Compositional Equivalence Checking of Imperative Programs: A Game-Semantic Approach

Luke Ong

Oxford University Computing Laboratory

Intel Symposium, Technion, 8 Sep 2009

Model checking: Extremely successful in verifying finite-state processes. E.g. digital circuits and communication protocols.

Over the past decade, huge strides made in **verification of 1st-order imperative programs**. Many tools: SLAM, Blast, SatAbs, etc.

State-of-the-art tools use abstraction techniques, as exemplified by CEGAR (Counter-Example Guided Abstraction Refinement), and acceleration methods such as SAT- and SMT-solvers.

An Alternative Approach

Start from an accurate denotational semantics of the program; then derive an appropriate model of computation sufficiently concrete (and tractable) for verification.

Advantages: Soundness and completeness inherited by the model; method remains compositional.

Is there such a semantics?

Software Model Checking

Model checking: Extremely successful in verifying finite-state processes. E.g. digital circuits and communication protocols.

Over the past decade, huge strides made in **verification of 1st-order imperative programs**. Many tools: SLAM, Blast, SatAbs, etc.

State-of-the-art tools use abstraction techniques, as exemplified by CEGAR (Counter-Example Guided Abstraction Refinement), and acceleration methods such as SAT- and SMT-solvers.

An Alternative Approach

Start from an accurate denotational semantics of the program; then derive an appropriate model of computation sufficiently concrete (and tractable) for verification.

Advantages: Soundness and completeness inherited by the model; method remains compositional.

Is there such a semantics?

[Abramsky, Jagadeesan, Malacaria, Hyland, O., Hanno, McCusker, etc.]

Game semantics has emerged as a powerful paradigm for giving semantics to a wide range of programming languages (procedural, higher-order functional, polymorphic, reference types, non-local control, concurrent, probabilistic, etc.).

These models are highly accurate (fully abstract).

Promising features of game semantics

- Clear operational content, while admitting compositional methods in the style of denotational semantics.
- Strategies are highly-constrained processes, admitting automata-theoretic representations.
- Rich mathematical structures yielding accurate models of advanced high-level programming languages.

[Abramsky, Jagadeesan, Malacaria, Hyland, O., Hanno, McCusker, etc.]

Game semantics has emerged as a powerful paradigm for giving semantics to a wide range of programming languages (procedural, higher-order functional, polymorphic, reference types, non-local control, concurrent, probabilistic, etc.).

These models are highly accurate (fully abstract).

Promising features of game semantics

- Clear operational content, while admitting compositional methods in the style of denotational semantics.
- Strategies are highly-constrained processes, admitting automata-theoretic representations.
- Rich mathematical structures yielding accurate models of advanced high-level programming languages.

Challanges of the Approach

To carry over methods of model checking to much more structured, modern programming situations, in which the following features are important:

- data-types: references (pointers), recursive types
- non-local control flow: exceptions, call-cc, etc.
- modularity principles: e.g. object orientation: inheritance and subtyping
- higher-order features: higher-order procedures; closures; components
- variables and names: passing mechanisms, life-span, scoping rules
- concurrency and non-determinism: synchronization, multithreading, etc.

Aim:

Combine results and insights in (game) semantics, with techniques in verification.

Outline

1. Idealized Algol and Observational Equivalence
2. Game Semantics: An Impressionistic Introduction
3. Using Game Semantics to Decide Observational Equivalence
4. Homer: **H**igher-order **O**bservational-equivalence **M**odel check**ER**

Outline

1. Idealized Algol and Observational Equivalence
2. Game Semantics: An Impressionistic Introduction
3. Using Game Semantics to Decide Observational Equivalence
4. Homer: **H**igher-order **O**bservational-equivalence **M**odel check**ER**

A compact language that elegantly combines state-based procedural and higher-order functional programming, using a simple type-theoretic framework. IA is essentially a call-by-name variant of Core ML.

IA Types:

$$T ::= \mathsf{exp} \mid \mathsf{com} \mid \mathsf{var} \mid T \to T$$

$T ::=$	exp	numbers-valued expressions
$\mid$	com	commands
$\mid$	var	assignable variables
$\mid$	$T \to T$	function space

IA Terms:

- imperative constructs
- block-allocated local assignable variables
- PCF (= simply-typed λ-calculus + basic arithmetics + conditionals + fixpoint operators).

In this talk, we suppress higher-order features, though not completely. (E.g. Recursive 1st-order procedures are fixpoints of 2nd-order functionals.)

$\mathrm{ord}(o) := 0 \qquad \mathrm{ord}(A \to B) := \max(\mathrm{ord}(A) + 1, \mathrm{ord}(B))$

Idealized Algol (IA) [Reynolds 80]

A compact language that elegantly combines state-based procedural and higher-order functional programming, using a simple type-theoretic framework. IA is essentially a call-by-name variant of Core ML.

