

Game Semantics and its Algorithmic Applications

Lecture 5: A Game-Semantic Approach to Software Model Checking

Luke Ong

Oxford University Computing Laboratory

`www.comlab.ox.ac.uk/oucl/work/luke.ong/`

Game semantics has emerged as a paradigm for giving semantics to a wide range of programming languages.

All of these models are highly accurate: **fully abstract**.

Algorithmic game semantics is concerned with the extraction of algorithms for program verification from (appropriate representations of) the game semantics of programs.

Promising features:

- 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.

Algorithmic Games Semantics: Theory and Practice

Practice: An alternative approach to Software Model Checking.

- Start from an **accurate denotational semantics** of the program.
- Derive an appropriate **model of computation sufficiently concrete (and tractable)** for the purpose of verification.

Advantages: **Soundness** and **completeness** inherited by the model; method remains **compositional** – it can treat **open terms**.

This talk: Foundational (complexity) results in Algorithmic Games Semantics.

Questions. For which fragments of Idealized Algol is observational equivalence decidable? What are their complexities?

Game Semantics can help! Case study: **higher-order procedural programs**

Survey recent applications to Software Model Checking.

Outline of the Talk

- I. Idealized Algol and Observational Equivalence
- II. Game Semantics: An Impressionistic Introduction
- III. Some Results in Algorithmic Game Semantics: A Survey
- IV. Complete Classification of Decidable Fragments of IA
- V. Application to Software Model Checking: A Prototype Tool

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

$$\begin{array}{l} b ::= \mathbf{exp} \text{ (or } \mathbf{nat}) \text{ } \text{numeric expressions} \\ \quad | \mathbf{com} \text{ } \text{commands} \\ \quad | \mathbf{var} \text{ } \text{assignable variables (or locations)} \\ T ::= b \mid T \rightarrow T \end{array}$$

IA Terms

Simply-typed λ -calculus + basic arithmetic + conditional (definition-by-cases) + fixpoint + **imperative constructs** + **block-allocated local variables**.

Imperative constructs of IA

1. Null command. **skip** : **com**

2. Command sequencing.
$$\left\{ \begin{array}{l} \text{seq} : \text{com} \rightarrow \text{com} \rightarrow \text{com} \\ \text{seq} : \text{com} \rightarrow \text{exp} \rightarrow \text{exp} \end{array} \right.$$

(Expressions may have side-effects.) Write **seq** $C C'$ as $C ; C'$.

3. Assignment. **assign** : **var** \rightarrow **exp** \rightarrow **com**

Write **assign** MN as $M := N$.

4. Dereferencing (explicit in the syntax). **deref** : **var** \rightarrow **exp**

Write **deref** M as $!M$.

5. Block-allocated local (assignable) variables: $n \geq 0$, $b = \text{com}$ or **exp**

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

Examples

1. $x := 1 ; x := !x + 1$ becomes

$\text{seq}(\text{assign } x \ 1) (\text{assign } x \ (\text{deref } x + 1))$

2. We can express the while-loop $\text{while } B \text{ do } C$ as

$\mathbf{Y}(\lambda x : \text{com. if } B \text{ then } (C ; x) \text{ else skip})$

Selected rules defining evaluation relation $M, S \Downarrow V, S'$

$$\frac{M, S \Downarrow \mathbf{skip}, S' \quad N, S' \Downarrow V, S''}{M ; N, S \Downarrow V, S''} \quad V = n \text{ or } \mathbf{skip}$$

$$\frac{M, S \Downarrow l, S'}{\mathbf{!}M, S \Downarrow n, S'} \quad l \in \text{dom}(S') \wedge S'(l) = n$$

$$\frac{M, S \Downarrow \lambda x.P, S' \quad P[N/x], S' \Downarrow V, S''}{MN, S \Downarrow V, S''} \quad \frac{M(\mathbf{Y}(M)), S \Downarrow V, S'}{\mathbf{Y}(M), S \Downarrow V, S'}$$

$$\frac{M[l/x], S[l \mapsto n] \Downarrow V, S'[l \mapsto m]}{\mathbf{new } x := n \text{ in } M, S \Downarrow V, S'} \quad l \notin \text{dom}(S) \cup \text{dom}(S').$$

Intuitively $M \approx N$ means “ M may be replaced by N (and vice versa) in **every** program context with no observable difference in the resultant computational outcome”.

Formally $M \approx N$ iff for every context $C[]$ such that $C[M]$ and $C[N]$ are programs (i.e. **closed** terms of type **com**)

$$C[M], \emptyset \Downarrow \mathbf{skip}, \emptyset \iff C[N], \emptyset \Downarrow \mathbf{skip}, \emptyset.$$

- 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**.

The Theory of Observational Equivalence is Rich: Some Examples

1. **State changes are irreversible**: There is no “**snap back**” construct

Snapback : **com** \rightarrow **com**

which runs its argument and then undoes all its state-changes.

$p : \mathbf{com} \rightarrow \mathbf{com}$

$\vdash \mathbf{new } x := 0 \mathbf{ in } \{p(x := 1) ; \mathbf{if } !x = 1 \mathbf{ then } \Omega \mathbf{ else skip}\} \approx p(\Omega)$

2. **Parametricity**: Terms that have the “same underlying algorithm” are obs. eq.

$p : \mathbf{com} \rightarrow \mathbf{bool} \rightarrow \mathbf{com}$

$\vdash \mathbf{new } x := 1 \mathbf{ in } \{p(x := \neg !x) (!x > 0)\}$

$\approx \mathbf{new } y := \mathbf{t} \mathbf{ in } \{p(y := \neg y) (!y)\}$

OBS EQUIV_L: Given M and N in sublanguage L of IA, does $M \approx N$?

Note: M and N are **open** terms.

Question. For what fragment L of IA is OBS EQUIV_L decidable?

We use game semantics to answer the question.

Outline of the Talk

- I. Idealized Algol and Observational Equivalence
- II. **Game Semantics of Idealized Algol**
- III. Some Results in Algorithmic Game Semantics: A Survey
- IV. Complete Classification of Decidable Fragments of IA
- V. Application to Software Model Checking: A Prototype Tool

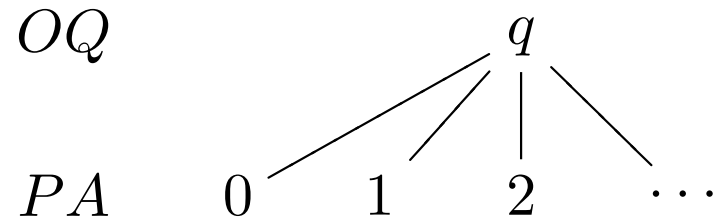
Game Semantics FAQs

1. How does game semantics treat **open** terms (equivalently, support compositional reasoning)?
2. Why is there **no mention of winning strategies**?

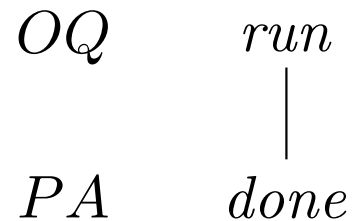
Example arenas

(We display arenas as “upside down” forests.)

