GAME SEMANTICS
FOR
INTERFACE MIDDLEWEIGHT JAVA

Andrzej S. Murawski
WARWICK

Steven J. Ramsay
OXFORD

Nikos Tzevelekos
QUEEN MARY LONDON
WHAT IS THIS TALK ABOUT?

a fully abstract denotational model for a core fragment of Java

classification of decidable cases for contextual equivalence
CAPTURING PROGRAM BEHAVIOUR

```
int CockLevel = 5;
int Red = ((int CockLevel - 5) / 5);
int Green = ((int CockLevel - 5) / 5);
int Blue = ((int CockLevel - 5) / 5);
if intRed < 0 then intRed = 0;
if intGreen < 0 then intGreen = 0;
if intBlue < 0 then intBlue = 0;
if intLevel = 1 then color limb intCurrentBread;
color limb intCurrentBread endif;
```

```
ciel = new Capitur.Ciel;
ciel.nombresetoiles = 50;
ciel.couleur = 'bleu';
ciel.dessinentoile = 'dessin.png';
navette = new Capitur.Navette;
navette.dessin = 'navette.png';
navette.vitesse = 100;
```
FULL ABSTRACTION

\[ [M_1] = [M_2] \]

if and only if

\[ M_1 \simeq M_2 \]
Game semantics: from PCF to ML

Full Abstraction for PCF (early 90's)

Games for variants of Idealized Algol
  • Non-determinism, exceptions, probability, concurrency, polymorphism, ...

Nominal game semantics (2004-)
  • *Use names for dynamic resource generation*
    → fragments of ML, CML, C, now Java
The need for names

References as *pairs*:

\[
\text{ref int } = (\text{unit } \to \text{int}) \times
\begin{align*}
\text{int } & \to \text{unit} \\
\Rightarrow & \quad (1 \to \text{Z}) \times (\text{Z } \to 1)
\end{align*}
\]

- Theoretically attractive
- but: \text{mkvar}(R, H), all R, H (bad variables)
The need for names

References as *pairs*:

\[ \text{ref } \text{int } = (\text{unit } \rightarrow \text{int}) \times (\text{int } \rightarrow \text{unit}) \]

\[ \implies (1 \rightarrow \mathbb{Z}) \times (\mathbb{Z} \rightarrow 1) \]

- Theoretically attractive
- but: \( \text{mkvar} (R, H) \), all \( R, H \) (bad variables)

References as *names*:

\[ \text{ref } \text{int } = \text{base type} \]

\[ \implies \mathbb{A} \ (\text{reference names}) \]

- Notion of resource (name):
  - atomic values
  - infinitely many
  - comparable for equality
Interface Middleweight Java (IMJ)

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

Interface definitions\[ \Theta ::= \emptyset \mid (f : \theta), \Theta \mid (m : \theta \rightarrow \theta), \Theta \]

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)

Terms

\[ M ::= \text{skip} \mid n \mid \text{null} \mid x \mid i \mid \text{M} \oplus \text{M} \mid \text{if M M M} \]
\[ \mid \text{let x=M in M} \mid \text{M=M} \mid (I)\text{M} \]
\[ \mid \text{new(x:I; M)} \mid \text{M.f} \mid \text{M.f := M} \mid \text{M.m} (\overline{M}) \]

Method implementations

\[ M ::= \emptyset \mid (m : \lambda x. M), M \]

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

- Objects, inheritance, casting, interfaces
$M_1 : \text{let } u = \text{new}(\text{Var}_{\text{Emp}}) \text{ in} $

new($M_1$) : Cell

$M_1 : \text{get: } \lambda().\ u.\text{val},$

set: $\lambda y.\ u.\text{val} := y$

$\Delta = \text{Empty: } \emptyset,$

Cell: (get: void $\rightarrow$ Empty,

set: Empty $\rightarrow$ void),

$\text{Var}_{\text{Emp}} : (\text{val: Empty}),$

$\text{Var}_{\text{Int}} : (\text{val: int})$

* Koutavas & Wand, 2007*
**IMJ example**

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

\[ M_1 : \text{get: } \lambda(). u.\text{val}, \]
\[ \text{set: } \lambda y. u.\text{val} := y \]

\[ M_2 : \text{let } b = \text{new}(\text{Var}_{Int}) \text{ in} \]
\[ \text{let } u_1 = \text{new}(\text{Var}_{Emp}) \text{ in} \]
\[ \text{let } u_2 = \text{new}(\text{Var}_{Emp}) \text{ in} \]
\[ \text{new}(M_2) : \text{Cell} \]

