

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

Yannick Forster and Dominique Larchey-Wendling

CPP 2019  
January 14



# Decidability

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

Classically

Fix a model of computation  $M$ :  
there is a decider in  $M$

For the cbv  $\lambda$ -calculus  $\exists u : \mathbf{T}. \forall x : X. (u\bar{x} \triangleright T \wedge P_x) \vee (u\bar{x} \triangleright F \wedge \neg P_x)$

# Decidability

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

Classically

Fix a model of computation  $M$ :  
there is a decider in  $M$

For the cbv  $\lambda$ -calculus

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

Type Theory

$\exists f : X \rightarrow \mathbb{B}. \forall x : X. P_x \leftrightarrow f_x = \text{true}$

# Decidability

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

Classically

Fix a model of computation  $M$ :  
there is a decider in  $M$

For the cbv  $\lambda$ -calculus

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

Type Theory

$\exists f : X \rightarrow \mathbb{B}. \forall x : X. P_x \leftrightarrow f_x = \text{true}$

dependent version

(Coq, Agda, Lean, ...)

$\text{dec } P := \forall x : X. \{P_x\} + \{\neg P_x\}$

# Undecidability

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

Classically

If there is no decider  $u$  in  $M$

# Undecidability

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

Classically

If there is no decider  $u$  in  $M$

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

# Undecidability

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

Classically

If there is no decider  $u$  in  $M$

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

Type Theory

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

# Undecidability

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

Classically

If there is no decider  $u$  in  $M$

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

Type Theory

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



# Undecidability

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

Classically

If there is no decider  $u$  in  $M$

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

Type Theory

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

In reality: most proofs are by reduction

## Definition

$P$  undecidable := Halting problem reduces to  $P$

# Inductive Undecidability via reductions

- a problem  $(X, P) : \Sigma(X : \text{Type}), X \rightarrow \text{Prop}$
- Inductive definition of undecidability over  $\Sigma_X(X \rightarrow \text{Prop})$

$$\frac{}{\text{undec } \mathit{Halt}} \qquad \frac{\text{dec } Q \rightarrow \text{dec } P \quad \text{undec } P}{\text{undec } Q}$$

# Inductive Undecidability via reductions

- a problem  $(X, P) : \Sigma(X : \text{Type}), X \rightarrow \text{Prop}$
- Inductive definition of undecidability over  $\Sigma_X(X \rightarrow \text{Prop})$

$$\frac{}{\text{undec } \textit{Halt}} \qquad \frac{\text{dec } Q \rightarrow \text{dec } P \quad \text{undec } P}{\text{undec } Q}$$

Lemma (Incompatibility between decidability and undecidability)

*If dec P and undec P then dec Halt*

# Inductive Undecidability via reductions

- a problem  $(X, P) : \Sigma(X : \text{Type}), X \rightarrow \text{Prop}$
- Inductive definition of undecidability over  $\Sigma_X(X \rightarrow \text{Prop})$

$$\frac{}{\text{undec } \textit{Halt}} \qquad \frac{\text{dec } Q \rightarrow \text{dec } P \quad \text{undec } P}{\text{undec } Q}$$

## Lemma (Incompatibility between decidability and undecidability)

*If dec P and undec P then dec Halt*

- Turing Reductions:  $\text{dec } Q \rightarrow \text{dec } P$
- Many-one reduction from  $(X, P)$  to  $(Y, Q)$ 
  - ▶ computable function  $f : X \rightarrow Y$  s.t.  $\forall x. P x \leftrightarrow Q(f x)$
  - ▶ “computable” requirement dropped in CTT
  - ▶ We write  $P \preceq Q$  when such reduction exists

# An undecidability proof for Intuitionistic Linear Logic

# An undecidability proof for Intuitionistic Linear Logic

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

Dominique Larchey-Wendling<sup>†</sup> and Didier Galmiche<sup>‡</sup>  
LORIA – CNRS<sup>†</sup> – UHP Nancy<sup>‡</sup> 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-

# An undecidability proof for Intuitionistic Linear Logic

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

Dominique Larchey-Wendling<sup>1</sup> and Didier Galmiche<sup>2</sup>  
LORIA – CNRS<sup>1</sup> – UHP Nancy<sup>2</sup> 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<sup>(66)</sup>, 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.

