A constructive Coq library for the mechanisation of undecidability

Yannick Forster and Dominique Larchey-Wendling

MLA 2019
March 13

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 $\exists u : \mathbf{T}. \forall x : X.\ (u\overline{x} \triangleright T \land Px) \lor (u\overline{x} \triangleright F \land \neg Px)$

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	$\exists f : X \to \mathbb{B}.\ \forall x : X.\ Px \leftrightarrow fx = \text{true}$

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	$\exists f : X \to \mathbb{B}.\ \forall x : X.\ Px \leftrightarrow fx = \text{true}$
dependent version (Coq, Agda, Lean, ...)	$\boxed{\text{dec } P := \forall x : X.\ \{P\,x\} + \{\neg P\,x\}}$

Undecidability

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

Classically

If there is no decider u in M

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 — $\neg \exists u : \mathbf{T}. \forall x : X.\ (u\overline{x} \triangleright T \wedge Px) \vee (u\overline{x} \triangleright F \wedge \neg Px)$

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	$\neg(\forall x : X.\ \{Px\} + \{\neg Px\})$

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	$\neg\exists u : \mathbf{T}. \forall x : X.\ (u\overline{x} \triangleright T \land Px) \lor (u\overline{x} \triangleright F \land \neg Px)$
Type Theory	~~$\neg(\forall x : X.\ \{Px\} + \{\neg Px\})$~~

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	~~$\neg(\forall x : X.\ \{Px\} + \{\neg Px\})$~~

In reality: most proofs are by reduction

Definition (Synthetic undecidability)

P undecidable := Halting problem reduces to P

The library

`https://github.com/uds-psl/coq-library-undecidability`

- Halting problems
  - Turing machines
  - Minsky machines
  - μ-recursive functions
  - call-by-value lambda-calculus
- Post correspondence problem
- Provability in linear logic and first-order logic
- Solvability of Diophantine equations, including a formalisation of the DPRM theorem

Today

1. Overview over PCP and H10 as entry points
2. Exemplary undecidability proof for intuitionistic linear logic
3. Overview over the library and future work

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$_X$

$$\frac{Na}{19in}\ \frac{MLA}{M}\ \frac{y}{cy}\ \frac{xuz}{ofze}\ \frac{19i}{LA}\ \frac{n}{\ }\ \frac{c}{Nan}$$

PCP_X

$$\frac{Na}{19in}\quad\frac{MLA}{M}\quad\frac{y}{cy}\quad\frac{xuz}{ofze}\quad\frac{19i}{LA}\quad\frac{n}{\;}\quad\frac{c}{Nan}$$

$$\frac{MLA}{M}$$

PCP_X

$$\boxed{\frac{Na}{19in}}\ \boxed{\frac{MLA}{M}}\ \boxed{\frac{y}{cy}}\ \boxed{\frac{xuz}{ofze}}\ \boxed{\frac{19i}{LA}}\ \boxed{\frac{n}{\epsilon}}\ \boxed{\frac{c}{Nan}}$$

$$\boxed{\frac{MLA}{M}}\ \boxed{\frac{19i}{LA}}$$

PCP_X

$\frac{Na}{19in}$ $\frac{MLA}{M}$ $\frac{y}{cy}$ $\frac{xuz}{ofze}$ $\frac{19i}{LA}$ $\frac{n}{}$ $\frac{c}{Nan}$

$\frac{MLA}{M}$ $\frac{19i}{LA}$ $\frac{n}{}$

PCP$_X$

$\frac{Na}{19in}$ $\frac{MLA}{M}$ $\frac{y}{cy}$ $\frac{xuz}{ofze}$ $\frac{19i}{LA}$ $\frac{n}{\ }$ $\frac{c}{Nan}$

$\frac{MLA}{M}$ $\frac{19i}{LA}$ $\frac{n}{\ }$ $\frac{Na}{19in}$

PCP_X

$\frac{Na}{19in}$ $\frac{MLA}{M}$ $\frac{y}{cy}$ $\frac{xuz}{ofze}$ $\frac{19i}{LA}$ $\frac{n}{\ }$ $\frac{c}{Nan}$

$\frac{MLA}{M}$ $\frac{19i}{LA}$ $\frac{n}{\ }$ $\frac{Na}{19in}$ $\frac{n}{\ }$

PCP_X

$$\boxed{\frac{Na}{19in}}\ \boxed{\frac{MLA}{M}}\ \boxed{\frac{y}{cy}}\ \boxed{\frac{xuz}{ofze}}\ \boxed{\frac{19i}{LA}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{c}{Nan}}$$

$$\boxed{\frac{MLA}{M}}\ \boxed{\frac{19i}{LA}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{Na}{19in}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{c}{Nan}}$$

PCP$_X$

$$\boxed{\frac{Na}{19in}}\ \boxed{\frac{MLA}{M}}\ \boxed{\frac{y}{cy}}\ \boxed{\frac{xuz}{ofze}}\ \boxed{\frac{19i}{LA}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{c}{Nan}}$$

$$\boxed{\frac{MLA}{M}}\ \boxed{\frac{19i}{LA}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{Na}{19in}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{c}{Nan}}\ \boxed{\frac{y}{cy}}$$

PCP_X

$$\boxed{\frac{Na}{19in}}\ \boxed{\frac{MLA}{M}}\ \boxed{\frac{y}{cy}}\ \boxed{\frac{xuz}{ofze}}\ \boxed{\frac{19i}{LA}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{c}{Nan}}$$

$$\boxed{\frac{MLA}{M}}\ \boxed{\frac{19i}{LA}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{Na}{19in}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{c}{Nan}}\ \boxed{\frac{y}{cy}}$$

$$\frac{MLA19inNancy}{MLA19inNancy}$$

PCP$_X$

$$\frac{Na}{19in}\ \frac{MLA}{M}\ \frac{y}{cy}\ \frac{xuz}{ofze}\ \frac{19i}{LA}\ \frac{n}{}\ \frac{c}{Nan}$$

$$\frac{MLA}{M}\ \frac{19i}{LA}\ \frac{n}{}\ \frac{Na}{19in}\ \frac{n}{}\ \frac{c}{Nan}\ \frac{y}{cy}$$

$$\frac{MLA19inNancy}{MLA19inNancy}$$

- Symbols a, b, c: symbols of type X
- Strings x, y, z: lists of symbols
- Card x/y: pairs of strings
- Card set R: finite set of cards
- Stacks A: lists of cards

