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

SAARLAND UNIVERSITY COMPUTER SCIENCE

UNIVERSITY OF CAMBRIDGE

Loria

cnrs

UNIVERSITEIT VAN AMSTERDAM


Yannick Forster


Dominique Larchey-Wendling


Andrej Dudenhefner

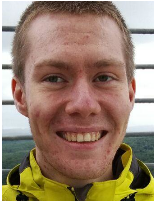

Edith Heiter


Dominik Kirst


Fabian Kunze


Gert Smolka


Simon Spies


Dominik Wehr


Maximilian Wuttke

Overview

1. Synthetic Undecidability in Coq
2. Overview over the Library
3. Development Issues
4. Future Work

Synthetic Decidability

A problem $P : X \to \mathbb{P}$ is decidable if there is $f : X \to \mathbb{B}$ s.t. $\forall x.\ Px \leftrightarrow fx = \texttt{true}$.

Q.: Why is this definition okay?

A.: Because Coq is a programming language and every definable function $f : X \to \mathbb{B}$ is computable in a model of computation, provided X is a datatype like $\mathbb{N}$, $\mathbb{N} \times \mathbb{B}$, or $\texttt{list}\,(\mathbb{N}) \times \texttt{list}\,(\texttt{option}\,\mathbb{B})$.

Synthetic Undecidability

A problem $P : X \to \mathbb{P}$ is **un**decidable if
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)

Synthetic Turing and many-one reductions

A problem $P : X \to \mathbb{P}$ is undecidable if
$\texttt{dec}\, P \to \texttt{dec}\, \mathit{Halt}$.

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

Lemma

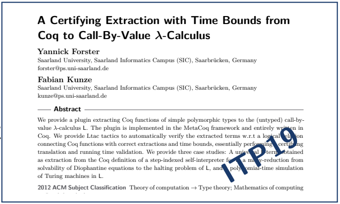
If $Q \preceq_T P$ and Q is undecidable, then P is undecidable.

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

Lemma

If $Q \preceq P$ then $Q \preceq_T P$.

Traditional Models of Computation


A Certifying Extraction with Time Bounds from Coq to Call-By-Value λ-Calculus

Yannick Forster
Saarland University, Saarland Informatics Campus (SIC), Saarbrücken, Germany
forster@ps.uni-saarland.de

Fabian Kunze
Saarland University, Saarland Informatics Campus (SIC), Saarbrücken, Germany
kunze@ps.uni-saarland.de

Abstract

We provide a plugin extracting Coq functions of simple polymorphic types to the (untyped) call-by-value λ-calculus L. The plugin is implemented in the MetaCoq framework and entirely written in Coq. We provide Ltac tactics to automatically verify the extracted terms w.r.t a logical relation connecting Coq functions with correct extractions and time bounds, essentially performing a certifying translation and running time validation. We provide three case studies: A universal L-term obtained as extraction from the Coq definition of a step-indexed self-interpreter for L, a many-reduction from solvability of Diophantine equations to the halting problem of L, and a polynomial-time simulation of Turing machines in L.

2012 ACM Subject Classification Theory of computation → Type theory; Mathematics of computing

Q: How to prove
“There is no Turing machineλ-term comput

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

$$\text{Coq function } f : \mathbb{N} \to \texttt{list}\,\mathbb{N}$$
$$\rightsquigarrow$$
$$\lambda\text{-term } t \text{ with } t\ \overline{n} \triangleright \overline{fn}.$$

