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

Yannick Forster and Dominique Larchey-Wendling

LOLA 2018
July 12

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Loria

cnrs

Decidability

A problem $P : X \to \mathbb{P}$ is decidable if ...

Classically	Fix a model of computation M: there is a decider in M
For the cbv λ-calculus	$\exists u : \mathbf{T}. \forall x : X.\ (u\overline{x} \triangleright T \wedge Px) \vee (u\overline{x} \triangleright F \wedge \neg Px)$
Type Theory	$\exists f : X \to \mathbb{B}.\ \forall x : X.\ Px \leftrightarrow fx = \mathrm{true}$
dependent version (Coq, Agda, Lean, ...)	$\forall x : X.\ \{Px\} + \{\neg Px\}$

Undecidability

A problem $P : X \to \mathbb{P}$ is undecidable if ...

Classically — If there is no decider u in M

For the cbv λ-calculus — $\neg\exists u : \mathbf{T}. \forall x : X.\ (u\overline{x} \triangleright T \wedge Px) \vee (u\overline{x} \triangleright F \wedge \neg Px)$

Type Theory — $\neg(\forall x : X.\ \{Px\} + \{\neg Px\})$ $\cancel{\neg(\forall x : X.\ \{Px\} + \{\neg Px}$

In reality: most proofs are by reduction

Definition

P undecidable $:=$ Halting problem reduces to P

Reduction

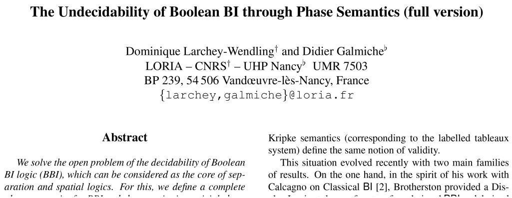
A problem is a type X and a unary predicate $P : X \to \mathbb{P}$

A reduction of (X, P) to (Y, Q) is a function $f : X \to Y$ s.t.
$\forall x.\ Px \leftrightarrow Q(fx)$

Write

$$P \preceq Q$$

An undecidability proof for intuitionistic linear logic



The Undecidability of Boolean BI through Phase Semantics (full version)

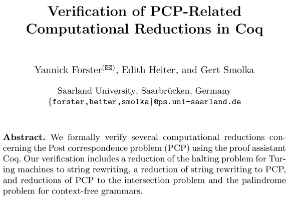
Dominique Larchey-Wendling† and Didier Galmiche♭
LORIA – CNRS† – UHP Nancy♭ UMR 7503
BP 239, 54 506 Vandœuvre-lès-Nancy, France
{larchey,galmiche}@loria.fr

Abstract

We solve the open problem of the decidability of Boolean BI logic (BBI), which can be considered as the core of separation and spatial logics. For this, we define a complete

Kripke semantics (corresponding to the labelled tableaux system) define the same notion of validity.

This situation evolved recently with two main families of results. On the one hand, in the spirit of his work with Calcagno on Classical BI [2], Brotherston provided a Dis-

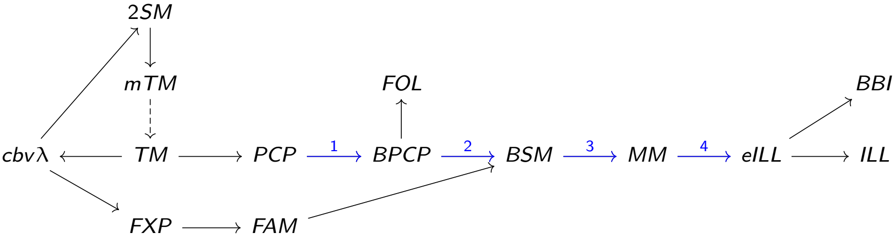


Verification of PCP-Related Computational Reductions in Coq

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.



Post correspondence problem

From Wikipedia, the free encyclopedia

The **Post correspondence problem** is an undecidable decision problem that was introduced by Emil Post in 1946.[1] Because it is simpler than the halting problem and the *Entscheidungsproblem* it is often used in proofs of undecidability.

