Introduction

Formalizing the theorems and proofs of modern mathematics is important, because it can certify our knowledge and help us develop new proofs. Certificates of correctness for computer programs are examples of proofs, verification of code is an important application. I'd like to describe some recent advances in which many people are participating.

Modern mathematics has been based, in principle, on "set theory" since its axioms were stated in 1922. In that theory, there are two types of things: the sets and the propositions about sets.

A competing foundation for mathematics is “type theory”, under development since 1908, offers various advantages:

- the types of mathematical objects are more meaningful (e.g., 3, instead of being a set, is a natural number);
- it comes with a completely specified formal language for expressing mathematical statements and their proofs;
- writing a proof (or defining a function) is the same as coding the body of a subroutine (or method) and certifying that it computes the right thing
- programmers are accustomed to declaring the types of their variables (in languages such as C, ML, Haskell, etc.)

Giuseppe Peano, in 1889, formulated precise axioms for the natural numbers: that 0 is one, that each one has a "successor", and that during a proof, a variable x of type natural number can be dealt with by assuming it is either 0 or the successor of something. (Compare with `case` in Ruby.) These axioms can be expressed compactly in type theory.

Martin-Löf, in 1975, saw that the notion of equality ($x = y$) could be similarly formulated in type theory – a variable p representing a proof that $x = y$ (where y is also a variable) can be dealt with in code by assuming it is the trivial proof of $x = x$ (and that y is x).

Voevodsky, in 2004 and 2009, figured how to deal with possible inequality of two proofs of the same equality (!) in a sensible way, and introduced his *Univalence Axiom*. Beneficial outcomes:

- implementation details are hidden
- transportation of proofs from one situation to an equivalent situation (reusability of code)
- sensible definitions of “set” and “proposition” are given, merging logic with set theory seamlessly
- a new, more fundamental, mathematical world (of types that are not sets) is revealed

Some *Homotopy Type Theory* links:

- these slides: http://dangrayson.com/Lectures/
- the web site: http://homotopytypetheory.org/
- the blog: http://homotopytypetheory.org/blog/
- the book: http://homotopytypetheory.org/book/
- code (proofs) under development based on Voevodsky's foundations: https://github.com/UniMath/UniMath
- another development based on the same ideas: https://github.com/HoTT/HoTT
- the paper I'm working to check the proofs of: http://arxiv.org/abs/1310.8644
- a research grant: http://bit.ly/1o4Jae7
- CACM article: http://bit.ly/1mL8XXA