Game Semantic Analysis of Equivalence in Interface Middleweight Java

Andrzej Murawski
Univ. of Warwick

Steven Ramsay
Univ. of Oxford

Nikos Tzevelekos
Queen Mary U. of London

ATVA, Shanghai, October 2015

Supported by a Royal Academy of Engineering Research Fellowship

what this talk is about

We examine contextual equivalence of Java programs

We work in an fragment of Middleweight Java

- open code modelled by use of interfaces
- denotational approach based on game semantics

Full characterisation based on type disciplines:

- queue machine encodings (negative cases)
- pushdown register automata (algorithms)

Program equivalence

$$M \cong M'$$

same observable behaviour in every context

for all closing contexts C:

$$(C[M],\phi) \rightarrow^* (\mathsf{skip},S) \iff (C[M'],\phi) \rightarrow^* (\mathsf{skip},S')$$

- subsumes standard verification properties
- note: quantification over *every* context
- several methods for proving given equivalences yet no general procedures

Interface Middleweight Java (IMJ)

Types $\quad \theta ::= \text{void} \mid \text{int} \mid \mathcal{I}$

Interface definitions

$$\Theta ::= \phi \mid (\mathsf{f} : \theta), \Theta \mid (\mathsf{m} : \overline{\theta} \rightarrow \theta), \Theta$$

Interface tables

$$\Delta ::= \phi \mid (\mathcal{I} : \Theta), \Delta \mid (\mathcal{I}\langle\mathcal{I}\rangle : \Theta), \Delta$$

Object calculus based on MJ [Bierman, Parkinson, Pitts]

- Objects, inheritance, casting, interfaces

Interface Middleweight Java (IMJ)

Types $\theta ::= \text{void} \mid \text{int} \mid \mathcal{I}$

Interface definitions

$\Theta ::= \phi \mid (\mathsf{f} : \theta), \Theta \mid (\mathsf{m} : \overline{\theta} \to \theta), \Theta$

Interface tables

$\Delta ::= \phi \mid (\mathcal{I} : \Theta), \Delta \mid (\mathcal{I}\langle\mathcal{I}\rangle : \Theta), \Delta$

interface ident.

field identifier

method identif.

Object calculus based on MJ [Bierman, Parkinson, Pitts]

- Objects, inheritance, casting, interfaces

Interface Middleweight Java (IMJ)

Terms

$$M ::= \text{skip} \mid a \mid \text{null} \mid x \mid i \mid M \oplus M \mid \text{if } M\ M\ M$$
$$\mid \text{let } x = M \text{ in } M \mid M = M \mid (\mathcal{I})M \mid \text{while } M\ M$$
$$\mid \text{new}(x : \mathcal{I};\ \mathcal{M}) \mid M.\text{f} \mid M.\text{f} := M \mid M.\text{m}(\overline{M})$$

Method implementations $\mathcal{M} ::= \phi \mid (\text{m} : \lambda\overline{x}.M),\ \mathcal{M}$

Interface Middleweight Java (IMJ)

$\mathcal{M} = \{ \mathsf{m}_i : \lambda \overline{x}_i.M_i \mid 1 \le i \le n \}$

$$\frac{\Delta|\Gamma \vdash M : \mathsf{int} \quad \Delta|\Gamma \vdash M', M'' : \theta}{\Delta|\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ M'\ \mathsf{else}\ M'' : \theta}$$

$$\frac{\bigwedge_{i=1}^{n}(\Delta|\Gamma \uplus \{\vec{x}_i : \vec{\theta}_i\} \vdash M_i : \theta_i)}{\Delta|\Gamma \vdash \mathcal{M} : \{\mathsf{m}_i : \vec{\theta}_i \to \theta_i \mid 1 \le i \le n\}}$$

$$\frac{\Delta|\Gamma \vdash M, M' : I}{\Delta|\Gamma \vdash M = M' : \mathsf{int}} \qquad \frac{\Delta|\Gamma \vdash M : \mathsf{void} \quad \Delta|\Gamma \vdash M' : \theta}{\Delta|\Gamma \vdash M; M' : \theta} \qquad \frac{\Delta|\Gamma \vdash M : I \quad \Delta|\Gamma \vdash M' : \theta}{\Delta|\Gamma \vdash M.\mathsf{f} := M' : \mathsf{void}}\ \Delta(I).\mathsf{f}=\theta$$

$$\frac{\Delta|\Gamma \vdash M : I}{\Delta|\Gamma \vdash M.\mathsf{f} : \theta}\ \Delta(I).\mathsf{f}=\theta \qquad \frac{\Delta|\Gamma \vdash M : I'}{\Delta|\Gamma \vdash (I)M : I}\ \begin{array}{l}\Delta \vdash I \le I' \\ \vee I' \le I\end{array}$$

$$\frac{\Delta|\Gamma, x : I \vdash \mathcal{M} : \Theta}{\Delta|\Gamma \vdash \mathsf{new}\{\, x : I; \mathcal{M}\,\} : I}\ \Delta(I) \restriction Meths = \Theta$$

$$\frac{\Delta|\Gamma \vdash M : I \quad \bigwedge_{i=1}^{n}(\Delta|\Gamma \vdash M_i : \theta_i)}{\Delta|\Gamma \vdash M.\mathsf{m}(M_1, \cdots, M_n) : \theta}\ \Delta(I).\mathsf{m}=\vec{\theta}\to\theta \qquad \frac{\Delta|\Gamma \vdash M : \theta' \quad \Delta|\Gamma, x : \theta' \vdash M' : \theta}{\Delta|\Gamma \vdash \mathsf{let}\ x = M\ \mathsf{in}\ M' : \theta}$$

