Project Description

Andreas Salhus Bakseter

January 23, 2023

1 Background

1.1 Formalization of mathematical problems

When solving mathematical problems, one often uses proofs to justify some claim. We can group proofs into two types; *informal* and *formal* proofs.

An informal proof is often written in a natural language, and the proof is adequate if most readers are convinced by the proof [1]. As a proof grows larger and more complex, it becomes harder to follow, which can ultimately lead to errors in the proof's reasoning. This might cause the whole proof to be incorrect [2].

A formal proof is written in a formal language, and can be compared to a computer program written in a programming language. Writing a formal proof is more difficult than writing an informal proof.

1.2 Proof assistants

Using a *proof assistant*, we can verify a formal proof mechanically. *Coq* is an example of a proof assistant. Coq uses type theory to formulate and verify proofs, but can also be used as a functional programming language [3]. Other examples of proof assistants include Agda, Isabelle, Lean and HOL.

1.3 Type theory & propositions as types

Type theory groups mathematical objects with similar properties together by assigning them a "type". Similarily to data types in computer programming, we can use types to represent mathematical objects. For example, we can use the data type `nat` to represent natural numbers.

The concept of propositions as types sees the proving of a mathematical proposition as the same process as constructing a value of that type. For example, to prove a proposition P which states "all integers are the sum of four squares", we must construct a value of the type P that shows that this is true for all integers. Proofs are mathematical objects; thus a proposition can be viewed as having the type of all its proofs (if any!). We can use this correspondance to model a proof as a typed computer program. The power of this concept comes from the fact that we can use a type checker to verify that our program is typed correctly, and thus that the corresponding proof is valid.

Coq also enables us to extract and execute programs from our proofs, once they have been verified.

2 The project

We will use the Coq proof assistant to formalize parts of the proofs of the following paper, Bezem and Coquand [4]. This paper solves two problems that occur in dependent type systems where typings depend on universe-level constraints. Since this proof is complex enough that mistakes are possible it is a good candidate for formalization. We can also use this process to gain further insight into the algorithm that lies behind the proof. It might also be interesting to use Coq to extract programs from the final proofs.

References

[1] Benjamin C. Pierce. *Software Foundations: Volume 1: Logical Foundations*. 2022. URL: https://softwarefoundations.cis.upenn.edu/current/lf-current/index.html (visited on 01/17/2022).

[2] Roxanne Khamsi. *Mathematical proofs are getting harder to verify*. 2006. URL: https://www.newscientist.com/article/dn8743-mathematical-proofs-getting-harder-to-verify (visited on 01/18/2022).

[3] The Coq Team. *A short introduction to Coq*. URL: https://coq.inria.fr/a-short-introduction-to-coq (visited on 01/18/2022).

[4] Marc Bezem and Thierry Coquand. "Loop-checking and the uniform word problem for join-semilattices with an inflationary endomorphism". In: *Theoretical Computer Science* (2022). ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2022.01.017. URL: https://www.sciencedirect.com/science/article/pii/S0304397522000317.