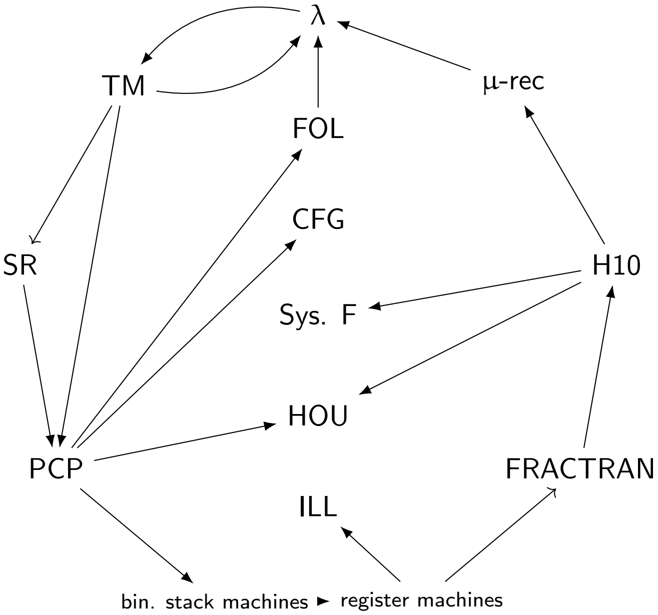
as a Coq plugin based on MetaCoq

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

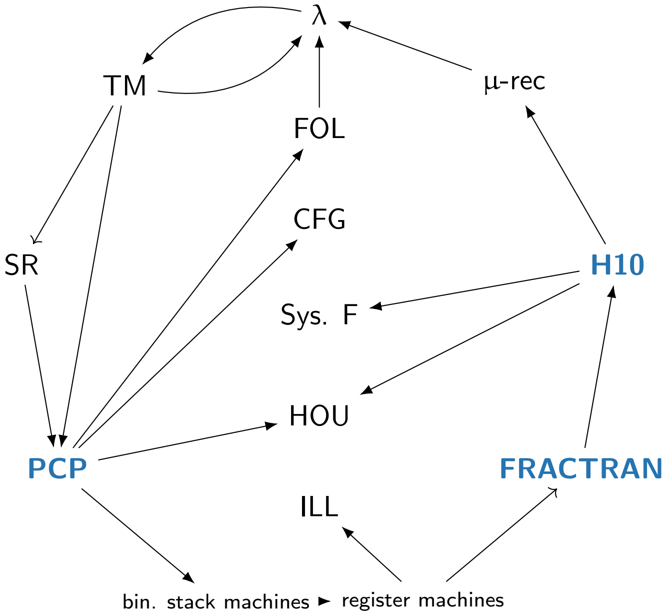
Vision

Undecidability proofs should be mechanisable

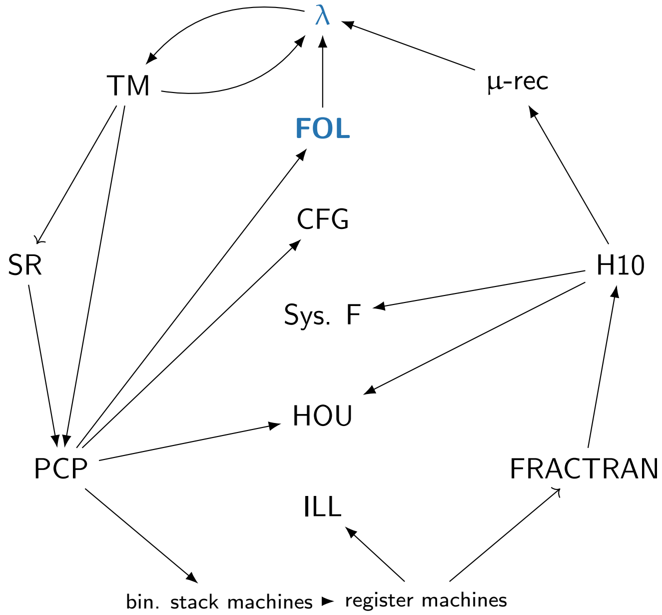
We need a library of potential starting points for proofs by reduction


λ
TM
μ-rec
FOL
CFG
SR
H10
Sys. F
HOU
PCP
FRACTRAN
ILL
bin. stack machines
register machines

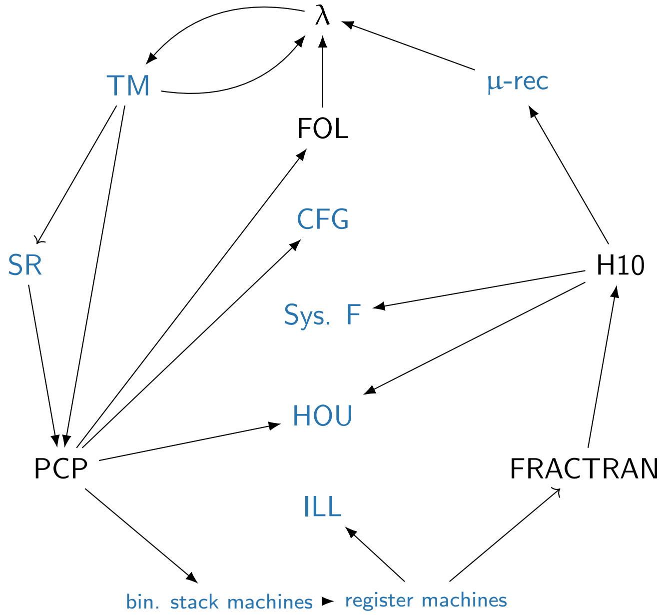
Seeds


λ
TM
μ-rec
FOL
CFG
SR
H10
Sys. F
HOU
PCP
FRACTRAN
ILL
bin. stack machines
register machines