IA Types:

$$\begin{cases} T & ::= & \mathsf{exp} & \text{numbers-valued expressions} \\ & \mid & \mathsf{com} & \text{commands} \\ & \mid & \mathsf{var} & \text{assignable variables} \\ & \mid & T \to T & \text{function space} \end{cases}$$

IA Terms:

- imperative constructs
- block-allocated local assignable variables
- PCF (= simply-typed λ-calculus + basic arithmetics + conditionals + fixpoint operators).

In this talk, we suppress higher-order features, though not completely. (E.g. Recursive 1st-order procedures are fixpoints of 2nd-order functionals.)

$\mathrm{ord}(\circ) := 0 \qquad \mathrm{ord}(A \to B) := \max(\mathrm{ord}(A) + 1, \mathrm{ord}(B))$

Examples

```
x : exp |- new X in
               new Y in
                   X := x;
                   Y := 1;
                   while !X > 0 do
                   {
                         Y := !Y * !X;
                         X := !X - 1
                   };
                   !Y
```

Notation. Assignable variables ranged over by X, Y, etc.

```
|- fun f : (exp -> exp) -> exp .
             f (fun x : exp . f (fun y : exp . x))
```

$$\lambda f.f\,(\lambda x.f\,(\lambda y.x)) : ((\mathrm{exp} \to \mathrm{exp}) \to \mathrm{exp}) \to \mathrm{exp}$$

Examples

```
x :  exp |- new X in
            new Y in
                X := x;
                Y := 1;
                while !X > 0 do
                {
                        Y := !Y * !X;
                        X := !X - 1
                };
                !Y
```

Notation. Assignable variables ranged over by X, Y, etc.

```
|- fun f :  (exp -> exp) -> exp .
            f (fun x :  exp .  f (fun y :  exp .  x))
```

$$\lambda f.f\,(\lambda x.f\,(\lambda y.x)) : ((\mathrm{exp} \to \mathrm{exp}) \to \mathrm{exp}) \to \mathrm{exp}$$

Observational (or Contextual) Equivalence

[Milner 1975, Plotkin 1977, ... Full Abstraction Problem for PCF]
Intuitively $M \approx N$ means

> *"M and N are mutually substitutable in every program context without causing any difference in the computational outcome".*

Definition $M \approx N$ just if for every context $C[\,]$ such that $C[M]$ and $C[N]$ are programs (i.e. closed terms of base type), for every value v

$$C[M] \Downarrow v \quad \Longleftrightarrow \quad C[N] \Downarrow v.$$

- Quantification over all program contexts $C[-]$ ensures that potential side effects of M and N are taken fully into account.
- $\approx$ is an intuitively compelling notion of program equivalence, but very hard to reason about.
- An appropriate notion of equivalence for regression verification, for maintaining backwards compatibility of code. (Cf. Strichman's lecture)

Observational (or Contextual) Equivalence

[Milner 1975, Plotkin 1977, ... Full Abstraction Problem for PCF]

Intuitively $M \approx N$ means

> *"M and N are mutually substitutable in every program context without causing any difference in the computational outcome".*

Definition $M \approx N$ just if for every context $C[\,]$ such that $C[M]$ and $C[N]$ are programs (i.e. closed terms of base type), for every value v

$$C[M] \Downarrow v \quad \Longleftrightarrow \quad C[N] \Downarrow v.$$

- Quantification over all program contexts $C[\text{-}]$ ensures that potential side effects of M and N are taken fully into account.
- $\approx$ is an intuitively compelling notion of program equivalence, but very hard to reason about.
- An appropriate notion of equivalence for regression verification, for maintaining backwards compatibility of code. (Cf. Strichman's lecture)

Example 1: In Algol-like languages, state changes are irreversible.

I.e. "Snap-back", a construct

$$\text{Snapback} \quad : \quad \text{com} \rightarrow \text{com}$$

that runs its command-argument and then immediately undoes all the state-changes caused by the command, is not definable in IA.

Non-definability of snap-back is equivalent to:

$$\begin{array}{ll} & p : \text{com} \longrightarrow \text{com} \\ \vdash & \text{new}\, X := 0 \,\text{in}\, \{p\ (X := 1)\, \mathbf{;}\, \text{if}\ \mathbf{!}X = 1\ \text{then}\ \mathbf{\Omega}\ \text{else}\,\text{skip}\} \\ \approx & p\ \mathbf{\Omega} \end{array}$$

The theory of observational equivalence is rich

Example 2: Parametricity

Terms that have the "same underlying algorithm" are observationally equivalent.

$$
\begin{array}{ll}
 & p : \mathsf{com} \to \mathsf{bool} \to \mathsf{com} \\
\vdash & \mathsf{new}\, X := 1 \,\mathsf{in}\, \{p\,(X := -!X)\,(!X > 0)\} \\
\approx & \mathsf{new}\, Y := \mathsf{t} \,\mathsf{in}\, \{p\,(Y := \neg !Y)\,(!Y)\}
\end{array}
$$

IA is Turing powerful: observational equivalence is not decidable.