Terms

$$M ::= \mathsf{skip} \mid a \mid \mathsf{null} \mid x \mid i \mid M \oplus M \mid \mathsf{if}\ M\ M\ M$$
$$\mid \mathsf{let}\ x = M\ \mathsf{in}\ M \mid M = M \mid (\mathcal{I})M$$
$$\mid \mathsf{new}(x : \mathcal{I}; \mathcal{M}) \mid M.\mathsf{f} \mid M.\mathsf{f} := M \mid M.\mathsf{m}(\overline{M})$$

Method implementations $\mathcal{M} ::= \phi \mid (\mathsf{m} : \lambda \overline{x}.M), \mathcal{M}$

IMJ: operational semantics

$$S,\ M \ \longrightarrow\ S',\ M'$$

S stores object names + their types and values

Obj : set of obj. names

$$S,\ \text{let}\ x = v\ \text{in}\ M \ \longrightarrow\ S,\ M[v/x]$$

$$S,\ a = a' \ \longrightarrow\ S,\ 0/1$$

$a, a' \in \text{Obj}$

$$S,\ \text{new}(x : \mathcal{I};\ M) \ \longrightarrow\ S \uplus \{(a,\ \mathcal{I},\ (V_{\mathcal{I}},\ M[a/x])\},\ a$$

$V_{\mathcal{I}}$: default field values

Program equivalence – our plan

$$M \cong M'$$

same observable behaviour in every context

for all closing contexts C:

$$(C[M],\phi) \rightarrow^* (\mathsf{skip},S) \iff (C[M'],\phi) \rightarrow^* (\mathsf{skip},S')$$

Program equivalence – our plan

$$M \cong M'$$

same observable behaviour in every context

for all closing contexts C:

$$(C[M],\phi) \to^* (\mathsf{skip},S) \iff (C[M'],\phi) \to^* (\mathsf{skip},S')$$

termination → inequivalence

we consider a finitary restriction IMJ_{fin} and first identify the fragment decidable for termination

we use game semantics and types to characterise the fragment of IMJ_{fin} decidable for equivalence

Termination

M *terminates* if:
$(M,\phi) \rightarrow^* (\text{skip},S)$

Two distinct sources of undecidability:

I. recursive methods

$\text{new}(x : \mathcal{I};\ \text{main: } \lambda y.\ \{\ldots\ x.\text{main}(..)\ \ldots\ \})$

II. objects with methods can be stored

$\text{let } a = \text{new}(x : \mathcal{I};\ \text{main: } \lambda y.\ \{\ldots\ \})\ \text{ in}$
$\text{let } b = \text{new}(x' : \mathcal{I}';\ )\ \text{ in }\ b.\text{var} := a$

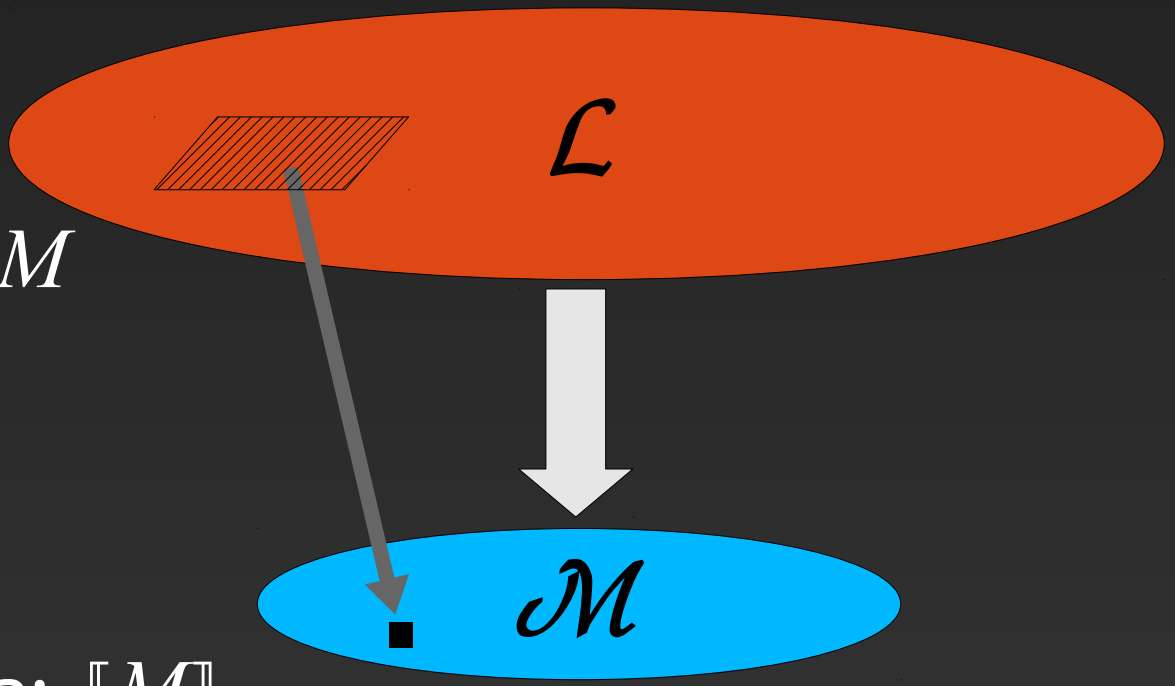
Theorem: Termination is undecidable for IMJ_{fin} terms that may feature I or II.
Conversely, given an IMJ_{fin} term M that does not feature I or II, we can always decide whether it terminates.