# An undecidability proof for Intuitionistic Linear Logic

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

Dominique Larchey-Wendling<sup>1</sup> and Didier Galmiche<sup>2</sup>  
LORIA – CNRS<sup>1</sup> – UHP Nancy<sup>2</sup> 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<sup>(66)</sup>, 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.

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



# An undecidability proof for Intuitionistic Linear Logic

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

Dominique Larchey-Wendling<sup>1</sup> and Didier Galmiche<sup>2</sup>  
LORIA – CNRS<sup>1</sup> – UHP Nancy<sup>2</sup> 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<sup>(66)</sup>, 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.

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

# 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.<sup>[1]</sup> Because it is simpler than the [halting problem](#) and the *Entscheidungsproblem* it is often used in proofs of undecidability.

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{CPP}{C}$
-----------------

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{CPP}{C}$	$\frac{19i}{PP}$
-----------------	------------------

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{CPP}{C}$	$\frac{19i}{PP}$	$\frac{n}{19in}$
-----------------	------------------	------------------

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{CPP}{C}$	$\frac{19i}{PP}$	$\frac{n}{19in}$	$\frac{Ca}{-}$
-----------------	------------------	------------------	----------------

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{CPP}{C}$	$\frac{19i}{PP}$	$\frac{n}{19in}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
-----------------	------------------	------------------	----------------	------------------



$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{CPP}{C}$	$\frac{19i}{PP}$	$\frac{n}{19in}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$	$\frac{Ca}{-}$
-----------------	------------------	------------------	----------------	------------------	----------------

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{CPP}{C}$	$\frac{19i}{PP}$	$\frac{n}{19in}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$	$\frac{Ca}{-}$	$\frac{is}{ais}$
-----------------	------------------	------------------	----------------	------------------	----------------	------------------

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{\quad}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	--------------------	------------------

$\frac{CPP}{C}$	$\frac{19i}{PP}$	$\frac{n}{19in}$	$\frac{Ca}{\quad}$	$\frac{s}{CasC}$	$\frac{Ca}{\quad}$	$\frac{is}{ais}$
-----------------	------------------	------------------	--------------------	------------------	--------------------	------------------

$$\frac{CPP19inCasCais}{CPP19inCasCais}$$

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{CPP}{C}$	$\frac{19i}{PP}$	$\frac{n}{19in}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$	$\frac{Ca}{-}$	$\frac{is}{ais}$
-----------------	------------------	------------------	----------------	------------------	----------------	------------------

$$\frac{CPP19inCasCais}{CPP19inCasCais}$$

- 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

$\frac{n}{19in}$	$\frac{CPP}{C}$	$\frac{is}{ais}$	$\frac{xuz}{ofze}$	$\frac{19i}{PP}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$
------------------	-----------------	------------------	--------------------	------------------	----------------	------------------

$\frac{CPP}{C}$	$\frac{19i}{PP}$	$\frac{n}{19in}$	$\frac{Ca}{-}$	$\frac{s}{CasC}$	$\frac{Ca}{-}$	$\frac{is}{ais}$
-----------------	------------------	------------------	----------------	------------------	----------------	------------------

$$\frac{CPP19inCasCais}{CPP19inCasCais}$$

- 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

$$\begin{aligned} \square^1 &:= \epsilon & \square^2 &:= \epsilon \\ (x/y :: A)^1 &:= x(A^1) & (x/y :: A)^2 &:= y(A^2) \end{aligned}$$

$$PCP(R) := \exists A \subseteq R. A \neq \square \wedge A^1 = A^2$$

# Contribution

$$PCP \xrightarrow{1} BPCP \longrightarrow BSM \longrightarrow MM \longrightarrow eILL \longrightarrow ILL$$

PCP  $\preceq$  BPCP

PCP  $\leq$  BPCP

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Define inverse function  $g$ , easy

