Reasoning Support for Undefinedness and Soft Typing in Formal Mathematics

Jonas Betzendahl
jonas.betzendahl@fau.de

FAU Erlangen-Nürnberg

Motivation

One aspect of mathematics that is only beginning to be incorporated into formal systems is undefinedness, the notion of terms or expressions that are syntactically well-formed but cannot be assigned a value. This typically arises when applying a *partial* function (or predicate, or operator, ...) to a value that lies outside of its domain. This phenomenon occurs naturally, both in mathematical expressions (e.g. $\frac{5}{0}$) as well as to natural language (e.g. "Fantasia lies east of the equator").

Equation (1) is an especially interesting example because it clearly demonstrates the split opinions about how to deal with undefinedness.

$$\forall x, y, z : \mathbb{R} \,.\, x = \frac{z}{y} \Rightarrow x \cdot y = z \tag{1}$$

Earlier contributions in the literature reference this example, but with contradictory connotations. One paper posits that (1) should be considered true, *even without the restriction that* $y \neq 0$; another emphasises that such a restriction *must* be present. Both pieces refer to mathematical consensus in one fashion or another to justify their respective approach. This demonstrates not only that there is no mathematical consensus about when exactly equations like the one above should be treated as defined, or that such a consensus is not as obvious as previously thought, but that the situation is actually even worse than that. There is no consensus, yet people believe there is.

We learn from this and other examples in the literature that there exists a multitude of different approaches to undefinedness, each with advantages and conducive to certain situations. Hence, our goal shall not be to find the next "one-to-rule-them-all" method of dealing with undefinedness, but to provide the possibility of easily developing and incorporating into an ongoing formalisation project, whatever notion of undefinedness is applicable to the task at hand. For this, we will be using the MMT framework.

Another trend in mathematical formalisms is that many of them make use of *hard* types, as opposed to *soft* ones. Hard type systems dominate the current landscape of formal systems since they lend themselves the easiest to automation (since e.g. type checking is not reduced to undecidable theorem proving, necessitating complex and potentially hard to predict heuristics).

However, in hard and even in semi-soft type systems, the user is usually forced to make the somewhat artificial choice of whether to encode a given piece of information in the type system or the logical system (compare $\forall x : \mathbb{N} \,.\, P(x)$ and $\forall x \,.\, x \in \mathbb{N} \Rightarrow P(x)$). In untyped systems that mimic types through predicates, the type system becomes a feature of the logical system.

Softer type systems are furthermore interesting because they naturally mirror the way many mathematicians work in their everyday efforts. Most of the time, not all information about any given object is known from the start and it is a mathematical *discovery* (subject to proof) that a certain object also fits the definition of another class of objects.

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

Related Work

Given the plethora of different approaches and opinions on undefinedness, it is no surprise that systems for doing mathematics reflect a similar diversity of thought. There have been multiple efforts, both in theory and in implementation, to come up with reasoning systems that support some sense of mathematical undefinedness or another. Our own investigation started with `IMPS` (which shares certain design decisions with MMT and is therefore an obvious choice). Other notable systems are Robin Millner et al.'s LCF (*Logic for Computable Functions*, based on Dana Scott's *Logic of Computable functions*) and the `mural` System by Jones et al. which is based on the three-valued LPF.

In the realm of formal systems for mathematics, soft type systems are not as abundant as hard ones `Mizar`'s type system is of particular interest since it is maybe the most prominent example, has been in use since before the feature of type classes was introduced in other formal systems and also because there is a large library of `Mizar` formalisation in the `Mizar` mathematical library. Furthermore, the Naproche-SAD system's approach to types can also be interpreted as a soft type system and the type system of the computer algebra system GAP also has soft aspects that are interesting in this context.

Effort & Evaluation

The most prominent goal of my PhD research remains to allow for native (un)definedness reasoning within MMT, ideally both on the manual and the automated reasoning level.

Currently, neither MMT nor **LF** have a way of dealing with undefinedness, so a key research question here is to find out at which level this feature should be introduced. Can we give MMT a primitive, yet foundation-independent notion of undefinedness that all logics and systems that are implemented in MMT then could benefit from on an opt-in basis? If so, any system that was implemented without support for undefinedness should be able to simply "import" a suited notion of undefinedness instead of having to re-implement all the machinery again, then and there.

The other topic I hope to tackle that of a soft type system for mathematics. Like undefinedness, soft type systems have the potential to capture realities of mathematics as performed by mathematicians often overlooked in formal systems. An object is shown to be an instance of one structure but can later be proven to also meet the criteria of another structure. Both of these angles (and any that follow) can be used in proofs and for further definitions.

Both prominent features are likely to introduce a significant number of proof obligations that are not necessarily hard to prove, but do need to be proved nevertheless (think "Type τ is not empty." or "The application of this function to this value is defined."). Requiring the user to prove them herself would place her under an undue burden of "busywork", likely leading to justified frustration. So, ideally, these obligations would be proved by the system.

All of the examples mentioned above are of course instances of larger problems that become intractable (or even unsolvable!) in larger contexts. Yet in many circumstances, the actual incarnations are solvable in a reasonable timespan often enough.

MMT already has a simple (yet still foundation-independent) iterative theorem proving system, yet it does not meet standards of efficacy and efficiency. I am therefore working on extending the MMT reasoning system by another automated component. I am using λProlog (or, to be more precise, ELPI) as the programming language for the proving component. This approach makes use of the fact that higher-order unification is already reliably implemented and can be used as a mechanism for proof search.

The pipeline looks as follows: For any given theory, MMT generates an ELPI *kernel*, a file that introduces the declarations of a theory (in the case of propositional logic, these would include constants for truth and falsehood, the connectives, as well as types for propositional formulae and proof certificates) and ELPI rules that reflect the inference rules of the theory (e.g. $\wedge_I$ or $\neg_E$). These are supported by *experts*, i.e. handwritten or partially-generated ELPI modules that apply strategies and/or apply help predicates to reduce the enormous search space of proof terms that would otherwise be intractable. First results in testing this approach appear promising.

Proof obligations that arise in MMT are automatically translated and handed to ELPI (running in server mode). The fact that MMT is calling an external system for proving introduces an issue of trust. How can it rely on the proof found in ELPI actually being correct? For this, we are working on including *proof certificates* (on which Dale Miller also has done extensive work during the *ProofCert* project), that could be generated by ELPI and communicated back to MMT to be checked for correctness.