Natural number arena, **nat**:



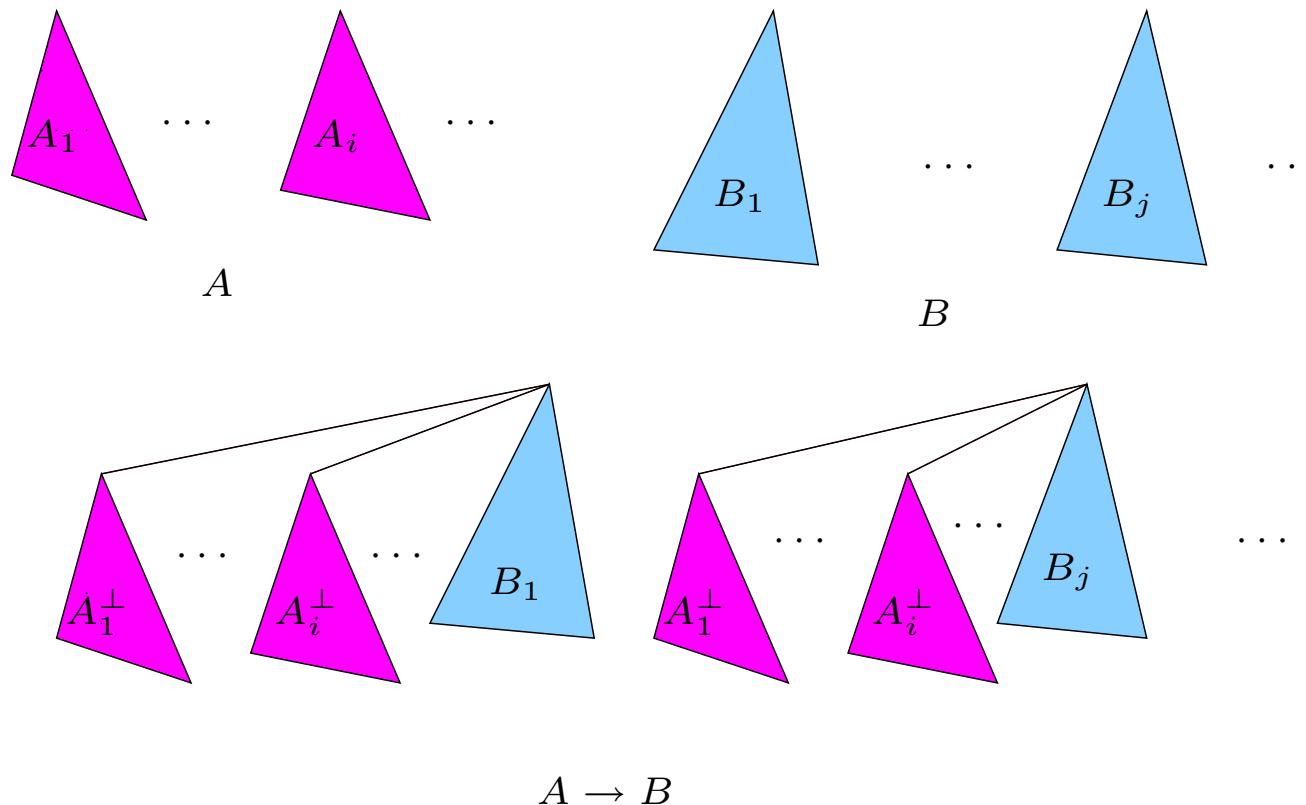
Command arena, **com**:



Arena constructions

Product arena $A \times B$: Disjoint union of (the underlying forest of) A and B .

Function space arena $A \rightarrow B$: “First invert the P and O moves of A , then graft one such copy of A just below every root of B .”



What are the plays?

A **justified sequence** (over an arena A) is a sequence of P/O-**alternating** moves such that each move m , except the first, has a **pointer** to an earlier m' such that $m' \vdash_A m$.

The **P-view** $\ulcorner s \urcorner$ of a justified sequence s is a subsequence consisting only of moves which P considers relevant (for determining his response). (Similarly for O-view.)

A **play** (over A) is a justified sequence satisfying:

- (1) **Visibility**: Each P-move points to some move that appears in the P-view at that point; similarly for O.
- (2) **Well-Bracketing**: Each answer points to the last **pending** question.

A category of arenas and strategies

Formally a **strategy** σ (over A) is a non-empty prefix-closed set of plays satisfying: for any s, a and b

- (i) (**P-Determinacy**). If $s a, s b \in \sigma$ and a is a P-move, then $a = b$.
- (ii) If $s \in \sigma$ and $s a$ is a play with a an O-move, then $s a \in \sigma$.

Thus strategy is a subtree of the game tree (of plays) that is single-branching (deterministic) whenever P is to play.

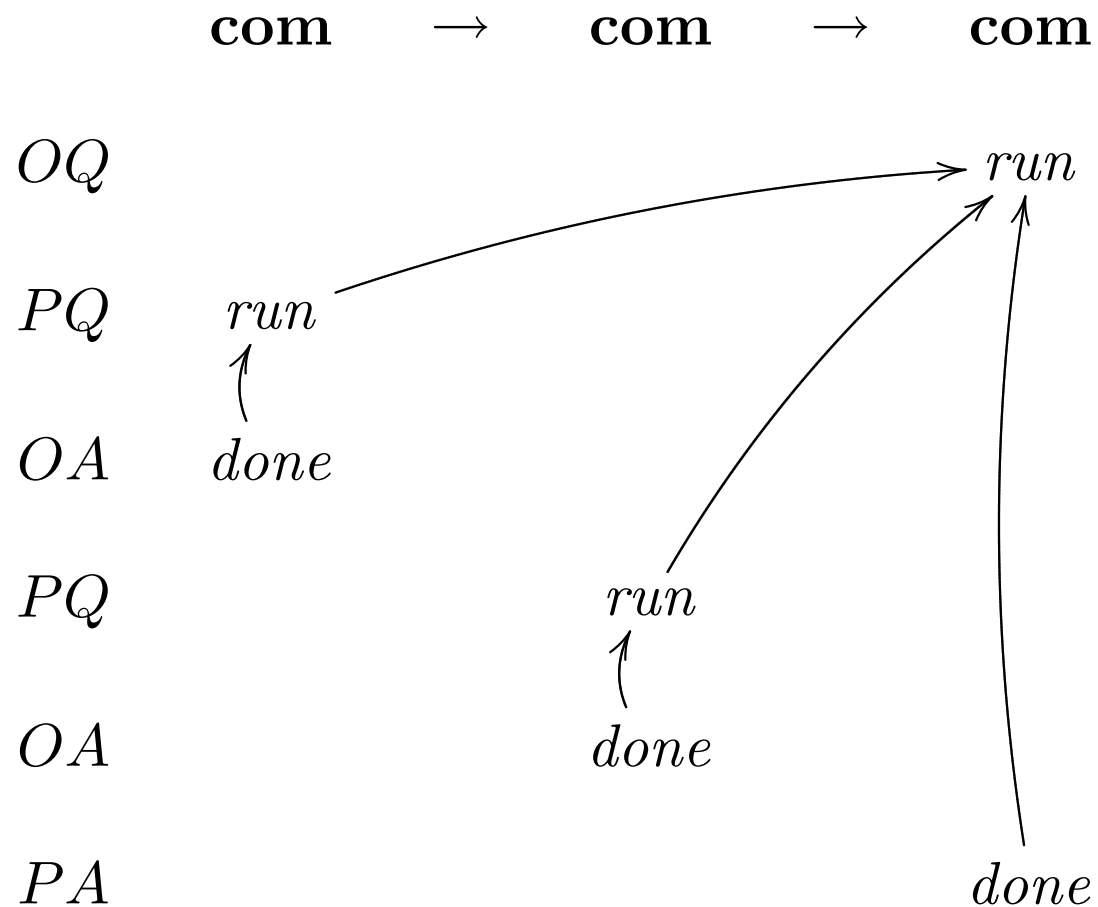
Theorem. (Abramsky-McCusker 1997) The category \mathbb{G} whose

- objects are **arenas**
- maps $A \longrightarrow B$ are **strategies** over function space arena $A \rightarrow B$

is cartesian closed, and gives rise to a **fully abstract** model of Idealized Algol.