# Low-level Code

# Code and subcode

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

$$i : \rho_0; \quad i + 1 : \rho_1; \quad \dots \quad i + n - 1 : \rho_{n-1};$$

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

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

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

# Small Step Semantics for Code

- Instructions as state transformers
- states  $(i, v)$ :  $i$  is PC value and  $v : \mathbb{C}$  a configuration
- a step relation  $\rho // (i_1, v_1) \succ (i_2, v_2)$ 
  - ▶ instruction  $\rho$  at position  $i_1$  transforms state  $(i_1, v_1)$  into  $(i_2, v_2)$
- extends to codes:  $(i, P) // (i_1, v_1) \succ^n (i_2, v_2)$  means
  - ▶ Code  $(i, P)$  transforms state  $(i_1, v_1)$  into  $(i_2, v_2)$  in  $n$  steps
- described by two inductive rules

$$\frac{\overline{(i, P) // (i_1, v_1) \succ^0 (i_1, v_1)} \quad (i_1, [\rho]) <_{sc} (i, P) \quad \rho // (i_1, v_1) \succ (i_2, v_2) \quad (i, P) // (i_2, v_2) \succ^n (i_3, v_3)}{(i, P) // (i_1, v_1) \succ^{n+1} (i_3, v_3)}$$

# 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  $n = 0 \wedge s = (j, v)$
- reflexive and transitive closure of step relation

$$\mathcal{P} // s \succ^* s' := \exists n, \mathcal{P} // s \succ^n s'$$

- 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)

Example (emptying stack  $\alpha$  in 3 instructions)

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

# Binary stack machines (BSM)

## Example (emptying stack $\alpha$ in 3 instructions)

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

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

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

- Step semantics for POP and PUSH (pseudo code)

$\text{POP } \alpha \ p \ q$ :    if  $\alpha = \square$  then  $\text{PC} \leftarrow q$   
                          if  $\alpha = 0 :: \beta$  then  $\alpha \leftarrow \beta$ ;  $\text{PC} \leftarrow p$   
                          if  $\alpha = 1 :: \beta$  then  $\alpha \leftarrow \beta$ ;  $\text{PC} \leftarrow \text{PC} + 1$

$\text{PUSH } \alpha \ b$ :       $\alpha \leftarrow b :: \alpha$ ;  $\text{PC} \leftarrow \text{PC} + 1$

# Binary stack machines (BSM)

## Example (emptying stack $\alpha$ in 3 instructions)

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

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

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

- Step semantics for POP and PUSH (pseudo code)

$\text{POP } \alpha \ p \ q$ : if  $\alpha = \square$  then  $\text{PC} \leftarrow q$   
if  $\alpha = 0 :: \beta$  then  $\alpha \leftarrow \beta$ ;  $\text{PC} \leftarrow p$   
if  $\alpha = 1 :: \beta$  then  $\alpha \leftarrow \beta$ ;  $\text{PC} \leftarrow \text{PC} + 1$

$\text{PUSH } \alpha \ b$ :  $\alpha \leftarrow b :: \alpha$ ;  $\text{PC} \leftarrow \text{PC} + 1$

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

## 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.

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

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:  $\bowtie : \mathbb{C}_X \rightarrow \mathbb{C}_Y \rightarrow \text{Prop}$
- a certified compiler from model  $X$  to model  $Y$

# Certified compilation (assumptions)

- model  $X$  (resp.  $Y$ ): language + step semantics
- a simulation:  $\bowtie : \mathbb{C}_X \rightarrow \mathbb{C}_Y \rightarrow \text{Prop}$
- 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:  $\bowtie : \mathbb{C}_X \rightarrow \mathbb{C}_Y \rightarrow \text{Prop}$
- 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  $j = \text{start } \mathcal{Q} = lnk(\text{start } \mathcal{P})$ ;  $\forall i, \text{ out } i \mathcal{P} \rightarrow lnk i = \text{end } \mathcal{Q}$ ;

### Lemma (Soundness)

$$\begin{aligned} & v_1 \bowtie w_1 \wedge \mathcal{P} //_X (i_1, v_1) \rightsquigarrow (i_2, v_2) \\ \rightarrow & \exists w_2, v_2 \bowtie w_2 \wedge \mathcal{Q} //_Y (lnk i_1, w_1) \rightsquigarrow (lnk i_2, w_2) \end{aligned}$$

## 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  $j = \text{start } \mathcal{Q} = lnk(\text{start } \mathcal{P})$ ;  $\forall i, \text{ out } i \mathcal{P} \rightarrow lnk i = \text{end } \mathcal{Q}$ ;

