Mechanising Undecidability results in Coq: Elementary Linear Logic and Boolean BI

Dominique Larchey-Wendling
TYPES team, ANR TICAMORE

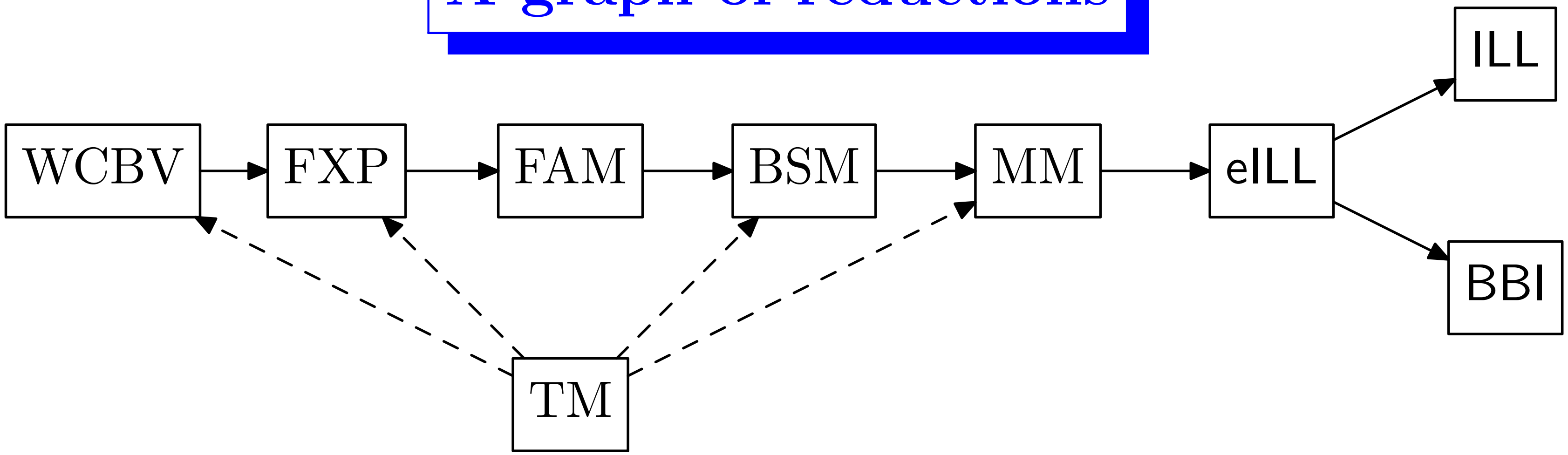
LORIA – CNRS
Nancy, France

Second SYSMICS Workshop
Vienna, Feb. 2018

Introduction

- We mechanize undecidability results in Coq
  - Results from LICS'10 (see also ToCL 2013)
  - Linear logic (ILL and eILL) and Boolean BI
  - still controversial results? (do not laught...)
- How do we proceed? We build reductions:
  - from Minsky machines (MM) to eILL
  - but also ...
  - Weak Call by Value λ-calculus (Forster&Smolka 2017)
  - A first-order functional language with recursion (FXP)
  - Functional Abstract Machines (FAM)
  - Binary Stacks Machines (BSM)
  - Turing machines (TM, in progress)

A graph of reductions



- most reductions are certified compilers
- except WCBV $\rightarrow$ FXP (certified interpreter)
- and eILL $\rightarrow$ ILL or eILL $\rightarrow$ BBI (faithfull embeddings)
- can plug Turing machines (TM) at several stages (in progress)
- WCBV is the more obvious, but not necessarily the easiest

Coq and (un)decidability, Type Theory stops here !