Interpreting IA in \mathbb{G} : Arenas and Strategies

Interpreting sequential composition. $;$: $\text{com} \rightarrow \text{com} \rightarrow \text{com}$ is modelled by the prefix-closure of:



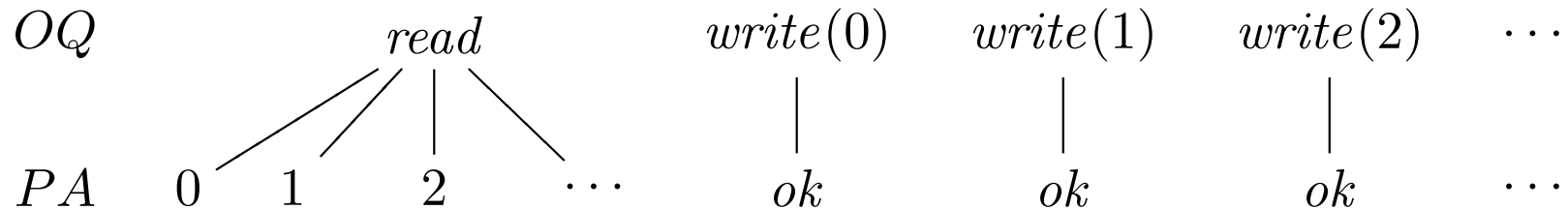
Interpreting var

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

$$\mathbf{var} = \mathbf{nat} \times \left(\prod_{i \in \omega} \mathbf{com} \right)$$

- first component is value held at that location
- second component contains countably many commands to write 0, 1, 2, etc. respectively to that location.

Thus arena **var** is the product arena $\mathbf{nat} \times \left(\prod_{i \in \omega} \mathbf{com} \right)$:



The strategy $\mathbf{new}_n : (\mathbf{var} \rightarrow \mathbf{com}) \rightarrow \mathbf{com}$

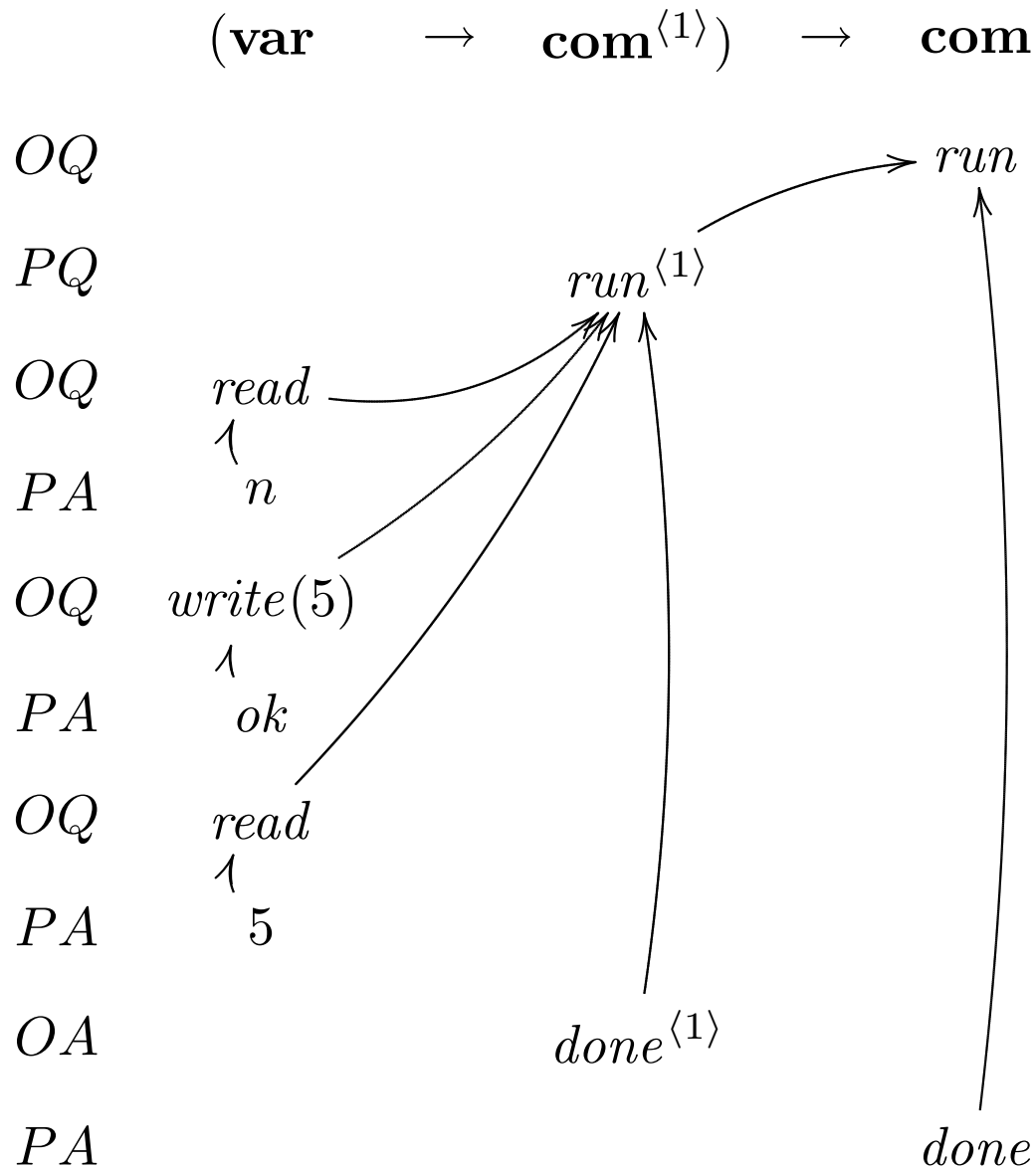
The plays in \mathbf{new}_n should correspond to the behaviour of a *prima facie* variable (or location) initialized to n , namely, **whenever the variable is read, it should yield the value last written to.**

The maximal plays are words matching the regular expression:

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

- The alphabet is the move-set of $(\mathbf{var} \rightarrow \mathbf{com}^{\langle 1 \rangle}) \rightarrow \mathbf{com}$ (subject to the labelling convention to distinguish copies of the same subarena).
- Pointers can be safely omitted since they are uniquely reconstructible for plays of up to order-2.

Equivalently \mathbf{new}_n is the prefix-closure of complete plays of the form:



Outline of the Talk

- I. Idealized Algol and Observational Equivalence
- II. Game Semantics of Idealized Algol
- III. **Some Results in Algorithmic Game Semantics: A Survey**
- IV. Complete Classification of Decidable Fragments of IA
- V. Application to Software Model Checking: A Prototype Tool

Finitary Idealized Algol IA_f

IA_f = Recursion-free IA over Finite Ground Types

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

- **IA_i** : collection of i -th order IA_f -terms.

- **$\text{IA}_i + \text{while}$** is IA_i augmented by

$$\frac{\Gamma \vdash M : \mathbf{bool} \quad \Gamma \vdash N : \mathbf{com}}{\Gamma \vdash \mathbf{while} M \mathbf{do} N : \mathbf{com}}$$

- **$\text{IA}_i + \mathbf{Y}_j$ (where $j < i$)** is IA_i augmented by

$$\frac{\Gamma, x : T \vdash M : T}{\Gamma \vdash \mathbf{Y}(\lambda x^T. M) : T}$$

where terms in the rule are i -th order, and $\text{ord}(T) \leq j$.

First steps in Algorithmic Game Semantics (Ghica-McCusker'00)

At low types, game semantics admits a concrete, algorithmic representation.