### Lemma (Soundness)

$$\begin{aligned} & v_1 \bowtie w_1 \wedge \mathcal{P} //_X (i_1, v_1) \rightsquigarrow (i_2, v_2) \\ \rightarrow & \exists w_2, v_2 \bowtie w_2 \wedge \mathcal{Q} //_Y (lnk i_1, w_1) \rightsquigarrow (lnk i_2, w_2) \end{aligned}$$

### Lemma (Completeness)

$$\begin{aligned} & v_1 \bowtie w_1 \wedge \mathcal{Q} //_Y (lnk i_1, w_1) \rightsquigarrow (j_2, w_2) \\ \rightarrow & \exists i_2 v_2, v_2 \bowtie w_2 \wedge \mathcal{P} //_X (i_1, v_1) \rightsquigarrow (i_2, v_2) \wedge j_2 = lnk i_2. \end{aligned}$$

- 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)

Example (transfert  $\alpha$  to  $\beta$  in 3 instructions,  $\gamma_0$  spare register)

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

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

Example (transfert  $\alpha$  to  $\beta$  in 3 instructions,  $\gamma_0$  spare register)

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

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

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

- Step semantics for INC and DEC (pseudo code)

$\text{INC } \alpha : \quad \alpha \leftarrow \alpha + 1; \text{PC} \leftarrow \text{PC} + 1$

$\text{DEC } \alpha p : \quad \text{if } \alpha = 0 \text{ then } \text{PC} \leftarrow p$   
 $\quad \text{if } \alpha > 0 \text{ then } \alpha \leftarrow \alpha - 1; \text{PC} \leftarrow \text{PC} + 1$

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

Example (transfer  $\alpha$  to  $\beta$  in 3 instructions,  $\gamma_0$  spare register)

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

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

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

- Step semantics for INC and DEC (pseudo code)

$\text{INC } \alpha : \quad \alpha \leftarrow \alpha + 1; \text{PC} \leftarrow \text{PC} + 1$

$\text{DEC } \alpha p : \quad \text{if } \alpha = 0 \text{ then } \text{PC} \leftarrow p$   
 $\quad \quad \quad \text{if } \alpha > 0 \text{ then } \alpha \leftarrow \alpha - 1; \text{PC} \leftarrow \text{PC} + 1$

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

## BSM $\preceq$ MM (simulating stacks)

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

# BSM $\preceq$ MM (simulating stacks)

## ■ Simulation $\bowtie$ between stacks ( $\mathbb{L}\mathbb{B}$ ) and $\mathbb{N}$

- ▶ stack 100010 simulated by  $1 \cdot 010001$
- ▶  $s2n / : \mathbb{N}$  using:  $s2n [] := 1$   $s2n (b :: l) := b + 2 \cdot s2n l$
- ▶  $\vec{v} \bowtie \vec{w}$  iff for any  $\alpha$ ,  $s2n(\vec{v}[\alpha]) = \vec{w}[\alpha]$

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 · 010001
- ▶  $s2n \ / \ : \ \mathbb{N}$  using:  $s2n \ [] := 1$   $s2n \ (b :: l) := b + 2 \cdot s2n \ l$
- ▶  $\vec{v} \bowtie \vec{w}$  iff for any  $\alpha$ ,  $s2n(\vec{v}[\alpha]) = \vec{w}[\alpha]$

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

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

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

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

where  $\vec{w} = \text{vec\_map } s2n \vec{v}$

# Contribution

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

$MM \preceq eILL \preceq ILL$

# Intuitionistic Linear Logic

Definition ( $S_{ILL}$  sequent calculus for the  $(!, \multimap, \&)$  fragment)

$$\begin{array}{c} \frac{}{A \vdash A} \text{ [id]} \quad \frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B} \text{ [cut]} \\ \\ \frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} \text{ [!}_L\text{]} \quad \frac{! \Gamma \vdash B}{! \Gamma \vdash !B} \text{ [!}_R\text{]} \quad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B} \text{ [w]} \quad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} \text{ [c]} \\ \\ \frac{\Gamma, A \vdash C}{\Gamma, A \& B \vdash C} \text{ [&}_L^1\text{]} \quad \frac{\Gamma, B \vdash C}{\Gamma, A \& B \vdash C} \text{ [&}_L^2\text{]} \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} \text{ [&}_R\text{]} \\ \\ \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C} \text{ [}\multimap\text{}_L\text{]} \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \text{ [}\multimap\text{}_R\text{]} \end{array}$$