Questions

1. For which fragment of IA is observational equivalence decidable?
2. Classify these fragments.

Game semantics helps to answer these questions.

The theory of observational equivalence is rich

Example 2: Parametricity

Terms that have the "same underlying algorithm" are observationally equivalent.

$$
\begin{array}{ll}
 & p : \mathsf{com} \rightarrow \mathsf{bool} \rightarrow \mathsf{com} \\
\vdash & \mathsf{new}\, X := 1 \,\mathsf{in}\, \{p\,(X := -!X)\,(!X > 0)\} \\
\approx & \mathsf{new}\, Y := \mathsf{t} \,\mathsf{in}\, \{p\,(Y := \neg !Y)\,(!Y)\}
\end{array}
$$

IA is Turing powerful: observational equivalence is not decidable.

Questions

1. For which fragment of IA is observational equivalence decidable?
2. Classify these fragments.

Game semantics helps to answer these questions.

Outline

1. Idealized Algol and Observational Equivalence
2. Game Semantics: An Impressionistic Introduction
3. Using Game Semantics to Decide Observational Equivalence
4. Homer: **H**igher-order **O**bservational-equivalence **M**odel check**ER**

Game Interpretation of Types and Programs: Some Generalities

Types of a programming language are interpreted as (2-person) games.

Player	Point of View	
P (Proponent)	System	Term being modelled
O (Opponent)	Environment	Program context

Programs are interpreted as strategies for playing these games.

Game semantics is inherently a semantics of open systems; the meaning of a program is given by its potential interactions with the environment.

Compositionality: The key operation is plugging two strategies together, so that each actualizes part of the environment of the other.

$$\frac{\sigma : A \longrightarrow B \qquad \tau : B \longrightarrow C}{\sigma ; \tau : A \longrightarrow C}$$

This exploits the P/O duality: σ's P-move at B become an O-move of τ (and vice versa).

Game Interpretation of Types and Programs: Some Generalities

Types of a programming language are interpreted as (2-person) games.

Player	Point of View	
P (Proponent)	System	Term being modelled
O (Opponent)	Environment	Program context

Programs are interpreted as strategies for playing these games.

Game semantics is inherently a semantics of open systems; the meaning of a program is given by its potential interactions with the environment.

Compositionality: The key operation is plugging two strategies together, so that each actualizes part of the environment of the other.

$$\frac{\sigma : A \longrightarrow B \qquad \tau : B \longrightarrow C}{\sigma;\tau : A \longrightarrow C}$$

This exploits the P/O duality: σ's P-move at B become an O-move of τ (and vice versa).

Innocent Game or HON Game: Some Intuitions

[Hyland+O. (1995) Info & Comp 2000; Nickau 96. Precursors: Lorenz and Lorenzen, Conway, Joyal, Blass, Berry + Curien.]

Play as dialogue between O and P

Four types of move: P-questions, O-answers, O-questions, P-answers.

A play is an O/P-alternating sequence of moves, satisfying:

Rules of Civil Conversation

Justification:

- A question is asked only if the dialogue warrants it at that point.
- An answer is proferred only if a question expecting it is pending.

Well-Bracketing: "Last asked first answered."

Outcome of play: A dialogue ends when the opening question is answered. (We don't care about winning: just play to the bitter end!)

[Hyland+O. (1995) Info & Comp 2000; Nickau 96. Precursors: Lorenz and Lorenzen, Conway, Joyal, Blass, Berry + Curien.]

Play as dialogue between O and P

Four types of move: P-questions, O-answers, O-questions, P-answers.

A play is an O/P-alternating sequence of moves, satisfying:

Rules of Civil Conversation

Justification:

- A question is asked only if the dialogue warrants it at that point.
- An answer is proferred only if a question expecting it is pending.

Well-Bracketing: "Last asked first answered."

Outcome of play: A dialogue ends when the opening question is answered. (We don't care about winning: just play to the bitter end!)

Take $\vdash M : A$.

- Type A is interpreted as (a 2-player game called) arena $[\![A]\!]$.
- Term M is interpreted as a P-strategy $[\![M]\!]$ for playing in arena $[\![A]\!]$.

An arena is a forest (the nodes are the moves; edge relation is called enabling); each move has a label from $\{ PQ, PA, OQ, OA \}$.

