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.
**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
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**

\[
b ::= \text{exp} \quad \text{(or nat)} \\ | \quad \text{com} \\ | \quad \text{var} \\
\]

\[
T ::= b \quad | \quad T \rightarrow T 
\]

**IA Terms**

Simply-typed \(\lambda\)-calculus + basic arithmetic + conditional (definition-by-cases) + fixpoint + imperative constructs + block-allocated local variables.
1. Null command. \texttt{skip} : \texttt{com}

2. Command sequencing.
\begin{align*}
\text{seq} & : \texttt{com} \rightarrow \texttt{com} \rightarrow \texttt{com} \\
\text{seq} & : \texttt{com} \rightarrow \texttt{exp} \rightarrow \texttt{exp}
\end{align*}

(Expressions may have side-effects.) Write \texttt{seq} \( C \ C' \) as \( C ; C' \).

3. Assignment. \texttt{assign} : \texttt{var} \rightarrow \texttt{exp} \rightarrow \texttt{com}

Write \texttt{assign} \( MN \) as \( M := N \).

4. Dereferencing (explicit in the syntax). \texttt{deref} : \texttt{var} \rightarrow \texttt{exp}

Write \texttt{deref} \( M \) as \( !M \).

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

\begin{align*}
\Gamma, x : \texttt{var} \vdash M : b \\
\Gamma \vdash \texttt{new} x := n \texttt{ in } M : b
\end{align*}
Examples

1. \( x := 1 ; x := !x + 1 \) becomes

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

2. We can express the while-loop \textbf{while} \( B \) \textbf{do} \( C \) as

\[
Y(\lambda x : \text{com.} \text{if } B \text{ then } (C ; x) \text{ else skip})
\]
Selected rules defining evaluation relation $M, S \downarrow V, S'$

\[
\frac{M, S \downarrow \text{skip}, S'}{M ; N, S \downarrow V, S''}
\]
\[
V = n \text{ or skip}
\]

\[
\frac{M, S \downarrow l, S'}{!M, S \downarrow n, S'}
\]
\[
l \in \text{dom}(S') \land S'(l) = n
\]

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

\[
\frac{M[l/x], S[l \rightarrow n] \downarrow V, S'[l \rightarrow m]}{\text{new } x := n \text{ in } M, S \downarrow V, S'}
\]
\[
l \notin \text{dom}(S) \cup \text{dom}(S').
\]
Observational Equivalence \( M \approx N \)

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 \texttt{com})

\[
C[M], \emptyset \downarrow \text{skip}, \emptyset \iff C[N], \emptyset \downarrow \text{skip}, \emptyset.
\]

- 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 → com

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

   \[ p : \text{com} \to \text{com} \]

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

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

   \[ p : \text{com} \to \text{bool} \to \text{com} \]

   \[ \vdash \text{new } x := 1 \text{ in } \{ p(x := \neg !x) \ ; \text{if } !x > 0 \} \]

   \[ \approx \text{new } y := t \text{ in } \{ p(y := \neg y) \ ; \text{if } !y \} \]
**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 \( \text{OBS EQUIV}_L \) 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
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:

\[
\begin{array}{c}
OQ \\
P A \quad 0 \quad 1 \quad 2 \quad \cdots
\end{array}
\]

Command arena, com:

\[
\begin{array}{c}
OQ \quad run \\
P A \quad done
\end{array}
\]
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** $\langle s \rangle$ 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 $\sigma$ (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 $G$ whose

- objects are arenas
- maps $A \rightarrow B$ 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 $G$: Arenas and Strategies

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

$$\text{com} \rightarrow \text{com} \rightarrow \text{com}$$

```
OQ
PQ run
OA done
PQ run
OA done
PA done
```
Interpreting \texttt{var}

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

\[
\texttt{var} = \texttt{nat} \times (\prod_{i \in \omega} \texttt{com})
\]