Games for program equivalence

We focus on: $\mathrm{IMJ}_{fin} - \{\mathrm{I},\mathrm{II}\}$

and employ the fully abstract game semantics for IMJ:

$$M \cong M' \Leftrightarrow [\![M]\!] = [\![M']\!]$$



Games for program equivalence

We focus on: $\mathrm{IMJ}_{fin} - \{\mathrm{I},\mathrm{II}\}$

and employ the fully abstract game semantics for IMJ:

$$M \cong M' \Leftrightarrow [\![M]\!] = [\![M']\!]$$

- for negative results, given a queue machine Q, devise terms M, M':

$$Q\Uparrow \Leftrightarrow [\![M]\!] = [\![M']\!]$$

- for decidability, given terms M, M', devise automata A, A':

$$[\![M]\!] = [\![M']\!] \Leftrightarrow A \sim A'$$

Game semantics

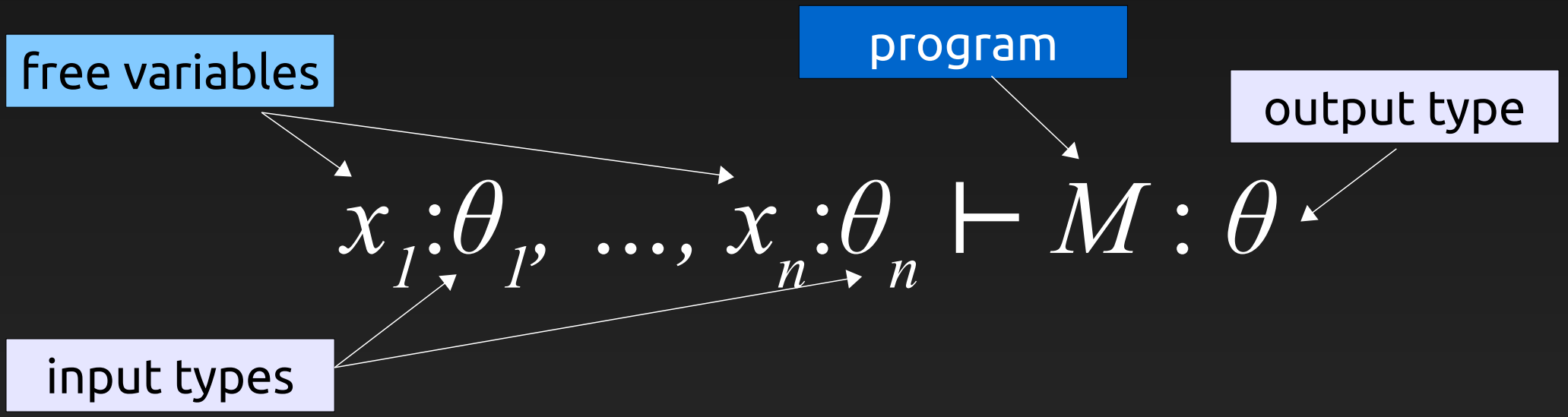
We can model the execution of open programs as a 2-player game between:

- *Opponent* (the environment, O)
- *Proponent* (the program, P)

