

Model Checking Program Equivalence in Interface Middleweight Java

Andrzej Murawski
Uni. of Warwick

Steven Ramsay
Uni. of Oxford

Nikos Tzevelekos
Queen Mary U. of London

Higher-Order Model Checking, Shonan, Mar 2016

Supported by a Royal Academy of Engineering Research Fellowship

what this talk is about

We examine **contextual equivalence** of Java programs

- these are 'higher'-order

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)

Implementation of decidable fragment in **Coneqct**

Program equivalence

$$M \cong M'$$

same observable behaviour in every context

for all closing contexts C :

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

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

Interface Middleweight Java (IMJ)

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

Interface definitions

$\Theta ::= \emptyset \mid (f : \theta), \Theta \mid (m : \bar{\theta} \rightarrow \theta), \Theta$

Interface tables

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

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

$\Theta ::= \emptyset \mid (f : \theta), \Theta \mid (m : \bar{\theta} \rightarrow \theta), \Theta$

Interface tables

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

interface ident.

field identifier

method identif.

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

- Objects, inheritance, casting, **interfaces**

Interface Middleweight Java (IMJ)

Terms

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

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

Interface Middleweight Java (IMJ)

Terms

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

Method implementations $\mathcal{M} ::= \emptyset \mid (m : \lambda \bar{x}. M), \mathcal{M}$

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

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

$$\mathcal{M} = \{m_i : \lambda \bar{x}_i. M_i \mid 1 \leq i \leq n\}$$

$$\frac{\Delta | \Gamma \vdash M, M' : I}{\Delta | \Gamma \vdash M = M' : \text{int}}$$

$$\frac{\Delta | \Gamma \vdash M : \text{void} \quad \Delta | \Gamma \vdash M' : \theta}{\Delta | \Gamma \vdash M; M' : \theta}$$

$$\frac{\Delta | \Gamma \vdash M : I \quad \Delta | \Gamma \vdash M' : \theta}{\Delta | \Gamma \vdash M.f := M' : \text{void}} \quad \Delta(I).f = \theta$$

$$\frac{\Delta | \Gamma \vdash M : I}{\Delta | \Gamma \vdash M.f : \theta} \quad \Delta(I).f = \theta$$

$$\frac{\Delta | \Gamma \vdash M : I'}{\Delta | \Gamma \vdash (I)M : I} \quad \Delta \vdash I \leq I' \quad \forall I' \leq I$$

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

$$\frac{\Delta | \Gamma \vdash M : I \quad \bigwedge_{i=1}^n (\Delta | \Gamma \vdash M_i : \theta_i)}{\Delta | \Gamma \vdash M.m(M_1, \dots, M_n) : \theta} \quad \Delta(I).m = \vec{\theta} \rightarrow \theta$$

$$\frac{\Delta | \Gamma \vdash M : \theta' \quad \Delta | \Gamma, x : \theta' \vdash M' : \theta}{\Delta | \Gamma \vdash \text{let } x \equiv M \text{ in } M' : \theta}$$

IMJ: operational semantics

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

S stores object names
+ their types and values

Obj : set of obj. names

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

$$S, a = a' \longrightarrow S, 0/1 \quad a, a' \in \text{Obj}$$

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

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

Program equivalence – the plan

$$M \cong M'$$

same observable behaviour in every context

for all closing contexts C :

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

Program equivalence – the plan

$$M \cong M'$$

same observable behaviour in every context

for all closing contexts C :

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

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

Two distinct sources of undecidability:

I. recursive methods

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

II. objects with methods can be stored

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

Theorem: Termination is undecidable for IMJ_{fin} terms that may feature I or II.