- $\texttt{dec}\ \{X\}\ (P : X \to \texttt{Prop}) := \texttt{inhabited}(\forall x, \{P\,x\} + \{\neg P\,x\});$
- Works when Coq is normalizing (no axioms ...)
- Undecidability of P is **not equivalent** to $\neg(\texttt{dec}\ P)$;
- $\texttt{reduction}\ \{X\ Y\}\ P\ Q\ (f : X \to Y) := \forall x, P\,x \Leftrightarrow Q(f\,x);$
- Lemma: $(\exists f, \texttt{reduction}\ P\ Q\ f) \to \texttt{dec}\ Q \to \texttt{dec}\ P$
- Inductive for $\texttt{undec}\ \{X\} : (X \to \texttt{Prop}) \to \texttt{Prop}$

$$\frac{}{\texttt{undec TM\_Halting}} \qquad \frac{\texttt{dec}\ P \to \texttt{dec}\ Q \quad \texttt{undec}\ Q}{\texttt{undec}\ P}$$

- Lemma: $(\exists f, \texttt{reduction}\ P\ Q\ f) \to \texttt{undec}\ P \to \texttt{undec}\ Q$
- Theorem: $\texttt{undec}\ P \to \texttt{dec}\ P \to \texttt{dec TM\_Halting}$

Weak Call-by-Value (WCBV) lambda calculus

- untyped λ-terms: $t ::= x \mid t \cdot t \mid \lambda x.t$ with $x \in \texttt{Var}$
- Weak Call-by-Value (Dal Lago & Martini 2008):

$$\frac{}{(\lambda x.f) \cdot (\lambda y.g) \rightarrow_{\text{wcbv}} f\{x \leftarrow \lambda y.g\}} \qquad \frac{f \rightarrow_{\text{wcbv}} f' \quad g \rightarrow_{\text{wcbv}} g'}{f \cdot g \rightarrow_{\text{wcbv}} f' \cdot g'}$$

- WCBV is non-deterministic but *strongly confluent*
- Evaluation:

$$\frac{}{\lambda x.f \triangleright \lambda x.f} \qquad \frac{f \triangleright \lambda x.u \quad g \triangleright v \quad u\{x \leftarrow v\} \triangleright w}{f \cdot g \triangleright w}$$

- Equivalence: $f \rightarrow^{\star}_{\text{wcbv}} \lambda x.g$ iff $f \triangleright \lambda x.g$
- Problem: given f and g, does $f \triangleright g$?

Functional expressions (FXP), w. mutual recursion

- datatype is `bt` (binary trees): $\tau ::= \emptyset_{\mathrm{t}} \mid (\tau, \tau)_{\mathrm{t}}$
- trivial to encode data structures (pair, lists, trees, etc.)
- expressions ($v \in \texttt{Var}$ and $f \in \texttt{Fun}$):

  $$\texttt{expr} : \; e ::= v \mid f(e) \mid \emptyset_{\mathrm{e}} \mid (e, e)_{\mathrm{e}} \mid \texttt{match}\; e\; \texttt{with}\; \emptyset_{\mathrm{e}} \Rightarrow e\; \texttt{or}\; (x, y)_{\mathrm{e}} \Rightarrow e$$

- programs:

$$\begin{array}{rlcl} \texttt{let rec} & f_1(x_1) & = & body_1 \\ \texttt{with} & \cdots & & \\ \texttt{with} & f_n(x_n) & = & body_n \end{array} \quad \text{with} \begin{cases} f_1, \ldots, f_n \in \texttt{Fun} \\ x_1, \ldots, x_n \in \texttt{Var} \\ body_1, \ldots, body_n \in \texttt{expr} \end{cases}$$

Bigstep semantics for FXP

- for $e \in \texttt{expr}$, $\sigma \in \texttt{Var} \to \texttt{bt}$ and $\tau \in \texttt{bt}$
- bigstep semantics: $e \dashv\![\, \sigma \,]\!\mapsto \tau$,

$$\frac{}{v \dashv\![\, \sigma \,]\!\mapsto \sigma_v} \qquad \frac{}{\emptyset_{\mathrm{e}} \dashv\![\, \sigma \,]\!\mapsto \emptyset_{\mathrm{t}}} \qquad \frac{e_1 \dashv\![\, \sigma \,]\!\mapsto \tau_1 \quad e_2 \dashv\![\, \sigma \,]\!\mapsto \tau_2}{(e_1, e_2)_{\mathrm{e}} \dashv\![\, \sigma \,]\!\mapsto (\tau_1, \tau_2)_{\mathrm{t}}}$$

$$\frac{e_1 \dashv\![\, \sigma \,]\!\mapsto \emptyset_{\mathrm{t}} \qquad\qquad e_2 \dashv\![\, \sigma \,]\!\mapsto \tau}{\texttt{match } e_1 \texttt{ with } \emptyset_{\mathrm{e}} \Rightarrow e_2 \ \ldots \dashv\![\, \sigma \,]\!\mapsto \tau}$$