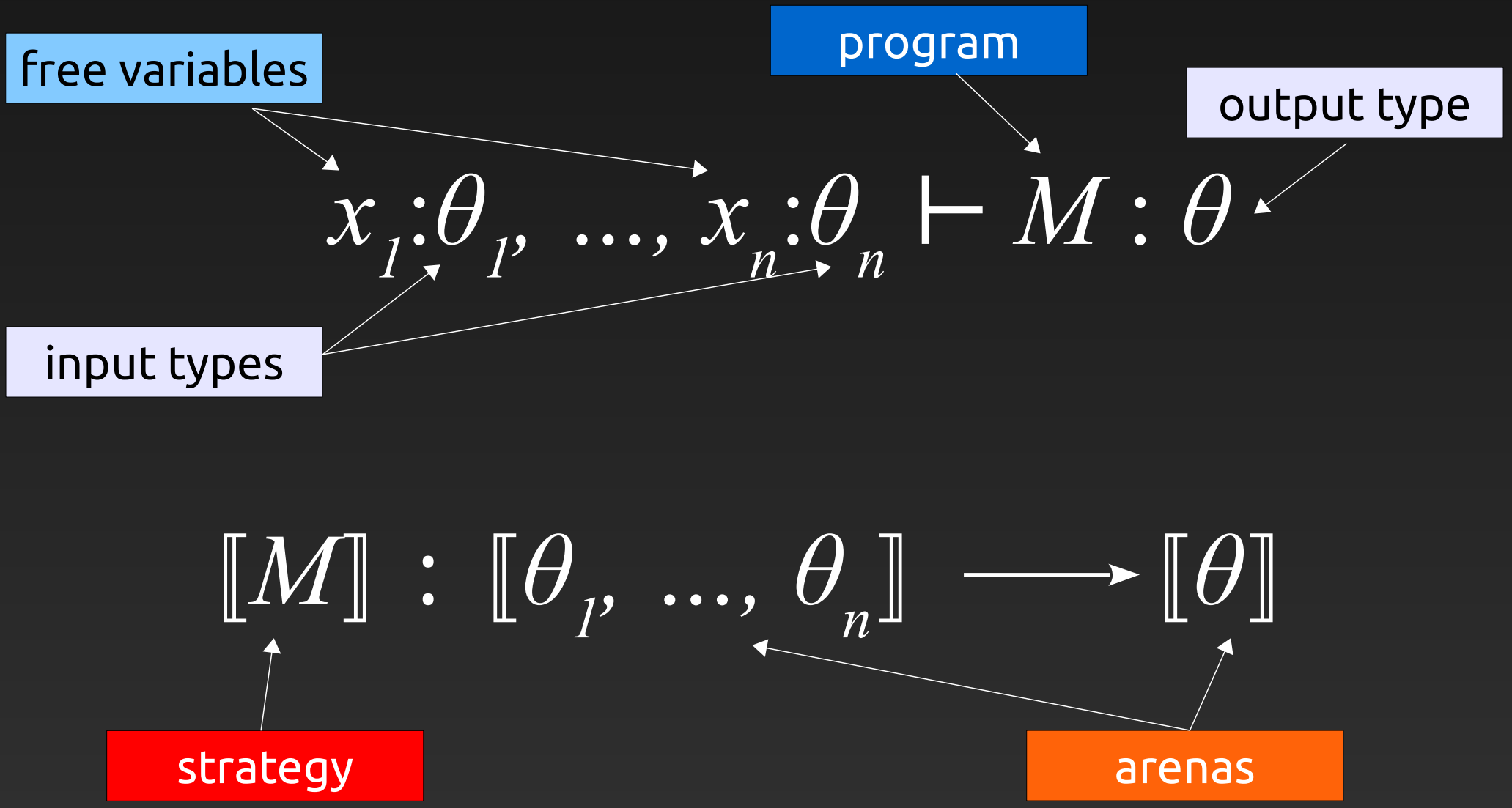
→ think of these as open program traces

Programs = sets of traces (for P), called *strategies*

Game apparatus



Game apparatus



Games for IMJ programs

Program interactions are modelled by sequences of moves-with-store, written m^{Σ}, where:

- move m is either:
  - a value/arena move
  - an object method call/return

call c.set(c'), ret q.get(), …

- stores are of the form:

$$\Sigma = \{ n \mapsto (\mathsf{Emp}, \emptyset),\ a \mapsto (\mathsf{Ptr}, \mathsf{val} = a),\ p \mapsto (\mathsf{Pt2}, \mathsf{x} = 1, \mathsf{y} = 0),\ c \mapsto (\mathsf{Ptr'}, \emptyset),\ q \mapsto (\mathsf{Pt2'}, \emptyset),\ \ldots \}$$

Ptr' : (get : Ptr' → Ptr', set : Ptr' → Ptr')
Pt2' : (get : void → (int,int), set : (int,int) → void)

[Murawski & T. '14]

IMJ example: game semantics

M_1 : let u = new(Var_{Emp}) in
new(M_1) : Cell

M_1 : get: $\lambda().\ u.\mathrm{val}$,
set: $\lambda y.\ u.\mathrm{val} := y$

Δ = Empty: $\emptyset$,
Cell: (get: void → Empty,
set: Empty → void),
Var_{Emp}: (val: Empty),
Var_{Int}: (val: int)

$$[\![M_1]\!] = * \; n^{\Sigma_0} \; (\; \underset{O}{\text{call } n.\text{get}()^{\Sigma_0}} \; \underset{P}{\text{ret } n.\text{get}(\text{nul})^{\Sigma_0}} \;)^* \; \underset{O}{\text{call } n.\text{set}(n_1)^{\Sigma_1}} \; \underset{P}{\text{ret } n.\text{set}()^{\Sigma_1}} \; (\; \text{call } n.\text{get}()^{\Sigma_1} \; \text{ret } n.\text{get}(n_1)^{\Sigma_1} \;)^* \; \text{call } n.\text{set}(n_2)^{\Sigma_2} \; \text{ret } n.\text{set}()^{\Sigma_2} \; \ldots$$

$$\Sigma_i = \{\; (n : \text{Cell},) \;\} \cup \{\; (n_j : \text{Empty},) \mid 1 \le j \le i \;\}$$