PCP_X

$$\boxed{\frac{Na}{19in}}\ \boxed{\frac{MLA}{M}}\ \boxed{\frac{y}{cy}}\ \boxed{\frac{xuz}{ofze}}\ \boxed{\frac{19i}{LA}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{c}{Nan}}$$

$$\boxed{\frac{MLA}{M}}\ \boxed{\frac{19i}{LA}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{Na}{19in}}\ \boxed{\frac{n}{\ }}\ \boxed{\frac{c}{Nan}}\ \boxed{\frac{y}{cy}}$$

$$\frac{MLA19inNancy}{MLA19inNancy}$$

- Symbols a, b, c: symbols of type X
- Strings x, y, z: lists of symbols
- Card x/y: pairs of strings
- Card set R: finite set of cards
- Stacks A: lists of cards

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

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

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

PCP $\preceq$ BPCP

PCP $\preceq$ BPCP

PCP is $\mathrm{PCP}_{\mathbb{N}}$

BPCP is $\mathrm{PCP}_{\mathbb{B}}$

PCP $\preceq$ BPCP

PCP is $\text{PCP}_{\mathbb{N}}$

BPCP is $\text{PCP}_{\mathbb{B}}$

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

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

Lift f to cards, card sets and stack by pointwise application

PCP $\preceq$ BPCP

PCP is $\text{PCP}_{\mathbb{N}}$

BPCP is $\text{PCP}_{\mathbb{B}}$

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

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

Lift f to cards, card sets and stack by pointwise application

To prove: PCP R $\leftrightarrow$ BPCP$(f\ R)$

PCP $\preceq$ BPCP

PCP is $\text{PCP}_{\mathbb{N}}$

BPCP is $\text{PCP}_{\mathbb{B}}$

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

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

Lift f to cards, card sets and stack by pointwise application

To prove: $\quad \text{PCP } R \quad \leftrightarrow \quad \text{BPCP}(f\ R)$

Define inverse function g, easy

Hilbert's tenth problem, constraints version

$$c : \mathsf{constr} ::= x \mathbin{\dot{+}} y \doteq z \;\mid\; x \mathbin{\dot{\times}} y \doteq z \;\mid\; x \doteq 1$$

$$\begin{aligned} [\![x \mathbin{\dot{+}} y \doteq z]\!]_\rho &:= \rho\, x + \rho\, y = \rho\, z \\ [\![x \mathbin{\dot{\times}} y \doteq z]\!]_\rho &:= \rho\, x \cdot \rho\, y = \rho\, z \\ [\![x \doteq 1]\!]_\rho &:= \rho\, x = 1 \end{aligned}$$

$$\mathsf{H10c}(L : \mathbb{L}\,\mathsf{constr}) := \exists \rho, \forall c \in L,\; [\![c]\!]_\rho$$

Undecidability of Intuitionistic Linear Logic (CPP '19)

Undecidability of Intuitionistic Linear Logic (CPP '19)

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

LICS 2010

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-

Undecidability of Intuitionistic Linear Logic (CPP '19)

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

LICS 2010

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

ITP 2018

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.

Undecidability of Intuitionistic Linear Logic (CPP '19)

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

LICS 2010

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

ITP 2018

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.

$$TM \xrightarrow[ITP18]{} PCP \longrightarrow BPCP \longrightarrow BSM \longrightarrow MM \xrightarrow[LICS10]{} eILL \xrightarrow[LICS10]{} ILL$$

Undecidability of Intuitionistic Linear Logic (CPP '19)

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

LICS 2010

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

ITP 2018

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.

$$TM \xrightarrow[ITP18]{} PCP \xrightarrow{1} BPCP \xrightarrow{2} BSM \xrightarrow{3} MM \xrightarrow[LICS10]{4} eILL \xrightarrow[LICS10]{5} ILL$$

Low-level Code

Code and subcode

- Given a type $\mathbb{I}$ of instructions
- Codes are $\mathbb{N}$-indexed programs: $(i, P = [\rho_0; \ldots; \rho_{n-1}])$ of type $\mathbb{N} \times \mathbb{L}\,\mathbb{I}$

$$i : \rho_0; \qquad i+1 : \rho_1; \qquad \ldots \qquad i+n-1 : \rho_{n-1};$$

- labels $i, \ldots, i+n-1$ identify PC values inside the program
- Subcode relation $(i, P) <_{\mathrm{sc}} (j, Q)$

$$(i, P) <_{\mathrm{sc}} (j, Q) := \exists\, L\, R,\ \wedge \begin{cases} Q = L +\!\!+ P +\!\!+ R \\ i = j + |L| \end{cases}$$

- instruction ρ occurs at pos. i in (j, Q): $(i, [\rho]) <_{\mathrm{sc}} (j, Q)$
- "Sub-programs" are contiguous segments

Small Step Semantics for Code

- Instructions as state transformers
- states (i, v): i is $\texttt{PC}$ value and $v : \mathbb{C}$ a configuration
- a step relation $\rho \mathbin{/\!/} (i_1, v_1) \succ (i_2, v_2)$
  - instruction ρ at position i_1 transforms state (i_1, v_1) into (i_2, v_2)
- extends to codes: $(i, P) \mathbin{/\!/} (i_1, v_1) \succ^n (i_2, v_2)$ means
  - Code (i, P) transforms state (i_1, v_1) into (i_2, v_2)
  - $$\frac{(i_1, [\rho]) <_{\mathrm{sc}} (i, P) \quad \rho \mathbin{/\!/} (i_1, v_1) \succ (i_2, v_2)}{(i, P) \mathbin{/\!/} (i_1, v_1) \succ (i_2, v_2)}$$
  - Reflexive transitive closure: $\mathcal{P} \mathbin{/\!/} s \succ^* s'$

Terminating computations and Big Step Semantics

- denote $\mathcal{P}$ for codes like (i, P) and s for states like (j, v)
- which termination condition: $\boxed{\text{out } j\ \mathcal{P}}$
  - no instruction at j in $\mathcal{P}$, computation is blocked (sufficient)
  - $\mathcal{P} /\!\!/ (j, v) \succ^n s \wedge \text{out } j\ \mathcal{P}$ implies $n = 0 \wedge s = (j, v)$
- Terminating computations

$$\mathcal{P} /\!\!/ s \rightsquigarrow (j, w) \ := \ \mathcal{P} /\!\!/ s \succ^* (j, w) \wedge \text{out } j\ \mathcal{P}$$