Theorem. (Ghica + McCusker 2000). For IA_2 -terms M and N

$$M \approx N \iff \underbrace{\llbracket \Gamma \vdash M : A \rrbracket^{\text{reg}}}_{R} \equiv \underbrace{\llbracket \Gamma \vdash N : A \rrbracket^{\text{reg}}}_{S}$$

Moreover $R \equiv S$, equivalence of regular expressions, is decidable.

(ICALP'00 Best Paper Award)

An important semantic characterization of observational equivalence.

Theorem. (Abramsky + McCusker 1997) Observational equivalence of IA is characterized by **complete plays**. I.e.

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

Representing (game semantics of) IA_2 -terms by regular expressions

Note. Up to order 2, justification pointers can be uniquely reconstructed from the underlying sequence of moves, and so, may be ignored!

Assign to each type a finite alphabet of moves of the corresponding arena.

Plays are just words over the alphabet of moves — pointers can be ignored at 2nd-order!

Lemma. (Ghica+McCusker 2000) The set of complete plays in (fully abstract) game semantics $\llbracket \Gamma \vdash M : A \rrbracket$ is regular.

Decidability and undecidability results

Two orthogonal directions of extension:

Fragments of IA	Is observational equivalence decidable?
IA ₂	Yes. (Ghica+McCusker ICALP'00)

Can Ghica-McCusker's results be extended to larger fragments?

Decidability and undecidability results

Two orthogonal directions of extension:

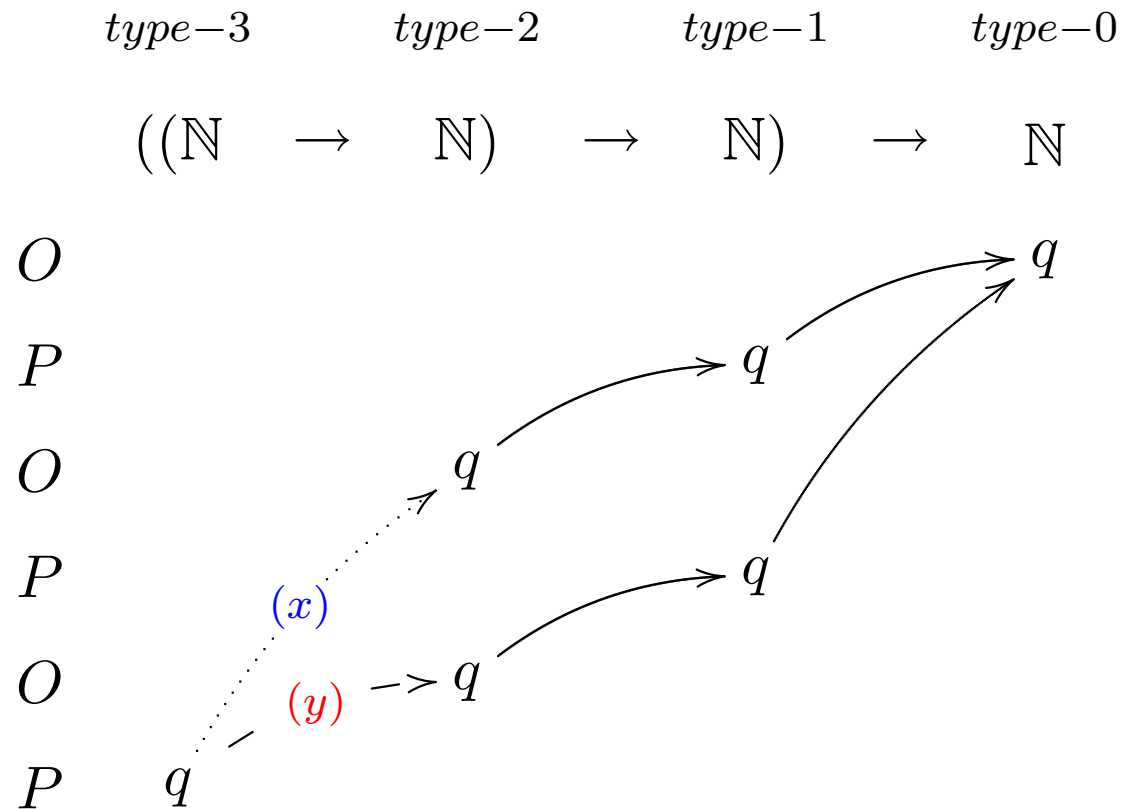
Fragments of IA	Is observational equivalence decidable?
IA_2	Yes. (Ghica+McCusker ICALP'00)
$IA_2 + \mathbf{while}$	Yes (Ghica+McCusker ICALP'00)
$IA_2 + \mathbf{Y}_1$	No. (Ong LICS'02)

What about higher-type fragments of IA?

Are pointer really necessary? (Yes, at 3rd-order or higher.)

Kierstead terms

$$\begin{cases} (y) & \lambda f.f(\lambda x.f(\lambda y.y)) \\ (x) & \lambda f.f(\lambda x.f(\lambda y.x)) \end{cases}$$



The two justified sequences have the same underlying move-sequence!

Decidability and undecidability results

Two orthogonal directions of extension:

Fragments of IA	Is observational equivalence decidable?
IA_2	Yes. (Ghica+McCusker ICALP'00)
$IA_2 + \mathbf{while}$	Yes (Ghica+McCusker ICALP'00)
$IA_2 + \mathbf{Y}_1$	No. (Ong LICS'02)
IA_3	Yes: Reduction to DPDA Equivalence. (Ong LICS'02)
$IA_i, i \geq 4$	No. (Murawski LICS'03)

Decidability and undecidability results

Two orthogonal directions of extension:

Fragments of IA	Is observational equivalence decidable?
IA_2	Yes. (Ghica+McCusker ICALP'00)
$IA_2 + \mathbf{while}$	Yes (Ghica+McCusker ICALP'00)
$IA_2 + \mathbf{Y}_1$	No. (Ong LICS'02)
IA_3	Yes: Reduction to DPDA Equivalence. (Ong LICS'02)
$IA_i, i \geq 4$	No. (Murawski LICS'03)
$IA_3 + \mathbf{while}$?

Decidability and undecidability results

Two orthogonal directions of extension:

Fragments of IA	Is observational equivalence decidable?
IA_2	Yes. (Ghica+McCusker ICALP'00)
$IA_2 + \mathbf{while}$	Yes (Ghica+McCusker ICALP'00)
$IA_2 + \mathbf{Y}_1$	No. (Ong LICS'02)
IA_3	Yes: Reduction to DPDA Equivalence. (Ong LICS'02)
$IA_i, \quad i \geq 4$	No. (Murawski LICS'03)
$IA_3 + \mathbf{while}$	Yes. (Murawski + Walukiewicz FOSSACS'05) (Ong MFPS'04)

What about $IA_i + \mathbf{Y}_0$, for $i = 1, 2, 3$? Complexity?

Outline of the Talk

- I. Idealized Algol and Observational Equivalence
- II. Game Semantics of Idealized Algol
- III. Some Results in Algorithmic Game Semantics: A Survey
- IV. Complete Classification of Decidable Fragments of IA
- V. Applications to Software Model Checking: A Prototype Tool

A Complete Classification of Decidable Fragments of IA

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

	pure	+while	+Y ₀	+Y ₁
IA ₀	P TIME	–	–	–
IA ₁	C ONP	P SPACE	DPDA EQUIV	undecidable
IA ₂	P SPACE	P SPACE	DPDA EQUIV	undecidable
IA ₃	E XPTIME	E XPTIME	DPDA EQUIV	undecidable
IA _{<i>i</i>} , $i \geq 4$	undecidable	undecidable	undecidable	undecidable

Undecidability results: Ong LICS'02 and Murawski LICS'03.