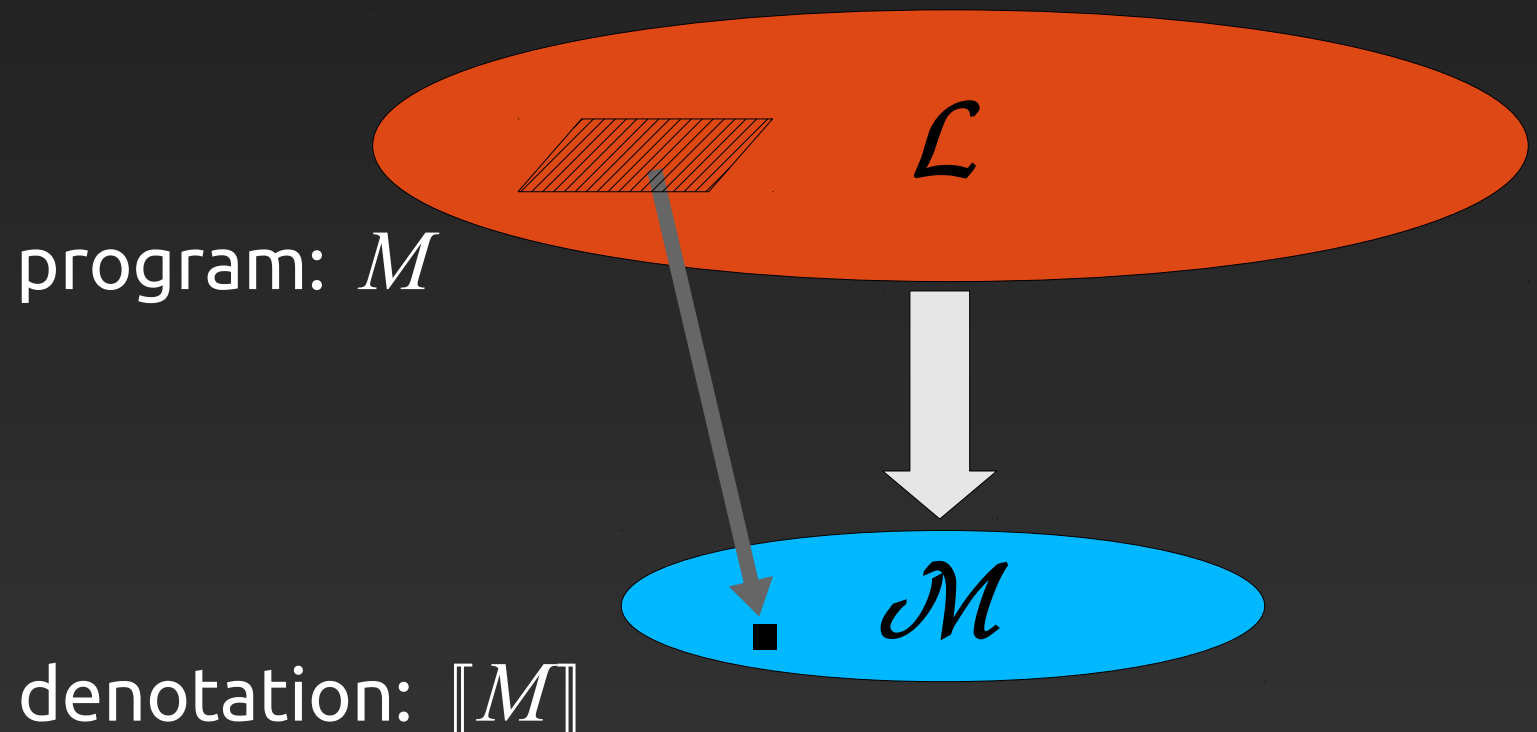
Conversely, given an IMJ_{fin} term M that does not feature I or II, we can always decide whether it terminates.

Games for program equivalence

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

and employ the fully abstract game semantics for IMJ:

$$M \cong M' \Leftrightarrow \llbracket M \rrbracket = \llbracket M' \rrbracket$$



Games for program equivalence

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

and employ the fully abstract game semantics for IMJ:

$$M \cong M' \Leftrightarrow \llbracket M \rrbracket = \llbracket M' \rrbracket$$

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

$$Q \uparrow \Leftrightarrow \llbracket M \rrbracket = \llbracket M' \rrbracket$$

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

$$\llbracket M \rrbracket = \llbracket M' \rrbracket \Leftrightarrow A \sim A'$$

Nominal game semantics

Model the execution of programs as a 2-player game:

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

Qualitative games (no winning conditions)

Computations = *plays* of a specified game

Programs = *strategies* for P

Categories of games, extended with names & effects

- storage: moves with **stores**
- conditions for name **privacy** and **propagation**