Example. The arena $[\![\text{exp}]\!]$

OQ

PA

$[\![2 : \text{exp}]\!]$ is the P-strategy:

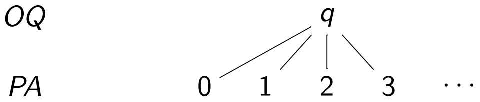
OQ	q
PA	2

Take $\vdash M : A$.

- Type A is interpreted as (a 2-player game called) arena $[\![A]\!]$.
- Term M is interpreted as a P-strategy $[\![M]\!]$ for playing in arena $[\![A]\!]$.

An arena is a forest (the nodes are the moves; edge relation is called enabling); each move has a label from $\{ PQ, PA, OQ, OA \}$.

Example. The arena $[\![\mathrm{exp}]\!]$



$[\![2 : \mathrm{exp}]\!]$ is the P-strategy:

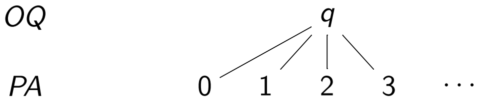
OQ	q
PA	2

Take $\vdash M : A$.

- Type A is interpreted as (a 2-player game called) arena $[\![A]\!]$.
- Term M is interpreted as a P-strategy $[\![M]\!]$ for playing in arena $[\![A]\!]$.

An arena is a forest (the nodes are the moves; edge relation is called enabling); each move has a label from $\{ PQ, PA, OQ, OA \}$.

Example. The arena $[\![\mathrm{exp}]\!]$



$[\![2 : \mathrm{exp}]\!]$ is the P-strategy:

OQ	q
PA	2

Write "if B then M else N" in prefix form:

$$\text{if } B\ M\ N : \text{exp}$$

- if : exp → exp → exp → exp is interpreted as a P-strategy
- program context $[\,]\ B\ M\ N$ determines an O-strategy

for playing in the arena $[\![\text{exp} \to \text{exp} \to \text{exp} \to \text{exp}]\!]$

	B		M		N		
	exp	→	exp	→	exp	→	exp
OQ							q
PQ	q						
OA	t						
PQ			q				
OA			3				
PA							3

(assuming O-strategy given by $[\,]\ \text{t}\ 3\ 4$).

Write "if B then M else N" in prefix form:

$$\text{if } B\ M\ N : \text{exp}$$

- if : exp → exp → exp → exp is interpreted as a P-strategy
- program context $[\,]\ B\ M\ N$ determines an O-strategy

for playing in the arena $[\![\, \text{exp} \rightarrow \text{exp} \rightarrow \text{exp} \rightarrow \text{exp} \,]\!]$

	B		M		N		
	exp	→	exp	→	exp	→	exp
OQ							q
PQ	q						
OA	t						
PQ			q				
OA			3				
PA							3

(assuming O-strategy given by $[\,]\ \text{t}\ 3\ 4$).

Write "if B then M else N : exp" in prefix-form:

$$\text{if } B\ M\ N : \text{exp}$$

- if : exp $\rightarrow$ exp $\rightarrow$ exp $\rightarrow$ exp is interpreted as a P-strategy
- program context $[\,]\ B\ M\ N$ determines an O-strategy

for playing in the arena $[\![\, \text{exp} \rightarrow \text{exp} \rightarrow \text{exp} \rightarrow \text{exp} \,]\!]$

	B exp	$\rightarrow$	M exp	$\rightarrow$	N exp	$\rightarrow$	exp
OQ							q
PQ	q						
OA	f						
PQ					q		
OA					4		
PA							4

(assuming O-strategy given by $[\,]\ \text{f}\ 3\ 4$).

The arena $[\![\mathsf{com}]\!]$

OQ	run
PA	$done$

$[\![\mathsf{skip} : \mathsf{com}]\!]$ is the P-strategy:

OQ	run
PA	$done$

Interpreting sequential composition: $\mathbf{;} : \mathsf{com} \to \mathsf{com} \to \mathsf{com}$

	com	→	com	→	com
OQ					run
PQ	run				
OA	$done$				
PQ			run		
OA			$done$		
PA					$done$

The arena $[\![\, \mathrm{com} \,]\!]$

OQ	*run*
PA	*done*

$[\![\, \mathrm{skip} : \mathrm{com} \,]\!]$ is the P-strategy:

OQ	*run*
PA	*done*

Interpreting sequential composition: **;** $: \mathrm{com} \rightarrow \mathrm{com} \rightarrow \mathrm{com}$

	com	$\rightarrow$	com	$\rightarrow$	com
OQ					*run*
PQ	*run*				
OA	*done*				
PQ			*run*		
OA			*done*		
PA					*done*