C**ONP** + P**SPACE** results: Murawski TCS 2005(?).

E**XPTIME** results: Murawski + Walukiewicz FOSSACS'05.

DPDA EQUIV results: Murawski, Walukiewicz + Ong (submitted).

Deciding \approx for IA_1 is coNP-complete (Murawski 2005)

IA_1 -term: $x_1 : b_1, \dots, x_n : b_n \vdash M : b$ with b_i, b ground

A coNP-procedure for deciding \approx .

Given M and N :

- Construct a DFA recognising $comp(\llbracket M \rrbracket)$ of size linear in M . Similarly for N .
- Guess a play witnessing inequivalence of IA_1 -terms M and N , and verify it (in polytime).

Deciding \approx for IA_1 is coNP-complete (cont'd)

coNP-Hardness. Recall: TAUTOLOGY is coNP-complete.

Boolean formulas: $B ::= X_i \mid B \vee B \mid B \wedge B \mid \neg B$

Translation of Boolean formulas $B(X_1, \dots, X_k)$ to IA_1 -terms M_B :

$$\begin{aligned} M_X &= \mathbf{if} \ !X \ \mathbf{t} \ \mathbf{f} & M_{\neg B} &= \mathbf{if} \ M_B \ \mathbf{f} \ \mathbf{t} \\ M_{B_1 \vee B_2} &= \mathbf{if} \ M_{B_1} \ \mathbf{t} \ M_{B_2} & M_{B_1 \wedge B_2} &= \mathbf{if} \ M_{B_1} \ M_{B_2} \ \mathbf{f} \end{aligned}$$

Theorem. $B(X_1, \dots, X_k)$ tautology iff following are obs. equivalent:

- $x : \mathbf{bool} \vdash \mathbf{new} \ X_1, \dots, X_k \ \mathbf{in} \ (X_1 := x ; \dots ; X_k := x ; M_B) : \mathbf{bool}$
- $x : \mathbf{bool} \vdash \mathbf{new} \ X_1, \dots, X_k \ \mathbf{in} \ (X_1 := x ; \dots ; X_k := x ; \mathbf{t}) : \mathbf{bool}$

P asks for the value of x exactly k times; O's answers (k of them) correspond to a valuation of the Boolean variables X_1, \dots, X_k .

Deciding \approx for IA_3+while is EXPTIME-complete (M.+W.'05)

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

$$A = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle \quad \Sigma = \Sigma_{\text{push}} + \Sigma_{\text{pop}} + \Sigma_{\text{noop}}$$

$$\delta \subseteq (Q \times \Sigma_{\text{push}} \times \Gamma \times Q) \cup (Q \times \Sigma_{\text{pop}} \times \Gamma \times Q) \cup (Q \times \Sigma_{\text{noop}} \times Q)$$

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

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

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

Theorem. (Murawski+Walukiewicz 05) Complete plays of $(\text{IA}_3+\text{while})$ -terms are VPA-recognizable. (FOSSACS05 Best Paper Award)

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

Deciding \approx for $\mathbf{IA}_i + \mathbf{Y}_0$ (for $i = 1, 2, 3$) is equivalent to DPDA EQUIV

(Murawski, O. + Walukiewicz ICALP 2005.)

$\mathbf{IA}_i + \mathbf{Y}_0$: Only terms of **ground type** can call themselves recursively.

Example. Non tail-recursive ground recursion:

$$\underbrace{c : \mathbf{com}, b : \mathbf{bool} \vdash \mathbf{Y}(\lambda p^{\mathbf{com}}. \mathbf{if } b (p ; c ; p) \mathbf{skip}) : \mathbf{com}}_{\Xi}$$

We construct a DPDA accepting $\llbracket \Xi \rrbracket$ (“*q-semantics*”) given by

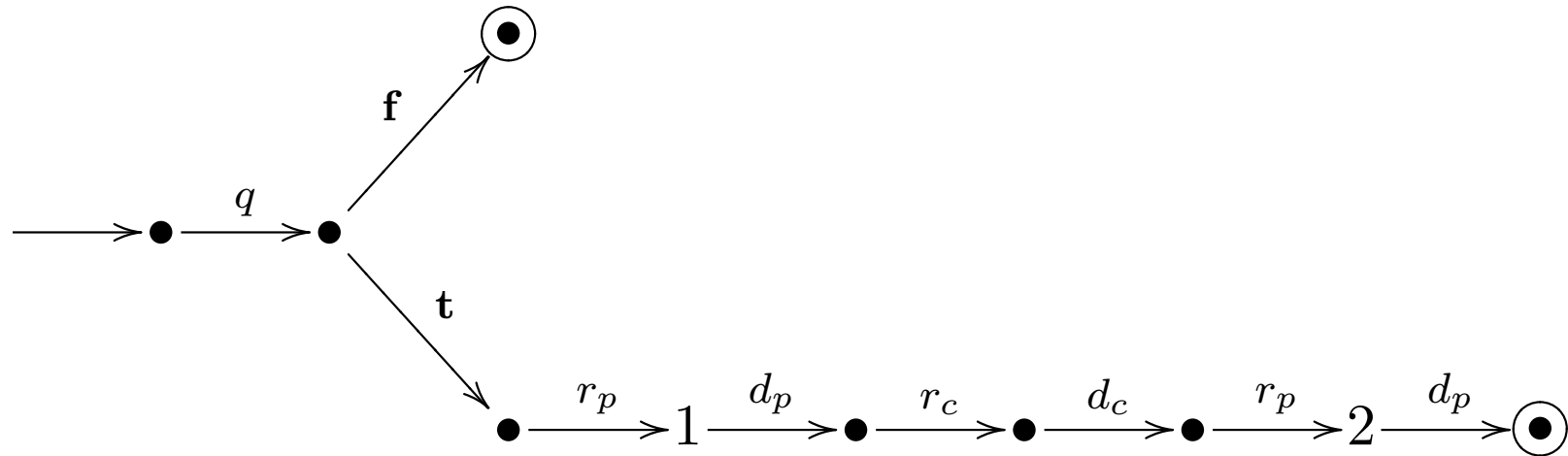
$$\mathit{comp}(\llbracket \Xi \rrbracket) = \mathit{run} \cdot \llbracket \Xi \rrbracket \cdot \mathit{done}$$

from the *q-semantics* of

$$c : \mathbf{com}, b : \mathbf{bool}, p : \mathbf{com} \vdash \mathbf{if } b (p ; c ; p) \mathbf{skip} : \mathbf{com}$$

in stages.