PCP

$$\boxed{\frac{C2}{LoC2018}}\ \boxed{\frac{xfor}{}}\ \boxed{\frac{nf}{d}}\ \boxed{\frac{FLo}{F}}\ \boxed{\frac{d}{ord}}\ \boxed{\frac{018inO}{inOxf}}$$

$$\boxed{\frac{FLo}{F}}\ \boxed{\frac{C2}{LoC2018}}\ \boxed{\frac{018inO}{inOxf}}\ \boxed{\frac{xfor}{}}\ \boxed{\frac{d}{ord}}$$

$$\frac{FLoC2018inOxford}{FLoC2018inOxford}$$

- Symbols a, b, c: $\mathbb{N}$
- Strings x, y, z: lists of symbols
- Card c: pairs of strings
- Stacks A: lists of cards

$$[]^1 := \epsilon \qquad []^2 := \epsilon$$

$$(x/y :: A)^1 := x(A^1) \quad (x/y :: A)^2 := y(A^2)$$

$$PCP(P) \ := \ \exists A \subseteq P.\ A \neq [] \land A^1 = A^2$$

$\text{PCP} \preceq \text{BPCP}$

generalised BPCP

BPCP:

- Symbols a, b, c: $\mathbb{B}$
- Strings x, y, z: lists of symbols
- Card c: pairs of strings
- Stacks A, R: lists of string

generalised PCP:

- Symbols a, b, c: X
- Strings x, y, z: lists of symbols
- Card c: pairs of strings
- Stacks A, R: lists of stacks

$$[]^1 := \epsilon$$
$$(x/y :: A)^1 := x(A^1)$$

$$[]^2 := \epsilon$$
$$(x/y :: A)^2 := y(A^2)$$

$$BPCP\,(P : \mathsf{Stack}_{\mathbb{B}}) \;:=\; \exists A \subseteq P.\ A \neq [] \land A^1 = A^2$$

$$PCP_X\,(P : \mathsf{Stack}_X) \;:=\; \exists A \subseteq P.\ A \neq [] \land A^1 = A^2$$

PCP $\preceq$ BPCP

$$f : \mathbb{N}^* \to \mathbb{B}^*$$

$$f(a_1 \dots a_n : \mathbb{N}^*) := 1^{a_1}0 \dots 1^{a_n}0$$

Lift f to cards and stack by pointwise application

To prove: PCP$P \leftrightarrow$ BPCP$(f\ P)$
Define inverse function g, easy

Contribution

$$PCP \longrightarrow BPCP \longrightarrow BSM \longrightarrow MM \longrightarrow eILL \longrightarrow ILL$$

$\mathsf{BPCP} \preceq \mathsf{BSM}$

Binary stack machines

- n stacks of 0s and 1s (`list bool`) for a fixed n
- instructions (with $0 \leqslant x < n$ and $b \in \texttt{bool}$ and $i \in \mathbb{N}$)

$$\texttt{bsm\_instr} \ ::= \ \texttt{POP}\ x\ i\ j \mid \texttt{PUSH}\ x\ b \mid \texttt{HALT}$$

- state: $(\texttt{PC} \in \mathbb{N}, \vec{S} \in (\texttt{list bool})^n)$
- Small step semantics (HALT is blocking):

$$\begin{array}{ll} \texttt{POP}\ x\ i: & \text{if } x \text{ is empty, then } \texttt{PC} \leftarrow j \\ & \text{else pop } b \text{ from stack } x; \\ & \text{if } b \text{ is 0 then } \texttt{PC} \leftarrow i \text{ else } \texttt{PC} \leftarrow \texttt{PC}+1; \\ \texttt{PUSH}\ x\ b: & \text{push } b \text{ on stack } x; \texttt{PC} \leftarrow \texttt{PC}+1; \end{array}$$

- BSM program $\mathcal{B}_{i,j}$: $i : \texttt{bsm\_instr}_i; i+1 : \ldots; j : \texttt{bsm\_instr}_j$
- $\text{BSM}(\mathcal{B}_{i,j}, \vec{S}) := \exists \vec{S}'. \mathcal{B} : (i, \vec{S}) \longrightarrow^* (j+1, \vec{S}')$

BPCP $\preceq$ BSM

- Keep stacks for top and bottom row
- Hard code every card as `PUSH` instructions
- Iterate all possible stacks
- Check for stack equality

```
Definition compare_stacks x y i p q :=
  (*   i    *) [POP x (4+i) (7+i) ;
  (* 1+i    *)  POP y q q ;
  (* 2+i    *)  PUSH x Zero ; POP x i i ;
  (* 4+i    *)  POP y i q ;
  (* 5+i    *)  PUSH y Zero ; POP y q i ;
  (* 7+i    *)  POP y q p ;
  (* 8+i    *)  PUSH x' Zero ; POP x' q q ].
```

Lemma

For all stack configurations v,

$$\texttt{compare\_stacks}\ x\ y\ i\ p\ q : (i, v) \longrightarrow^* (r, w)$$

where $r = p$ if the value of x is the value of y and $r = q$ otherwise. The value of all stacks apart from x and y in w is equal to the value of all stacks in v.