Game apparatus

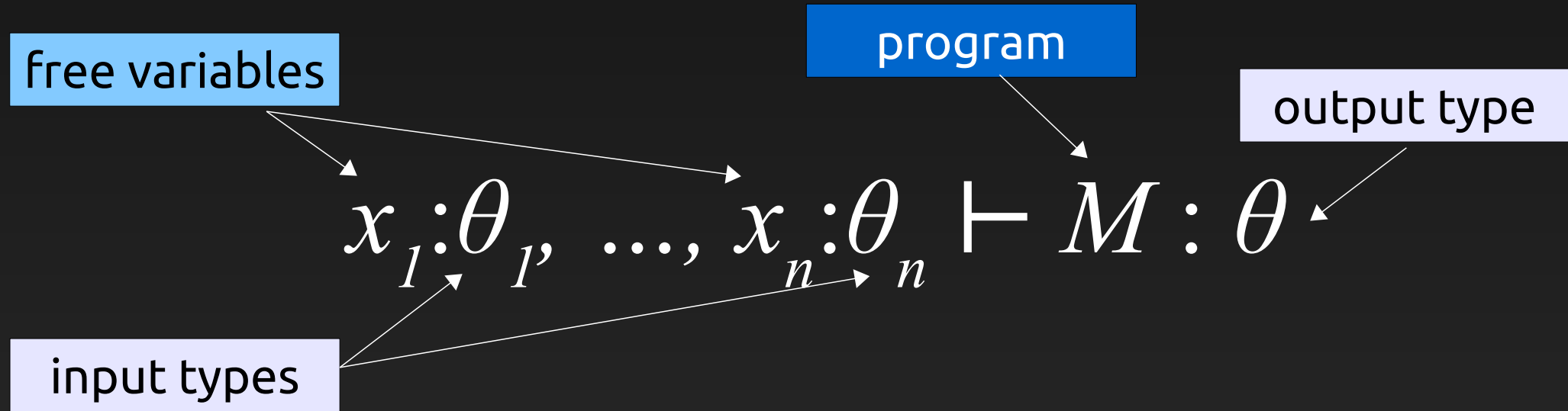
free variables

program

output type

$$x_1 : \theta_1, \dots, x_n : \theta_n \vdash M : \theta$$

input types



Game apparatus

free variables

program

output type

$$x_1:\theta_1, \dots, x_n:\theta_n \vdash M:\theta$$

input types

$$\llbracket M \rrbracket : \llbracket \theta_1, \dots, \theta_n \rrbracket \longrightarrow \llbracket \theta \rrbracket$$

strategy

arenas

Initial game moves

$$[[M]] : [[\theta_1, \dots, \theta_n]] \longrightarrow [[\theta]]$$

strategy

arenas

$$[[\text{void}]] = \{ * \}$$

$$[[\text{int}]] = \{ 0, 1, -1, \dots \}$$

$$[[\mathcal{I}]] = \{ a, b, \dots, n, \dots \} \quad a, b, n, \dots \in \mathcal{N}$$

moves

\mathcal{N} a set of *names*

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:

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

$\text{Ptr}' : (\text{get} : \text{Ptr}' \rightarrow \text{Ptr}', \text{set} : \text{Ptr}' \rightarrow \text{Ptr}')$
 $\text{Pt2}' : (\text{get} : \text{void} \rightarrow (\text{int}, \text{int}), \text{set} : (\text{int}, \text{int}) \rightarrow \text{void})$

IMJ example: game semantics

M_1 : let $u = \text{new}(\text{Var}_{Emp})$ in
 $\text{new}(M_1)$: Cell

M_1 : get: $\lambda_ . u.\text{val}$,
 set: $\lambda y . u.\text{val} := y$

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

O P O P

$\llbracket M_1 \rrbracket = * c^{\Sigma_0} (\text{call } c.\text{get}()^{\Sigma_0} \text{ ret } c.\text{get}(\text{nul})^{\Sigma_0})^*$
 $\text{call } c.\text{set}(n_1)^{\Sigma_1} \text{ ret } c.\text{set}()^{\Sigma_1}$
 $(\text{call } c.\text{get}()^{\Sigma_1} \text{ ret } c.\text{get}(n_1)^{\Sigma_1})^*$
 $\text{call } c.\text{set}(n_2)^{\Sigma_2} \text{ ret } c.\text{set}()^{\Sigma_2} \dots$

$\Sigma_i = \{ c \mapsto (\text{Cell}, \emptyset) \} \cup \{ n_j \mapsto (\text{Empty}, \emptyset) \mid 1 \leq j \leq i \}$

IMJ example: game semantics

$M_2 : \text{let } b = \text{new}(\text{Var}_{Int}) \text{ in}$ $\text{let } u_1 = \text{new}(\text{Var}_{Emp}) \text{ in}$ $\text{let } u_2 = \text{new}(\text{Var}_{Emp}) \text{ in}$ $\text{new}(M_2) : \text{Cell}$	$M_2 : \text{get: } \lambda_ . \text{if } b.\text{val}$ $\text{then } b.\text{val} := 0; u_1.\text{val}$ $\text{else } b.\text{val} := 1; u_2.\text{val},$ $\text{set: } \lambda y . u_1.\text{val} := y;$ $u_2.\text{val} := y$
--	---

$$\begin{aligned}
 \llbracket M_1 \rrbracket = & \quad \color{red}{O} \quad \color{blue}{P} \quad \quad \quad \color{red}{O} \quad \quad \quad \color{blue}{P} \\
 & * c^{\Sigma_0} (\text{call } c.\text{get}()^{\Sigma_0} \text{ ret } c.\text{get}(\text{nul})^{\Sigma_0})^* \\
 & \quad \text{call } c.\text{set}(n_1)^{\Sigma_1} \text{ ret } c.\text{set}()^{\Sigma_1} \\
 & \quad (\text{call } c.\text{get}()^{\Sigma_1} \text{ ret } c.\text{get}(n_1)^{\Sigma_1})^* \\
 & \quad \text{call } c.\text{set}(n_2)^{\Sigma_2} \text{ ret } c.\text{set}()^{\Sigma_2} \dots = \llbracket M_2 \rrbracket
 \end{aligned}$$

$$\Sigma_i = \{ c \mapsto (\text{Cell}, \emptyset) \} \cup \{ n_j \mapsto (\text{Empty}, \emptyset) \mid 1 \leq j \leq i \}$$

Games for program equivalence

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

and employ the fully abstract game semantics for IMJ:

$$M \cong M' \Leftrightarrow \llbracket M \rrbracket = \llbracket M' \rrbracket$$