* [Koutavas & Wand' 07]

IMJ example: game semantics

M_2 : let b = new(Var_{Int}) in
let u_1 = new(Var_{Emp}) in
let u_2 = new(Var_{Emp}) in
new(M_2) : Cell

M_2 : get: $\lambda()$. if b.val
then b.val := 0; u_1.val
else b.val := 1; u_2.val,
set: λy. u_1.val := y;
u_2.val := y

O **P** **O** **P**

$[\![M_1]\!]$ = $*$ n^{Σ_0} (call n.get()$^{\Sigma_0}$ ret n.get(nul)$^{\Sigma_0}$)*
call n.set(n_1)$^{\Sigma_1}$ ret n.set()$^{\Sigma_1}$
(call n.get()$^{\Sigma_1}$ ret n.get(n_1)$^{\Sigma_1}$)*
call n.set(n_2)$^{\Sigma_2}$ ret n.set()$^{\Sigma_2}$ ... $= [\![M_2]\!]$

$\Sigma_i = \{ (n : \mathrm{Cell},) \} \cup \{ (n_j : \mathrm{Empty},) \mid 1 \leq j \leq i \}$

* [Koutavas & Wand' 07]

Games for program equivalence

We focus on: $\mathrm{IMJ}_{fin} - \{\mathrm{I},\mathrm{II}\}$

and employ the fully abstract game semantics for IMJ:

$$M \cong M' \Leftrightarrow [\![M]\!] = [\![M']\!]$$

- for negative results, given a queue machine Q, devise terms M, M':

$$Q\Uparrow \Leftrightarrow [\![M]\!] = [\![M']\!]$$

- for decidability, given terms M, M', devise automata A, A':

$$[\![M]\!] = [\![M']\!] \Leftrightarrow A \sim A'$$

One source of Undecidability

P passes objects of type $\mathcal{I}$, and $\mathcal{I}$ contains methods

$\mathcal{I}$: (step: void$\rightarrow$void)
N : (val: int)

(run: $\mathcal{I}\rightarrow$void) $\vdash$ void

we can encode computations of queue machines!

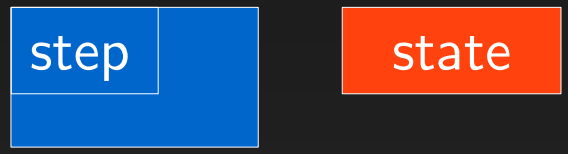
One source of Undecidability

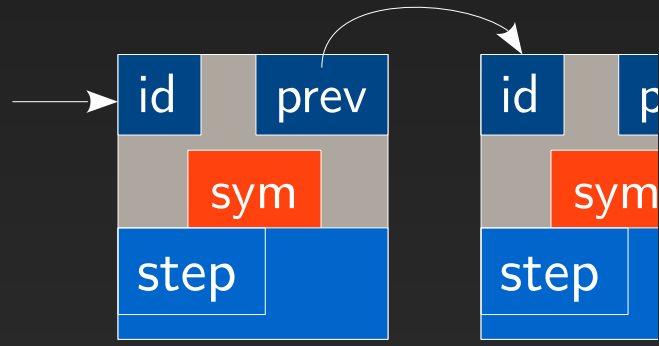
P passes objects of type $\mathcal{I}$, and $\mathcal{I}$ contains methods

$\mathcal{I}$: (step: void→void)
N : (val: int)

(run: $\mathcal{I}$→void) ⊢ void

we can encode computations of queue machines!





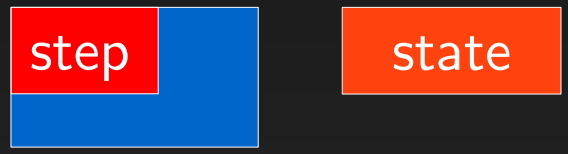
One source of Undecidability

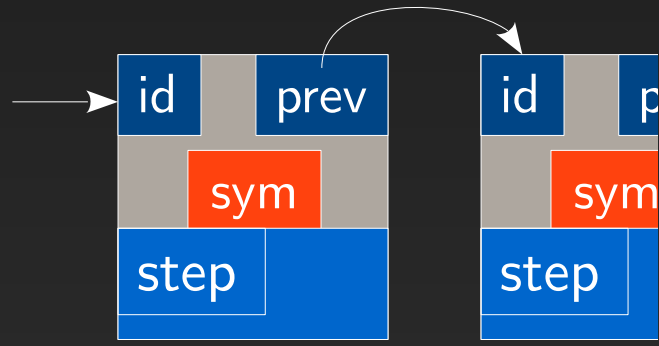
P passes objects of type $\mathcal{I}$, and $\mathcal{I}$ contains methods

$\mathcal{I}$: (step: void→void)
N : (val: int)

(run: $\mathcal{I}$→void) ⊢ void

we can encode computations of queue machines!





