Formal Languages for Mathematics

Jonas Betzendahl
Dept. of Computer Science
FAU Erlangen-Nürnberg, Germany
jonas.betzendahl@fau.de

Motivation

The main focus of my PhD thesis will be formal languages for mathematics, a topic that has gathered a lot of attention in recent years. By now, there are multiple popular formalisms for mathematics, but most fail in capturing the flexible and often informal nature of chalk-and-whiteboard mathematics. Usually, they are also either flexible *or* have good tool support, not both.

Type systems are ubiquitous in modern efforts to formalise mathematics, since they allow for helpful support by the underlying system via type-checking and -inference. So far, however, most of the systems used in the field lack some key features that are often used in practise (e.g. subtyping) because these are undecidable in the general case.

A focus on decidable type systems is understandable, since these guarantee a certain kind of productivity and determinism, yet the fact of the matter remains that many computationally undecidable features are in fact used by mathematicians "in the wild". This means that to formalise mathematics in a system with decidable types, one often has to fall back to encodings that are difficult to handle or otherwise awkward.

Since they're common in everyday mathematics, I would also like to facilitate the use of partial functions and definite descriptions. This necessitates a rigorous and unified treatment of (un)definedness by the system, something that is found in only a few of the attempts at formalising mathematics popular today (see also the Related Work section).

So, we would want a strictly formal system to provide a central formalisation ontology that is still flexible enough to capture real-world mathematics, and yet with enough computational support to take some of the more tedious tasks of the developers' hands and allow access to knowledge management services.

Ideally, we would also like to import formalisations from other systems and the ability to translate back into those systems to allow for communication between systems that doesn't need n^2 ad-hoc translations between systems (see also the ideas behind the *Math In The Middle* ontology).

This development would continue the efforts of the OAF (Open Archive of Formalisations) and ODK (OpenDreamKit) projects (see also Figure 1) that the KWARC group has been involved with for some time now.

Related Work

First of all, there are the *logical frameworks*, like LF and Isabelle. These are usually either specialised for meta-level reasoning (like proving consistency of a system developed in the framework) or concentrate on object-level reasoning, sometimes even mostly within one or two fixed systems. Tool support for reasoning with logical frameworks is also often thin.

There is also the class of *proof assistants*. Systems like, Coq, Agda, and HOL Light are widely in use for formalising mathematics, but restrict themselves to decidable type theories.

In: O. Hasan, D. Gallois-Wong (eds.): Proceedings of CICM 2018, Hagenberg, Austria, 13-08-2018, published at http://ceur-ws.org

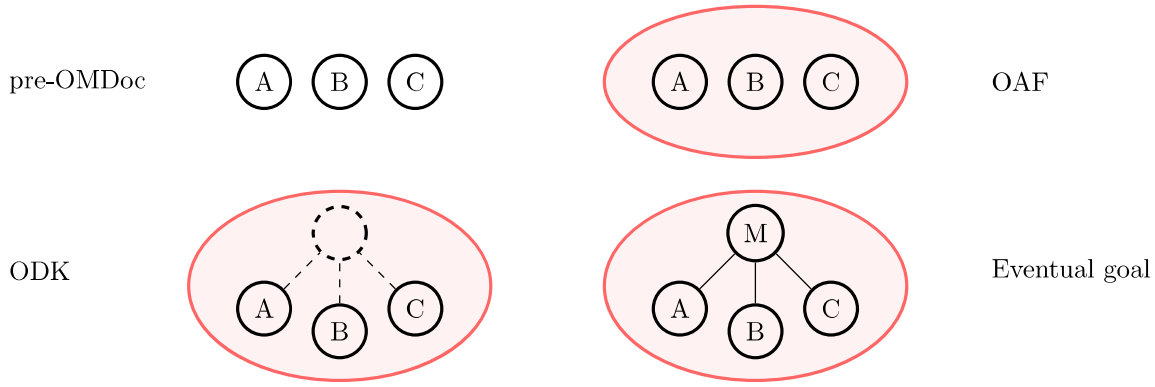

Figure 1: Development towards a unifying ontology for formalisation (M). A, B and C are different, pre-existing mathematical systems and the red background corresponds to formalisation in OMDoc.

The IMPS system already features subsorting and undefinedness, but has fallen out of use and is not set up to interact with other systems.

Fully automated theorem provers (like E or Vampire) don't meet or expectations either because they usually focus on first order logic, which is an important part of mathematics, but not general enough. Our system would need to support higher order reasoning as well, if we intend to capture the nature of chalk-and-whiteboard mathematics accurately.

Lastly, there are multiple *computation systems* (e.g. GAP, Sage, and Mathematica) that are built (often by mathematicians) for a very specific purpose *other* than formalising the wider mathematical landscape. They also are often not well equipped to deal with undefined values. As a result, these systems are not suitable to use for our goal, but since they are a useful resource and sometimes encapsulate a smaller domain extremely well, they ought to be considered as inputs.

Effort & Evaluation

Catching up to the state of the art (i.e. developing a full reasoning system that has all the features we discussed so far) in one PhD-thesis is not feasible, so I will focus particularly on proof obligations that pop up during the process of type-checking and the best ways to discard them automatically.

As a first step I plan on developing a softly typed set theory (i.e. a form of untyped set theory where types are added after the fact via unary predicates). This would offer a certain flexibility, since one is not strictly bound to a type system that is fundamentally baked into the system. This theory will hopefully lead to some insights on the effectiveness of such a system for formalisations

The main challenges on this would be to keep the system manageable to the user and find a good way to handle undefined values within the system without falling back to exceptions or errors.

MMT as of today has support for annotating declarations so that the simplifier is able to use them as additional rules at runtime. Even a lot of basic notions are not be well-typed in LF itself, but they are in the quotient of LF over some previously introduced rules.

However, MMT does not support such annotations for rules that would have *premises*, only simple equalities. So one of the tasks would be to implement support for these as well.

A major hassle with such rules is that since they have premises, they generate a lot of minor proof obligations (judgements for equality, typing, subtyping, definedness, ...). These take a lot of time to prove by hand, yet they don't contribute substantially to the idea behind the structure or the proof that is being constructed. So it would be desirable to have some algorithms that are efficient and specialised to discharge these automatically wherever possible. This is another task that I will take a closer look at, especially for the type of premises mentioned above).

I will also survey other systems for formalising mathematics (like those mentioned earlier) about their treatments of undefined values and their ways of automatically dealing with tiny proof obligations, especially in the context of type checking.