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

$$Q \uparrow \Leftrightarrow \llbracket M \rrbracket = \llbracket M' \rrbracket$$

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

$$\llbracket M \rrbracket = \llbracket M' \rrbracket \Leftrightarrow A \sim A'$$

One source of Undecidability

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

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

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

we can encode
computations of
queue machines!

step

state

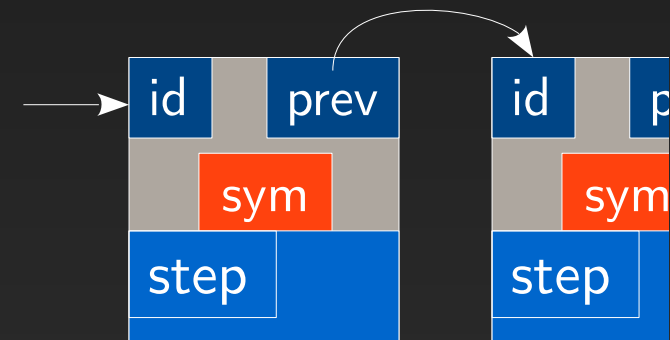
One source of Undecidability

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

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

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

we can encode
computations of
queue machines!



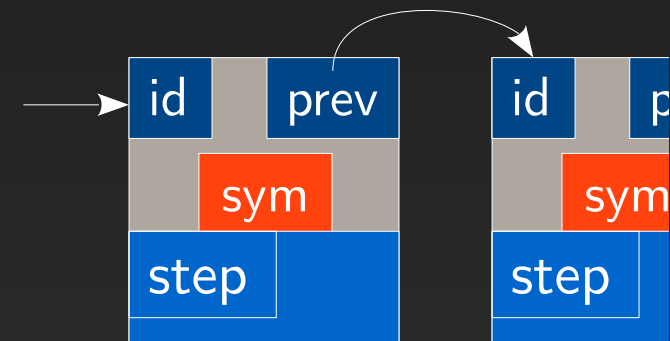
One source of Undecidability

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

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

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

we can encode
computations of
queue machines!



One source of Undecidability

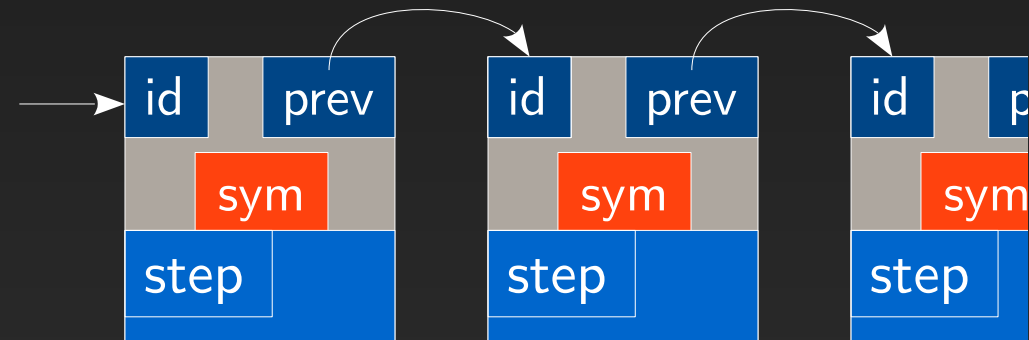
P passes objects of type \mathcal{I} , and \mathcal{I} contains methods

$\mathcal{I} : (\text{step}: \text{void} \rightarrow \text{void})$

$N : (\text{val}: \text{int})$

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

we can encode
computations of
queue machines!



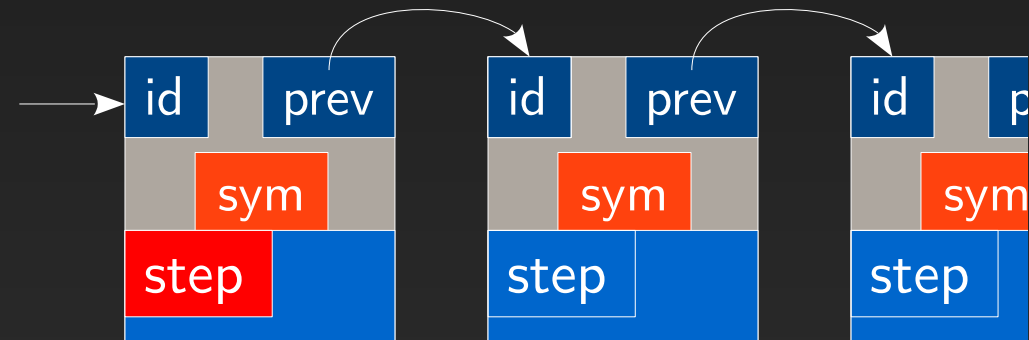
One source of Undecidability

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

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

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

we can encode
computations of
queue machines!