Contribution

$$PCP \longrightarrow BPCP \longrightarrow BSM \longrightarrow MM \longrightarrow eILL \longrightarrow ILL$$

BSM $\preceq$ MM

Minsky Machines

- n registers $\{x_1, \ldots, x_n\}$ of value in $\mathbb{N}$ for a fixed n
- instructions (with $x \in \{x_1, \ldots, x_n\}$ and $i \in \mathbb{N}$)

$$\texttt{mm\_instr} ::= \texttt{INC}\ x \mid \texttt{DEC}\ x\ i$$

- Small step semantics, state: $(\texttt{PC} \in \mathbb{N}, \vec{v} \in \mathbb{N}^n)$

$$\begin{array}{ll} \texttt{INC}\ x: & x \leftarrow x + 1; \texttt{PC} \leftarrow \texttt{PC} + 1; \\ \texttt{DEC}\ x\ i: & \text{if } x \text{ is } 0 \text{ then } \texttt{PC} \leftarrow i \text{ else } x \leftarrow x - 1; \texttt{PC} \leftarrow \texttt{PC} + 1; \end{array}$$

- MM program $\mathcal{M}_{i,j}$: $i : \texttt{mm\_instr}_i; i+1 : \ldots; j : \texttt{mm\_instr}_j$
- $\mathrm{MM}(\mathcal{M}_{i,j}, \vec{v}) := \mathcal{M} : (i, \vec{v}) \longrightarrow^* (j+1, \vec{0})$

BSM $\preceq$ MM

Certified Compiler

Stacks are registers, interpret bitstring as binary number

Implement `DIV2`, `MOD2`, `MUL2` ... for push and pop operations

Contribution

$$PCP \longrightarrow BPCP \longrightarrow BSM \longrightarrow MM \longrightarrow eILL \longrightarrow ILL$$

MM $\preceq$ eILL

Intuitionistic Linear Logic

- We "restrict" to the $(!, \multimap, \&)$ fragment, system G-ILL

$$\frac{}{A \vdash A}\ [\mathrm{id}] \qquad \frac{\Gamma \vdash A \qquad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\ [\mathrm{cut}]$$

$$\frac{\Gamma, A \vdash B}{\Gamma, !\,A \vdash B}\ [!_L] \qquad \frac{!\,\Gamma \vdash B}{!\,\Gamma \vdash !\,B}\ [!_R] \qquad \frac{\Gamma \vdash B}{\Gamma, !\,A \vdash B}\ [\mathrm{w}] \qquad \frac{\Gamma, !\,A, !\,A \vdash B}{\Gamma, !\,A \vdash B}\ [\mathrm{c}]$$

$$\frac{\Gamma, A \vdash C}{\Gamma, A \,\&\, B \vdash C}\ [\&_L^1] \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \,\&\, B \vdash C}\ [\&_L^2] \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\ [\&_R]$$

$$\frac{\Gamma \vdash A \qquad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C}\ [\multimap_L] \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ [\multimap_R]$$

- Full linear logic faithfully embedded into that fragment
- $\mathrm{ILL}(\Gamma, A) := \mathrm{provable}(\Gamma \vdash A)$

eILL

- Elementary sequents: $!\,\Sigma, g_1, \ldots, g_k \vdash d$ $\quad$ (g_i, a, b, c, d variables)
- Σ contains *commands*:
  - $(a \multimap b) \multimap c$, correponding to `INC`
  - $a \multimap (b \multimap c)$, correponding to `DEC`
  - $(a \,\&\, b) \multimap c$, correponding to `FORK`
- goal directed rules for eILL (sound and complete w.r.t. G-ILL):
- TPS (even $\mathbb{N}^k$) is (sound and) complete for eILL.
- Hence a fragment of both ILL and BBI