\[ \Delta = \text{Empty}: \emptyset, \]
\[ \text{Cell: } (\text{get: } \text{void} \rightarrow \text{Empty}, \]
\[ \text{set: } \text{Empty} \rightarrow \text{void}), \]
\[ \text{Var}_{Emp} : (\text{val: } \text{Empty}), \]
\[ \text{Var}_{Int} : (\text{val: } \text{int}) \]

\[ M_2 : \text{get: } \lambda(). \text{if } b.\text{val} \]
\[ \text{then } b.\text{val} := 0; u_1.\text{val} \]
\[ \text{else } b.\text{val} := 1; u_2.\text{val}, \]
\[ \text{set: } \lambda y. u_1.\text{val} := y; \]
\[ u_2.\text{val} := y \]

* Koutavas & Wand, 2007
Game Semantics

Computation is modelled as a 2-player game between:

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

Qualitative games ($\neq$ Game Theory)

Computations = *plays* of a specified game

Programs = *strategies* for $P$

Strategy composition $\rightarrow$ *categories* of games
Plays, strategies

\[ M : [I_{inp}] \rightarrow [I_{out}] \]

**Plays**: sequences of *moves-with-store*

**Strategies**: sets of plays

- moves have polarities \((O/P)\), which alternate
- \(P\) calls methods of \(O\), and vice versa; dually for returns
- calls and returns obey the object interfaces
- strategies are closed wrt to \(O\)-name subtyping

\[
\text{call } n.set(12) \ (n \mapsto \text{IntCell}, \text{val=5}), \ ...
\]
IMJ example: game semantics

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

\[ M_1 : \] get: \( \lambda(). u.\text{val}, \)
set: \( \lambda y. u.\text{val} := y \)

\[ \Delta = \] Empty: \( \emptyset \),
Cell: (get: void \( \rightarrow \) Empty,
set: Empty \( \rightarrow \) void),
\text{Var}_{\text{Emp}}: (\text{val: Empty}),
\text{Var}_{\text{int}}: (\text{val: int})

\[ [M_1] = \ast n^{\Sigma_0} \]
\( ( \text{call } n.\text{get()}^{\Sigma_0} \text{ ret } n.\text{get}(\text{nul})^{\Sigma_0} )^* \)
\( \text{call } n.\text{set}(n_1)^{\Sigma_1} \text{ ret } n.\text{set}()^{\Sigma_1} \)
\( ( \text{call } n.\text{get()}^{\Sigma_1} \text{ ret } n.\text{get}(n_1)^{\Sigma_1} )^* \)
\( \text{call } n.\text{set}(n_2)^{\Sigma_2} \text{ ret } n.\text{set}()^{\Sigma_2} \ldots \)

\[ \Sigma_i = \{ n \mapsto (\text{Cell, } \emptyset) \} \cup \{ n_j \mapsto (\text{Empty, } \emptyset), 1 \leq j \leq i \} \]
**IMJ example: game semantics**

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

\[ \text{new}( M_1 ) : \text{Cell} \]

\[ M_1 : \text{get: } \lambda(). u.\text{val}, \]

\[ \text{set: } \lambda y. u.\text{val} : = y \]

\[ \Delta = \text{Empty: } \emptyset, \]

\[ \text{Cell: } (\text{get: void } \to \text{Empty}, \]

\[ \text{set: Empty } \to \text{void}), \]

\[ \text{Var}_{\text{Emp}} : (\text{val: Empty}), \]

\[ \text{Var}_{\text{Int}} : (\text{val: int}) \]

\[
\begin{align*}
O & \quad P \\
[M_1] = \ast & n^{\Sigma_0} (\text{call } n.\text{get}()^{\Sigma_0} \text{ ret } n.\text{get}(\text{nul})^{\Sigma_0} )^* \\
& \text{call } n.\text{set}(n_1)^{\Sigma_1} \text{ ret } n.\text{set}()^{\Sigma_1} \\
& (\text{call } n.\text{get}()^{\Sigma_1} \text{ ret } n.\text{get}(n_1)^{\Sigma_1} )^* \\
& \text{call } n.\text{set}(n_2)^{\Sigma_2} \text{ ret } n.\text{set}()^{\Sigma_2} \\
& \ldots \\
\Sigma_i = \{ n \mapsto (\text{Cell, } \emptyset) \} \cup \{ n_j \mapsto (\text{Empty, } \emptyset), 1 \leq j \leq i \} 
\end{align*}
\]
IMJ example: game semantics