The arena $[\![\mathrm{com}]\!]$

$$\begin{array}{cc} OQ & run \\ & | \\ PA & done \end{array}$$

$[\![\mathrm{skip} : \mathrm{com}]\!]$ is the P-strategy:

$$\begin{array}{cc} OQ & run \\ PA & done \end{array}$$

Interpreting sequential composition: $\mathbf{;} : \mathrm{com} \to \mathrm{com} \to \mathrm{com}$

$$\begin{array}{ccccccc} & \mathrm{com} & \to & \mathrm{com} & \to & \mathrm{com} \\ OQ & & & & & run \\ PQ & run & & & & \\ OA & done & & & & \\ PQ & & & run & & \\ OA & & & done & & \\ PA & & & & & done \end{array}$$

Interpreting var

Following Reynolds, we view a variable type as given (in an object-oriented style) by a product of its read method and its write method. Thus

$$\mathsf{var} \quad := \quad \mathsf{exp} \times (\prod_{i \in \omega} \mathsf{com})$$

- read-part: first component is the value held at that location
- write-part: second component contains countably many commands, namely, to write 0 (respectively 1, 2, etc.) to that location.

Thus arena $[\![\, \mathsf{var} \,]\!]$ is the product arena $[\![\, \mathsf{exp} \,]\!] \times \prod_{i \in \omega} [\![\, \mathsf{com} \,]\!]$:

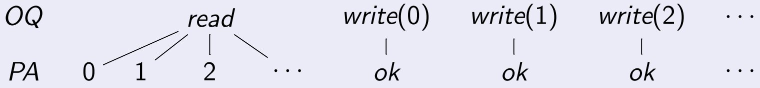
Interpreting var

Following Reynolds, we view a variable type as given (in an object-oriented style) by a product of its read method and its write method. Thus

$$\mathsf{var} \quad := \quad \mathsf{exp} \times (\prod_{i \in \omega} \mathsf{com})$$

- read-part: first component is the value held at that location
- write-part: second component contains countably many commands, namely, to write 0 (respectively 1, 2, etc.) to that location.

Thus arena $[\![\, \mathsf{var} \,]\!]$ is the product arena $[\![\, \mathsf{exp} \,]\!] \times \prod_{i \in \omega} [\![\, \mathsf{com} \,]\!]$:



Write $X := M$ as **assign** $X\ M$. Thus

- **assign** : var → exp → com is interpreted as a P-strategy
- context $[\,]\ X\ M$ determines an O-strategy

for playing in the arena $[\![$ var → exp → com $]\!]$.

	X		M		
	var	→	exp	→	com
OQ					run
PQ			q		
OA			5		
PQ	$write(5)$				
OA	ok				
PA					$done$

(assuming O-strategy is given by context $[\,]\ X\ 5$)

Interpreting assignment: $X := M$

Write $X := M$ as **assign** $X\ M$. Thus

- **assign** : var $\rightarrow$ exp $\rightarrow$ com is interpreted as a P-strategy
- context $[\,]\,X\ M$ determines an O-strategy

for playing in the arena $[\![$ var $\rightarrow$ exp $\rightarrow$ com $]\!]$.

	X var	$\rightarrow$	M exp	$\rightarrow$	com
OQ					run
PQ			q		
OA			5		
PQ	$write(5)$				
OA	ok				
PA					$done$

(assuming O-strategy is given by context $[\,]\,X\ 5$)

Interpreting block-allocated local variables: new

We decompose the formation

$$\frac{\Gamma, x : \mathsf{var} \vdash M : \mathsf{com}}{\Gamma \vdash \mathsf{new}\, x := n\, \mathsf{in}\, M : \mathsf{com}}$$

into two constructions:

1. Currying: $\Gamma \vdash \boldsymbol{\lambda} x : \mathsf{var}.M : \mathsf{var} \to \mathsf{com}$
2. Application by a constant: $\mathbf{new}_n : (\mathsf{var} \to \mathsf{com}) \to \mathsf{com}$.

Thus we have

$$\mathsf{new}\, x := n\, \mathsf{in}\, M \;:=\; \mathbf{new}_n\, (\boldsymbol{\lambda} x : \mathsf{var}.M)$$

Accordingly $[\![\, \mathsf{new}\, x := n\, \mathsf{in}\, M \,]\!]$ is the composite

$$[\![\, \Gamma \,]\!] \xrightarrow{[\![\, \Gamma \vdash \boldsymbol{\lambda} x:\mathsf{var}.M \,]\!]} (\mathsf{var} \to \mathsf{com}) \xrightarrow{\mathbf{new}_n} \mathsf{com}$$

Question. What is the strategy $\mathbf{new}_n$?