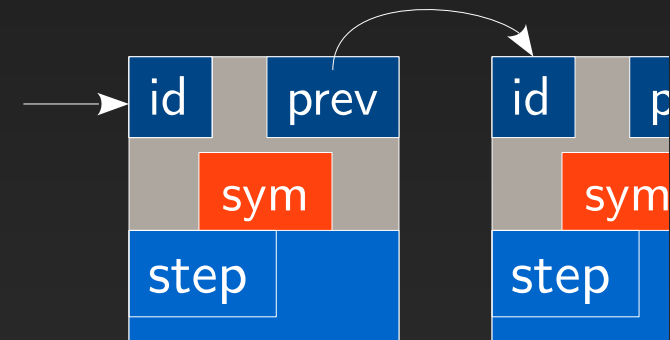
One source of Undecidability

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

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

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

we can encode
computations of
queue machines!



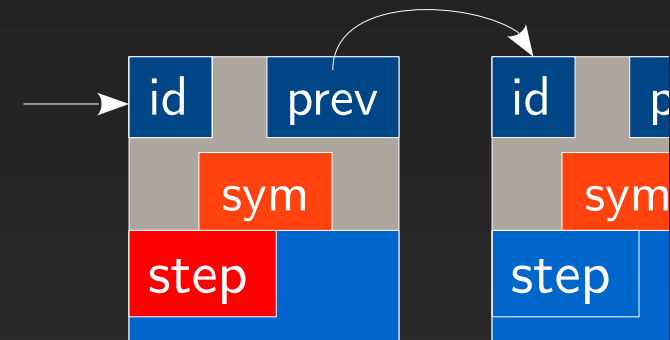
One source of Undecidability

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

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

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

we can encode
computations of
queue machines!



One source of Undecidability

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

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

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

we can encode
computations of
queue machines!

step

state

id p
sym
step

Undecidability

$$x : L \vdash M : R$$

Three cases for undecidability:

- $L = (\dots, m : \mathcal{I} \rightarrow \theta)$
- $R = (\dots, m : \theta \rightarrow \mathcal{I})$
- $R = (\dots, m : \mathcal{I}' \rightarrow \theta)$ where $\mathcal{I}' = (\dots, m : \mathcal{I} \rightarrow \theta)$

and \mathcal{I} contains methods (i.e. is higher-order)

Decidable types – IMJ*

$x : L \vdash M : R$

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

$L ::= \text{void} \mid \text{int} \mid (f : G, m : G \rightarrow L)$

$R ::= \text{void} \mid \text{int} \mid (f : G, m : L \rightarrow G)$

Games for program equivalence

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

and employ the fully abstract game semantics for IMJ:

$$M \cong M' \Leftrightarrow \llbracket M \rrbracket = \llbracket M' \rrbracket$$

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

$$Q \uparrow \Leftrightarrow \llbracket M \rrbracket = \llbracket M' \rrbracket$$

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

$$\llbracket M \rrbracket = \llbracket M' \rrbracket \Leftrightarrow A \sim A'$$

Model checking equivalence

$$\begin{array}{c} M \cong M' \\ \Leftrightarrow \\ \llbracket M \rrbracket = \llbracket M' \rrbracket \end{array}$$

M Programs



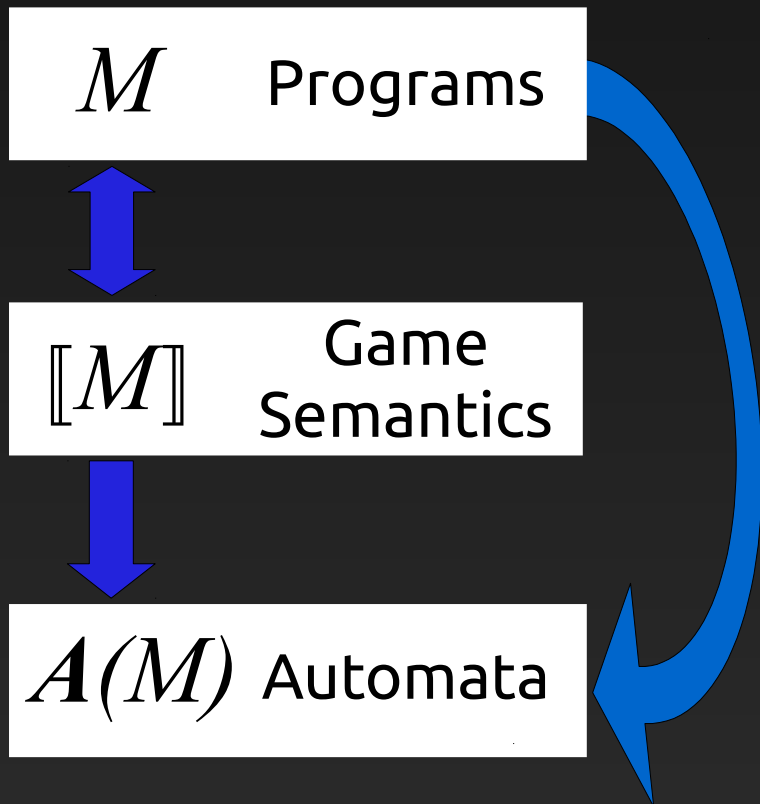
$\llbracket M \rrbracket$ Game Semantics