$M_1: \begin{align*}
\text{let } u &= \text{new}(\text{Var}_{\text{Emp}}) \text{ in } \\
\text{new}(M_1) &\quad : \text{Cell} \\
M_1: &\quad \text{get: } \lambda(). u.\text{val}, \\
&\quad \text{set: } \lambda y. u.\text{val} \coloneqq y
\end{align*}$

$\Delta = \begin{align*}
\text{Empty: } \emptyset, \\
\text{Cell: } (\text{get: } \text{void} \to \emptyset, \\
&\quad \text{set: } \emptyset \to \text{void}), \\
\text{Var}_{\text{Emp}}: (\text{val: } \emptyset), \\
\text{Var}_{\text{Int}}: (\text{val: } \text{int})
\end{align*}$

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

$\mathcal{O} \quad \mathcal{P} \quad \mathcal{O} \quad \mathcal{P}$

$$[M_1] = \ast n^{\Sigma_0} ( \text{call } n.\text{get()}^{\Sigma_0} \text{ ret } n.\text{get(nul)}^{\Sigma_0} \ast^* \\
\text{call } n.\text{set}(n_1)^{\Sigma_1} \text{ ret } n.\text{set()}^{\Sigma_1} \\
( \text{call } n.\text{get()}^{\Sigma_1} \text{ ret } n.\text{get}(n_1)^{\Sigma_1} \ast^* \\
\text{call } n.\text{set}(n_2)^{\Sigma_2} \text{ ret } n.\text{set()}^{\Sigma_2} \ldots$$
**IMJ example: game semantics**

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

\[ M_1 : \text{get: } \lambda(). u.\text{val}, \]
\[ \text{set: } \lambda y. u.\text{val} := y \]

\[
\begin{align*}
\Delta & = \text{Empty: } \emptyset, \\
& \quad \text{Cell: } (\text{get: } \text{void} \rightarrow \text{Empty}, \text{set: } \text{Empty} \rightarrow \text{void}), \\
& \quad \text{Var}_{\text{Emp}} : (\text{val: } \text{Empty}), \\
& \quad \text{Var}_{\text{Int}} : (\text{val: } \text{int})
\end{align*}
\]

\[
\begin{align*}
[M_1] & = \ast n \Sigma_0 (\text{call } n.\text{get()} \Sigma_0 \text{ ret } n.\text{get}(\text{nul}) \Sigma_0 )^* \\
& \quad \text{call } n.\text{set}(n_1) \Sigma_1 \text{ ret } n.\text{set()} \Sigma_1 \\
& \quad (\text{call } n.\text{get()} \Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \\
& \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set()} \Sigma_2 \ldots
\end{align*}
\]

\[
\Sigma_i = \{ n \mapsto (\text{Cell, } \emptyset) \} \cup \{ n_j \mapsto (\text{Empty, } \emptyset), 1 \leq j \leq i \}
\]
IMJ example: game semantics

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

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

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

$O \quad P$

$[M_1] = * n \Sigma_0 ( \text{call } n.\text{get}(\Sigma_0 \text{ ret } n.\text{get}(\text{null}) \Sigma_0 )^* \quad \text{call } n.\text{set}(n_1) \Sigma_1 \text{ ret } n.\text{set}(\Sigma_1)$

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

$O \quad P$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$

$\quad \text{call } n.\text{get}(\Sigma_1 \text{ ret } n.\text{get}(n_1) \Sigma_1 )^* \quad \text{call } n.\text{set}(n_2) \Sigma_2 \text{ ret } n.\text{set}(\Sigma_2)$
**IMJ example: game semantics**

<table>
<thead>
<tr>
<th>$M_2$ :</th>
<th>let $b = \text{new(Var}_{\text{Int}}) \text{ in}$</th>
<th>$M_2$ : get: $\lambda()$. if $b$.val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>let $u_1 = \text{new(Var}_{\text{Emp}}) \text{ in}$</td>
<td>then $b$.val := 0; $u_1$.val</td>
</tr>
<tr>
<td></td>
<td>let $u_2 = \text{new(Var}_{\text{Emp}}) \text{ in}$</td>
<td>else $b$.val := 1; $u_2$.val, set: $\lambda y. u_1$.val := $y$;</td>
</tr>
<tr>
<td></td>
<td>new($M_2$) : Cell</td>
<td>$u_2$.val := $y$</td>
</tr>
</tbody>
</table>

| \[ [M_1] = \ast n \sum_0 (\text{call } n.\text{get()} \sum_0 \text{ ret } n.\text{get(nul)} \sum_0 )^{*} \]
|---|---|
| \[ \text{call } n.\text{set}(n_1) \sum_1 \text{ ret } n.\text{set()} \sum_1 \]
| \[ (\text{call } n.\text{get()} \sum_1 \text{ ret } n.\text{get}(n_1) \sum_1 )^{*} \]
| \[ \text{call } n.\text{set}(n_2) \sum_2 \text{ ret } n.\text{set()} \sum_2 \cdots \]
| \[ = [M_2] \]