- 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 \texttt{var} is the product arena \texttt{nat} \times (\prod_{i \in \omega} \texttt{com}):

\[
\begin{array}{ccccccc}
OQ & \quad \text{read} \quad & \quad \text{write}(0) \quad & \quad \text{write}(1) \quad & \quad \text{write}(2) \quad & \cdots \\
PA & 0 & 1 & 2 & \cdots & ok & ok & ok & \cdots \\
\end{array}
\]
Interpreting assignment. Recall: \( \text{assign} \ x \ M \) is just \( x := M \)

The strategy (interpreting) \( \text{assign} : \text{var} \to \text{nat} \to \text{com} \) is the prefix-closure of (complete) plays of the form (with \( n \) ranging over \( \mathbb{N} \))

\[
\begin{array}{ccc}
\text{var} & \to & \text{nat} & \to & \text{com} \\
OQ & \rightarrow & run \\
PQ & \rightarrow & read \\
OA & \rightarrow & n \\
PQ & \rightarrow & write(n) \\
OA & \rightarrow & ok \\
PA & \rightarrow & done
\end{array}
\]
The strategy \( \text{new}_n : (\text{var} \to \text{com}) \to \text{com} \)

The plays in \( \text{new}_n \) should correspond to the behaviour of a \textit{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:

\[
\text{run} \cdot \text{run}^{(1)} \cdot (\text{read} \cdot n)^* \cdot \left( \sum_{i \geq 0} \text{write}(i) \cdot \text{ok} \cdot (\text{read} \cdot i)^* \right)^* \cdot \text{done}^{(1)} \cdot \text{done}
\]

- The alphabet is the move-set of \((\text{var} \to \text{com}^{(1)}) \to \text{com}\) (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 $\text{new}_n$ is the prefix-closure of complete plays of the form:

$$\text{(var} \quad \rightarrow \quad \text{com}^{(1)}) \quad \rightarrow \quad \text{com}$$
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
Finitary Idealized Algol \( \text{IA}_f \)

\( \text{IA}_f \) = Recursion-free IA over Finite Ground Types

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

- \( \text{IA}_i \): collection of \( i \)-th order \( \text{IA}_f \)-terms.
- \( \text{IA}_i + \text{while} \) is \( \text{IA}_i \) augmented by
  \[
  \Gamma \vdash M : \text{bool} \quad \Gamma \vdash N : \text{com} \\
  \Gamma \vdash \text{while } M \text{ do } N : \text{com}
  \]
- \( \text{IA}_i + Y_j \) (where \( j < i \)) is \( \text{IA}_i \) augmented by
  \[
  \Gamma, x : T \vdash M : T \\
  \Gamma \vdash Y(\lambda x^T.M) : T
  \]
  where terms in the rule are \( i \)-th order, and \( \text{ord}(T) \leq j \).
At low types, game semantics admits a concrete, algorithmic representation.

**Theorem.** (Ghica + McCusker 2000). For IA$_2$-terms $M$ and $N$

\[ M \approx N \iff \left[ \Gamma \vdash M : A \right]_{\text{reg}}^R \equiv \left[ \Gamma \vdash N : A \right]_{\text{reg}}^S \]

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.

\[ M \approx N \iff \text{comp}(\left[ \Gamma \vdash M : A \right]) = \text{comp}(\left[ \Gamma \vdash N : A \right]) \]
Representing (game semantics of) $\text{IA}_2$-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 $\left[ \Gamma \vdash M : A \right]$ is regular.
Decidability and undecidability results

Two orthogonal directions of extension:

<table>
<thead>
<tr>
<th>Fragments of IA</th>
<th>Is observational equivalence decidable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA_2</td>
<td>Yes. (Ghica+McCusker ICALP’00)</td>
</tr>
</tbody>
</table>

Can Ghica-McCusker’s results be extended to larger fragments?
Decidability and undecidability results

Two orthogonal directions of extension:

<table>
<thead>
<tr>
<th>Fragments of IA</th>
<th>Is observational equivalence decidable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{IA}_2$</td>
<td>Yes. (Ghica+McCusker ICALP’00)</td>
</tr>
<tr>
<td>$\text{IA}_2 + \text{while}$</td>
<td>Yes (Ghica+McCusker ICALP’00)</td>
</tr>
<tr>
<td>$\text{IA}_2 + \text{Y}_1$</td>
<td>No. (Ong LICS’02)</td>
</tr>
</tbody>
</table>

What about higher-type fragments of $\text{IA}$?
Are pointer really necessary? (Yes, at 3rd-order or higher.)

Kierstead terms

\[
\begin{align*}
\lambda f. f(\lambda x. f(\lambda y. y)) \\
\lambda f. f(\lambda x. f(\lambda y. x))
\end{align*}
\]

\( \text{type} - 3 \quad \text{type} - 2 \quad \text{type} - 1 \quad \text{type} - 0 \)

\[((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \to \mathbb{N} \]

The two justified sequences have the same underlying move-sequence!
Decidability and undecidability results

Two orthogonal directions of extension:

<table>
<thead>
<tr>
<th>Fragments of IA</th>
<th>Is observational equivalence decidable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA₂</td>
<td>Yes. (Ghica+McCusker ICALP’00)</td>
</tr>
<tr>
<td>IA₂ + while</td>
<td>Yes (Ghica+McCusker ICALP’00)</td>
</tr>
<tr>
<td>IA₂ + Y₁</td>
<td>No. (Ong LICS’02)</td>
</tr>
<tr>
<td>IA₃</td>
<td>Yes: Reduction to DPDA Equivalence. (Ong LICS’02)</td>
</tr>
<tr>
<td>IAᵢ, i ≥ 4</td>
<td>No. (Murawski LICS’03)</td>
</tr>
</tbody>
</table>
Decidability and undecidability results

Two orthogonal directions of extension:

<table>
<thead>
<tr>
<th>Fragments of IA</th>
<th>Is observational equivalence decidable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA$_2$</td>
<td>Yes. (Ghica+McCusker ICALP’00)</td>
</tr>
<tr>
<td>IA$_2$ + while</td>
<td>Yes (Ghica+McCusker ICALP’00)</td>
</tr>
<tr>
<td>IA$_2$ + Y$_1$</td>
<td>No. (Ong LICS’02)</td>
</tr>
<tr>
<td>IA$_3$</td>
<td>Yes: Reduction to DPDA Equivalence. (Ong LICS’02)</td>
</tr>
<tr>
<td>IA$_i$, $i \geq 4$</td>
<td>No. (Murawski LICS’03)</td>
</tr>
<tr>
<td>IA$_3$ + while</td>
<td>?</td>
</tr>
</tbody>
</table>
Decidability and undecidability results

Two orthogonal directions of extension:

<table>
<thead>
<tr>
<th>Fragments of IA</th>
<th>Is observational equivalence decidable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA₂</td>
<td>Yes. (Ghica+McCusker ICALP’00)</td>
</tr>
<tr>
<td>IA₂ + while</td>
<td>Yes (Ghica+McCusker ICALP’00)</td>
</tr>
<tr>
<td>IA₂ + Y₁</td>
<td>No. (Ong LICS’02)</td>
</tr>
<tr>
<td>IA₃</td>
<td>Yes: Reduction to DPDA Equivalence. (Ong LICS’02)</td>
</tr>
<tr>
<td>IAᵢ,  i ≥ 4</td>
<td>No. (Murawski LICS’03)</td>
</tr>
<tr>
<td>IA₃ + while</td>
<td>Yes. (Murawski + Walukiewicz FOSSACS’05) (Ong MFPS’04)</td>
</tr>
</tbody>
</table>

What about IAᵢ + Y₀, for i = 1, 2, 3? Complexity?
Example: DPDA defined by $\lambda f. f(\lambda x.x) : ((o \rightarrow o) \rightarrow o) \rightarrow o$

