions
$\sum_{(x:A)} B(x)$	$\exists_{x:A} B(x)$	disjoint sum
$\prod_{(x:A)} B(x)$	$\forall_{x:A} B(x)$	product
Id_A	equality $=$	$\{ (x, x) \mid x \in A \}$

Figure: Overview from the HoTT Book

Dependent Type Theory and Lean 4

Generically Lean 4 requires constructive proofs.

Lean's Calculus of Constructions:
Calculus of Constructions is equivalent to ZF with intuitionistic logic.

Lean 4 can also work non-constructively, i.e. with classical logic.

Lean library for standard logic: `Classical`
Provides full ZFC with first order logic.

A "Simple" Lean 4 Example: There are infinitely many primes

- Lean 4 uses **tactics** to construct proofs.
- Tactic **by** starts a proof.
- Tactic **show** allows to show a goal (helpful for large proofs).
- Tactic **have** allows to introduce a new hypothesis.
- Tactic **sorry** declares a proof without providing it.

```
theorem primes_infinite : ∀ n, ∃ p > n, Nat.Prime p := by
  intro n
  have : 2 ≤ Nat.factorial (n + 1) + 1 := by
    sorry
  rcases exists_prime_factor this with ⟨p, pp, pdvd⟩
  refine ⟨p, ?_, pp⟩
  show p > n
  by_contra ple
  push_neg at ple
  have : p ∣ Nat.factorial (n + 1) := by
    sorry
  have : p ∣ 1 := by
    sorry
  show False
  sorry
```

[0]Formalisation from "Mathematics in Lean".

Connes' Reconstruction Theorem

Why formalising Noncommutative Geometry in Lean 4?

- Algebraic fields of mathematics are easier to formalise.
- NCG may provide back door to formalising Differential Geometry.
- Because I like NCG and want to formalise it.

The Aim: Reconstruction Theorem

Let (A, H, D) be a p-dimensional commutative spectral triple. There exist a compact oriented Riemannian p-manifold X, a Hermitian vector bundle $E \to X$, and an essentially self-adjoint Dirac-type operator D_E on E, such that

$$(A, H, D) \cong (C^\infty(X), L^2(X, E), D_E).$$

Where we stand: Functional Analysis & C^*-algebras

Definition/Theorem	**In mathlib**
Hilbert space and bounded/self-adjoint/compact operators	Yes
Compact operator and compact resolvent	Yes
*-algebra and *-representation on Hilbert space	Partial
GNS construction from a state on a C*-algebra	No
C*-algebra and C*-norm identity	Yes
Gelfand–Naimark theorem for C*-algebras	Partial
Gelfand duality for commutative C*-algebras	Yes
Compact operators $K(H)$	Yes
Fredholm operators and Fredholm modules	Partial
Functional calculus for self-adjoint operators	Partial

Where we stand: Smooth Manifolds & Vector Bundles

Definition/Theorem	**In mathlib**
Smooth manifold (Hausdorff, second-countable, C^∞-atlas)	Yes
Algebra $C^\infty(X)$ of smooth functions	Yes
Complex vector bundle and smooth sections	Partial
Serre–Swan theorem	No
Riemannian metric on a manifold	Partial
Orientation and oriented manifold	No
Spin and spin^c structures	No
Spinor bundle from spin^c structure	No
Dirac-type operator on Hermitian bundle	No
Spectral theorem for self-adjoint elliptic operators	Partial

Where we stand: Spectral Triples

Definition/Theorem	**In mathlib**
Spectral triple (A, H, D) with bounded commutators	No
Summability: $\mathrm{Tr}(\lvert D\rvert^{-s}) < \infty$ for $s > p$	No
Dimension spectrum of a spectral triple	No
Regularity: commutators with $\lvert D\rvert$ densely defined	No
Smooth domain of D	No
$H_\infty = \bigcap \mathrm{Dom}(D^k)$	No
Finiteness: H_∞ is finitely generated projective A-module	No
Orientability: Hochschild p-cycle with $\pi_D(c) = \gamma$	No
Hochschild complex and Hochschild homology	No
Grading γ with $\gamma D = -D\gamma$	No
Real structure J on H	No
First-order condition $[[D, a], Jb^*J^{-1}] = 0$	No