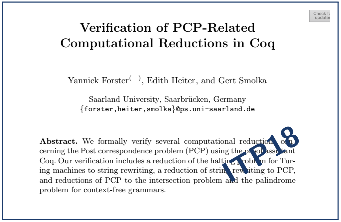
Targets


λ
TM
μ-rec
FOL
CFG
SR
H10
Sys. F
HOU
PCP
FRACTRAN
ILL
bin. stack machines
register machines

Advanced problems



Seed 1: PCP


Verification of PCP-Related Computational Reductions in Coq

Yannick Forster, Edith Heiter, and Gert Smolka

Saarland University, Saarbrücken, Germany
{forster,heiter,smolka}@ps.uni-saarland.de

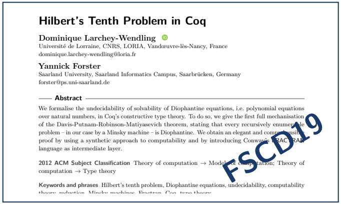
Abstract. We formally verify several computational reductions concerning the Post correspondence problem (PCP) using the proof assistant Coq. Our verification includes a reduction of the halting problem for Turing machines to string rewriting, a reduction of string rewriting to PCP, and reductions of PCP to the intersection problem and the palindrome problem for context-free grammars.

Base type: $\texttt{list}\,(\texttt{list}\,\mathbb{B} \times \texttt{list}\,\mathbb{B})$
Definition: $\mathsf{PCP}(L) := \exists x : \texttt{list}\,\mathbb{B}.\ L \triangleright (x, x)$

$$\frac{(u, v) \in L}{L \triangleright (u, v)} \qquad \frac{L \triangleright (x, y) \quad (u, v) \in L}{L \triangleright (x \mathbin{+\!\!+} u, y \mathbin{+\!\!+} v)}$$

Good seed for target problems that can express string concatenation and simple inductive predicates.

Seed 2: Diophantine constraints



Hilbert's Tenth Problem in Coq

Dominique Larchey-Wendling
Université de Lorraine, CNRS, LORIA, Vandœuvre-lès-Nancy, France
dominique.larchey-wendling@loria.fr

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

Abstract

We formalise the undecidability of solvability of Diophantine equations, i.e. polynomial equations over natural numbers, in Coq's constructive type theory. To do so, we give the first full mechanisation of the Davis-Putnam-Robinson-Matiyasevich theorem, stating that every recursively enumerable problem – in our case by a Minsky machine – is Diophantine. We obtain an elegant and comprehensible proof by using a synthetic approach to computability and by introducing Conway's FRACTRAN language as intermediate layer.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Type theory

Keywords and phrases Hilbert's tenth problem, Diophantine equations, undecidability, computability theory, reduction, Minsky machines, Fractran, Coq, type theory

Base type: `list` C where $C ::= x \dot{+} y \doteq z \mid x \dot{\times} y \doteq z \mid x \doteq 1$.
Definition: $\mathsf{H10}(C) := \exists \delta : \mathsf{var} \to \mathbb{N}.\ \delta \vDash C$ where

$$\delta \vDash x \dot{+} y \doteq z := \delta x + \delta y = \delta z$$

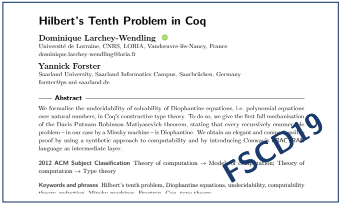
$$\delta \vDash x \dot{\times} y \doteq z := \delta x \cdot \delta y = \delta z$$

$$\delta \vDash x \doteq n := \delta x = 1$$

$$\delta \vDash C := \forall c \in C.\ \delta \vDash c$$

Good seed for target problems that can express addition and multiplication.