- Termination

$$\mathcal{P} /\!\!/ s\downarrow \ := \ \exists s', \mathcal{P} /\!\!/ s \rightsquigarrow s'$$

Contribution

$$PCP \longrightarrow BPCP \xrightarrow{2} BSM \longrightarrow MM \longrightarrow eILL \longrightarrow ILL$$

BPCP $\preceq$ BSM

Binary stack machines (BSM)

- n stacks of 0s and 1s ($\mathbb{L}\,\mathbb{B}$) for a fixed n
- state of type $(\mathtt{PC}, \vec{v}) \in \mathbb{N} \times (\mathbb{L}\,\mathbb{B})^n$
- instructions (with $\alpha \in [0, n-1]$ and $b \in \mathbb{B}$ and $p, q \in \mathbb{N}$)

$$\mathtt{bsm\_instr} \ ::= \ \mathtt{POP}\ \alpha\ p\ q \mid \mathtt{PUSH}\ \alpha\ b$$

- Step semantics for POP and PUSH (pseudo code)

$$\begin{array}{ll} \mathtt{POP}\ \alpha\ p\ q: & \text{if } \alpha = [] \text{ then } \mathtt{PC} \leftarrow q \\ & \text{if } \alpha = 0 :: \beta \text{ then } \alpha \leftarrow \beta;\ \mathtt{PC} \leftarrow p \\ & \text{if } \alpha = 1 :: \beta \text{ then } \alpha \leftarrow \beta;\ \mathtt{PC} \leftarrow \mathtt{PC} + 1 \\ \mathtt{PUSH}\ \alpha\ b: & \alpha \leftarrow b :: \alpha;\ \mathtt{PC} \leftarrow \mathtt{PC} + 1 \end{array}$$

Binary stack machines (BSM)

- n stacks of 0s and 1s ($\mathbb{L}\,\mathbb{B}$) for a fixed n
- state of type $(\mathtt{PC}, \vec{v}) \in \mathbb{N} \times (\mathbb{L}\,\mathbb{B})^n$
- instructions (with $\alpha \in [0, n-1]$ and $b \in \mathbb{B}$ and $p, q \in \mathbb{N}$)

$$\mathtt{bsm\_instr} ::= \mathtt{POP}\ \alpha\ p\ q \mid \mathtt{PUSH}\ \alpha\ b$$

- Step semantics for POP and PUSH (pseudo code)

$$\begin{array}{ll} \mathtt{POP}\ \alpha\ p\ q: & \text{if } \alpha = [] \text{ then } \mathtt{PC} \leftarrow q \\ & \text{if } \alpha = 0 :: \beta \text{ then } \alpha \leftarrow \beta;\ \mathtt{PC} \leftarrow p \\ & \text{if } \alpha = 1 :: \beta \text{ then } \alpha \leftarrow \beta;\ \mathtt{PC} \leftarrow \mathtt{PC}+1 \\ \mathtt{PUSH}\ \alpha\ b: & \alpha \leftarrow b :: \alpha;\ \mathtt{PC} \leftarrow \mathtt{PC}+1 \end{array}$$

- BSM termination problem: $\boxed{\mathrm{BSM}(n, i, \mathcal{B}, \vec{v}) := (i, \mathcal{B}) \mathbin{/\!/} (i, \vec{v})\downarrow}$

Binary stack machines (BSM)

- n stacks of 0s and 1s ($\mathbb{L}\,\mathbb{B}$) for a fixed n
- state of type $(\mathtt{PC}, \vec{v}) \in \mathbb{N} \times (\mathbb{L}\,\mathbb{B})^n$
- instructions (with $\alpha \in [0, n-1]$ and $b \in \mathbb{B}$ and $p, q \in \mathbb{N}$)

$$\mathtt{bsm\_instr} \ ::= \ \mathtt{POP}\ \alpha\ p\ q \mid \mathtt{PUSH}\ \alpha\ b$$

- Step semantics for POP and PUSH (pseudo code)

$$\begin{array}{ll} \mathtt{POP}\ \alpha\ p\ q: & \text{if } \alpha = [] \text{ then } \mathtt{PC} \leftarrow q \\ & \text{if } \alpha = 0 :: \beta \text{ then } \alpha \leftarrow \beta;\ \mathtt{PC} \leftarrow p \\ & \text{if } \alpha = 1 :: \beta \text{ then } \alpha \leftarrow \beta;\ \mathtt{PC} \leftarrow \mathtt{PC} + 1 \\ \mathtt{PUSH}\ \alpha\ b: & \alpha \leftarrow b :: \alpha;\ \mathtt{PC} \leftarrow \mathtt{PC} + 1 \end{array}$$

- BSM termination problem: $\boxed{\mathrm{BSM}(n, i, \mathcal{B}, \vec{v}) := (i, \mathcal{B}) \mathbin{/\!/} (i, \vec{v}) \downarrow}$

Example (emptying stack α in 3 instructions)

$$i : \mathtt{POP}\ \alpha\ i\ (i+3) \qquad i+1 : \mathtt{PUSH}\ \alpha\ 0 \qquad i+2 : \mathtt{POP}\ \alpha\ i\ i$$

BPCP $\preceq$ BSM

- Iterate all possible lists of card (indices)
- Hard code every card as `PUSH` instructions
- Given a list of cards, compute top and bottom words in two stacks
- Check for those two stacks equality

BPCP $\preceq$ BSM

- Iterate all possible lists of card (indices)
- Hard code every card as PUSH instructions
- Given a list of cards, compute top and bottom words in two stacks
- Check for those two stacks 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 ;     (* JMP i *)
  (* 4+i *)   POP y i q ;
  (* 5+i *)   PUSH y Zero ; POP y q i ;     (* JMP q *)
  (* 7+i *)   POP y q p ;
  (* 8+i *)   PUSH x Zero ; POP x q q ].    (* JMP q *)
```

BPCP $\preceq$ BSM

- Iterate all possible lists of card (indices)
- Hard code every card as PUSH instructions
- Given a list of cards, compute top and bottom words in two stacks
- Check for those two stacks 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 ;     (* JMP i *)
  (* 4+i *)   POP y i q ;
  (* 5+i *)   PUSH y Zero ; POP y q i ;     (* JMP q *)
  (* 7+i *)   POP y q p ;
  (* 8+i *)   PUSH x Zero ; POP x q q ].    (* JMP q *)
```

Lemma (Comparing two distinct stacks for identical content)

When $x \neq y$, for any stack configuration $\vec{v}$, there exists j and $\vec{w}$ s.t.

$$(i, \texttt{compare\_stacks}\ x\ y\ p\ q\ i)\ /\!\!/\ (i, \vec{v}) \succ^* (j, \vec{w})$$

where $j = p$ if $\vec{v}[x] = \vec{v}[y]$ and $j = q$ otherwise. For any $\alpha \notin \{x, y\}$ we have $\vec{w}[\alpha] = \vec{v}[\alpha]$.