3rd-order questions and answers activate pushes and pops; automaton is a Visibly PA.
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

**\textbf{OBS EQUIV}_L:** Given \(\beta\)-nfs \(M\) and \(N\) in sublanguage \(L\) of IA, does \(M \approx N\)?

<table>
<thead>
<tr>
<th>(IA_i), (i \geq 4)</th>
<th>pure</th>
<th>+\textbf{while}</th>
<th>+(\textbf{Y}_0)</th>
<th>+(\textbf{Y}_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(IA_0)</td>
<td>\textbf{PTIME}</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>(IA_1)</td>
<td>\textbf{coNP}</td>
<td>\textbf{PSPACE}</td>
<td>\textbf{DPDA EQUIV}</td>
<td>undecidable</td>
</tr>
<tr>
<td>(IA_2)</td>
<td>\textbf{PSPACE}</td>
<td>\textbf{PSPACE}</td>
<td>\textbf{DPDA EQUIV}</td>
<td>undecidable</td>
</tr>
<tr>
<td>(IA_3)</td>
<td>\textbf{EXPTIME}</td>
<td>\textbf{EXPTIME}</td>
<td>\textbf{DPDA EQUIV}</td>
<td>undecidable</td>
</tr>
<tr>
<td>(IA_i), (i \geq 4)</td>
<td>undecidable</td>
<td>undecidable</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>

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

\textbf{coNP} + \textbf{PSPACE} results: Murawski TCS 2005(?).

\textbf{EXPTIME} results: Murawski + Walukiewicz FOSSACS’05.

\textbf{DPDA EQUIV} results: Murawski, Walukiewicz + Ong (submitted).
Deciding $\approx$ for $IA_1$ is coNP-complete (Murawski 2005)

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

A coNP-procedure for deciding $\approx$.

Given $M$ and $N$:

- Construct a DFA recognising $comp([M])$ 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 $\text{IA}_1$ is $\text{coNP}$-complete (cont’d)

**coNP-Hardness.** Recall: $\text{TAUTOLOGY}$ is $\text{coNP}$-complete.

Boolean formulas: $B ::= X_i \mid B \lor B \mid B \land B \mid \neg B$

Translation of Boolean formulas $B(X_1, \cdots, X_k)$ to $\text{IA}_1$-terms $M_B$:

\[
\begin{align*}
M_X &= \text{if } !X \text{ t f} \\
M_{\neg B} &= \text{if } M_B \text{ f t} \\
M_{B_1 \lor B_2} &= \text{if } M_{B_1} \text{ t } M_{B_2} \\
M_{B_1 \land B_2} &= \text{if } M_{B_1} M_{B_2} \text{ f }
\end{align*}
\]

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

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

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

Deciding ≈ for \( lA_3 + \text{while} \) 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 \( lA_3 + \text{while} \)-terms are VPA-recognizable. (FOSSACS05 Best Paper Award)

EXPTIME-hardness: by reducing the EXPTIME-complete problem \( \text{FINITE TREE AUTOMATA EQUIVALENCE} \) (Seidl 1990) to it.
Deciding \( \equiv \) for \( \exists A_i + Y_0 \) (for \( i = 1, 2, 3 \)) is equivalent to DPDA \( \text{EQUIV} \) (Murawski, O. + Walukiewicz ICALP 2005.)

\( \exists A_i + Y_0 \): Only terms of ground type can call themselves recursively.