Seed 3: FRACTRAN


Hilbert's Tenth Problem in Coq

Dominique Larchey-Wendling
Université de Lorraine, CNRS, LORIA, Vandœuvre-lès-Nancy, France
dominique.larchey-wendling@loria.fr

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

Abstract

We formalise the undecidability of solvability of Diophantine equations, i.e. polynomial equations over natural numbers, in Coq's constructive type theory. To do so, we give the first full mechanisation of the Davis-Putnam-Robinson-Matiyasevich theorem, stating that every recursively enumerable problem – in our case by a Minsky machine – is Diophantine. We obtain an elegant and comprehensible proof by using a synthetic approach to computability and by introducing Conway's FRACTRAN language as intermediate layer.

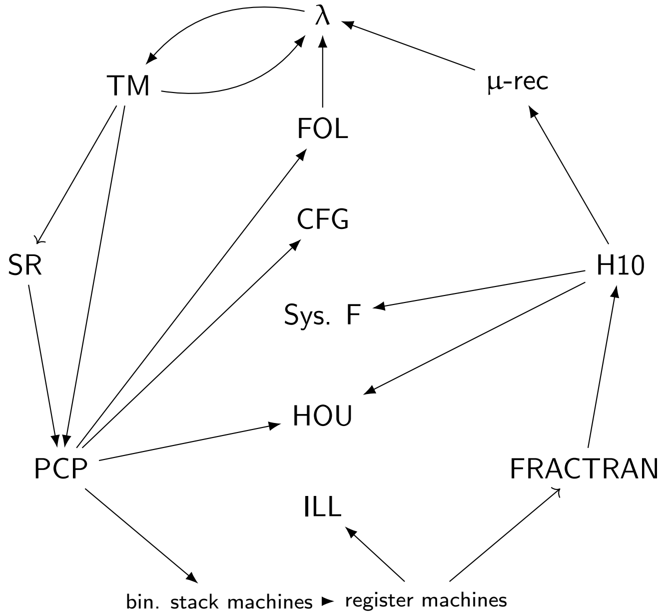
2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Type theory

Keywords and phrases Hilbert's tenth problem, Diophantine equations, undecidability, computability theory, reduction, Minsky machines, Fractran, Coq, type theory

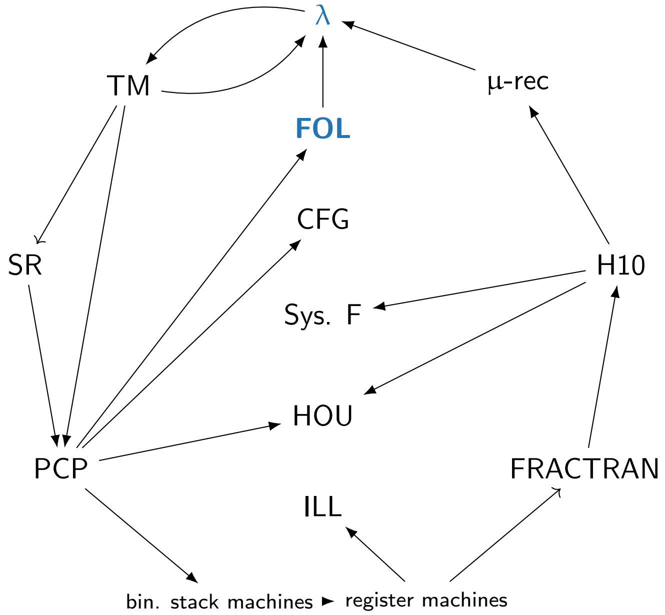
Base type: $\mathbb{N} \times \texttt{list}\,(\mathbb{N} \times \mathbb{N})$
Definition: $\text{FRACTRAN}(x, P) := x$ is terminating under $P \vdash \_ \succ \_$

$$\frac{q \cdot y = p \cdot x}{(p, q) :: P \vdash x \succ y} \qquad \frac{q \nmid p \cdot x \quad P \vdash x \succ y}{(p, q) :: P \vdash x \succ y}$$

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


λ
TM
FOL
μ-rec
CFG
SR
H10
Sys. F
HOU
PCP
FRACTRAN
ILL
bin. stack machines
register machines

Target problems



First-order logic

On Synthetic Undecidability in Coq, with an Application to the Entscheidungsproblem