We decompose the formation

$$\frac{\Gamma, x : \mathsf{var} \vdash M : \mathsf{com}}{\Gamma \vdash \mathsf{new}\, x := n \,\mathsf{in}\, M : \mathsf{com}}$$

into two constructions:

1. Currying: $\Gamma \vdash \boldsymbol{\lambda} x : \mathsf{var}.M : \mathsf{var} \rightarrow \mathsf{com}$
2. Application by a constant: $\mathbf{new}_n : (\mathsf{var} \rightarrow \mathsf{com}) \rightarrow \mathsf{com}$.

Thus we have

$$\mathsf{new}\, x := n \,\mathsf{in}\, M \;\; := \;\; \mathbf{new}_n\, (\boldsymbol{\lambda} x : \mathsf{var}.M)$$

Accordingly $[\![\, \mathsf{new}\, x := n \,\mathsf{in}\, M \,]\!]$ is the composite

$$[\![\, \Gamma \,]\!] \xrightarrow{[\![\, \Gamma \vdash \boldsymbol{\lambda} x:\mathsf{var}.M \,]\!]} (\mathsf{var} \rightarrow \mathsf{com}) \xrightarrow{\mathbf{new}_n} \mathsf{com}$$

Question. What is the strategy $\mathbf{new}_n$?

The plays in $\mathbf{new}_n$ should correspond to the behaviour of a *prima facie* variable (initialized to n). Namely, they should satisfy:

Good Variable Property

Whenever the variable is read, it yields the value last written to it.

Thus, the (maximal) plays are defined to be words matching the regular expression:

$$q \cdot q^{\langle 1 \rangle} \cdot (read \cdot n)^* \cdot \left(\sum_{i \geq 0} write(i) \cdot ok \cdot (read \cdot i)^* \right)^* \cdot done^{\langle 1 \rangle} \cdot done$$

The (infinite) alphabet is the move-set of $(\mathsf{var} \to \mathsf{com}^{\langle 1 \rangle}) \to \mathsf{com}$ (subject to the labelling convention to distinguish copies of the same subarena).

Good Variable Behaviour: An Example Play in $\mathbf{new}_n$

	(var	$\rightarrow$	$\mathsf{com}^{\langle 1\rangle}$)	$\rightarrow$	com
OQ					run
PQ			$run^{\langle 1\rangle}$		
OQ	$read$				
PA	n				
OQ	$write(5)$				
PA	ok				
OQ	$read$				
PA	9				
OA			$done^{\langle 1\rangle}$		
PA					$done$

Outline

1. Idealized Algol and Observational Equivalence
2. Game Semantics: An Impressionistic Introduction
3. Using Game Semantics to Decide Observational Equivalence
4. Homer: **H**igher-order **O**bservational-equivalence **M**odel check**ER**

Finitary Idealized Algol IA_f: recursion-free, finite base types

Recall: $\mathrm{ord}(b) := 0 \quad \mathrm{ord}(T_1 \to T_2) := \max(\mathrm{ord}(T_1)+1, \mathrm{ord}(T_2))$

An IA_f-term $\quad x_1 : T_1, \cdots, x_n : T_n \vdash M : T \quad$ is an ***i*-th order term** just if $\mathrm{ord}(T_j) < i$ and $\mathrm{ord}(T) \le i$.

- IA_i: collection of *i*-th order IA_f-terms.
- IA_i+**while** is IA_i augmented by while-loops
- $\mathsf{IA}_i+\mathbf{Y}_j$ **(where $j < i$)** is IA_i augmented by

$$\frac{\Gamma, f : T \vdash M : T}{\Gamma \vdash \mu f^T.M : T}$$

where the premise is *i*-th order, and $\mathrm{ord}(T) \le j$.
I.e. $\mathsf{IA}_i+\mathbf{Y}_j$ consists of IA_i and recursively-defined terms of order at most j.

Theorem (Full Abstraction, Abramsky + McCusker 1997)

Observational equivalence of IA is characterized by complete plays (i.e. plays ending with a move that answers the opening question):

$$M \approx N \iff \mathbf{cplays}([\![\Gamma \vdash M : A]\!]) = \mathbf{cplays}([\![\Gamma \vdash N : A]\!])$$

At low types, game semantics admits a concrete representation.

Theorem (Ghica + McCusker 2000)

In IA_2+**while**:

1. $\mathbf{cplays}([\![\Gamma \vdash M : A]\!])$ *is regular. Further*
2. $\mathbf{cplays}([\![\Gamma \vdash M : A]\!])$*, given as a DFA (or regular expression), can be constructed by recursion over syntax.*

Hence $\approx$ *in* IA_2 *reduces to the problem of DFA-equivalence.*

Theorem (Full Abstraction, Abramsky + McCusker 1997)

Observational equivalence of IA is characterized by complete plays (i.e. plays ending with a move that answers the opening question):

$$M \approx N \iff \mathbf{cplays}(\llbracket \Gamma \vdash M : A \rrbracket) = \mathbf{cplays}(\llbracket \Gamma \vdash N : A \rrbracket)$$

At low types, game semantics admits a concrete representation.