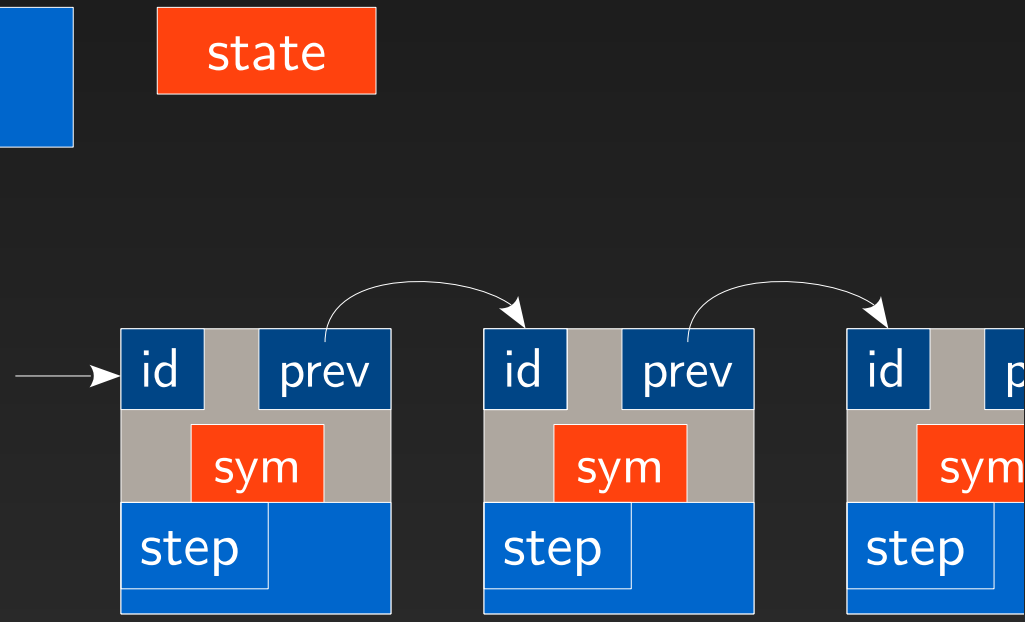
One source of Undecidability

P passes objects of type $\mathcal{I}$, and $\mathcal{I}$ contains methods

$\mathcal{I}$: (step: void→void)
N : (val: int)

(run: $\mathcal{I}$→void) ⊢ void

we can encode computations of queue machines!



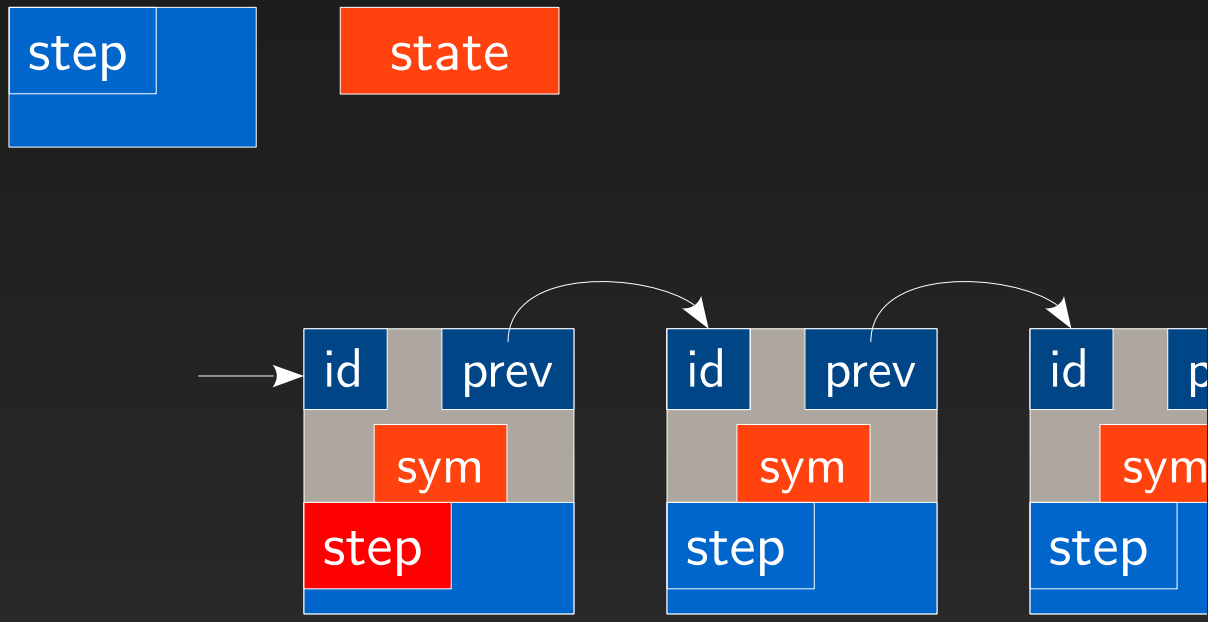
One source of Undecidability

P passes objects of type $\mathcal{I}$, and $\mathcal{I}$ contains methods

$\mathcal{I}$: (step: void→void)
N : (val: int)

(run: $\mathcal{I}$→void) ⊢ void

we can encode computations of queue machines!