(i) $(c : \mathbf{com}, b : \mathbf{bool}, p : \mathbf{com} \vdash \mathbf{if } b (p ; c ; p) \mathbf{skip} : \mathbf{com})$



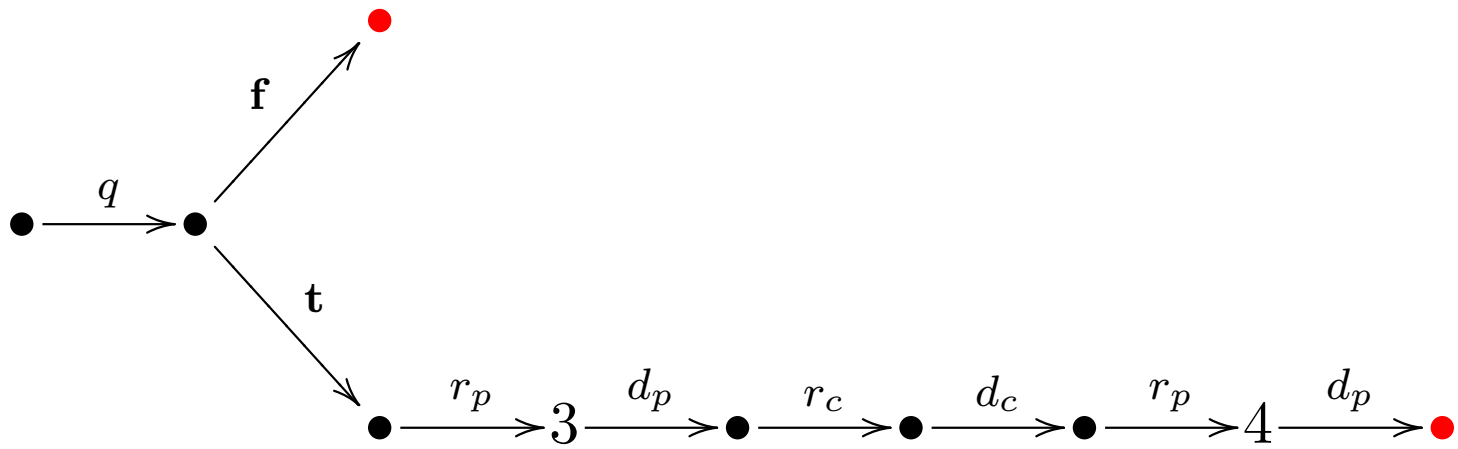
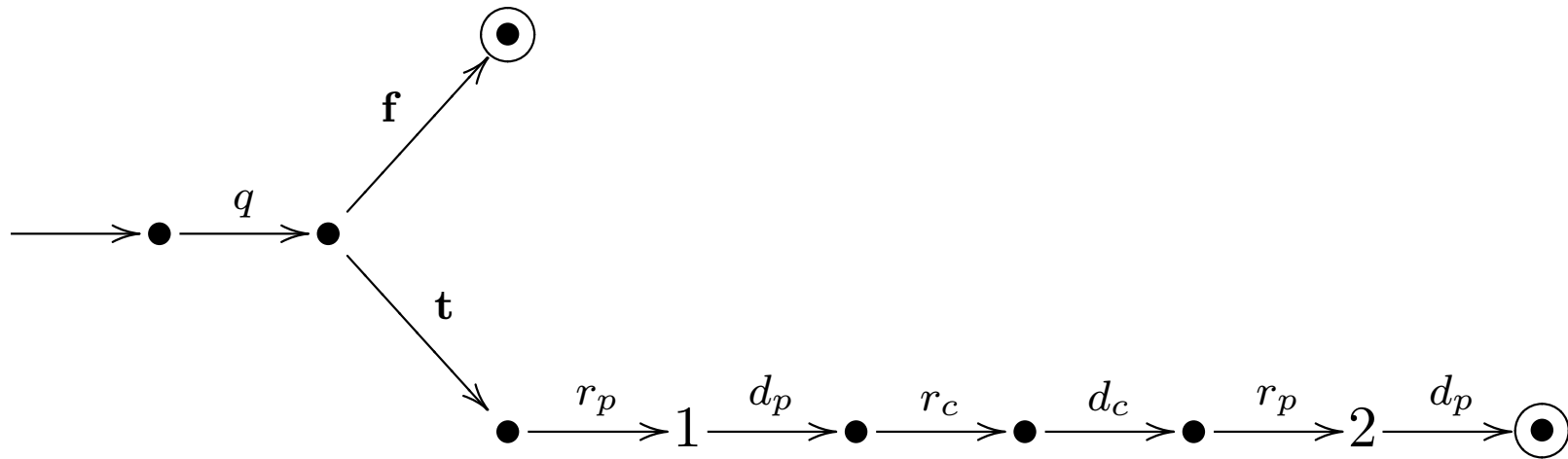
Set $G = \mathbf{com} \rightarrow \mathbf{bool} \rightarrow \mathbf{com} \rightarrow \mathbf{com}$.

Input symbols are $\{ r_c, d_c, q, \mathbf{t}, \mathbf{f}, r_p, d_p \}$.

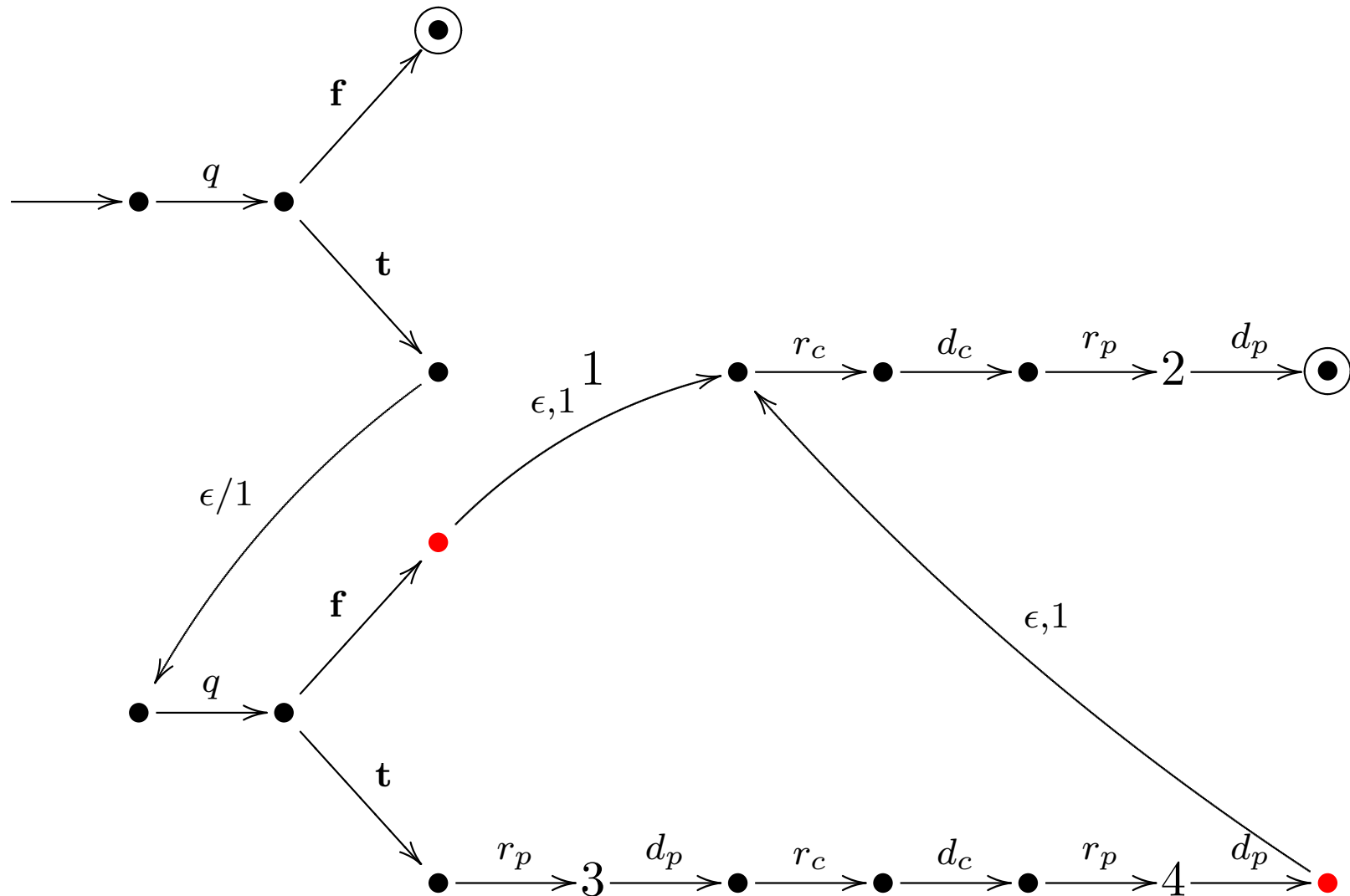
States are partitioned into P-states and O-states.