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

Dominik Kirst
Saarland University
Saarbrücken, Germany
kirst@ps.uni-saarland.de

Gert Smolka
Saarland University
Saarbrücken, Germany
smolka@ps.uni-saarland.de

Abstract

We formalise the computational undecidability of validity, satisfiability, and provability of first-order formulas following a synthetic approach based on the computation native to Coq's constructive type theory. Concretely, we consider Tarski and Kripke semantics as well as classical and intuitionistic natural deduction systems and provide compact many-one reductions from the Post correspondence problem (PCP). Moreover, developing a basic framework for synthetic computability theory in Coq, we formalise standard results concerning decidability, enumerability, and reducibility without reference to a concrete model of computation. For instance, we prove the equivalence of Post's theorem with Markov's principle and provide a convenient technique

like decidability, enumerability, and reductions are available without reference to a concrete model of computation such as Turing machines, general recursive functions, or the λ-calculus. For instance, representing a given decision problem by a predicate p on a type X, a function $f : X \to \mathbb{B}$ with $\forall x.\, p\,x \leftrightarrow f\,x = \mathsf{tt}$ is a decision procedure, a function $g : \mathbb{N} \to X$ with $\forall x.\, p\,x \leftrightarrow (\exists n.\, g\,n = x)$ an enumeration, and a function $h : X \to Y$ with $\forall x.\, p\,x \leftrightarrow q\,(h\,x)$ for a predicate q on a type Y is a many-one reduction from p to q. Working formally with concrete models instead is cumbersome, given that every defined procedure needs to be shown representable by a concrete entity of the model. To avoid this tedium, many presentations resort to informal arguments regarding the algorithmic properties at the core

CPP19

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

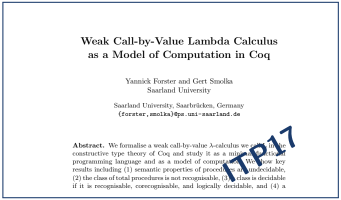
$$P(x) \leftrightarrow \Gamma \vdash \varphi_x$$

Usual proof strategy:

$\rightarrow$ By induction on the definition of P.

$\leftarrow$ By defining the standard model for P and soundness.

The cbv λ-calculus L



Weak Call-by-Value Lambda Calculus
as a Model of Computation in Coq

Yannick Forster and Gert Smolka
Saarland University

Saarland University, Saarbrücken, Germany
{forster,smolka}@ps.uni-saarland.de

Abstract. We formalise a weak call-by-value λ-calculus we call L in the constructive type theory of Coq and study it as a minimal functional programming language and as a model of computation. We show key results including (1) semantic properties of procedures are undecidable, (2) the class of total procedures is not recognisable, (3) a class is decidable if it is recognisable, corecognisable, and logically decidable, and (4) a

Definition

A problem $P : X \to \mathbb{P}$ is L-enumerable if there is an L-computable function $f : \mathbb{N} \to \texttt{option}\,X$ s.t. $Px \leftrightarrow \exists n.\ fn = \lfloor x \rfloor$.

Theorem

A problem $P : X \to \mathbb{P}$ 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 $f : \mathbb{N} \to \texttt{option}\,X$ (purely in Coq)
2. Give equality decider $X \to X \to \mathbb{B}$ (purely in Coq)
3. Give encoding for X in L (mostly automatic)
4. Use extraction to L from ITP '19 (fully automatic)

Advanced problems

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
s8mawutt@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 Ricciotti 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 time and space complexity compositionally. The user can do so 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

not clear at all how to compose a two-tape Turing machine with a three-tape Turing machine that works on a different alphabet. Therefore, it is common to rely on pseudo code or prose describing the intended behaviour. The exact implementation as well as its correctness or resource analysis is left as an exercise to the reader. In a mechanised proof, those details cannot be left out. Luckily, it is possible to hide those details behind suitable abstractions.

We present a framework that aims to have the cake and eat it too when it comes to mechanising computation in terms of Turing machines: Algorithms are stated in the style of a register based while-language; a corresponding Turing machine is automatically constructed behind the scene. Our framework furthermore characterises the semantics by deriving two relations for each machine, one witnessing partial

CPP 20

Turing Machine halting

A Simpler Undecidability Proof for System F Inhabitation

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

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

Abstract

Provability in the intuitionistic second-order propositional logic (resp. inhabitation in the polymorphic lambda-calculus) was shown by Löb to be undecidable in 1976. Since the original proof is heavily condensed, Arts in collaboration with Dekkers provided a fully unfolded argument in 1992 spanning approximately fifty pages. Later in 1997, Urzyczyn developed a different, syntax oriented proof. Each of the above approaches embeds (an undecidable fragment of) first-order predicate logic into second-order propositional logic.