$$\frac{e_1 \dashv\![\, \sigma \,]\!\mapsto (\tau_1, \tau_2)_{\mathrm{t}} \quad e_3 \dashv\![\, \sigma\{x \leftarrow \tau_1, y \leftarrow \tau_2\} \,]\!\mapsto \tau_3}{\texttt{match } e_1 \texttt{ with } \ldots \texttt{ or } (x, y)_{\mathrm{e}} \Rightarrow e_3 \dashv\![\, \sigma \,]\!\mapsto \tau_3}$$

$$\frac{e \dashv\![\, \sigma \,]\!\mapsto \tau_1 \quad \mathit{body}_f \dashv\![\, \sigma\{x \leftarrow \tau_1\} \,]\!\mapsto \tau_2}{f(e) \dashv\![\, \sigma \,]\!\mapsto \tau_2}$$

- Problem: given a program, e and τ, does $e \dashv\![\, \_ \mapsto \emptyset_{\mathrm{t}} \,]\!\mapsto \tau$ hold ?

Functional Abstract Machines (FAM)

- instructions (with $v \in \texttt{Var}$ and $i, k \in \mathbb{N}$)

$$\begin{array}{rcl} \texttt{fam\_instr} & ::= & \texttt{LDV}\ v \mid \texttt{STV}\ v \mid \texttt{LDA}\ i \mid \texttt{IJP} \mid \texttt{FJK}\ k \\ & \mid & \texttt{NULL} \mid \texttt{PAIR} \mid \texttt{MATCH}\ k \mid \texttt{HALT} \end{array}$$

- state: $(\mathrm{PC}, \sigma, \mathrm{addr}, \mathrm{data})$
  - a program counter $\mathrm{PC} \in \mathbb{N}$, and environment $\sigma \in \texttt{Var} \to \texttt{bt}$
  - a (return) address stack ($\texttt{list}\ \mathbb{N}$) and a data stack ($\texttt{list bt}$)
- Small step semantics:

$$\begin{array}{ll} \texttt{LDV}\ v: & \text{push } \sigma_v \text{ on data stack}; \mathrm{PC} \leftarrow \mathrm{PC} + 1 \\ \texttt{STV}\ v: & \text{pop } \tau \text{ from data stack}; \sigma \leftarrow \sigma\{v \leftarrow \tau\}; \mathrm{PC} \leftarrow \mathrm{PC} + 1 \\ \texttt{LDA}\ i: & \text{push } i \text{ on addr stack}; \mathrm{PC} \leftarrow \mathrm{PC} + 1 \end{array}$$

FAM small step semantics (continued)

`IJP` :	pop i from addr stack; $\mathrm{PC} \leftarrow i$
`FJP` k :	$\mathrm{PC} \leftarrow \mathrm{PC} + k$
`NULL`	push $\emptyset_{\mathrm{t}}$ on data stack; $\mathrm{PC} \leftarrow \mathrm{PC} + 1$
`PAIR`	pop τ_1 then pop τ_2 from data stack;
	push $(\tau_1, \tau_2 2)_{\mathrm{t}}$ on data stack; $\mathrm{PC} \leftarrow \mathrm{PC} + 1$
`MATCH` k	pop τ from data stack;
	if τ is $\emptyset_{\mathrm{t}}$ then $\mathrm{PC} \leftarrow \mathrm{PC} + k$;
	otherwise τ is $(\tau_1, \tau_2)_{\mathrm{t}}$ and
	push τ_2 then push τ_1 on data stack; $\mathrm{PC} \leftarrow \mathrm{PC} + 1$

- FAM program: $i : \texttt{fam\_instr}_i; i+1 : \ldots; j : \texttt{fam\_instr}_j$
- Problem: $(i, \sigma, \emptyset, \emptyset) \longrightarrow^{\star} (j+1, \sigma', \emptyset, \emptyset)$

Binary Stack Machines

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

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

- state: $(\mathrm{PC}, (\texttt{list bool})^n)$
- Small step semantics (`HALT` is blocking):