Note the distinguished O-states 1 and 2.

(ii) Make a duplicate for processing recursive calls

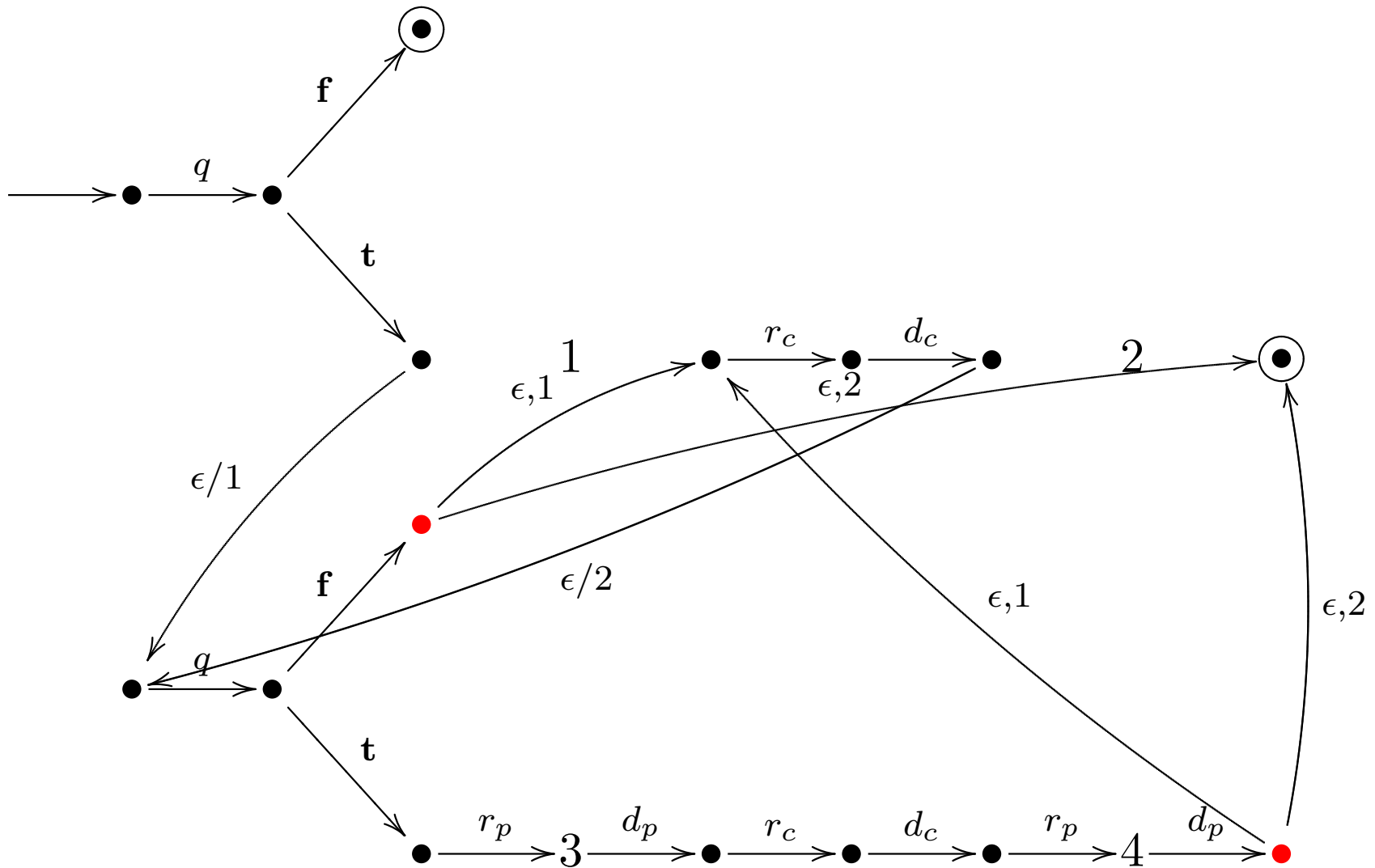


(iii) Connecting the two copies: getting rid of r_p, d_p or 1, 2, 3 and 4

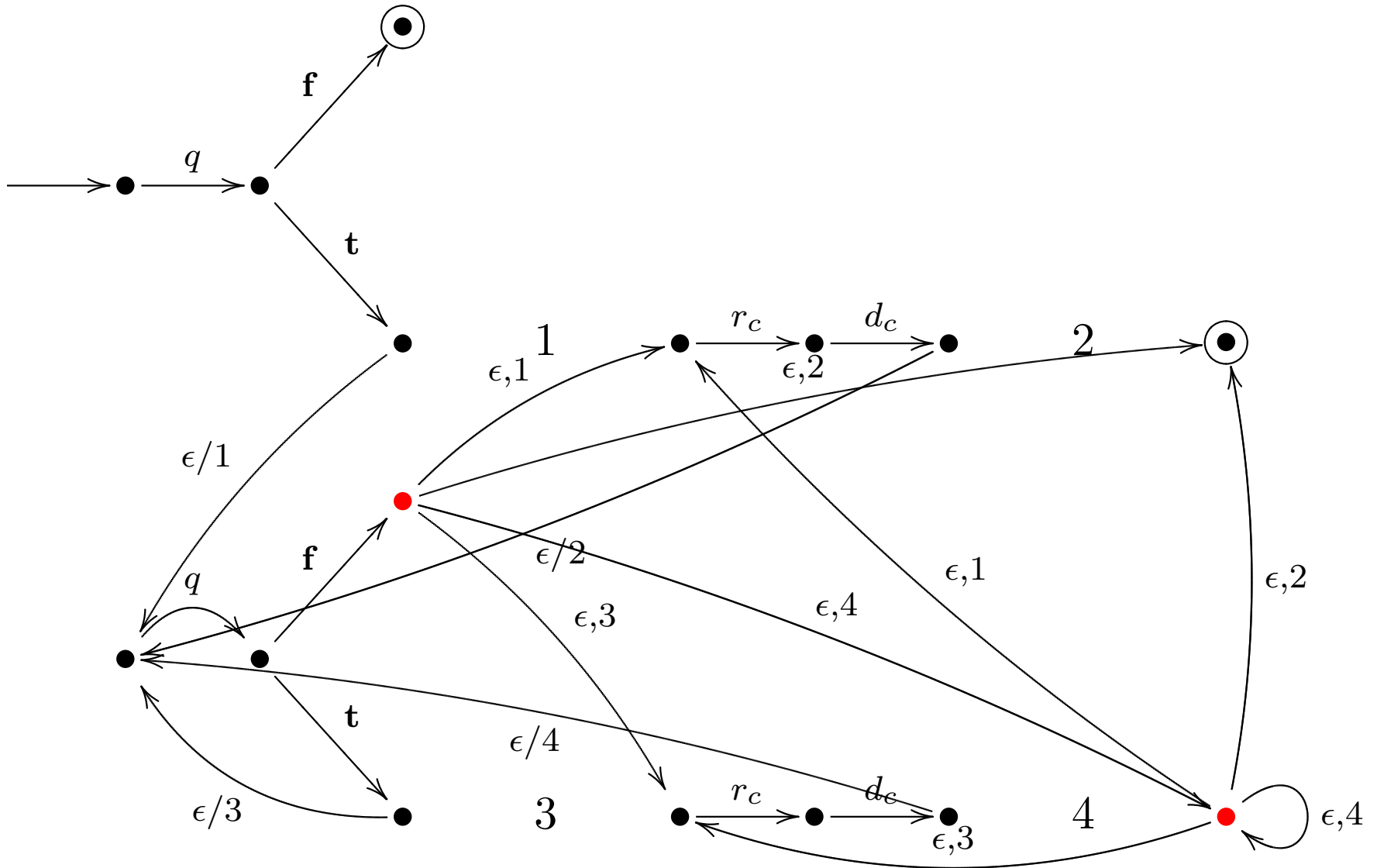


$\epsilon/1$ = “read no input symbol and push 1”; $\epsilon, 1$ = “read no input symbol and pop 1”.