$A(M)$ Automata



Model checking equivalence



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**


$$\begin{aligned} M \cong M' &\Leftrightarrow \llbracket M \rrbracket = \llbracket M' \rrbracket \\ &\Leftrightarrow A \sim A' \Leftrightarrow A \otimes A' = 0 \end{aligned}$$

Implemented in **Coneqct!**

Coneqct

Atlassian, Inc. (US) <https://bitbucket.org/sjr/coneqct/wiki/Home> Search

Atlassian Bitbucket Features Pricing Find a repository... English Sign up Log in

 sjr coneqct

ACTIONS

- Clone
- Compare
- Fork

NAVIGATION

- Overview
- Source
- Commits
- Branches
- Pull requests
- Wiki
- Downloads 1

Wiki

Clone wiki

coneqct / Home View History

Coneqct: a contextual equivalence checking tool for Interface Middleweight Java

[Home](#) | [Downloads](#) | [Syntax](#) | [Examples](#)

Requirements

The checker runs on the .NET platform (≥ 4.5), and hence requires a recent implementation of the .NET Common Language Infrastructure (CLI) to be installed on your system.

- On Windows we recommend Microsoft's ".NET Framework", the latest stable version is 4.5.2: <https://www.microsoft.com/en-us/download/details.aspx?id=42643>.
- On Linux or Mac we recommend Xamarin's "Mono": <http://www.mono-project.com/download/>

Installation

- Download the latest assemblies from [downloads](#). All the required assemblies are packaged together in a zip file named "coneqct-XXX.zip" where "XXX" denotes the revision number.
- Unzip to any convenient location, this creates a new directory "coneqct-XXX" in which resides the executable "coneqct.exe".
- To verify that all is well, on the command line navigate to the directory "coneqct-XXX" and run the command:
 - "coneqct.exe" on Windows or,
 - "mono ./coneqct.exe" on Linux or Mac. If the installation is working correctly, the usage message will be printed out to the terminal.

Usage

On Windows, to check the equivalence of two IMJ terms defined in the file "terms.inp", run:

```
> .\coneqct.exe \path\to\terms.inp
```

On Linux or Mac you should prefix this command by "mono" (and use appropriate slashes):

```
> mono ./coneqct.exe /path/to/terms.inp
```

A number of example inputs are bundled with the installation. For example, after navigating to the root of the directory "coneqct-XXX", to verify the "extended types" equivalence adapted from Benton and Leperchey's "Relational Reasoning in a Nominal Semantics for Storage" on Mac, run:

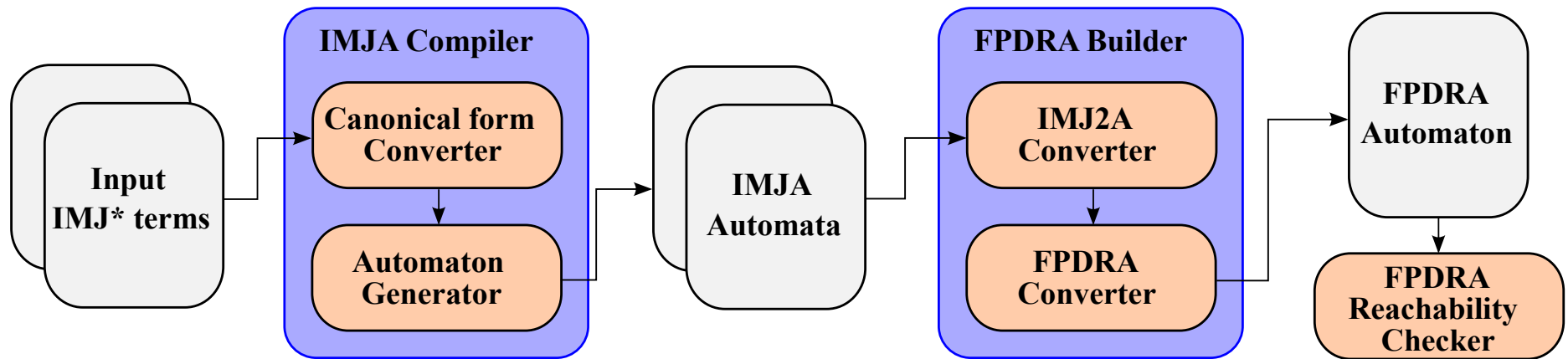
```
> mono ./coneqct.exe inputs/inp2.imj
```

See [Syntax](#) for a detailed description of the syntax of the input file format.