$\sum_i = \{ n \mapsto (\text{Cell, } \emptyset) \} \cup \{ n_j \mapsto (\text{Empty, } \emptyset), 1 \leq j \leq i \}
Full abstraction for IMJ

**Lemma.** The game model is sound

**Lemma.** Every finitary strategy is IMJ-definable

**Theorem.** The game model is fully abstract

\[ P \subseteq P' \iff [P] \subseteq [P'] \]
Related work on objects

Domain models
Related work on objects

Domain models

Environmental bisimulations

TAOOP'94

Two Semantic Models of Object-Oriented Languages
Samuel N. Kamim
Uday S. Reddy
University of Illinois at Urbana-Champaign
October 4, 1993

Abstract
We present and compare two models of object-oriented language closure model behavior by using closures to encapsulate side effects.

FOOL/WOOD'99

Dynamic Denotational Semantics of Java
Jim Alves-Foss and Fong Shing Lam
Center for Secure and Dependable Software, Department of Computer Science, University of Idaho, Moscow, ID 83844-1100, USA

FOOL/WOOD'07

Reasoning about Class Behavior
Vladan Josifovic and Mitchell Wand
Northeastern University
vjosefovic@cs.northeastern.edu

1. Introduction
The class is a facility to divide programs into small units that encapsulate different parts of the entire program behavior. This makes classes attractive for reuse and re-implementation. But changing the implementation of a class that is being used in a program can cause the program to become non-terminating because of changes in the program's behavior. This can lead to the need for changes in the behavior of the program.

FSSJava'99

Dynamic Denotational Semantics of Java
Jim Alves-Foss and Fong Shing Lam
Center for Secure and Dependable Software, Department of Computer Science, University of Idaho, Moscow, ID 83844-1100, USA

To formalize the notion of equivalent implementations of classes we adopt the standard notion of contextual equivalence between expressions from functional languages [19] to an equivalent between classes in class-based languages. Classes C and C' are contextually equivalent, if and only if, for all executable contexts C[], expressions e, and the empty state s, the program configurations C[e], C's, and C'[e], C's have the same operational behavior.

Another way of reasoning about the behavior of class implementations is by using denotational methods (see [6, 23]). Denotations are usually compositional in the sense that they give the meaning of program fragments without the quantification over contexts. Nevertheless, the usual denotational methods distinguish equivalent class-implementations that have different local state behaviors. For example, the two implementations of a Cell class in Figure 1 would have different denotations because the two different kinds of class implementations can be dealt with by methods that build logical relations of denotations [2], or exploit properties of some programs, such as ownership confinement [3]. However, these methods are still not complete with respect to contextual equivalence.

A more natural way to reason about the behavior of two program fragments is by using bisimulations. Bisimulations were introduced by Hennessy and Milner [9] for reasoning about the behavior of concurrent programs. They were applied in sequential calculi by Abramsky [7] and by Riecke [13] for proving that they are a congruence. Sann and Pierce later gave a big-step bisimulation proof technique which is sound and complete with respect to contextual equivalence in a language with dynamic scoping [22] and in a language with recursive and polymorphic types [21]. Their key innovation was to split inputs into parts, and associate each part with the conditions of knowledge under which that part holds, building


Related work on objects

Domain models

Environmental bisimulations

Trace models

**TAOOP'94**

Two Semantic Models of Object-Oriented Languages
Samuel N. Kamin
Uday S. Reddy
University of Illinois at Urbana-Champaign
October 4, 1993

**FSSJava'99**

Dynamic Denotational Semantics of Java
Jim Alves-Foss and Fong-Shing Lam
Center for Secure and Dependable Software, Department of Computer Science, University of Idaho, Moscow, ID 83844-0100, USA

**FOOL/WOOD'07**

Reasoning about Class Behavior
Vladislav Kostovs
Northeastern University
vkosov@ccs.neu.edu

**ESOP'03**

Java Jr.: Fully abstract trace semantics for a core Java language.
Alan Jeffrey*1 and Julian Rathke*
1 Bell Labs, Lucent Technologies, Murray Hill, NJ, USA
2 University of Sussex, Brighton, UK