- Full Linear Logic faithfully embedded by  $((\cdot) \multimap b) \multimap b$  translation
- $ILL(\Gamma, A) := \text{provable}(\Gamma \vdash A)$
- the reduction for MM occurs in the eILL sub-fragment

# Elementary ILL (eILL)

- Elementary sequents:  $! \Sigma, g_1, \dots, g_k \vdash d$  ( $g_i, a, b, c, d$  variables)
- $\Sigma$  contains *commands*:
  - ▶  $(a \multimap b) \multimap c$ , corresponding to INC
  - ▶  $a \multimap (b \multimap c)$ , corresponding to DEC
  - ▶  $(a \& b) \multimap c$ , corresponding to FORK

# Elementary ILL (eILL)

- Elementary sequents:  $! \Sigma, g_1, \dots, g_k \vdash d$  ( $g_i, a, b, c, d$  variables)
- $\Sigma$  contains *commands*:
  - ▶  $(a \multimap b) \multimap c$ , corresponding to INC
  - ▶  $a \multimap (b \multimap c)$ , corresponding to DEC
  - ▶  $(a \& b) \multimap c$ , corresponding to FORK

## Definition ( $G_{\text{eILL}}$ goal directed rules for eILL)

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

# Elementary ILL (eILL)

- Elementary sequents:  $! \Sigma, g_1, \dots, g_k \vdash d$  ( $g_i, a, b, c, d$  variables)
- $\Sigma$  contains *commands*:
  - ▶  $(a \multimap b) \multimap c$ , corresponding to INC
  - ▶  $a \multimap (b \multimap c)$ , corresponding to DEC
  - ▶  $(a \& b) \multimap c$ , corresponding to FORK

## Definition ( $G_{\text{eILL}}$ goal directed rules for eILL)

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

- Sound and complete w.r.t.  $S_{\text{ILL}}$  for eILL sequents



# Elementary ILL (eILL)

- Elementary sequents:  $! \Sigma, g_1, \dots, g_k \vdash d$  ( $g_i, a, b, c, d$  variables)
- $\Sigma$  contains *commands*:
  - ▶  $(a \multimap b) \multimap c$ , corresponding to INC
  - ▶  $a \multimap (b \multimap c)$ , corresponding to DEC
  - ▶  $(a \& b) \multimap c$ , corresponding to FORK

## Definition ( $G_{\text{eILL}}$ goal directed rules for eILL)

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

- Sound and complete w.r.t.  $S_{\text{ILL}}$  for eILL sequents
- Trivial Phase Semantics (commutative monoid, closure is identity)
  - ▶  $S_{\text{ILL}}$  and  $G_{\text{eILL}}$  sound for TPS
  - ▶ eILL complete for  $\text{TPS}(\mathbb{N}^k)$  (when  $k$  big enough)
- The reduction  $\text{eILL} \preceq \text{ILL}$  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  $\bar{x}_i$
  - ▶ for every position/state ( $PC = i$ ) in  $\mathcal{M}$ , a variable  $q_i$

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

# Encoding Minsky machines in eLL

- 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  $\bar{x}_i$
  - ▶ for every position/state ( $\text{PC} = i$ ) in  $\mathcal{M}$ , a variable  $q_i$

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

- 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  $\vec{v} = (p_0, \dots, p_{n-1})$  then  $\Delta_{\vec{v}} = p_0.x_0, \dots, p_{n-1}.x_{n-1}$
  - ▶ the commands in  $\Sigma_{\mathcal{M}}$  are determined by instructions in  $\mathcal{M}$

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

# Encoding Minsky machines in eLL

- 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  $\bar{x}_i$
  - ▶ for every position/state (PC =  $i$ ) in  $\mathcal{M}$ , a variable  $q_i$

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

- 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  $\vec{v} = (p_0, \dots, p_{n-1})$  then  $\Delta_{\vec{v}} = p_0.x_0, \dots, p_{n-1}.x_{n-1}$
  - ▶ the commands in  $\Sigma_{\mathcal{M}}$  are determined by instructions in  $\mathcal{M}$

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

Theorem (Simulating MM termination at zero with  $G_{\text{eLL}}$  entailment)

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

# Encoding Minsky machines in eLL

- 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  $\bar{x}_i$
  - ▶ for every position/state (PC =  $i$ ) in  $\mathcal{M}$ , a variable  $q_i$