Encoding Minsky machines in eILL

- Given $\mathcal{M}$ as a list of MM instructions
  - for every register x_i in $\mathcal{M}$, two logical variables x_i and $\underline{x}_i$
  - for every position/state ($\mathtt{PC} = i$) in $\mathcal{M}$, a variable q_i
- the state $(i, \vec{v})$ is represented by $!\,\Sigma; \Delta_{\vec{v}} \vdash q_i$
  - where if $\vec{v} = (p_1, \ldots, p_n)$ then $\Delta_{\vec{v}} = p_1.x_1, \ldots, p_n.x_n$
  - Variables: $\{x_1, \ldots, x_n\} \uplus \{\underline{x}_1, \ldots, \underline{x}_n\} \uplus \{q_0, q_1, \ldots\}$
  - the commands in Σ are determined by instructions in $\mathcal{M}$

$$i :\ \mathtt{INC}\, x \in \mathcal{M} \;\left|\; \begin{array}{l} x \leftarrow x + 1 \\ \mathtt{PC} \leftarrow i + 1 \end{array} \;\right|\; \dfrac{\dfrac{\cdots}{!\,\Sigma, x, \Delta \vdash q_{i+1}}}{!\,\Sigma, \Delta \vdash q_i} \; ((x \multimap q_{i+1}) \multimap q_i \in \Sigma)$$

MM to eILL, (continued)

- Decrement

$$i:\ \texttt{DEC}\ x\ j \in \mathcal{M} \quad \left|\quad \begin{array}{l} \text{if } x = 0 \text{ then } \texttt{PC} \leftarrow j \\ \text{else } x \leftarrow x - 1; \texttt{PC} \leftarrow i + 1 \end{array}\right.$$

- corresponds to two proofs $x > 0$ and $x = 0$:

$$\dfrac{\dfrac{}{!\,\Sigma, x \vdash x}\,(\mathrm{Ax}) \qquad \dfrac{\cdots}{!\,\Sigma, \Delta \vdash q_{i+1}}}{!\,\Sigma, x, \Delta \vdash q_i}\,(x \multimap (q_{i+1} \multimap q_i) \in \Sigma)$$

$$\dfrac{\dfrac{\cdots}{!\,\Sigma, \Delta \vdash \underline{x}}\,(x \notin \Delta) \qquad \dfrac{\cdots}{!\,\Sigma, \Delta \vdash q_j}}{!\,\Sigma, \Delta \vdash q_i}\,((\underline{x} \,\&\, q_j) \multimap q_i \in \Sigma)$$

Zero test $x \notin \Delta$ in eILL

- $!\,\Sigma; \Delta \vdash \underline{x}$ provable iff $x \notin \Delta$
- Proof for y, Δ with $y \neq x$:

$$\dfrac{\dfrac{}{!\,\Sigma, y \vdash y}\,(\mathsf{Ax}) \qquad \dfrac{\cdots}{!\,\Sigma, \Delta \vdash \underline{x}}}{!\,\Sigma, y, \Delta \vdash \underline{x}}\,(y \multimap (\underline{x} \multimap \underline{x}) \in \Sigma)$$

- Proof for empty context $\Delta = \emptyset$:

$$\dfrac{\dfrac{}{!\,\Sigma, \underline{x} \vdash \underline{x}}\,(\mathsf{Ax})}{!\,\Sigma, \emptyset \vdash \underline{x}}\,((\underline{x} \multimap \underline{x}) \multimap \underline{x} \in \Sigma)$$

Correctness proof $\Rightarrow$

- Termination, for k halting state, i.e. k *outside of* $\mathcal{M}$

$$\dfrac{\dfrac{}{!\,\Sigma, q_k \vdash q_k}\ (\mathrm{Ax})}{!\,\Sigma, \emptyset \vdash q_k}\ ((q_k \multimap q_k) \multimap q_k \in \Sigma)$$

- We define $\Sigma_{\mathcal{M},k}$ by:

$$\begin{array}{rcl}
\Sigma_{\mathcal{M},k} & = & \{(q_k \multimap q_k) \multimap q_k\} \\
 & \cup & \{y \multimap (\underline{x} \multimap \underline{x}), (\underline{x} \multimap \underline{x}) \multimap \underline{x} \mid x \neq y \in [1,n]\} \\
 & \cup & \{(x \multimap q_{i+1}) \multimap q_i \mid i : \mathtt{INC}\ x \in \mathcal{M}\} \\
 & \cup & \{(\underline{x} \mathbin{\&} q_j) \multimap q_i, x \multimap (q_{i+1} \multimap q_i) \mid i : \mathtt{DEC}\ x\ j \in \mathcal{M}\}
\end{array}$$