**FMCO'04**

Observability, Connectivity, and Replay in a Sequential Calculus of Classes
Erika Ábrahám and Marcello M. Bonsangue and Frank S. de Boer and Andreas Grüner and Martin Stettin
1 Christian-Albrechts-University Kiel, Germany
2 Albert-Ludwigs-University Freiburg, Germany
3 University of Amsterdam, The Netherlands
4 CWI Amsterdam, The Netherlands

**TCS'05**

A fully abstract may testing semantics for concurrent objects
Alan Jeffrey*1,2,3, Julian Rathke*1,2
1 School of CTS, Delft University, 242 E. Washburn Ave, Chicago, IL 60614, USA
2 Bell Labs, Lucent Technologies, 7901 South Lake, Loja, IL 68652, USA
3 School of Informatics, University of Sussex, Brighton, UK

Received 24 October 2002, received in revised from 30 December 2002, accepted 6 October 2004

Abstract
This paper provides a fully abstract semantics for a variant of the concurrent object calculus.
VERIFICATION

\[ M_1, M_2 \text{ contextually equivalent} \iff [M_1] = [M_2] \iff A_{M_1} \approx A_{M_2} \]
SOURCES OF UNDECIDABILITY

- arithmetic
- recursive definitions (datatypes and methods)
- storage of method-carrying objects in fields
- “higher-order” types

\[ o_1 : I_1, \ldots, o_k : I_k \vdash M : I \]
HIGHER-ORDER TYPES

bad

\[ \vdash \bullet \rightarrow (\bullet \rightarrow \bullet) \]
\[ \vdash (\bullet \rightarrow \bullet) \rightarrow \bullet \]
\[ (\bullet \rightarrow \bullet) \rightarrow \bullet \]
\[ \vdash \]

good

\[ G ::= \text{void} \mid \text{int} \mid f : \vec{G} \]
\[ L ::= \text{void} \mid \text{int} \mid (f : \vec{G}, \ m : \vec{G} \rightarrow L) \]
\[ R ::= \text{void} \mid \text{int} \mid (f : \vec{G}, \ m : \vec{L} \rightarrow G) \]
AUTOMATA THEORY
OVER INFINITE ALPHABETS

\[ n_1 \# n_2 \cdots n_r \]

• RA
  language equivalence (det) \textit{co-NP-complete} [LICS’15]
  bisimilarity \textit{PSPACE-complete} [LICS’15]

• PDRA
  emptiness \textit{EXPTIME-complete} [MFCS’14]
  bisimilarity \textit{undecidable} [LICS’15]
  HO emptiness \textit{undecidable} [MFCS’14]

• FPDRA
  emptiness \textit{EXPTIME-complete} [ICALP’12, MFCS’14]
These are pairs of maps which describe how registers from the two constituent IMJA which makes
then they will both consume this call move and step into states
respectively and, moreover, the same object name is contained in register
IMJA and register
names contained in the registers of the two IMJA which makes
To synchronise, matching names have
emptiness checking
From IMJA to FPDRA.
Overall, the translation in
alphabet \([2]\) and, hence, their stacks can be synchronised. This is possible, because IMJA operate over a visibly pushdown al-
construct a kind of symmetric difference automaton, which accepts exactly those words
From IMJA to IMJ2A.
reachable in the finite transition graph of the IMJA).
IMJA are produced, we remove states that are not graph-reachable from the initial state
is faithful to that description, so we shall not discuss it further here. As soon as the
pair of IMJA. This translation is extensively documented in \([8]\) and our implementation
\(q_1\) and \(q_2\) of the second IMJA and there is a correspondence between the
\(q_1\) and \(q_2\) of the first IMJA \(q_0\), \(q_1\) and \(q_2\) of the second IMJA \(q_0\).

The first transformation is from the pair input terms to a

\(\text{IMJA Compiler} \rightarrow \text{IMJA Automata} \rightarrow \text{FPDRA Builder} \rightarrow \text{FPDRA Automaton} \rightarrow \text{FPDRA Reachability Checker} \)

\(\text{IMJ}^* \rightarrow \text{IMJA} \rightarrow \text{IMJ2A} \rightarrow \text{FPDRA} \rightarrow \text{Reachability Checker} \)

\([\text{POPL’14}]\)
\([\text{MFCS’14}]\)
\([\text{ICALP’12}]\)

\([\text{ATVA’15}]\)
\([\text{ATVA’15}]\)
\([\text{MFCS’14}]\)

Andrzej S. Murawski, Steven J. Ramsay, Nikos Tzevelekos: A Contextual Equivalence
Checker for IMJ *. ATVA 2015: 234-240
COMPARISON


FUTURE WORK

• polymorphism
• soundness and incompleteness