Theorem (Ghica + McCusker 2000)

In IA_2+**while***:*

1. $\mathbf{cplays}(\llbracket \Gamma \vdash M : A \rrbracket)$ *is regular. Further*
2. $\mathbf{cplays}(\llbracket \Gamma \vdash M : A \rrbracket)$*, given as a DFA (or regular expression), can be constructed by recursion over syntax.*

Hence $\approx$ *in* IA_2 *reduces to the problem of DFA-equivalence.*

A Complete Classification of Decidable Fragments of Finitary IA

OBS EQUIV$_L$: Given β-nfs M and N in sublanguage L of IA, does $M \approx N$?

	pure	+**while**	$+\mathbf{Y}_0$	$+\mathbf{Y}_1$
IA_0	PTIME	–	–	–
IA_1	coNP	PSPACE	DPDA EQUIV	undecidable
IA_2	PSPACE	PSPACE	DPDA EQUIV	undecidable
IA_3	EXPTIME	EXPTIME	DPDA EQUIV	undecidable
$\mathsf{IA}_i, i \geq 4$	undecidable	undecidable	undecidable	undecidable

Undecidability results: O. LICS'02 and Murawski LICS'03.
coNP + PSPACE results: Murawski TCS 2005.
EXPTIME results: O. LICS'02; Murawski + Walukiewicz FOSSACS'05.
DPDA EQUIV results: Murawski, Walukiewicz + O. ICALP'05.

Visibly pushdown automata (Alur+Madhusudan, STOC'04)

- Stack action is determined by input alphabet read:
  $\Sigma = \Sigma_{\mathrm{push}} + \Sigma_{\mathrm{pop}} + \Sigma_{\mathrm{noop}}$
- Excellent closure properties (almost as good as regular languages)

VPA-languages: Closed under complementation and intersection (*cf.* DPDA).

$$L(A) \subseteq L(B) \iff L(A) \cap \overline{L(B)} = \varnothing$$

is EXPTIME-complete, and in PTIME if B deterministic.

Theorem (Murawski+Walukiewicz 2005)

*The complete plays of (IA_3+**while**)-terms are VPA-recognizable.*

EXPTIME-hardness: by reducing the EXPTIME-complete problem FINITE TREE AUTOMATA EQUIVALENCE (Seidl 1990) to it.

[Murawski, O. + Walukiewicz ICALP'05]

$\mathsf{IA}_i+\mathbf{Y}_0$: Only terms of base type can call themselves recursively. This includes all tail-recursive functions (i.e. iterations) and:

Example. Non tail-recursive ground recursion:

$$c : \mathsf{com}, b : \mathsf{bool} \vdash \mu p^{\mathsf{com}}.\mathsf{if}\ b\ \mathsf{then}\ (p\ ;\ c\ ;\ p)\ \mathsf{else}\ \mathsf{skip} : \mathsf{com}$$

DPDA-Equivalenc hardness

Theorem

There is a translation that maps a DPDA A to a $(\mathsf{IA}_1+\mathbf{Y}_0)$-term $x : \mathsf{exp} \vdash M_A : \mathsf{com}$ such that for any A, B, we have $L(A) = L(B)$ iff $M_A \approx M_B$.

Outline

1. Idealized Algol and Observational Equivalence
2. Game Semantics: An Impressionistic Introduction
3. Using Game Semantics to Decide Observational Equivalence
4. Homer: **H**igher-order **O**bservational-equivalence **M**odel check**ER**

HOMER: Higher-order Observational-equivalence Model checkER

[Hopkins + O. CAV 2009]

HOMER: a prototype tool implementing Murawski and Walukiewicz's algorithm.

HOMER maps IA_3+**while** terms to VPA representing the complete plays in their game semantics, then check for the equivalence of the VPA.
If the input term is at most 2nd-order (possibly with iteration), the VPA-compile is just a DFA.

Counterexample. If the terms are inequivalent, HOMER will produce both a game-semantic and an operational-semantic counterexample, in the form of a play and a separating context respectively.

Property checking. HOMER can also model check a term against a regular property or LTL formula.

HOMER is written in about 8 KLOC of F#, including about 600 LOC for the VPA toolkit. It is the first model checker for third-order programs.

HOMER: Higher-order Observational-equivalence Model checkER

[Hopkins + O. CAV 2009]

HOMER: a prototype tool implementing Murawski and Walukiewicz's algorithm.