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

- BSM program: $i : \texttt{bsm\_instr}_i; i+1 : \ldots; j : \texttt{bsm\_instr}_j$
- Problem: $(i, S) \longrightarrow^\star (j+1, S')$

Minsky Machines

- n registers of value in $\mathbb{N}$ for a fixed n
- instructions (with $0 \leqslant x < n$ and $i \in \mathbb{N}$)

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

- Small step semantics, state: $(\mathrm{PC}, \mathbb{N}^n)$

$$\texttt{INC}\ x: \quad x \leftarrow x + 1; \mathrm{PC} \leftarrow \mathrm{PC} + 1;$$

$$\texttt{DEC}\ x\ i: \quad \text{if } x \text{ is } 0 \text{ then } \mathrm{PC} \leftarrow i \text{ else } x \leftarrow x - 1; \mathrm{PC} \leftarrow \mathrm{PC} + 1;$$

- MM program: $i : \texttt{mm\_instr}_i; i + 1 : \ldots ; j : \texttt{mm\_instr}_j$
- Problem: $(i, R) \longrightarrow^\star (j + 1, R')$

Intuitionistic Linear Logic (ILL)

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

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

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

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

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

- Full linear logic faithfully embedded into that fragment

- Problem: given $\Gamma \vdash A$, is it provable ?

ILL, Trivial Phase Semantics and BBI

- A commutative monoid $(M, \circ, \epsilon)$, or even $(\mathbb{N}^k, +, 0)$
- Trivial phase interpretation of ILL operators:

$$
\begin{array}{rcl}
[\![!\, A]\!] & = & \{\epsilon\} \cap [\![A]\!] \\
[\![A \,\&\, B]\!] & = & [\![A]\!] \cap [\![B]\!] \\
m \in [\![A \multimap B]\!] & \text{iff} & \forall \alpha, \alpha \in [\![A]\!] \Rightarrow \alpha \circ m \in [\![B]\!]
\end{array}
$$

- Sound but incomplete for ILL, $\multimap / -\!\!*$ equivalent (BBI)
- Encoding of ILL in BBI (a sound embedding):

$$
(!\, A)^\star \rightsquigarrow \mathsf{I} \wedge A^\star \quad (A \,\&\, B)^\star \rightsquigarrow A^\star \wedge B^\star \quad (A \multimap B)^\star \rightsquigarrow A^\star -\!\!* B^\star
$$

- This embedding is faithfull for TPS
- Find a fragment of ILL for which TPS is complete

The elementary fragment eILL of ILL

- 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):

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

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

- eILL is sound and complete for TPS (even $\mathbb{N}^k$).

Encoding Minsky machines in eILL

- Given $\mathcal{M}$ as a list of MM instructions
- for every register x in $\mathcal{M}$, two logical variables x and $\underline{x}$
- the state $(i, (p_1, \ldots p_n))$ is represented by $!\,\Sigma; x_1^{p_1}, \ldots x_n^{p_n} \vdash q_i$

$$i : \ \texttt{INC}\ x \in \mathcal{M} \quad \left| \quad \begin{array}{l} x \leftarrow x + 1 \\ \mathrm{PC} \leftarrow i + 1 \end{array} \right.$$

- corresponds to proof:

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

MM to **elLL**, (continued)

- Decrement

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

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

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

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

Zero test $x \notin \Delta$ in **elLL**

- Proof for y, Δ with $y \neq x$:

$$\dfrac{\dfrac{}{!\,\Sigma; y \vdash y}\,\mathrm{Ax} \quad \dfrac{\cdots}{!\,\Sigma; \Delta \vdash \underline{x}}}{!\,\Sigma; y, \Delta \vdash \underline{x}}\; y \multimap (\underline{x} \multimap \underline{x}) \in \Sigma$$

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

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

Terminating the MM computation

- k is a halting state ($k \notin \mathcal{M}$)

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

- We define $\Sigma_{\mathcal{M},k} = \Sigma_{\mathcal{M}} \cup \{(q_k \multimap q_k) \multimap q_k\}$ where:

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

- Theorem (for k outside of $\mathcal{M}$):

$$\mathcal{M} : (i, \Delta) \longrightarrow^\star (k, \emptyset) \quad \text{iff} \quad !\,\Sigma_{\mathcal{M},k}; \Delta \vdash q_i$$