What formalized mathematics is all about?

Adam Topaz

University of Alberta

December 2, 2024

It is about encoding mathematical

definitions, *theorems* and *proofs*

using a formal language

so that they can be

checked, *generated*, *processed*, and *stored*

using a computer.

Tools

Formal methods have been used in computer science for a while to write specifications for hardware and software, and to verify that specifications are met.

In practice this involves using both *automated reasoning tools*, and *interactive tools* like *proof assistants*.

This talk will focus on a particular proof assistant called *Lean*, developed primarily by L. de Moura, originally at Microsoft Research, but now at AWS and the Lean FRO.

Other proof assistants exist: Isabelle, Mizar, HOL Light, Coq, ...

Motivation

- Formal verification of theorems.
- Digital Libraries of Mathematics.
- Various collaborative aspects.
- Helps in taming complexity.
- Mathematicial artifacts become digital.
- *This is just the beginning!*

Motivation

Theme: Scale

Now I'll move to the blackboard.

Now I'll move to the ~~blackboard~~ Lean editor.

mathlib

- Lean4's primary mathematics library.
- Currently $\sim$ 1.5 Million lines of code.
  - Vast majority is formalized mathematics.
  - Some infrastructure code, tactics, etc.
- Over 300 contributors, $\sim$ 30 maintainers, $\sim$ 50 reviewers.

mathlib

Algebra, AlgebraicGeometry, AlgebraicTopology, Analysis, CategoryTheory, Combinatorics, Computability, Condensed, Control, Data, Deprecated, Dynamics, FieldTheory, Geometry, GroupTheory, InformationTheory, Lean, LinearAlgebra, Logic, Mathport, MeasureTheory, ModelTheory, NumberTheory, Order, Probability, RepresentationTheory, RingTheory, SetTheory, Std, Tactic, Testing, Topology, Util,

`mathlib` docs and search

- Documentation webpage
- Loogle!
- Moogle!
- Lean Search

mathlib

- Maintain compatibility across all subcomponents.
- Promote the *right* level of abstraction.
- Organize concepts into hierarchies.
- Enable formalization of research-level mathematics.

Collaboration

Lean/`mathlib` opens new avenues for mathematical collaboration *at scale*.

Collaboration

- *The Liquid Tensor Experiment*: ~ 12 main contributors, ~ 30 in total.
- *The Sphere Eversion Project*: ~ 3 main contributors, ~ 15 in total.
- *The Polynomial Freiman-Ruzsa Conjecture*: ~ 30 contributors in total.
- *The prime number theorem and more*: over 20 contributors.

- *The Formalization of FLT*: over 30 contributors.

Collaboration

In essentially all cases, these projects had a few "*leaders*," a smaller group of *core contributors*, and a much larger group of *casual contributors*.

T. Tao:

> *One notable feature of proof formalization projects is that they lend themselves to large collaborations that do not require high pre-established levels of trust.*

Blueprints

One of the most effective methods for facilitating such projects is via the use of *blueprints*, which have become essentially standard practice for large Lean formalization projects.

LTE, PNT++

This new paradigm allows work to be *widely distributed* across many contributors.

- Contributors *with no Lean experience* can contribute to the blueprint.
- Contributors *with Lean experience* can contribute to the formalization.
- Primary contributors can set up *skeletons* for various targets.
- Casual contributors can focus on targets within their interests and ability.

Spec-driven development

Algorithm:

1. Isolate a desired *target* (defn, theorem, . . .)
2. Isolate an initial *specification* (spec) for the target.
3. Break down the target/spec into parts with lower complexity.
4. Repeat the above, with each new part acting as the target.

Key is a liberal use of `sorry`.

```
universe u

def AbsoluteGaloisGroup (K : Type u) [Field K] :
    Type u := sorry

variable (K : Type u) [Field K]

instance :
    TopologicalSpace (AbsoluteGaloisGroup K) :=
  sorry

instance :
    CompactSpace (AbsoluteGaloisGroup K) :=
  sorry
```

```
def FiniteGalExt
    (K : Type u) [Field K] :
    Type (u+1) := sorry

def FiniteGalExt.Gal
    (M : FiniteGalExt K) :
    Type u := sorry

instance (M : FiniteGalExt K) :
    Fintype M.Gal := sorry

instance (M : FiniteGalExt K) :
    TopologicalSpace M.Gal := ⊥
```

```
def AbsGal.ι :
    AbsGal K →* (M : FiniteGalExt K) → M.Gal :=
  sorry

lemma AbsGal.ι_injective :
    Function.Injective (AbsGal.ι K) :=
  sorry

lemma AbsGal.ι_inducing :
    Topology.IsInducing (AbsGal.ι K) :=
  sorry

lemma AbsGal.isClosed_range_ι :
    IsClosed (Set.range <| AbsGal.ι K) :=
  sorry
```

```
instance : CompactSpace (AbsGal K) := by
  constructor
  rw [(AbsGal.ι_inducing K).isCompact_iff]
  apply IsClosed.isCompact
  rw [Set.image_univ]
  apply AbsGal.isClosed_range_ι
```

Spec-driven development

Features:

- Non-blocking: For example, contributors can work with the assumption that `AbsGal K` is compact from the very beginning.
- Promotes good abstractions, since contributors *cannot rely* on implementation details early on.
- Allows for smoother refactors, as abstraction boundaries are more closely followed.
- Allows contributors to focus on targets within their knowledge and ability while still ensuring consistency throughout the project.

Spec-driven development

Proof assistants already eliminate much of the *accidental complexity* in a piece of mathematics (imprecise and unwritten assumptions, reference chasing, side conditions, . . .).

We propose that proof assistants can help to *tame* some of the *inherent* complexity in a piece of mathematics as well:

- Focusing on and creating abstraction boundaries.
- Distributing the complexity across many "subtargets".
- Relying on the proof assistant to maintain consistency.
- Reducing cognitive load as one can focus on local context.

Spec-driven development

Work is being done now to create tools that help facilitate this workflow.

What do we get out of formalized mathematics?

- (Essentially) absolute certainty that a piece of mathematics is correct.
- Large libraries of digitized mathematics, consistent across subfields.
- New methods for collaboration and exposition, at scale.
- New ways to distribute mathematical work, and tame complexity.

More generally, proof assistants encode mathematical artifacts in a *digital form* which can be:

- Stored
- Distributed
- Generated
- Processed

I think this paradigm fundamentally changes the relationship between a *mathematician* and the *mathematical objects* themselves.

It makes mathematical objects *tangible*, yet still a product of a mathematician's *creativity*.

Before we have seen how it can facilitate mathematical collaboration at scale. But it goes further.

Namely, the *digitization* of mathematical artifacts opens up completely new opportunities to obtain mathematical insight by processing and studying such objects *at scale*.

Thank you for listening!

The Lean community welcomes all interested participants, especially mathematicians, regardless of their experience with the language itself or other formalization experience.

If you find any of this intriguing, then please visit the community webpage. We have many tutorials (NNG) that helps mathematicians get started with the Lean language itself. Our zulip server is very active and welcoming to new members.