HOMER maps IA_3+**while** terms to VPA representing the complete plays in their game semantics, then check for the equivalence of the VPA.
If the input term is at most 2nd-order (possibly with iteration), the VPA-compile is just a DFA.

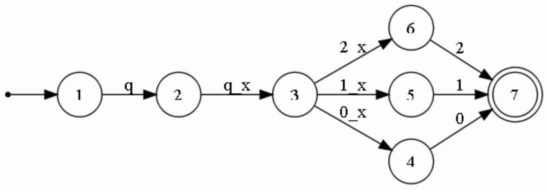
Counterexample. If the terms are inequivalent, HOMER will produce both a game-semantic and an operational-semantic counterexample, in the form of a play and a separating context respectively.

Property checking. HOMER can also model check a term against a regular property or LTL formula.

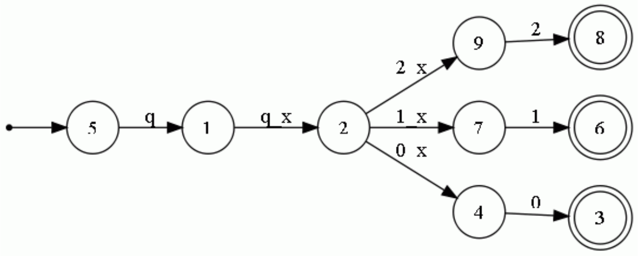
HOMER is written in about 8 KLOC of F#, including about 600 LOC for the VPA toolkit. It is the first model checker for third-order programs.

Example 1

```
x :  exp |- x
```

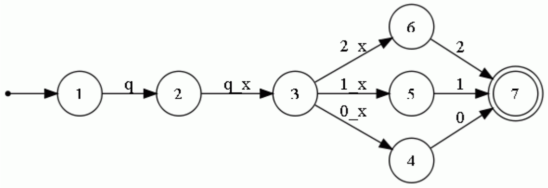



```
x :  exp |- new X in (X := x ; if !X = 0 then !X else !X)
```

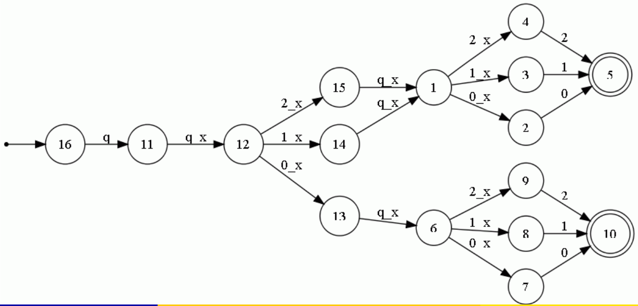


Example 1'

`x :  exp |- x`



`x :  exp |- if x = 0 then x else x`



Why sorting?

"... it seems impossible to use Model Checking to verify that a sorting algorithm is correct since sorting correctness is a data-oriented property involving several quantifications and data structures." [Bandera user manual]

Example: Equivalence of (respective implementations of) bubble sort and insertion sort.

- Program parameterized over array size (n) and basic data type ($\mathbb{Z}$ MOD 3).
- The DFA model is fully abstract. Only the actions of the non-local array are observable, and hence, represented.
- An array of size 20 (over integers MOD 3) has *circa* 3^{20} states (about 3.5 billion). Our model is highly abstract (though still accurate): it has only about 5500 states!

Conclusions and Further Directions

- Game semantics has clear operational content, while admitting compositional methods in the style of denotational semantics.
- The game-semantic approach to observational equivalence checking is fully automatic, sound and complete, and compositional.
- The model (given by complete plays) extracted is highly accurate, yet "compact".
- To extend to infinite data types, use abstraction refinement techniques (Bakewell + Ghica, TACAS08) or prove auxiliary data-independence results.

Further directions

1. Performance: Exploit abstraction refinement techniques (CEGAR), and acceleration technologies (SMT-solvers) to improve scalability.
2. Challenge of verifying highly structured programs (e.g. object orientation / functional features e.g. Javascript, Perl, etc).

Conclusions and Further Directions

- Game semantics has clear operational content, while admitting compositional methods in the style of denotational semantics.
- The game-semantic approach to observational equivalence checking is fully automatic, sound and complete, and compositional.
- The model (given by complete plays) extracted is highly accurate, yet "compact".
- To extend to infinite data types, use abstraction refinement techniques (Bakewell + Ghica, TACAS08) or prove auxiliary data-independence results.

Further directions

1. Performance: Exploit abstraction refinement techniques (CEGAR), and acceleration technologies (SMT-solvers) to improve scalability.
2. Challenge of verifying highly structured programs (e.g. object orientation / functional features e.g. Javascript, Perl, etc).