The tool can be configured using some command-line options:

- maxint <int>: set the value of maxint to <int>

Coneqct architecture



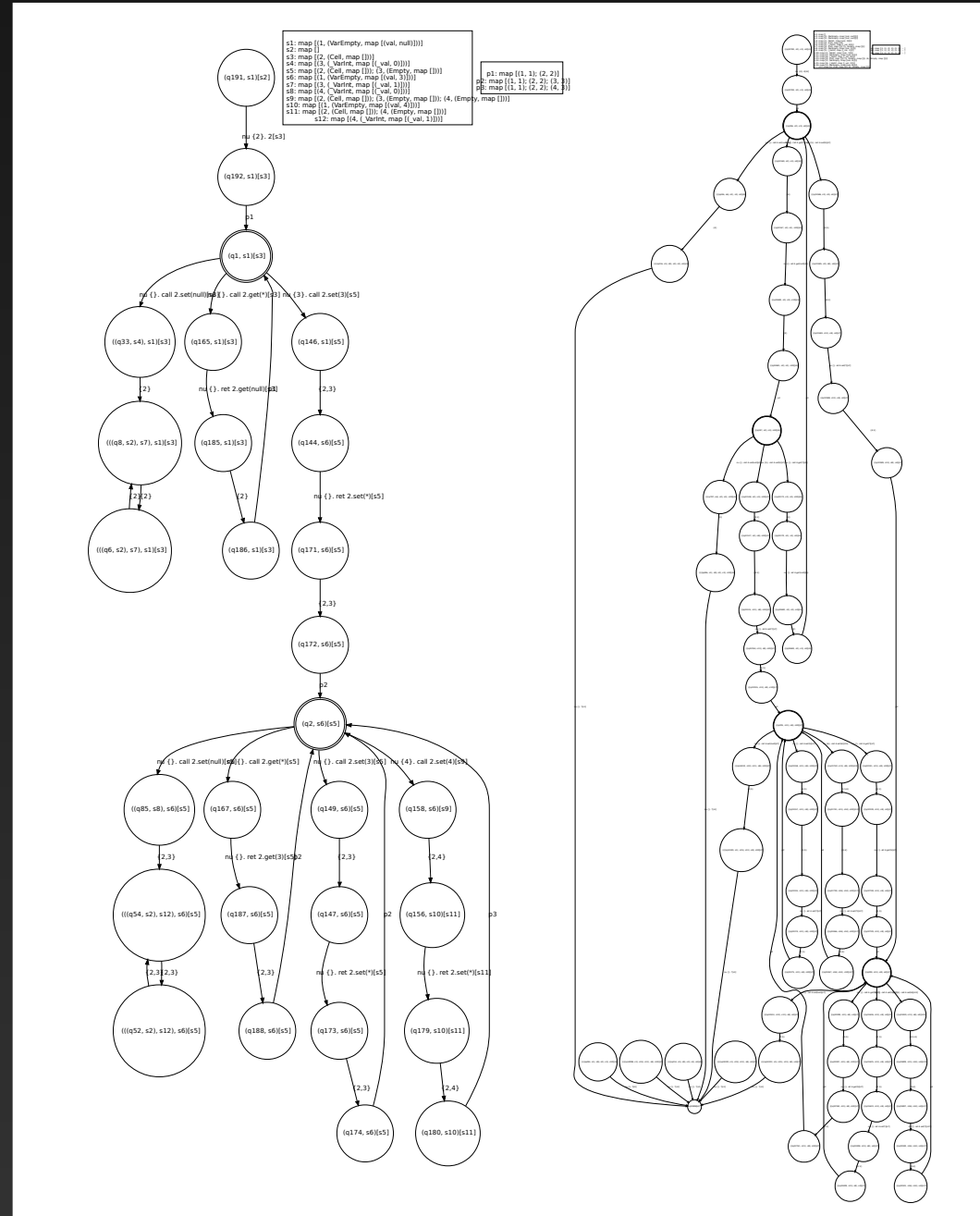
FPDRA : *Fresh Pushdown Register Automata*

- MRT: Reachability in Pushdown Register Automata, MFCS'14
- MT: Algorithmic Games for Full Ground References, ICALP'12

Coneqct example

M_1 : let $v = \text{new } \{ _ : \text{Var}_{\text{Empty}} \}$ in
 new $\{ _ : \text{Cell};$
 get : $\lambda _ . v.\text{val},$
 set : $\lambda y . \text{if } y = \text{null} \text{ then div}$
 else $v.\text{val} := y ; w.\text{val} := y$

