Overview on Formalization

George McNinch

2024-03-25

Proof assistants

Foundations of mathematics in proof assistants

Quoting from an answer by Jason Rute on the proofassistants.stackexchange

> Foundations of mathematics can mean a lot of things to a lot of people. Proof assistants present a sort of practical foundations of mathematics, where one actually comes up with a usable set of rules and axioms for doing mathematics formally in practice. Mizar and Metamath (set.mm) use a set theory foundation which is more similar to how foundations is taught in a typical university logic course. HOL-Light, HOL4, and Isabelle/HOL use higher order logic, which is a simpler type theory. And as we already mentioned Lean, Coq, and Agda use dependent type theory.

Here *type theory* is a system of logic that can be used as the *foundations of mathematics*. The objects discussed are *terms*, and every term has a *type*.

A **dependent type** is a *type* whose definition depends on a value; see e.g. the wikipedia discussion of dependent type theory.

In simple type theory, for example, there is a type `Nat`, or $\mathbb{N}$, of natural numbers. And there are *parametrized types*; e.g. `List Nat` is the type of *lists* of natural numbers.

So e.g. the notation

```
[1, 3, 5, 7] : List Nat
```

means that the indicated list `[1,3,5,7]` is a term of type `List Nat`.

Now we can give perhaps the most basic example of a *dependent type*: the type of vectors of a given fixed length; e.g. `Vect m Nat` is the type of vectors of natural numbers of length `m : Nat`

So for example

```
[1,3,5] : Vect 3 Nat
```

Observe that the type `Vect m Nat` *depends* on the term `m : Nat`.

Another example of a *dependent type* is the type of *equality* between natural numbers. For `n m : Nat`, there is a type `n = m`. This type has *no* terms if `n` and `m` are not the same natural number, and it has a unique term `rfl : n = m` (here `rfl` represents the "reflexive property").

This is an extremely quick summary; the point is that dependent types give one an ability to express mathematical statements and proofs, more-or-less as we already think about such expressions and proofs.

Resources:

- 'stack exchange' for proof assistants
- e.g. this post comparing proof assistants

List of Proof Assistants

- The Coq Proof Assistant
- Agda

- LEAN-prover community
- Isabelle/HOL

"audience"

There are a variety of users of proof assistants, mainly (I think) in two categories (the boundary between which can be blurry):

- theoretical computer science
- mathematics

The pure mathematical point of view I'd like to discuss is that taken by the Xena project.

There has been a good bit of energy aimed at formalizing the sort of mathematics that "pure mathematicians in mathematics departments" are familiar witth.

A substantial output from this activity is a library of formalized mathematics in Lean: mathlib

Some successes of proof assistants

- Four-color Theorem

  The four color theorem states that no more than four colors are required to color regions of a map so that two adjacent regions never have the same color.

  (from Wikipedia:)

  > "Kenneth Appel and Wolfgang Haken at the University of Illinois announced, on June 21, 1976,[16] that they had proved the theorem."

  Subsequently, there were some errors discovered and then corrected in this algorithmic proof.

  > "In 2005, Benjamin Werner and Georges Gonthier formalized a proof of the theorem inside the Coq proof assistant. This removed the need to trust the various computer programs used to verify particular cases; it is only necessary to trust the Coq kernel."

  Gonthier et al report
  Formal Proof—The Four- Color Theorem - Gonthier

  Quoting from the introduction to Gonthier's article in the *Notices*:

  > "Even Appel and Haken's 1976 triumph [2] had a hint of defeat: they'd had a computer do the proof for them! Perhaps the mathematical controversy around the proof died down with their book [3] and with the elegant 1995 revision [13] by Robertson, Saunders, Seymour, and Thomas. However something was still amiss: both proofs combined a textual argument, which could reasonably be checked by inspection, with computer code that could not. Worse, the empirical evidence provided by running code several times with the same input is weak, as it is blind to the most common cause of "computer" error: programmer error."
  >
  > …
  >
  > "For some thirty years, computer science has been working out a solution to this problem: formal program proofs. The idea is to write code that describes not only what the machine should do, but also why it should be doing it—a formal proof of correctness. The validity of the proof is an objective mathematical fact that can be checked by a different program, whose own validity can be ascertained empirically because it does run on many inputs.