Certified Low-Level Compiler

Certified compilation (assumptions)

- model X (resp. Y): language + step semantics
- a simulation: $\bowtie : \mathbb{C}_X \to \mathbb{C}_Y \to \mathbb{P}$
- a certified compiler from model X to model Y

Certified compilation (assumptions)

- model X (resp. Y): language + step semantics
- a simulation: $\bowtie : \mathbb{C}_X \to \mathbb{C}_Y \to \mathbb{P}$
- a certified compiler from model X to model Y
- given a Single Instruction Compiler (SIC):
  - transforms a single X instructions
  - into a list of Y instructions
  - needs a *linker* remapping PC values

Certified compilation (assumptions)

- model X (resp. Y): language + step semantics
- a simulation: $\bowtie : \mathbb{C}_X \to \mathbb{C}_Y \to \mathbb{P}$
- a certified compiler from model X to model Y

- given a Single Instruction Compiler (SIC):
  - transforms a single X instructions
  - into a list of Y instructions
  - needs a *linker* remapping `PC` values

- with the following assumptions:
  - X has total step sem.; Y has deterministic step sem.
  - length of SIC compiled instruction does not depend on linker
  - SIC is sound with respect to $\bowtie$

Certified compilation (results)

- INPUT: X program $\mathcal{P}$ and start target `PC` value $j : \mathbb{N}$

Certified compilation (results)

- INPUT: X program $\mathcal{P}$ and start target `PC` value $j : \mathbb{N}$
- OUTPUT: a linker *lnk* and Y program $\mathcal{Q}$

Certified compilation (results)

- INPUT: X program $\mathcal{P}$ and start target PC value $j : \mathbb{N}$
- OUTPUT: a linker *lnk* and Y program $\mathcal{Q}$
- such that $j = \text{start}\,\mathcal{Q} = \mathit{lnk}(\text{start}\,\mathcal{P})$; $\forall i,\ \text{out}\ i\ \mathcal{P} \to \mathit{lnk}\ i = \text{end}\,\mathcal{Q}$;

Lemma (Soundness)

$$v_1 \bowtie w_1 \wedge \mathcal{P} \mathbin{/\!\!/}_X (i_1, v_1) \leadsto (i_2, v_2)$$
$$\to \exists w_2,\ \ v_2 \bowtie w_2 \wedge \mathcal{Q} \mathbin{/\!\!/}_Y (\mathit{lnk}\ i_1, w_1) \leadsto (\mathit{lnk}\ i_2, w_2)$$

Certified compilation (results)

- INPUT: X program $\mathcal{P}$ and start target `PC` value $j : \mathbb{N}$
- OUTPUT: a linker lnk and Y program $\mathcal{Q}$
- such that $j = \text{start}\,\mathcal{Q} = lnk(\text{start}\,\mathcal{P})$; $\forall i,\ \text{out}\ i\ \mathcal{P} \to lnk\ i = \text{end}\,\mathcal{Q}$;

Lemma (Soundness)

$$v_1 \bowtie w_1 \wedge \mathcal{P} \mathbin{/\!/}_X (i_1, v_1) \leadsto (i_2, v_2)$$
$$\to \exists w_2,\ \ v_2 \bowtie w_2 \wedge \mathcal{Q} \mathbin{/\!/}_Y (lnk\ i_1, w_1) \leadsto (lnk\ i_2, w_2)$$

Lemma (Completeness)

$$v_1 \bowtie w_1 \wedge \mathcal{Q} \mathbin{/\!/}_Y (lnk\ i_1, w_1) \leadsto (j_2, w_2)$$
$$\to \exists\, i_2\, v_2,\ \ v_2 \bowtie w_2 \wedge \mathcal{P} \mathbin{/\!/}_X (i_1, v_1) \leadsto (i_2, v_2) \wedge j_2 = lnk\ i_2.$$

- Completeness essential for non-termination

Contribution

$$PCP \longrightarrow BPCP \longrightarrow BSM \xrightarrow{3} MM \longrightarrow eILL \longrightarrow ILL$$

BSM $\preceq$ MM

Minsky Machines ($\mathbb{N}$ valued register machines)

- n registers of value in $\mathbb{N}$ for a fixed n
- state: $(\texttt{PC}, \vec{v}) \in \mathbb{N} \times \mathbb{N}^n$
- instructions (with $\alpha \in [0, n-1]$ and $p \in \mathbb{N}$)

$$\texttt{mm\_instr} \;::=\; \texttt{INC}\ \alpha \mid \texttt{DEC}\ \alpha\ p$$

- Step semantics for `INC` and `DEC` (pseudo code)

$$\begin{array}{ll} \texttt{INC}\ \alpha: & \alpha \leftarrow \alpha + 1;\ \texttt{PC} \leftarrow \texttt{PC} + 1 \\ \texttt{DEC}\ \alpha\ p: & \text{if } \alpha = 0 \text{ then } \texttt{PC} \leftarrow p \\ & \text{if } \alpha > 0 \text{ then } \alpha \leftarrow \alpha - 1;\ \texttt{PC} \leftarrow \texttt{PC} + 1 \end{array}$$

Minsky Machines ($\mathbb{N}$ valued register machines)