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

- 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  $\vec{v} = (p_0, \dots, p_{n-1})$  then  $\Delta_{\vec{v}} = p_0.x_0, \dots, p_{n-1}.x_{n-1}$
  - ▶ the commands in  $\Sigma_{\mathcal{M}}$  are determined by instructions in  $\mathcal{M}$

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

Theorem (Simulating MM termination at zero with  $G_{\text{eLL}}$  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  $\text{MM} \preceq \text{eLL}$

# 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 eLL: elegant encoding of computational model in logics
- eLL to ILL: faithful embedding

# 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 eLL: elegant encoding of computational model in logics
- eLL to ILL: faithful embedding

Low verification overhead

# Wrap-up of this talk

## Reductions:

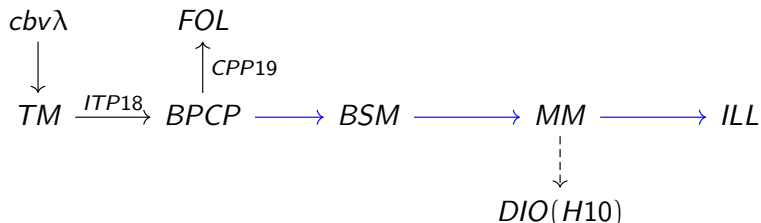
- PCP to BPCP: trivial binary encoding
- BPCP to BSM: verified exhaustive search
- BSM to MM: certified compiler between low-level languages
- MM to eLL: elegant encoding of computational model in logics
- eLL to ILL: faithful embedding

Low verification overhead

(compared to detailed paper proofs)



# Towards a library of undecidable problems



# Conclusion

- 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!

# Conclusion

- 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!

```
https://uds-psl.github.io/ill-undecidability
```

- PDF is hyperlinked with the repo.

# Conclusion

- 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!

```
https://uds-psl.github.io/ill-undecidability
```

- PDF is hyperlinked with the repo.

Advertisement: CPP 2019 talk

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

Tuesday, 16:00

# Conclusion

- 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!

`https://uds-psl.github.io/ill-undecidability`

- PDF is hyperlinked with the repo.

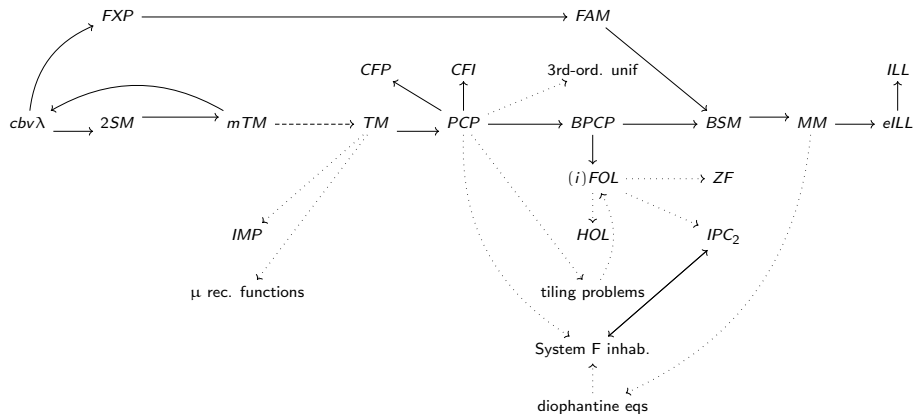
Advertisement: CPP 2019 talk

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

Tuesday, 16:00

Questions?

# Ongoing and Future Work



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

# Properties of step semantics

- Determinism (or functional):

$$\rho \parallel s \succ s_1 \rightarrow \rho \parallel s \succ s_2 \rightarrow s_1 = s_2$$

- Determinism then holds for  $\mathcal{P} \parallel s \succ^n s'$  and  $\mathcal{P} \parallel s \rightsquigarrow s'$
- But not for transitive closures:  $\mathcal{P} \parallel s \succ^* s'$  or  $\mathcal{P} \parallel s \succ^+ s'$
- example of non-determinism: parallel composition

- Totality:

$$\forall s \exists s', \rho \parallel s \succ s'$$

- then out  $j \mathcal{P}$  is the only way to block a computation
- example of blocking instructions: HALT or POP
- the upcoming BSM and MM programming languages both deterministic and total