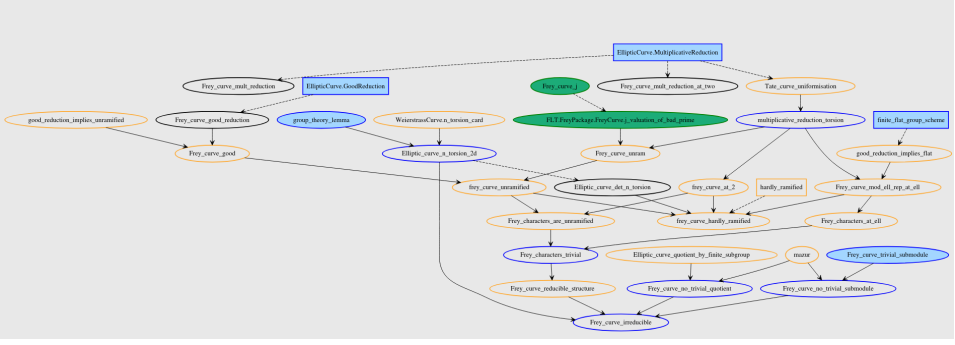
Challenges with Lean 4

- **Learning curve:** Lean 4 has a steep learning curve, especially for beginners.
- **Precision:** Everything has to be spelled out. No "put it into coordinates and you see..."', "as one easily sees...", "trivially...", etc.
- **Complexity:** Lean 4's syntax and semantics can be complex and difficult to understand (although it is improving and better than in Lean 3).
- **Limited libraries:** Many mathematical concepts are not yet formalized in Lean 4.
- **Documentation:** Documentation for Lean 4 is still evolving, making it challenging to find information.
- **Searchability:** Finding specific definitions or theorems in **mathlib** can be difficult due to the large amount of data and naming conventions.

The iPhone-moment for Lean 4 is still to come.

Graphical visualisation of Mathematics and Lean 4 code I

Lean blueprints: plasTeX plugin to generate graph representations of proofs

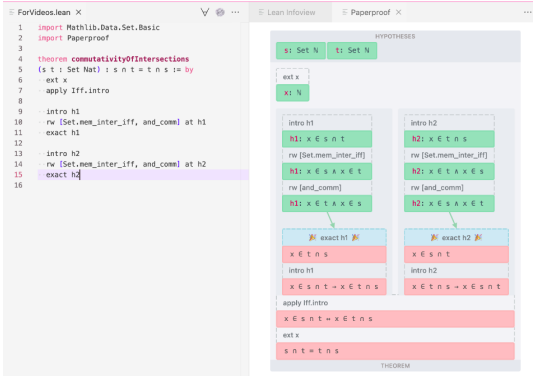


Graphical blueprint representation of the proof of Fermat's Theorem. (K. Buzzard, R. Taylor)

Graphical visualisation of Mathematics and Lean 4 code II

Paperproof:

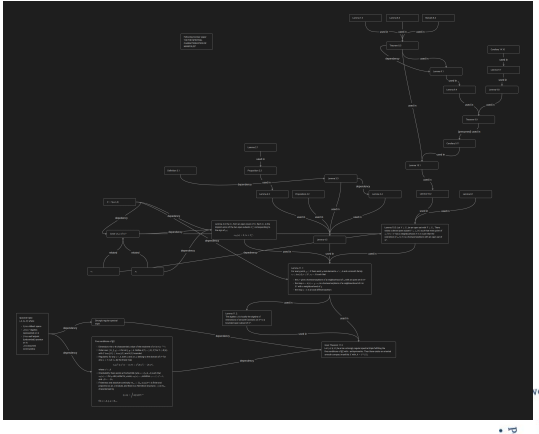
- VS Code extension for Lean 4
- Graph representation of Lean 4 code
- Not interactive yet



Current IT Project

Aim: Graph representation of mathematical texts and Lean 4 code.

- Interactive graph of definitions, theorems & proofs
- Bidirectional sync: graph ↔ Lean code
- Nodes/edges show proof status & presence in Lean
- Works with both math text and Lean code
- VS Code plugin for Lean 4
- AI assistance for graph/code generation



Graph of the reconstruction theorem.

Resources

Lean Game Server

Lean Main

Lean Community

Lean mathlib

Lean blueprint

Paperproof

People working on NCG in Lean 4

C. Stephan (Potsdam)

J. Taylor (Potsdam)

L. Ronge (Potsdam)

F. Hanisch (Potsdam)