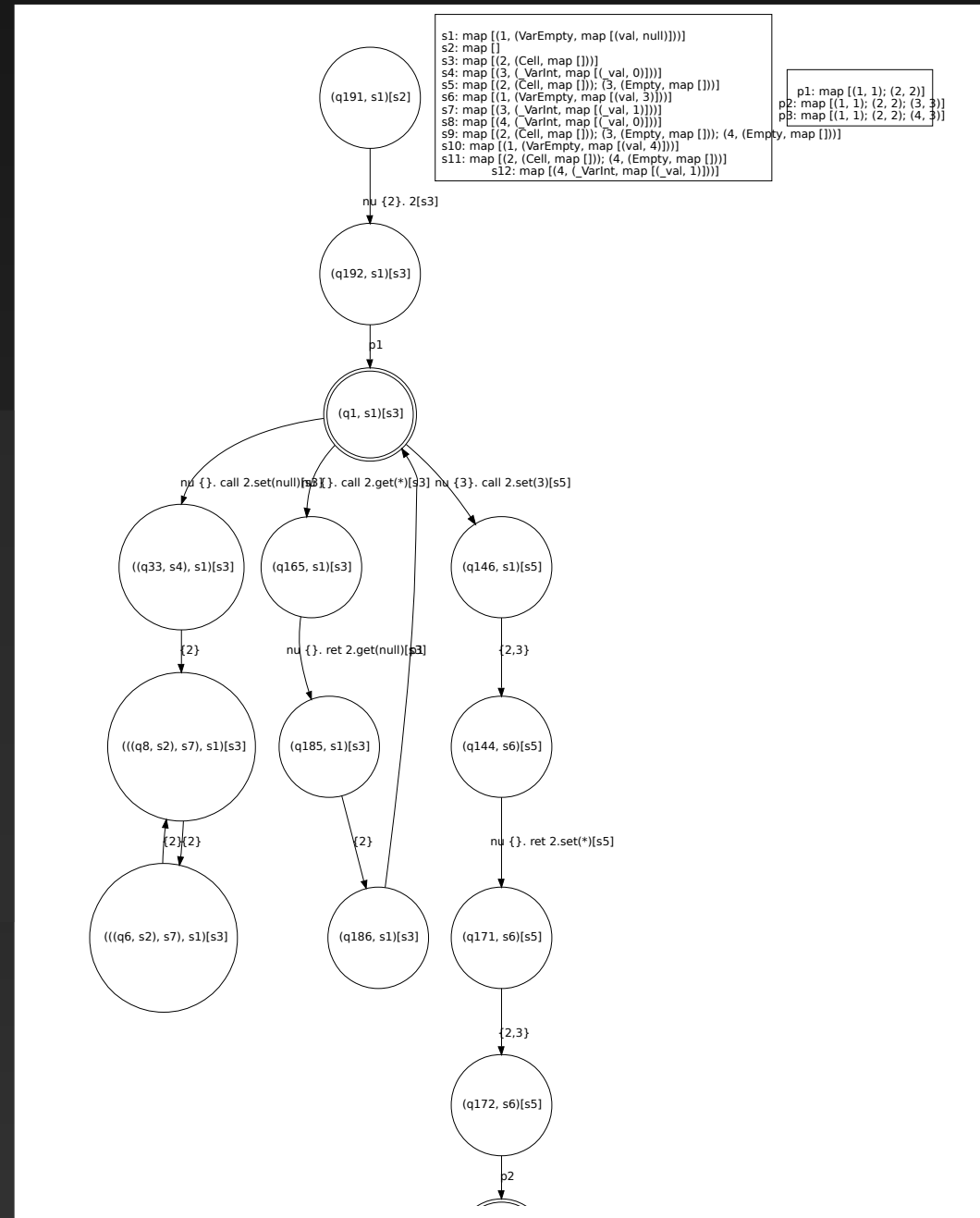
M_2 : let $b = \text{new } \{ _ : \text{Var}_{\text{Int}} \}$ in
 let $v = \text{new } \{ _ : \text{Var}_{\text{Empty}} \}$ in
 let $w = \text{new } \{ _ : \text{Var}_{\text{Empty}} \}$ in
 new $\{ _ : \text{Cell};$
 get : $\lambda _ . \text{if } b.\text{val} = 1 \text{ then } b.\text{val} := 0 ; v.\text{val}$
 else $b.\text{val} := 1 ; w.\text{val},$
 set : $\lambda y . \text{if } y = \text{null} \text{ then div}$
 else $v.\text{val} := y ; w.\text{val} := y$



Coneqct example

M_1 : let $v = \text{new } \{ _ : \text{Var}_{\text{Empty}} \}$ in
 new $\{ _ : \text{Cell};$
 get : $\lambda _ . v.\text{val},$
 set : $\lambda y . \text{if } y = \text{null} \text{ then div}$
 else $v.\text{val} := y ; w.\text{val} := y$

M_2 : let $b = \text{new } \{ _ : \text{Var}_{\text{Int}} \}$ in
 let $v = \text{new } \{ _ : \text{Var}_{\text{Empty}} \}$ in
 let $w = \text{new } \{ _ : \text{Var}_{\text{Empty}} \}$ in
 new $\{ _ : \text{Cell};$
 get : $\lambda _ . \text{if } b.\text{val} = 1 \text{ then } b.\text{val} := 0 ; v.\text{val}$
 else $b.\text{val} := 1 ; w.\text{val},$
 set : $\lambda y . \text{if } y = \text{null} \text{ then div}$
 else $v.\text{val} := y ; w.\text{val} := y$



Wrapping up

Contextual equivalence

- decidability characterised via games
- automated decision via FPDRA's → Coneqct

Further on

- sound methods for the whole of IMJ
- general model checking and logics

References

- MRT: Game Semantic Analysis of Equivalence in IMJ, ATVA'15
- MRT: A Contextual Equivalence Checker for IMJ*, ATVA'15