- Feit-Thompson's odd-order Theorem odd-order theorem

  Wikipedia summary:

  > "William Burnside (1911, p. 503 note M) conjectured that every nonabelian finite simple group has even order. Richard Brauer (1957) suggested using the centralizers of involutions of simple groups as the basis for the classification of finite simple groups, as the Brauer–Fowler theorem shows that there are only a finite number of finite simple groups with given centralizer of an involution. A group of odd order has no involutions, so to

> carry out Brauer's program it is first necessary to show that non-cyclic finite simple groups never have odd order. This is equivalent to showing that odd order groups are solvable, which is what Feit and Thompson proved."

A group of authors – Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi and Laurent Théry – formalized this proof in the Coq proof assistant – see A Machine-Checked Proof of the Odd Order Theorem.

- Kepler conjecture

  From the Wikipedia summary:

  > The Kepler conjecture, named after the 17th-century mathematician and astronomer Johannes Kepler, is a mathematical theorem about sphere packing in three-dimensional Euclidean space. It states that no arrangement of equally sized spheres filling space has a greater average density than that of the cubic close packing (face-centered cubic) and hexagonal close packing arrangements. The density of these arrangements is around 74.05%.

  Proof by T. Hales:

  > In 1998, Thomas Hales, following an approach suggested by Fejes Tóth (1953), announced that he had a proof of the Kepler conjecture. Hales' proof is a proof by exhaustion involving the checking of many individual cases using complex computer calculations. Referees said that they were "99% certain" of the correctness of Hales' proof, and the Kepler conjecture was accepted as a theorem. In 2014, the Flyspeck project team, headed by Hales, announced the completion of a formal proof of the Kepler conjecture using a combination of the Isabelle and HOL Light proof assistants. In 2017, the formal proof was accepted by the journal Forum of Mathematics, Pi.[1]

  publication

- some modern algebraic geometry

  some "early work" formalizing parts of the Stacks Project (of Johan de Jong et al) can be found her

  https://github.com/kbuzzard/lean-stacks-project.

  The author of that project write (in the `README` in that repo):

  > This project is deprecated. It was joint work by me (Kevin Buzzard), Kenny Lau and Chris Hughes. It was our first attempt to do MSc level mathematics in Lean (back in 2017/18) and we made some poor design decisions due to inexperience. Once I had understood better how to write this project, I supervised Ramon Fernandez Mir's masters project which achieved everything this repo achieved and more too, and better. I would look there instead.

  Raymon Fernandez Mir - an undergraduate student of Kevin Buzzard formalized the notion - due to A. Grothendieck - of a *scheme* in algebraic geometry.

  From the introduction to his project:

  > The starting point of this project is the work done by Kevin Buzzard, Chris Hughes and Kenny Lau on formally verifying parts of The Stacks Project [1].
  >
  > …
  >
  > the objective of this project is to, starting from scratch, define a scheme in Lean.

  A summary can be found in this project report

  The code can be found in this repository

  perfectoid spaces

  Kevin Buzzard, Johan Commelin, and Patrick Massot formalized Peter Scholze's definition of **perfectoid spaces** (see Scholze's paper "Étale cohomology of diamonds")

  The code comprisingt he formalization is here: https://leanprover-community.github.io/lean-perfectoid-spaces/

  See also the blog post at the xenaproject: https://xenaproject.wordpress.com/2019/05/11/perfectoid-spaces/ and the page

There is a useful "how to read this document" page here: https://leanprover-community.github.io/lean-perfectoid-spaces/how-to-read-lean.html

- Terence Tao [2023-10-23] LEAN helped catch a error in some recent work
- going forward: Fermat's Last Theorem
  - Kevin Buzzard reports here that he has been awarded a grant by the EPSRC to formalize (parts of?) the proof of Fermat's Last Theorem in Lean.
  - (an earlier talk of Buzzard described some of the requirements of this task
- Buzzard's talk at 2022 ICM

Bibliography