- n registers of value in $\mathbb{N}$ for a fixed n
- state: $(\mathtt{PC}, \vec{v}) \in \mathbb{N} \times \mathbb{N}^n$
- instructions (with $\alpha \in [0, n-1]$ and $p \in \mathbb{N}$)

$$\mathtt{mm\_instr} ::= \mathtt{INC}\ \alpha \mid \mathtt{DEC}\ \alpha\ p$$

- Step semantics for INC and DEC (pseudo code)

$$\begin{array}{ll} \mathtt{INC}\ \alpha: & \alpha \leftarrow \alpha + 1;\ \mathtt{PC} \leftarrow \mathtt{PC} + 1 \\ \mathtt{DEC}\ \alpha\ p: & \text{if } \alpha = 0 \text{ then } \mathtt{PC} \leftarrow p \\ & \text{if } \alpha > 0 \text{ then } \alpha \leftarrow \alpha - 1;\ \mathtt{PC} \leftarrow \mathtt{PC} + 1 \end{array}$$

- $\boxed{MM(n, \mathcal{M}, \vec{v}) := (1, \mathcal{M}) \mathbin{/\!/} (1, \vec{v}) \rightsquigarrow (0, \vec{0})}$ (termination at zero)

Minsky Machines ($\mathbb{N}$ valued register machines)

- n registers of value in $\mathbb{N}$ for a fixed n
- state: $(\mathtt{PC}, \vec{v}) \in \mathbb{N} \times \mathbb{N}^n$
- instructions (with $\alpha \in [0, n-1]$ and $p \in \mathbb{N}$)

$$\mathtt{mm\_instr} ::= \mathtt{INC}\ \alpha \mid \mathtt{DEC}\ \alpha\ p$$

- Step semantics for INC and DEC (pseudo code)

$$\begin{array}{ll} \mathtt{INC}\ \alpha : & \alpha \leftarrow \alpha + 1;\ \mathtt{PC} \leftarrow \mathtt{PC} + 1 \\ \mathtt{DEC}\ \alpha\ p : & \text{if } \alpha = 0 \text{ then } \mathtt{PC} \leftarrow p \\ & \text{if } \alpha > 0 \text{ then } \alpha \leftarrow \alpha - 1;\ \mathtt{PC} \leftarrow \mathtt{PC} + 1 \end{array}$$

- $\boxed{MM(n, \mathcal{M}, \vec{v}) := (1, \mathcal{M}) \mathbin{/\!/} (1, \vec{v}) \leadsto (0, \vec{0})}$ (termination at zero)

Example (transfers α to β in 3 instructions, γ_0 spare register)

$$i : \mathtt{DEC}\ \alpha\ (3+i) \qquad i+1 : \mathtt{INC}\ \beta \qquad i+2 : \mathtt{DEC}\ \gamma_0\ i$$

BSM $\preceq$ MM (simulating stacks)

- Simulation $\bowtie$ between stacks ($\mathbb{L}\,\mathbb{B}$) and $\mathbb{N}$
  - stack 100010 simulated by $1 \cdot 010001$
  - $\mathsf{s2n}\ l : \mathbb{N}$ using: $\quad \mathsf{s2n}\ [] := 1 \quad \mathsf{s2n}\ (b :: l) := b + 2 \cdot \mathsf{s2n}\ l$
  - $\vec{v} \bowtie \vec{w}$ iff for any α, $\mathsf{s2n}(\vec{v}[\alpha]) = \vec{w}[\alpha]$

BSM $\preceq$ MM (simulating stacks)

- Simulation $\bowtie$ between stacks ($\mathbb{L}\,\mathbb{B}$) and $\mathbb{N}$
  - stack 100010 simulated by $1 \cdot 010001$
  - s2n $l : \mathbb{N}$ using: $\quad$ s2n $[] := 1 \quad$ s2n $(b :: l) := b + 2 \cdot$ s2n l
  - $\vec{v} \bowtie \vec{w}$ iff for any α, s2n$(\vec{v}[\alpha]) = \vec{w}[\alpha]$

```
Definition mm_div2 :=
    (*   i *) [ DEC src (6+i) ;
    (* 1+i *)   INC rem ;
    (* 2+i *)   DEC src (i+6) ;
    (* 3+i *)   DEC rem (4+i) ;
    (* 4+i *)   INC quo ;
    (* 5+i *)   DEC rem i ].
```

BSM $\preceq$ MM (simulating stacks)

- Simulation $\bowtie$ between stacks ($\mathbb{L}\,\mathbb{B}$) and $\mathbb{N}$
  - stack 100010 simulated by $1 \cdot 010001$
  - $\text{s2n}\ l : \mathbb{N}$ using: $\quad \text{s2n}\ [] := 1 \quad \text{s2n}\ (b :: l) := b + 2 \cdot \text{s2n}\ l$
  - $\vec{v} \bowtie \vec{w}$ iff for any α, $\text{s2n}(\vec{v}[\alpha]) = \vec{w}[\alpha]$

```
Definition mm_div2 :=
    (*   i *) [ DEC src (6+i) ;
    (* 1+i *)   INC rem ;
    (* 2+i *)   DEC src (i+6) ;
    (* 3+i *)   DEC rem (4+i) ;
    (* 4+i *)   INC quo ;
    (* 5+i *)   DEC rem i ].
```

Lemma (Euclidian division by 2 of register src)

When $\texttt{quo} \neq \texttt{rem} \neq \texttt{src}$, $b \in \{0, 1\}$ *and* $k \in \mathbb{N}$

$$\vec{v}[\texttt{quo}] = 0 \wedge \vec{v}[\texttt{rem}] = 0 \wedge \vec{v}[\texttt{src}] = b + 2.k$$
$$\rightarrow (i, \texttt{mm\_div2}) \mathbin{/\!/} (i, \vec{v}) \succ^* (6 + i, \vec{v}[\texttt{src} := 0, \texttt{quo} := k, \texttt{rem} := b])$$

BSM $\preceq$ MM (simulating instructions)

- We implement an instruction compiler (BSM SIC)
  - simulating `PUSH` and `POP` operations
  - using `mm_div2`, `mm_mul2`, ...
  - we need two spare MM registers
  - n stacks, $2 + n$ registers

BSM $\preceq$ MM (simulating instructions)

- We implement an instruction compiler (BSM SIC)
  - simulating `PUSH` and `POP` operations
  - using `mm_div2`, `mm_mul2`, ...
  - we need two spare MM registers
  - n stacks, $2 + n$ registers
- As input for our certified low-level compiler
  - from (i, P), a n stacks BSM-program
  - we compute a $2 + n$ registers MM-program `bsm_mm`
  - which simulates termination

BSM $\preceq$ MM (simulating instructions)