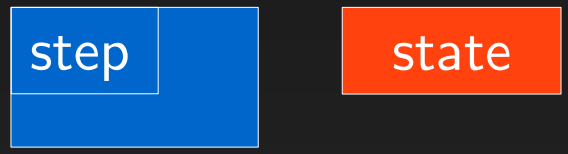
One source of Undecidability

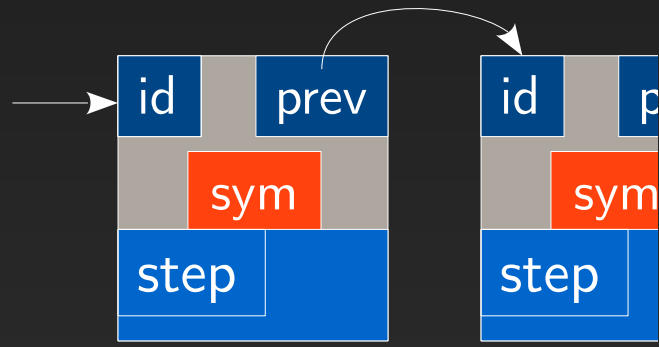
P passes objects of type $\mathcal{I}$, and $\mathcal{I}$ contains methods

$\mathcal{I}$: (step: void→void)
N : (val: int)

(run: $\mathcal{I}$→void) ⊢ void

we can encode computations of queue machines!





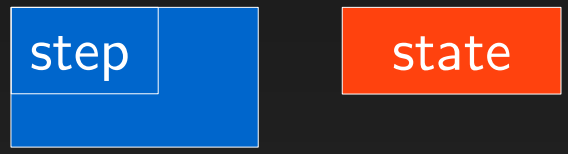
One source of Undecidability

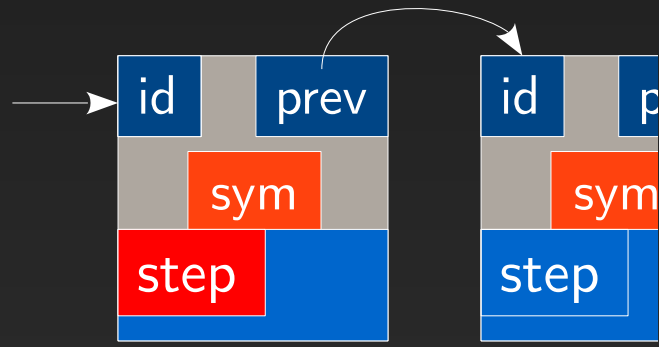
P passes objects of type $\mathcal{I}$, and $\mathcal{I}$ contains methods

$\mathcal{I}$: (step: void→void)
N : (val: int)

(run: $\mathcal{I}$→void) ⊢ void

we can encode computations of queue machines!





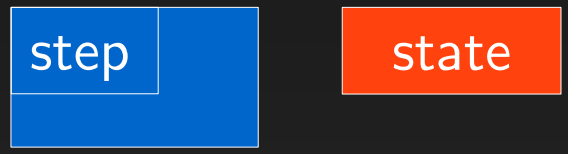
One source of Undecidability

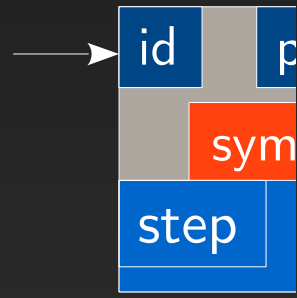
P passes objects of type $\mathcal{I}$, and $\mathcal{I}$ contains methods

$\mathcal{I}$: (step: void→void)
N : (val: int)

(run: $\mathcal{I}$→void) ⊢ void

we can encode computations of queue machines!





Decidable types – IMJ*

$x : \mathsf{L} \vdash M : \mathsf{R}$

$$\mathsf{G} \;::=\; \mathsf{void} \mid \mathsf{int} \mid (\mathsf{f} : \mathsf{G})$$

$$\mathsf{L} \;::=\; \mathsf{void} \mid \mathsf{int} \mid (\mathsf{f} : \mathsf{G},\ \mathsf{m} : \mathsf{G} \to \mathsf{L})$$

$$\mathsf{R} \;::=\; \mathsf{void} \mid \mathsf{int} \mid (\mathsf{f} : \mathsf{G},\ \mathsf{m} : \mathsf{L} \to \mathsf{G})$$

Games for program equivalence

We focus on: $\mathrm{IMJ}_{fin} - \{\mathrm{I},\mathrm{II}\}$

and employ the fully abstract game semantics for IMJ:

$$M \cong M' \Leftrightarrow [\![M]\!] = [\![M']\!]$$

- for negative results, given a queue machine Q, devise terms M, M':

