A Coq Library of Undecidable Problems

Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, Maximilian Wuttke

CoqPL 2020
January 25
Overview

1. Synthetic Undecidability in Coq
2. Overview over the Library
3. Development Issues
4. Future Work
A problem $P : X \rightarrow \mathbb{P}$ is decidable if there is $f : X \rightarrow \mathbb{B}$ s.t. $\forall x. \; Px \leftrightarrow fx = \text{true}$.

Q.: Why is this definition okay?

A.: Because Coq is a programming language and every definable function $f : X \rightarrow \mathbb{B}$ is computable in a model of computation, provided $X$ is a datatype like $\mathbb{N}$, $\mathbb{N} \times \mathbb{B}$, or $\text{list}(\mathbb{N}) \times \text{list}(\text{option}\mathbb{B})$. 
A problem $P : X \to \mathbb{P}$ is undecidable if $\text{dec } P \to \bot$
does not work, because you can consistently assume a decider for every $P$

Axiom halting_dec : dec Halt.

where $Halt$ is the halting problem of Turing machines is

1. consistent (because the set model validates it)
2. not provable (because all definable functions are computable)
A problem $P : X \to \mathbb{P}$ is undecidable if $\text{dec } P \to \text{dec } Halt$.

$Q \preceq_T P := \text{dec } P \to \text{dec } Q$

**Lemma**

If $Q \preceq_T P$ and $Q$ is undecidable, then $P$ is undecidable.

$Q \preceq P := \exists f. \forall x. Qx \leftrightarrow P(fx)$

**Lemma**

If $Q \preceq P$ then $Q \preceq_T P$. 
Traditional Models of Computation

Q: How to prove “There is no Turing machine $\lambda$-term computing $P$”?

A: Show that the many-one reductions are computable.

Coq function $f : \mathbb{N} \to \text{list } \mathbb{N}$

$\downarrow$

$\lambda$-term $t$ with $t \bar{n} \triangleright \bar{f}n$.

as a Coq plugin based on MetaCoq

$\Rightarrow$ Many-one reductions can be automatically translated to a “real” model of computation yielding a proof of $\exists t : \text{term. } t \text{ computes } P$.
Undecidability proofs should be mechanisable

We need a library of potential starting points for proofs by reduction
A Coq Library of Undecidable Problems

CoqPL 2020 – January 25
Seeds

TM \longrightarrow \lambda \longrightarrow \mu\text{-rec} \rightarrow H10

FOL \longrightarrow CFG \longrightarrow Sys. F \longrightarrow HOU \longrightarrow ILL

PCP \longrightarrow SR \longrightarrow bin. stack machines \longrightarrow register machines

FRACTRAN
Advanced problems

- TM
- FOL
- CFG
- Sys. F
- HOU
- H10
- PCP
- SR
- λ
- μ-rec
- A Coq Library of Undecidable Problems
- bin. stack machines
- register machines
Seed 1: PCP

Base type: \( \text{list}(\text{list } \mathbb{B} \times \text{list } \mathbb{B}) \)

Definition: \( \text{PCP}(L) := \exists x : \text{list } \mathbb{B}. \ L \triangleright (x, x) \)

\[
\begin{align*}
(u, v) \in L & \quad & \text{L} \triangleright (x, y) & \quad & (u, v) \in L \\
\hline
\text{L} \triangleright (u, v) & & \text{L} \triangleright (x \oplus u, y \oplus v)
\end{align*}
\]

Good seed for target problems that can express string concatenation and simple inductive predicates.
Seed 2: Diophantine constraints

Base type: `list C` where `C ::= x+ y = z  |  x\times y = z  |  x = 1`.

Definition: `H10(C) ::= \exists \delta : \text{var} \rightarrow \mathbb{N}. \delta \models C` where

\[
\begin{align*}
\delta \models x + y = z & := \delta x + \delta y = \delta z \\
\delta \models x \times y = z & := \delta x \cdot \delta y = \delta z \\
\delta \models x = n & := \delta x = 1 \\
\delta \models C & := \forall c \in C. \delta \models c
\end{align*}
\]

**Good seed for target problems that can express addition and multiplication.**
Seed 3: FRACTRAN

Base type: $\mathbb{N} \times \text{list}(\mathbb{N} \times \mathbb{N})$

Definition: $\text{FRACTRAN}(x, P) := x$ is terminating under $P \vdash \_ \succ \_\\\\
q \cdot y = p \cdot x \\
\frac{(p, q) :: P \vdash x \succ y}{(p, q) :: P \vdash x \succ y} \\
q \not| p \cdot x \\
\frac{P \vdash x \succ y}{(p, q) :: P \vdash x \succ y}$

Good seed for target problems that can express multiplication and reflexive transitive closure.
Target problems

Diagram:

- **TM**
- **SR**
- **PCP**
- **FOL**
- **CFG**
- **SyS. F**
- **HOU**
- **H10**
- **ILL**
- **FRACTRAN**

- **λ**
- **µ-rec**

- bin. stack machines
- register machines
First-order logic

To reduce a problem to first-order provability, show that its definition is expressible as a first-order formula.

\[ P(x) \iff \Gamma \vdash \varphi_x \]

Usual proof strategy:

→ By induction on the definition of \( P \).