- We implement an instruction compiler (BSM SIC)
  - simulating `PUSH` and `POP` operations
  - using `mm_div2`, `mm_mul2`, ...
  - we need two spare MM registers
  - n stacks, $2 + n$ registers
- As input for our certified low-level compiler
  - from (i, P), a n stacks BSM-program
  - we compute a $2 + n$ registers MM-program `bsm_mm`
  - which simulates termination

Lemma (BSM termination simulated by MM termination)

for any $\vec{v} \in \mathbb{N}^n$,

$$(i, P) \mathbin{/\!/} (i, \vec{v}) \downarrow \quad \leftrightarrow \quad (1, \texttt{bsm\_mm}) \mathbin{/\!/} (1, 0 :: 0 :: \vec{w}) \rightsquigarrow (0, \vec{0})$$

where $\vec{w} = \texttt{vec\_map}\ \mathrm{s2n}\ \vec{v}$

Contribution

$$PCP \longrightarrow BPCP \longrightarrow BSM \longrightarrow MM \xrightarrow{4} eILL \xrightarrow{5} ILL$$

$\mathsf{MM} \preceq \mathsf{eILL} \preceq \mathsf{ILL}$

Intuitionistic Linear Logic

Definition ($\mathbb{S}_{\mathsf{ILL}}$ sequent calculus for the $(!, \multimap, \&)$ fragment)

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

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

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

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

- $\mathsf{ILL}(\Gamma, A) := \mathsf{provable}(\Gamma \vdash A)$
- the reduction for MM occurs in the eILL sub-fragment

Elementary ILL (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`
  - $(a \mathbin{\&} b) \multimap c$, correponding to `FORK`

Elementary ILL (eILL)

- Elementary sequents: $!\,\Sigma, g_1, \ldots, g_k \vdash d$ (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`

Definition ($\mathrm{G}_{\mathsf{eILL}}$ goal directed rules for eILL)

$$\frac{}{!\,\Sigma, a \vdash a}\ \langle \mathrm{Ax} \rangle \qquad \frac{!\,\Sigma, \Gamma \vdash a \quad !\,\Sigma, \Delta \vdash b}{!\,\Sigma, \Gamma, \Delta \vdash c}\ a \multimap (b \multimap c) \in \Sigma$$

$$\frac{!\,\Sigma, a, \Gamma \vdash b}{!\,\Sigma, \Gamma \vdash c}\ (a \multimap b) \multimap c \in \Sigma \qquad \frac{!\,\Sigma, \Gamma \vdash a \quad !\,\Sigma, \Gamma \vdash b}{!\,\Sigma, \Gamma \vdash c}\ (a \,\&\, b) \multimap c \in \Sigma$$

Elementary ILL (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`

Definition ($\mathrm{G}_{\mathsf{eILL}}$ goal directed rules for eILL)

$$\frac{}{!\,\Sigma, a \vdash a}\ \langle\mathsf{Ax}\rangle \qquad \frac{!\,\Sigma, \Gamma \vdash a \quad !\,\Sigma, \Delta \vdash b}{!\,\Sigma, \Gamma, \Delta \vdash c}\ a \multimap (b \multimap c) \in \Sigma$$

$$\frac{!\,\Sigma, a, \Gamma \vdash b}{!\,\Sigma, \Gamma \vdash c}\ (a \multimap b) \multimap c \in \Sigma \qquad \frac{!\,\Sigma, \Gamma \vdash a \quad !\,\Sigma, \Gamma \vdash b}{!\,\Sigma, \Gamma \vdash c}\ (a \,\&\, b) \multimap c \in \Sigma$$

- Sound and complete w.r.t. $\mathrm{S}_{\mathsf{ILL}}$ for eILL sequents

Elementary ILL (eILL)

- Elementary sequents: $!\,\Sigma, g_1, \ldots, g_k \vdash d$ (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`

Definition ($\mathrm{G}_{\mathsf{eILL}}$ goal directed rules for eILL)

$$\frac{}{!\,\Sigma, a \vdash a}\ \langle \mathsf{Ax} \rangle \qquad \frac{!\,\Sigma, \Gamma \vdash a \quad !\,\Sigma, \Delta \vdash b}{!\,\Sigma, \Gamma, \Delta \vdash c}\ a \multimap (b \multimap c) \in \Sigma$$

$$\frac{!\,\Sigma, a, \Gamma \vdash b}{!\,\Sigma, \Gamma \vdash c}\ (a \multimap b) \multimap c \in \Sigma \qquad \frac{!\,\Sigma, \Gamma \vdash a \quad !\,\Sigma, \Gamma \vdash b}{!\,\Sigma, \Gamma \vdash c}\ (a \,\&\, b) \multimap c \in \Sigma$$

- Sound and complete w.r.t. $\mathrm{S}_{\mathsf{ILL}}$ for eILL sequents
- Trivial Phase Semantics (commutative monoid, closure is identity)
  - $\mathrm{S}_{\mathsf{ILL}}$ and $\mathrm{G}_{\mathsf{eILL}}$ sound for TPS
- The reduction eILL $\preceq$ ILL is the identity map

Encoding Minsky machines in eILL

- Given $\mathcal{M}$ as a list of MM instructions
  - for every register $i \in [0, n-1]$ in $\mathcal{M}$, two logical variables x_i and $\overline{x}_i$
  - for every position/state ($\texttt{PC} = i$) in $\mathcal{M}$, a variable q_i

$$\{x_0, \ldots, x_{n-1}\} \uplus \{\overline{x}_0, \ldots, \overline{x}_{n-1}\} \uplus \{q_0, q_1, \ldots\}$$

Encoding Minsky machines in eILL

- Given $\mathcal{M}$ as a list of MM instructions
  - for every register $i \in [0, n-1]$ in $\mathcal{M}$, two logical variables x_i and $\overline{x}_i$
  - for every position/state ($\mathtt{PC} = i$) in $\mathcal{M}$, a variable q_i

$$\{x_0, \ldots, x_{n-1}\} \uplus \{\overline{x}_0, \ldots, \overline{x}_{n-1}\} \uplus \{q_0, q_1, \ldots\}$$

