Towards a library of formalised undecidable problems in Coq: The undecidability of intuitionistic linear logic

YANNICK FORSTER, Saarland University, forster@ps.uni-saarland.de
DOMINIQUE LARCHEY-WENDLING, LORIA – CNRS, dominique.larchey-wendling@loria.fr

The undecidability of low-level problems like the halting problem is usually shown by deriving a logical contradiction from the assumption that the problem is decidable. For even slightly more advanced problems, such a direct approach becomes unfeasible and proofs are instead done by giving a (many-one) reduction from another undecidable problem to the problem to be shown undecidable. A proof by reduction in general amounts to defining the reduction function, showing that it is correct as a reduction and giving an argument for its computability. Most of these proofs rely on subtle details and have to be checked carefully. They could thus be a prime example for the use of interactive theorem provers to assist ongoing research. However, formalisations of undecidability proofs are yet to be found in the literature. There are two main obstacles in our eyes: first, proofs on paper mostly omit the invariants needed for the full verification of the reduction and rather rely on an intuitive justification of the correctness of the reduction. Second, they omit the actual computability proof, which would amount to the full formal verification of a program in the chosen model of computation implementing the potentially complex reduction. Both these gaps have to be closed for the formalisation of reductions in a proof assistant.

We propose a talk on our ongoing efforts to provide a library of undecidable problems in the Coq proof assistant. Coq is an implementation of constructive type theory, where, in contrast to classical proof assistants, every definable function is computable (assuming no extra axiom is added to Coq). Thus, without referring to an explicit model of computation, decidability proofs can be given by writing and verifying a function in the programming language underlying Coq. This well-known idea can be adapted to a notion of undecidability, using e.g. the halting problem as a seed of undecidability. We define a problem P to be undecidable if the halting problem for Turing machines is reducible to P by a Coq function. Notice that using this restricted notion, inside Coq, undecidability is not provably equivalent to the negation of decidability. The notion can however still be used to clear any doubts for undecidability proofs, because the only two non-formalised facts are that there is indeed no Turing machine deciding the halting problem and that indeed every Coq function is computable by a Turing machine.

Since most undecidability proofs in the literature do not directly reduce from the halting problem of Turing machines, but start from another either low- or high-level undecidable problem, we are working on a library of such problems in Coq, extending the work in [1]. Our library now covers the following low-level problems:

- Reachability and nullability in string rewriting systems.
- The Post correspondence problem using arbitrary (PCP) or binary symbols (BPCP).
- Emptyness of intersection and containment of palindromes for context-free grammars.
- The halting problem for binary stack machines and Minsky machines.

As an example for the use of the library, we report on our formalisation of the higher-level undecidability of provability in intuitionistic linear logic, following [4]. We start the explanation at PCP, which is reduced to intuitionistic linear logic via binary stack machines, Minsky machines, and elementary linear logic.

The proof involves many challenges, all exemplary for the formalisation of reductions. The reductions from Turing machines to string rewriting and from string rewriting to PCP are already published in [1] and require the formulation of intricate inductive invariants, which are usually left out in the literature. To reduce BPCP to binary stack machines, a relatively low level program has to be verified, performing a structured search for BPCP matches. The reduction from binary stack machines to Minsky machines is a typical compiler verification between low-level languages, and the verification of the reduction to elementary linear logic is an elegant example of how to encode computational models into logics.

We want to extend the library to include more computational models like the call-by-value λ-calculus from [3], recursive functions and combinatory logic, other tiling problems, and other problems related to logics, for instance provability in first-order logic. We also want to connect the notions of decidability and undecidability by proving the equivalence of the development of formalised computability theory in [3] with Turing machines. Giving explicit computability proofs for reduction functions then is routine using the framework from [2].

REFERENCES

[1] Yannick Forster, Edith Heiter, and Gert Smolka. Verification of PCP-related computational reductions in Coq. *arXiv preprint arXiv:1711.07023*, 2017. Accepted at ITP 2018.
[2] Yannick Forster and Fabian Kunze. Verified extraction from Coq to a lambda-calculus. *Coq Workshop 2016*, 2016.
[3] Yannick Forster and Gert Smolka. Weak call-by-value lambda calculus as a model of computation in Coq. In *ITP 2017*, pages 189–206. Springer, 2017.
[4] Dominique Larchey-Wendling and Didier Galmiche. The undecidability of boolean BI through phase semantics. In *LICS 2010*, pages 140–149. IEEE, 2010.