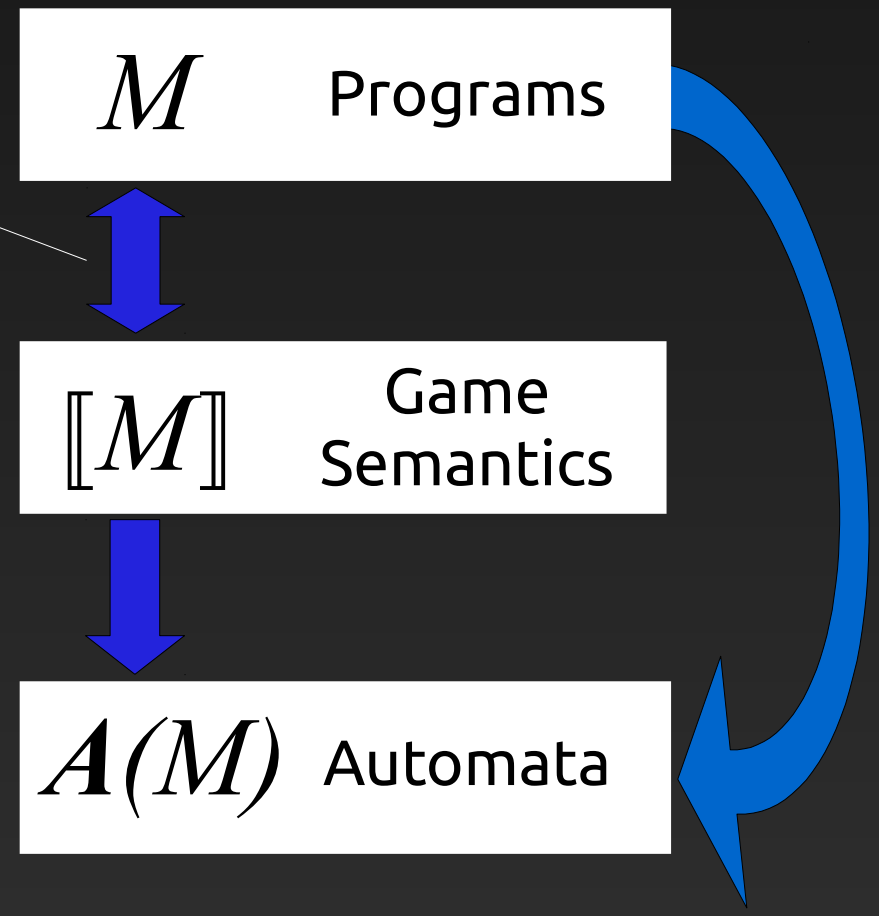
$$Q\Uparrow \Leftrightarrow [\![M]\!] = [\![M']\!]$$

- for decidability, given terms M, M', devise automata A, A':

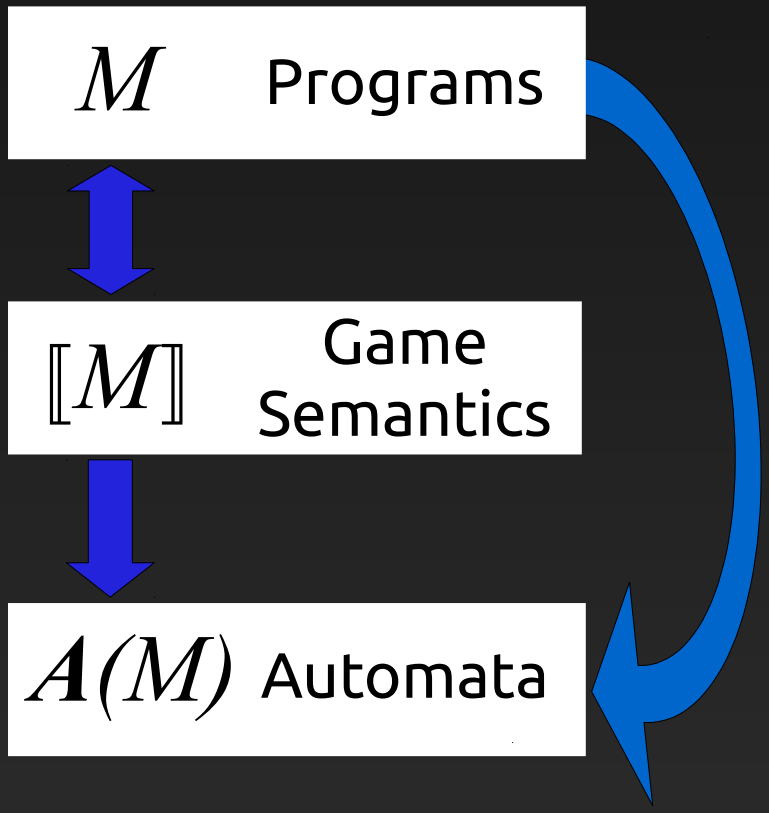
$$[\![M]\!] = [\![M']\!] \Leftrightarrow A \sim A'$$

From games to automata

$$M \cong M'$$
$$\Leftrightarrow$$
$$[\![M]\!] = [\![M']\!]$$



From games to automata



Translation done in two steps:

- terms reduced to canonical forms
- canonical terms are compositionally transformed into automata

We work with pushdown automata:

- over infinite alphabets
- visibly pushdown & deterministic

$$M \cong M' \Leftrightarrow [\![M]\!] = [\![M']\!]$$
$$\Leftrightarrow A \sim A' \Leftrightarrow A \otimes A' = 0$$

Implemented in Coneqct!

Wrapping up

Contextual equivalence

- captured accurately by game semantics
- decidability characterised via type disciplines
- automated decision via FPDRAs → Coneqct

Further on:

- sound methods for the whole of IMJ
- general model checking
- logics (over infinite alphabets)