- a computation $\mathcal{M} \mathbin{/\!\!/} (i, \vec{v}) \rightsquigarrow (0, \vec{0})$ is represented by $!\,\Sigma_{\mathcal{M}}; \Delta_{\vec{v}} \vdash q_i$
  - where if $\vec{v} = (p_0, \ldots, p_{n-1})$ then $\Delta_{\vec{v}} = p_0.x_0, \ldots, p_{n-1}.x_{n-1}$
  - the commands in $\Sigma_{\mathcal{M}}$ are determined by instructions in $\mathcal{M}$

$$\begin{array}{rcl}
\Sigma_{\mathcal{M}} & = & \{(q_0 \multimap q_0) \multimap q_0\} \\
& \cup & \{x_\beta \multimap (\overline{x}_\alpha \multimap \overline{x}_\alpha), (\overline{x}_\alpha \multimap \overline{x}_\alpha) \multimap \overline{x}_\alpha \mid \alpha \neq \beta \in [0, n-1]\} \\
& \cup & \{(x_\alpha \multimap q_{i+1}) \multimap q_i \mid i : \mathtt{INC}\ \alpha \in \mathcal{M}\} \\
& \cup & \{(\overline{x}_\alpha \,\&\, q_j) \multimap q_i, x_\alpha \multimap (q_{i+1} \multimap q_i) \mid i : \mathtt{DEC}\ \alpha\ j \in \mathcal{M}\}
\end{array}$$

Encoding Minsky machines in eILL

- Given $\mathcal{M}$ as a list of MM instructions
  - for every register $i \in [0, n-1]$ in $\mathcal{M}$, two logical variables x_i and $\overline{x}_i$
  - for every position/state $(\mathtt{PC} = i)$ in $\mathcal{M}$, a variable q_i

$$\{x_0, \ldots, x_{n-1}\} \uplus \{\overline{x}_0, \ldots, \overline{x}_{n-1}\} \uplus \{q_0, q_1, \ldots\}$$

- a computation $\mathcal{M} \mathbin{/\!/} (i, \vec{v}) \rightsquigarrow (0, \vec{0})$ is represented by $!\,\Sigma_{\mathcal{M}}; \Delta_{\vec{v}} \vdash q_i$
  - where if $\vec{v} = (p_0, \ldots, p_{n-1})$ then $\Delta_{\vec{v}} = p_0.x_0, \ldots, p_{n-1}.x_{n-1}$
  - the commands in $\Sigma_{\mathcal{M}}$ are determined by instructions in $\mathcal{M}$

$$\begin{array}{rcl}
\Sigma_{\mathcal{M}} & = & \{(q_0 \multimap q_0) \multimap q_0\} \\
& \cup & \{x_\beta \multimap (\overline{x}_\alpha \multimap \overline{x}_\alpha), (\overline{x}_\alpha \multimap \overline{x}_\alpha) \multimap \overline{x}_\alpha \mid \alpha \neq \beta \in [0, n-1]\} \\
& \cup & \{(x_\alpha \multimap q_{i+1}) \multimap q_i \mid i : \mathtt{INC}\ \alpha \in \mathcal{M}\} \\
& \cup & \{(\overline{x}_\alpha \,\&\, q_j) \multimap q_i, x_\alpha \multimap (q_{i+1} \multimap q_i) \mid i : \mathtt{DEC}\ \alpha\ j \in \mathcal{M}\}
\end{array}$$

Theorem (Simulating MM termination at zero with $\mathrm{G}_{\mathsf{eILL}}$ entailment)

$$\mathcal{M} \mathbin{/\!/} (i, \vec{v}) \rightsquigarrow (0, \vec{0}) \quad \leftrightarrow \quad !\,\Sigma_{\mathcal{M}}, \Delta_{\vec{v}} \vdash q_i$$

Encoding Minsky machines in eILL

- Given $\mathcal{M}$ as a list of MM instructions
  - for every register $i \in [0, n-1]$ in $\mathcal{M}$, two logical variables x_i and $\overline{x}_i$
  - for every position/state ($\texttt{PC} = i$) in $\mathcal{M}$, a variable q_i

$$\{x_0, \ldots, x_{n-1}\} \uplus \{\overline{x}_0, \ldots, \overline{x}_{n-1}\} \uplus \{q_0, q_1, \ldots\}$$

- a computation $\mathcal{M} /\!\!/ (i, \vec{v}) \rightsquigarrow (0, \vec{0})$ is represented by $!\,\Sigma_{\mathcal{M}}; \Delta_{\vec{v}} \vdash q_i$
  - where if $\vec{v} = (p_0, \ldots, p_{n-1})$ then $\Delta_{\vec{v}} = p_0.x_0, \ldots, p_{n-1}.x_{n-1}$
  - the commands in $\Sigma_{\mathcal{M}}$ are determined by instructions in $\mathcal{M}$

$$\begin{array}{rcl}
\Sigma_{\mathcal{M}} & = & \{(q_0 \multimap q_0) \multimap q_0\} \\
& \cup & \{x_\beta \multimap (\overline{x}_\alpha \multimap \overline{x}_\alpha), (\overline{x}_\alpha \multimap \overline{x}_\alpha) \multimap \overline{x}_\alpha \mid \alpha \neq \beta \in [0, n-1]\} \\
& \cup & \{(x_\alpha \multimap q_{i+1}) \multimap q_i \mid i : \texttt{INC}\ \alpha \in \mathcal{M}\} \\
& \cup & \{(\overline{x}_\alpha \,\&\, q_j) \multimap q_i, x_\alpha \multimap (q_{i+1} \multimap q_i) \mid i : \texttt{DEC}\ \alpha\ j \in \mathcal{M}\}
\end{array}$$

Theorem (Simulating MM termination at zero with $\mathrm{G}_{\mathsf{eILL}}$ entailment)

$$\mathcal{M} /\!\!/ (i, \vec{v}) \rightsquigarrow (0, \vec{0}) \quad \leftrightarrow \quad !\,\Sigma_{\mathcal{M}}, \Delta_{\vec{v}} \vdash q_i$$

- Hence the reduction MM $\preceq$ eILL

MM to eILL, (continued)

Increment:

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

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 $x = 0$:

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

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

Zero test $x \notin \Delta$ in eILL

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

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

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

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

Full reduction

Theorem

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

Full reduction

Theorem

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

other direction by soundness of TPS ($[\![A]\!] : \mathbb{N}^n \to \mathbb{P}$):

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

Wrap-up of this chain of reduction

Reductions:

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

Wrap-up of this chain of reduction

Reductions:

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

Low verification overhead

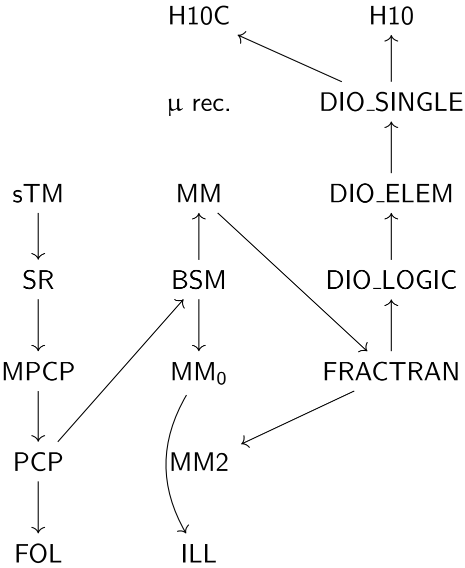
Wrap-up of this chain of reduction

Reductions:

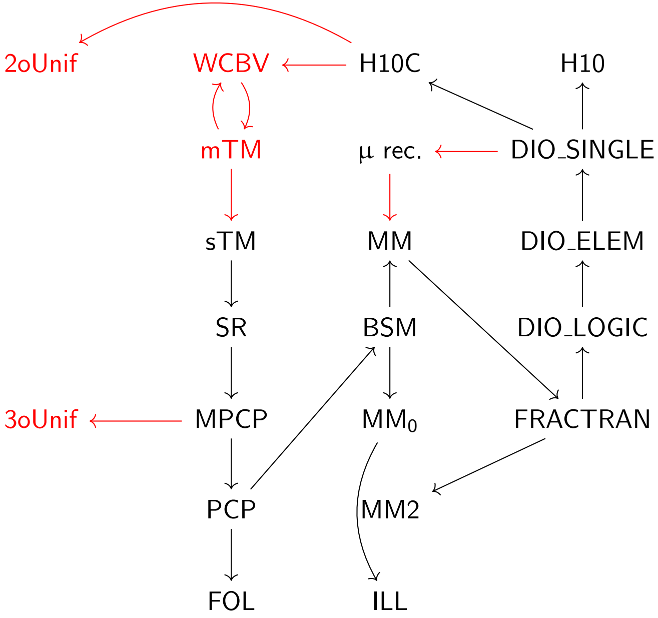
- PCP to BPCP: trivial binary encoding
- BPCP to BSM: verified exhaustive search
- BSM to MM: certified compiler between low-level languages
- MM to eILL: elegant encoding of computational model in logics
- eILL to ILL: faithfull embedding

Low verification overhead
(compared to detailed paper proofs)

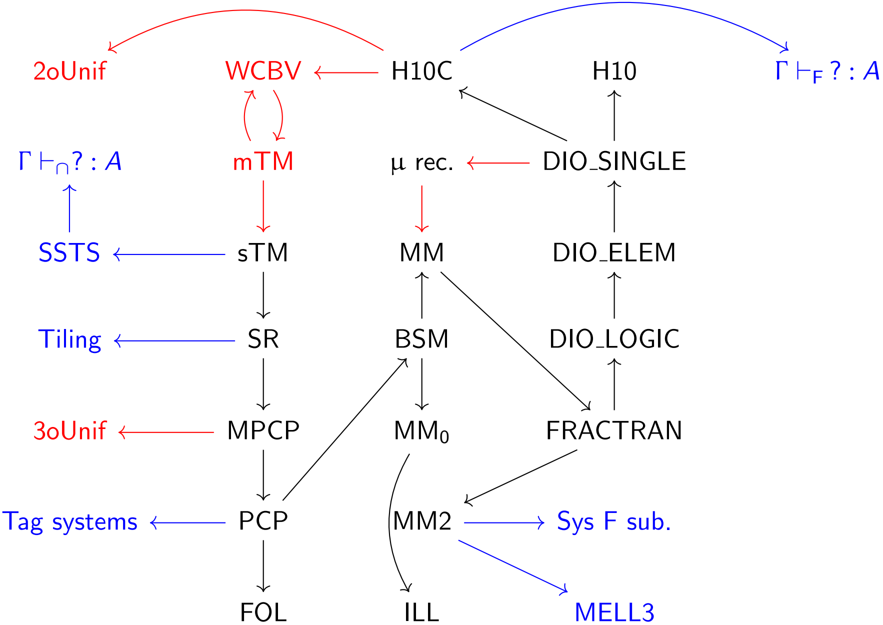
A library of undecidable problems in Coq



A library of undecidable problems in Coq



A library of undecidable problems in Coq



Papers

- Hilbert's Tenth Problem in Coq. Dominique Larchey-Wendling and Yannick Forster. Technical report (2019).
- Certified Undecidability of Intuitionistic Linear Logic via Binary Stack Machines and Minsky Machines. Yannick Forster and Dominique Larchey-Wendling. CPP '19.
- On Synthetic Undecidability in Coq, with an Application to the Entscheidungsproblem. Yannick Forster, Dominik Kirst, and Gert Smolka. CPP '19.
- Verification of PCP-Related Computational Reductions in Coq. Yannick Forster, Edith Heiter, and Gert Smolka. ITP 2018.
- Call-by-Value Lambda Calculus as a Model of Computation in Coq. Yannick Forster and Gert Smolka. Journal of Automated Reasoning (2018)

Conclusion

More future work:

- Realisability model of the calculus of inductive constructions witnessing (the propositional version) of excluded middle
- Automated translation of Coq function definitions into a concrete model of computation (e.g. call-by-value lambda calculus)

Conclusion

More future work:

- Realisability model of the calculus of inductive constructions witnessing (the propositional version) of excluded middle
- Automated translation of Coq function definitions into a concrete model of computation (e.g. call-by-value lambda calculus)

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

Conclusion

More future work:

- Realisability model of the calculus of inductive constructions witnessing (the propositional version) of excluded middle
- Automated translation of Coq function definitions into a concrete model of computation (e.g. call-by-value lambda calculus)

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

`https://github.com/uds-psl/coq-library-undecidability`

Conclusion

More future work:

- Realisability model of the calculus of inductive constructions witnessing (the propositional version) of excluded middle
- Automated translation of Coq function definitions into a concrete model of computation (e.g. call-by-value lambda calculus)

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

`https://github.com/uds-psl/coq-library-undecidability`

Questions?