(iii) Connecting the two copies: getting rid of more r_p, d_p



$(c : \text{com}, b : \text{bool} \vdash \mathbf{Y}(\lambda p^{\text{com}}. \text{if } b (p ; c ; p) \text{ skip}) : \text{com})$



Construction of the complete-play DPDA_M for $M \in \mathbf{IA}_i + \mathbf{Y}_0$: 3 stages

- (1) The underlying move-sequences of complete plays of **simple** $(\mathbf{IA}_i + \mathbf{Y}_0)$ -terms are DPDA-definable.

Simple terms: 3rd-order variables may occur at most once.

E.g. $\lambda f.f(\lambda x.f(\lambda x.y))$ not simple.

- (2) Complete plays of simple terms given by (1), augmented by words that encode pointers of 3rd-order questions, remain DPDA-recognizable.

Lemma. In the play of a simple term, any third-order question is justified by the last-occurring pending enabler of the question.

- (3) Identifying 3rd-order questions induces a renaming operation on DPDAs as given by (2) that identifies the corresponding input symbols.

This operation preserves determinacy of DPDAs.

DPDA-Equiv Hardness of deciding \approx for $\mathbf{IA}_i + \mathbf{Y}_0$

DPDA. $A = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0 \rangle$ where

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$$

Assume one-symbol pushes. Acceptance by empty stack.

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

Identify values of \mathbf{exp} with Σ . Consider arena $G = \mathbf{exp} \rightarrow \mathbf{com}$, so that $M_G = \{r, d\} \cup \Sigma \cup \{q\}$. Given $L \subseteq \Sigma^*$ define $\widehat{L} \subseteq M_G^*$ by

$$\widehat{L} = \{r q a_1 q a_2 \cdots q a_n d \mid a_1 \cdots a_n \in L\}$$

Note: $\widehat{L}_1 = \widehat{L}_2 \iff L_1 = L_2$.

The $(IA_1 + Y_0)$ -translate M_A of a DPDA A

$x : \mathbf{exp} \vdash$ **new** $Q := q_0, TOP := Z_0, CH$ **in**
 $\mu z^{\mathbf{com}}$. **new** $POP := 0, X := !TOP$ **in**
while (not $!POP$) **do**
 (if $\delta(!Q, \epsilon, !X) = (q', \alpha)$ **then**
 $(Q := q';$
 if $\alpha = \epsilon$ **then** $POP := 1$ **else** $((TOP := \alpha_1); z))$
 else
 $(CH := x;$
 if $\delta(!Q, !CH, !X) = (q', \alpha)$ **then**
 $(Q := q';$
 if $\alpha = \epsilon$ **then** $POP := 1$ **else** $((TOP := \alpha_1); z))$
 else $\Omega_{\mathbf{com}})) : \mathbf{com}$

Outline of the Talk

- I. Idealized Algol and Observational Equivalence
- II. Game Semantics of Idealized Algol
- III. Some Results in Algorithmic Game Semantics: A Survey
- IV. Complete Classification of Decidable Fragments of IA
- V. [Application to Software Model Checking: A Prototype Tool](#)

A new approach to Software Model Checking

Though we emphasize observational equivalence, **the same algorithmic representations of program meanings** can be used to verify a wide range of program properties $\Xi \models \phi$ where Ξ is a term-in-context, provided ϕ is a **regular property**.

E.g. take $\Xi = p : \text{nat} \rightarrow \text{nat}, x : \text{var} \vdash M : \text{com}$, and

$\phi =$ “in M , whenever p is called, its argument is read from x and its result is (immediately) written into x ”

can be captured by the regular expression: for appropriate move sets X, Y and Z

$$(X^* (q^1 \text{read}^3 \sum_{d \in D} (d^3 d^1) Y^* \sum_{d \in D} (d^3 \text{write}(d)^3) \text{ok}^3 Z)^*)^*$$

Fact. Definability by regular expressions is equivalent to definability in (Q)LTL. Thus can model check IA-terms against such properties.

Thus we obtain for free a model checker for a temporal style of program correctness assertions verifiable by checking for **emptiness of the intersection of the program automaton and the complement of the property automaton.**

Work in progress:

- Combine these ideas with the standard methods of **over-approximation** and **data-abstraction**
- Investigate applications in **inter-procedural dataflow analysis** and **reachability analysis.**

Goal: Build on the tools and methods of the verification community, while exploring the advantages offered by our semantics-based approach.

Prototype tool: A compiler from 2nd-order IA to DFA (D. Ghica^a)

- Parser + type inference + back-end processing in CAML
- (Most) back-end regular language processing: AT&T FSM Library
- Output: AT&T GraphViz and dot packages

A case study: Bubble sort

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]

Why bubble sort?

Not for any algorithmic virtues, but because it is a straightforward non-recursive sorting algorithm.

^aResearch Fellow, EPSRC project *Algorithmic Game Semantics and its Applications*.

Case study: Bubble Sort

Program parameterized over array size (n) and basic data type ($\mathbb{Z} \text{ MOD } 3$).

The DFA model is fully abstract. Only the actions of the non-local array are observable, and hence, represented.

Some performance data: (SunBlade 100, 2GB RAM)

array size n	model construction time
2	5 mins
5	10 mins
10	15 mins
20	4 hours
25	10 hours

An array of size 20 (over integers MOD 3) has *circa* 3^{20} states (about 3.5 billion). Our model is *highly abstract*: it has only about 5500 states!

Related work: Lazic has an IA-to-CSP compiler for FDR checking.

Game Semantics FAQs

1. How does game semantics treat **open** term?

A term is interpreted as a **P-strategy**, which is a rule telling P how to respond to **all** possible O-actions.

E.g. The strategy $\llbracket x : \text{nat} \vdash M : \text{nat} \rrbracket$ gives a response to every O-answer that correspond to a possible instantiation of x .

2. Why is there **no mention of winning strategies**?

We have use game semantics to construct a fully abstract model: Every point in the space is denoted by some IA term.

Notions of winning correspond properties of strategies

A possible notion is **totality**: P always has the last say!

References

- [MOW05] A. Murawski, C.-H. L. Ong, and I. Walukiewicz. Idealized Algol with ground recursion, and DPDA Equivalence. Preprint, submitted, 2005.
- [Mur03] A. Murawski. On program equivalence in languages with ground-type references. In *Proceedings of 18th IEEE Annual Symposium on Logic in Computer Science (LICS'03)*, pages 108–117. IEEE Computer Society Press, 2003.
- [Mur04] A. Murawski. Games for complexity of second-order call-by-name programs. To appear in *Theoretical Computer Science*, 2004.
- [MW05] A. Murawski and I. Walukiewicz. Third-order Idealized Algol with iteration is decidable. In *Proc. FOSSACS*, pages 202–218. Springer, 2005. LNCS Vol. 3441.
- [Ong02] C.-H. L. Ong. Observational equivalence of third-order Idealized

Algol is decidable. In *Proceedings of IEEE Symposium on Logic in Computer Science, 22-25 July 2002, Copenhagen Denmark*, pages 245–256. Computer Society Press, 2002.