- Theorem: $\mathcal{M} : (i, \vec{v}) \longrightarrow^* (k, \vec{0}) \Rightarrow\ !\,\Sigma_{\mathcal{M},k}, \Delta_{\vec{v}} \vdash q_i$

Correctness proof $\Leftarrow$

- let us show $!\,\Sigma_{\mathcal{M},k}, \Delta_{\vec{v}} \vdash q_i \Rightarrow \mathcal{M} : (i, \vec{v}) \longrightarrow^* (k, \vec{0})$
- we use trivial phase semantics: $[\![A]\!] : \mathbb{N}^n \to \texttt{Prop}$

$$\begin{aligned} [\![x]\!]\ \vec{v} &\iff \vec{v} = 1.x & \text{(i.e. } \vec{v}_y = \delta_{x,y}) \\ [\![\underline{x}]\!]\ \vec{v} &\iff \vec{v}_x = 0 \\ [\![q_i]\!]\ \vec{v} &\iff \mathcal{M} : (i, \vec{v}) \longrightarrow^* (k, \vec{0}) \end{aligned}$$

- we show: $[\![A]\!]\ \vec{0}$ for any $A \in \Sigma_{\mathcal{M},k}$, hence $[\![!\,\Sigma_{\mathcal{M},k}]\!] = \{\vec{0}\}$
- we also have $[\![\Delta_{\vec{v}}]\!] = \{\vec{v}\}$
- by soundness of TPS, from $!\,\Sigma_{\mathcal{M},k}; \Delta_{\vec{v}} \vdash q_i$ we get $[\![q_i]\!]\ \vec{v}$
- comp. reduction: $\boxed{\mathcal{M} : (i, \vec{v}) \longrightarrow^* (k, \vec{0}) \iff !\,\Sigma_{\mathcal{M},k}, \Delta_{\vec{v}} \vdash q_i}$

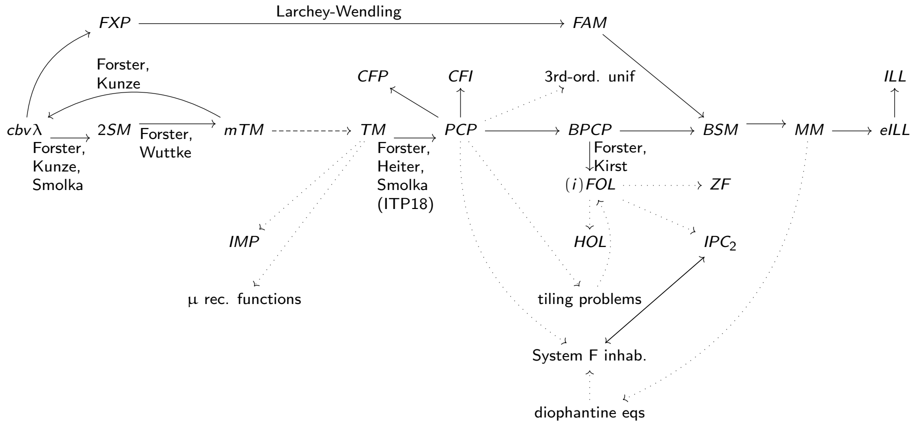
Wrap-up of this talk

Reductions:

- PCP to BPCP: trivial binary encoding
- BPCP to BSM: verified exhaustive search
- BSM to MM: certified compiler between low-level languages
- MM to iLL: elegant encoding of computational model in logics

Low verification overhead
(compared to detailed paper proofs)

Future Work



Forster, Kunze: Automated extraction from Coq to cbv λ-calculus yields computability proofs for all reductions

Lesson learned: Chunk your reductions!

Wrap-up

- A library of computational models and undecidable problems
- Exemplary undecidability proof for provability in linear logic
- Enabling loads of future work. Attach your own undecidable problems!

Advertisement: ITP 2018 talk

Verification of PCP-Related
Computational Reductions in Coq

Thursday, 10:00

Questions?