**Example.** Non tail-recursive ground recursion:

\[
\begin{align*}
  c : \text{com}, b : \text{bool} & \vdash Y(\lambda p^{\text{com}}.\text{if } b (p ; c ; p) \text{ skip}) : \text{com} \\
  \Xi
\end{align*}
\]

We construct a DPDA accepting \( [\Xi] \) ("\( q \)-semantics") given by

\[
\text{comp}(\langle \Xi \rangle) = \text{run} \cdot (\langle \Xi \rangle) \cdot \text{done}
\]

from the \( q \)-semantics of

\[
\begin{align*}
  c : \text{com}, b : \text{bool}, p : \text{com} & \vdash \text{if } b (p ; c ; p) \text{ skip} : \text{com}
\end{align*}
\]

in stages.
(i) \( c : \text{com}, b : \text{bool}, p : \text{com} \vdash \text{if } b (p ; c ; p) \text{ skip : com} \)

Set \( G = \text{com} \rightarrow \text{bool} \rightarrow \text{com} \rightarrow \text{com} \).

Input symbols are \( \{ r_c, d_c, q, t, f, r_p, d_p \} \).

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

Game Semantics and its Algorithmic Applications: Lecture 5, Newton Institute, February 2006.
(iii) Connecting the two copies: getting rid of $r_p, d_p$ or 1, 2, 3 and 4

$\epsilon/1 = \text{"read no input symbol and push 1"}; \quad \epsilon, 1 = \text{"read no input symbol and pop 1"}. $
(iii) Connecting the two copies: getting rid of more $r_p, d_p$
\((c : \text{com}, b : \text{bool} \vdash Y(\lambda p^\text{com}. \text{if } b (p ; c ; p) \text{ skip}) : \text{com})\)
Construction of the complete-play DPDA$_M$ for $M \in IA_i + Y_0$: 3 stages

(1) The underlying move-sequences of complete plays of simple $(IA_i + Y_0)$-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 $\mathbf{IA}_1 + \mathbf{Y}_0$

**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

$$\text{comp}([x : \mathbf{exp} \vdash M_A : \mathbf{com}]) = \text{comp}([x : \mathbf{exp} \vdash M_B : \mathbf{com}])$$

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

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

Note: $\hat{L}_1 = \hat{L}_2 \iff L_1 = L_2$. 
The \((IA_1 + Y_0)\)-translate \(M_A\) of a DPDA \(A\)

\[
x : \exp \vdash \begin{array}{l}
\text{new } Q := q_0, \ TOP := Z_0, \ CH \ \text{in}\\
\mu z^{\text{com}}. \ \text{new } POP := 0, \ X := !TOP \ \text{in}\\
\quad \text{while } (\text{not } !POP) \ \text{do}\\
\quad \quad (\text{if } \delta(!Q, \epsilon, !X) = (q', \alpha) \ \text{then}\\
\quad \quad \quad (Q := q';\\
\quad \quad \quad \quad \text{if } \alpha = \epsilon \ \text{then } POP := 1 \ \text{else } ((TOP := \alpha_1); z))\\
\quad \quad \text{else}\\
\quad \quad \quad (CH := x;\\
\quad \quad \quad \quad \text{if } \delta(!Q, !CH, !X) = (q', \alpha) \ \text{then}\\
\quad \quad \quad \quad (Q := q';\\
\quad \quad \quad \quad \quad \text{if } \alpha = \epsilon \ \text{then } POP := 1 \ \text{else } ((TOP := \alpha_1); z))\\
\quad \quad \quad \text{else } \Omega^{\text{com}})) : \text{com}
\end{array}
\]
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 $\Xi$ is a term-in-context, provided $\phi$ 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\textsuperscript{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.

\textsuperscript{a}Research 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)

<table>
<thead>
<tr>
<th>array size ( n )</th>
<th>model construction time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5 mins</td>
</tr>
<tr>
<td>5</td>
<td>10 mins</td>
</tr>
<tr>
<td>10</td>
<td>15 mins</td>
</tr>
<tr>
<td>20</td>
<td>4 hours</td>
</tr>
<tr>
<td>25</td>
<td>10 hours</td>
</tr>
</tbody>
</table>

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.
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 $\left[ x : \text{nat} \vdash M : \text{nat} \right]$ 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


Game Semantics and its Algorithmic Applications: Lecture 5, Newton Institute, February 2006.