In this work, we develop a simpler undecidability proof by reduction from solvability of Diophantine equations (is there an integer solution to $P(x_1, \ldots, x_n) = 0$ where P is a polynomial with integer coefficients?). Compared to the previous approaches, the given reduction is more accessible for formalization and more comprehensible for didactic purposes. Additionally, we formalize soundness

Types 18

System F inhabitation

Undecidability of Higher-Order Unification Formalised in Coq

Simon Spies
Saarland University
Saarland Informatics Campus, Saarbrücken, Germany
spies@ps.uni-saarland.de

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

Abstract

We formalise undecidability results concerning higher-order unification in the simply-typed λ-calculus with β-conversion in Coq. We prove the undecidability of general higher-order unification by reduction from Hilbert's tenth problem, the solvability of Diophantine equations, following a proof by Dowek. We sharpen the result by establishing the undecidability of second-order and third-order unification following proofs by Goldfarb and Huet, respectively.

Goldfarb's proof for second-order unification is by reduction from Hilbert's tenth problem. Huet's original proof uses the Post correspondence problem (PCP) to show the undecidability of third-order unification. We simplify and formalise

second-order unification only mentions terms with free variables of second-order type. In contrast, *third-order* unification is concerned with terms where variables have a type of at most order three.

In 1965, first-order unification was shown to be decidable by Robinson [1965] and even has linear decision algorithms [Martelli and Montanari 1976; Paterson and Wegman 1978]. In 1972, Huet [1972, 1973] and Lucchesi [1972] independently showed that third-order unification is undecidable, thereby establishing the undecidability of higher-order unification in general. In both cases the proof is by reduction from Post's correspondence problem [1946]. In 1981, Goldfarb [1981] proved that even second-order uni-

CPP 20

Higher-order unification

Certified Undecidability of Intuitionistic Linear Logic via Binary Stack Machines and Minsky Machines

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

Dominique Larchey-Wendling
Université de Lorraine, CNRS, LORIA
Vandœuvre-lès-Nancy, France
dominique.larchey-wendling@loria.fr

Abstract

We formally prove the undecidability of entailment in intuitionistic linear logic in Coq. We reduce the Post correspondence problem (PCP) via binary stack machines and Minsky machines to intuitionistic linear logic. The reductions rely on several technically involved formalisations, amongst them a binary stack machine simulator for PCP, a verified low-level compiler for instruction-based languages and a soundness proof for intuitionistic linear logic with respect to trivial phase semantics. We exploit the computability of all functions definable in constructive type theory and thus do not have to rely on a concrete model of computation, enabling the reduction proofs to focus on correctness properties.

the use of interactive theorem provers to assist ongoing research. However, formalisations of undecidability proofs are not common in the literature. There are two main obstacles in our eyes: (1) proofs on paper mostly omit the invariants needed for the verification of the reduction; (2) they omit the computability proof, which amounts to the formal verification of a program in the chosen model of computation.

Constructive type theory, as implemented in the proof assistant Coq [37], provides a particularly convenient setting for decidability and undecidability proofs. Since every function definable in constructive type theory is computable, one can use a synthetic approach to computability. This approach makes an explicit model of computation and explicit

CPP 19

Provability in Linear Logic

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

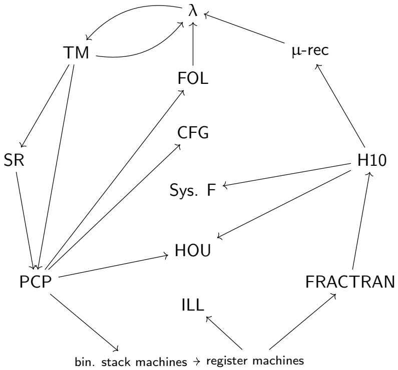
Include more problems

- Typability in $\lambda\Pi$ or type inference in System F_ω
- Intersection problem of two-way automata
- Wang tiling problems, Post's tag systems
- Subtyping in $F_{\leqslant}$, 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?