← By defining the standard model for \( P \) and soundness.
The cbv \(\lambda\)-calculus \(L\)

**Definition**

A problem \(P : X \rightarrow \mathbb{P}\) is \(L\)-enumerable if there is an \(L\)-computable function \(f : \mathbb{N} \rightarrow \text{option} X\) s.t. \(P x \leftrightarrow \exists n.\ fn = \lfloor x \rfloor\).

**Theorem**

A problem \(P : X \rightarrow \mathbb{P}\) reduces to the \(L\)-halting problem if it is \(L\)-enumerable and its base type has an equality decider in \(L\).

Strategy to reduce \(P\) to \(L\)-halting:

1. Give enumerating function \(f : \mathbb{N} \rightarrow \text{option} X\) (purely in Coq)
2. Give equality decider \(X \rightarrow X \rightarrow \mathbb{B}\) (purely in Coq)
3. Give encoding for \(X\) in \(L\) (mostly automatic)
4. Use extraction to \(L\) from ITP ’19 (fully automatic)
Verified Programming of Turing Machines in Coq

Yannick Forster
Saarland University
Saarbrücken, Germany
forster@ps.uni-saarland.de

Fabian Kunze
Saarland University
Saarbrücken, Germany
kunze@ps.uni-saarland.de

Maximilian Wuttke
Saarland University
Saarbrücken, Germany
wuttke@stud.uni-saarland.de

Abstract

We present a framework for the verified programming of multi-tape Turing machines in Coq. Improving on prior work by Asperti and Riccotti in Matita, we implement multiple layers of abstraction. The highest layer allows a user to implement nontrivial algorithms as Turing machines and verify their correctness, as well as their space and time complexity compositionally. The user can so do without ever mentioning states, symbols on tapes or transition functions. They write programs in an imperative language with registers containing values of encodable data types, and our framework constructs corresponding Turing machines.

As case studies, we verify a translation from multi-tape to single-tape machines as well as a universal Turing machine, both with polynomial time overhead and constant factor slowdown.

A Simpler Undecidability Proof for System F Inhabitation

Andrej Dudenhefner
Technical University of Dortmund, Dortmund, Germany
andrey.dudenhefner@cs.tu-dortmund.de

Jakob Rehof
Technical University of Dortmund, Dortmund, Germany
jacob.rehof@cs.tu-dortmund.de

Abstract

Provable decidability of intuitionistic linear logic via binary stack machines is known. We present a more direct, Coq-verified proof of the undecidability of intuitionistic linear logic with respect to inhabited formulas by unification.

The proof uses lemmas from Goldfarb's proof of undecidability of second-order unification in the simply-typed lambda-calculus. Using ideas from Goldfarb's proof of unification undecidability and Huet's original proof of undecidability of second-order unification, we provide a direct proof of the undecidability of inhabited formulas in System F.

The proof makes use of interactive theorem provers to assist in formalising the proof. We believe that our formalisation is a valuable resource for the formal verification community.
Development Issues

In total: 75.000 LOC

- Development on Github, Travis CI helps a lot
- Still figuring out best practices to target multiple Coq versions
  1. different branches?
  2. configure.sh files?
  3. only target most up-to-date Coq version?
- Some parts of the library depend on Equations or MetaCoq. Should the whole library depend now? What about Windows users?
- Different used standard libraries lead to incompatibility
- Different trust in axioms
Future Work

Include more problems

- Typability in $\lambda\Pi$ or type inference in System $F_\omega$
- Intersection problem of two-way automata
- Wang tiling problems, Post’s tag systems
- Subtyping in $F_{\leq}$, typability and type checking in System $F$
- ZF entailment, Trakhtenbrot’s theorem
- Semi unification

Work out foundations:

- What is a realizability model for Coq?
- Which axioms are compatible? Func. ext.? Prop. ext.? PI? EM?
Join us!

github.com/uds-psl/coq-library-undecidability/

Questions?