A Coq Library of Undecidable Problems

Yannick Forster  
Saarland University

Dominique Larchey-Wendling  
Université de Lorraine, CNRS, LORIA

Andrej Dudenhefner  
Saarland University

Edith Heiter  
U. van Amsterdam

Dominik Kirst  
Saarland University

Fabian Kunze  
Saarland University

Gert Smolka  
Saarland University

Simon Spies  
Saarland University

Dominik Wehr  
Saarland University

Maximilian Wuttke  
Saarland University

Abstract

We propose a talk on our library of mechanised reductions to establish undecidability results in Coq. The library is a collaborative effort, growing constantly and we are seeking more outside contributors willing to work on undecidability results in Coq.

1 Introduction

Undecidability proofs are usually carried out by giving a reduction from an undecidable problem to the problem to be shown undecidable. It involves the definition of a reduction function, proving that it is both computable and correct (as a reduction). We base our library on a synthetic approach to undecidability [7], meaning we rely on Coq's built-in notion of computation. Since every function on data types (such as $\lambda$) definable in (axiom free) Coq is computable in any standard model of computation, the computability requirement is automatically fulfilled, and giving a reduction only amounts to defining a Coq term and proving its correctness.

Most undecidability proofs work by many-one reductions. A problem $p : X \to Prop$ is many-one reducible to $q : Y \to Prop$, written $p \leq q$, if $\forall x \in X, p\ x \leftrightarrow q(f\ x)$ holds for some $f : X \to Y$, the correctness statement meaning that $f$ embeds $p$-validity into $q$-validity. Many-one reductions form a sub-class of Turing reductions: a problem $p$ is Turing-reducible to a problem $q$ if a decision for $q$ can be turned into a decision for $p$, i.e. if the type $dec\ q \to dec\ p$ is inhabited, where $dec\ p := \forall x, (p\ x) + (\neg p\ x)$.

In total, our library contains more than 20 different problems and about 70,000 lines of code. In this abstract, we give an overview of the problems in the library, the reductions between the problems, and sum up possible future work.

2 Problems in the Library

The problems in our library can mostly be categorized into seed problems, advanced problems, and target problems.

As they are simple to state, seeds make for good starting points leading to smooth reductions to other problems.

Advanced problems do not work well as seeds, but they highlight the potential of our library as a framework for mechanically checking pen&paper proofs of potentially hard undecidability results.

Target problems are very expressive and thus work well as targets for reduction, with the aim of closing loops in the reduction graph (Figure 1) to establish the inter-reducibility of problems.

2.1 Seeds

- The Post correspondence problem (PCP), which is a matching problem on strings, mechanised in [6]. PCP works well as seed for target problems that can express string concatenation and simple inductive predicates. In particular, PCP on Boolean strings (BPCP) is simpler than most halting problems, and thus is often-times used as the starting point of reductions.
- Halting problems for (n or just two) registers machines (MM, MM0 and MM2), also known as Minsky machines, mechanised in [10]. These machines have a (fixed) number of $\mathbb{N}$-valued registers and can increase/decrement registers or perform conditional jumps. The machines work well as seed for target problems based on simple arithmetic operations e.g. +1 or −1.
- The halting problem for the FRACTRAN language, which is a very simple language where states are natural numbers and programs are lists of fractions, mechanised in [15]. FRACTRAN works well as seed for target problems which can simulate more involved arithmetic, in this case multiplication.
- Satisfiability of Diophantine equations (H10) or elementary Diophantine constraints (H10c), which are equivalent formulations of Hilbert’s tenth problem, mechanised in [15].

2.2 Advanced Problems

- The halting problems for single-tape (TM1) and multi-tape Turing machines (TMm), mechanised in [9]. Turing machines are the standard model of computation, but their formal definition is quite involved compared to e.g. PCP.
- The halting problem for binary stack machines (BSM), which are simple machines working with stacks of Booleans, mechanised in [10]. They can push and pop Booleans, and conditionally jump.
- Standard first-order string rewriting (SR), mechanised in [6].
- Entailment in (elementary and standard) intuitionistic linear logic (EILL and ILL), mechanised in [10].
- Provability and satisfiability for intuitionistic and classical first-order logic (FOL), mechanised in [7].
- Type inhabitation for System F ($\Gamma \vdash \Pi : A$), described in [5].
- Finite satisfiability in (classical) first-order logic, known as Trakhtenbrot’s theorem (Trakht.)
- The termination problem for $\mu$-recursive algorithms ($\mu$-rec), a standard class of formal algorithms, mechanised in [14].
- The intersection problem (CFL) and palindrome problem (CFP) for context-free grammars, mechanised in [6].
- 2nd-order (2oUnif), 3rd-order (3oUnif) and thus higher-order unification in the simply-typed $\lambda$-calculus, mechanised in [20].
2.3 Target Problems

- The halting problem for the weak call-by-value $\lambda$-calculus $L$ (wCBV), mechanised in [11]. In order to reduce a problem $p$ to $L$-halting, one first shows that the problem is enumerable in Coq, i.e. that there is a term $f : \mathbb{N} \rightarrow list X s.t. \forall x, p x \leftrightarrow \exists n, x \in f n$. As a second step, one can use the automatic translation of Coq terms to $L$ from [8] to immediately obtain a reduction from $p$ to $L$-halting. This technique is used to obtain a compact reduction from provability in first-order logic to $L$-halting in [6].

- Provability or satisfiability in first-order logic, mechanised in [7].

If a problem is expressible with a first-order formula, this immediately serves as reduction to provability on first-order logic.

3 Future Work

First, we would like to strengthen the theoretical basis our work is built on, i.e. the folklore fact that every function of type $\mathbb{N} \rightarrow \mathbb{N}$ definable in (axiom free) Coq can be shown computable in a model of computation. A mechanised proof of this could for instance be carried out based on MetaCoq [19]. We also want to analyse which classical axioms are compatible with this computability assumption.

Secondly, we would like to include existing mechanisations of undecidability results in a uniform fashion, for instance [3] or [4].

Thirdly, there are many undecidability proofs we would like to mechanise in the future. Several known undecidability results build on the undecidability of unification problems we already include (2oUnif or 3oUnif), such as typability in the $\lambda$-calculus $L$.

Lastly, the undecidability proof of semi-unification [13] is by reduction from the mortality problem of Turing machines. It would be a challenge to find a different starting point for this proof to make it suitable for mechanisation in Coq.