Games Semantics: from Structures to Algorithmics

Luke Ong

Oxford University Computing Laboratory

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

(Acknowledgements: Samson Abramsky, Dan Ghica + Andrzej Murawski)

Model Checking

Extremely successful in verifying relatively flat “unstructured” finite-state processes.

E.g. digital circuits and communication protocols).

Naïve application of these techniques to software raises problems:

1. Programs are potentially infinite-state systems. Tools are (mainly) finite-state technologies.
2. Modern programs can only be accurately modelled using rich, highly-structured models, as studied in Semantics. These are not appropriate (as models) for Verification.

Software Model Checking: Challenges and Approaches

"Model Construction Problem": No systematic, fully automatic method exists for model construction. Tension between:

- Accuracy in modelling - tends to make models large
- Engineering feasibility - tends to make models small.

A leading paradigm: Abstract–Test–Refine Cycle.

An alternative: Start from an accurate denotational semantics of the program and then 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.

Question: Is there such a semantics?

Software Model Checking based on Game Semantics

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

All of these models are highly accurate: fully abstract.

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.

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; 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 of the Talk

I. A **H**igher-**O**rder **P**rocedural **L**anguage (**HOPL**): **Idealized Algol**

II. **Game Semantics**: An Impressionistic Introduction

III. Some Results in **Algorithmic Game Semantics**: Putting Game Semantics to Work

IV. Applications to **Software Model Checking**: A Prototype Tool

V. Further Directions

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

$$
\begin{array}{rcll}
T & ::= & \mathsf{int} & \text{integer-valued expressions} \\
 & | & \mathsf{com} & \text{commands} \\
 & | & \mathsf{loc} & \text{locations (or assignable variables)} \\
 & | & T \to T &
\end{array}
$$

IA Terms

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

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 com)

$$C[M], S \Downarrow \mathsf{skip}, S' \iff C[N], S \Downarrow \mathsf{skip}, S'.$$

Quantification over all program context ensures that all side effects of M and N are taken into account.

> Observational equivalence 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: No construct $\mathsf{Snapback} : \mathsf{com} \to \mathsf{com}$ that runs its argument and then undoes all its state-changes.

$$\begin{array}{ll} & p : \mathsf{com} \to \mathsf{com} \\ \vdash & \mathsf{new}\, x := 0\, \mathsf{in}\, \{p(x := 1)\, ;\, \mathsf{if}\, !x = 1\, \mathsf{then}\, \Omega\, \mathsf{else}\, \mathsf{skip}\} \quad \approx \quad p(\Omega) \end{array}$$

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

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

Question. Is observational equivalence decidable?

Game semantics can help to answer the question.

II. Game Semantics: An Informal Introduction

Game Semantics (for Idealized Algol)

Players	Point-of-view	Functional	Imperative	Concurrent
P	System	term	procedure	process
O	Environment	program context	context & store	rest-of-system

Basic Idea of Game Semantics: The meaning of a system is given in terms of its potential interaction with its environment.

Four kinds of moves: P-questions, P-answers, O-questions, O-answers.

Types are modelled by arenas.

Programs of a given type are modelled by strategies (for P) for playing in the arena which denotes that type.

An **evaluation** of the program (P), in a given program context (O), corresponds to a play between P and O, tracing out a dialogue of questions and answers.

Definition: Arenas

An arena $A = \langle M_A, \vdash_A, \lambda_A \rangle$ is a triple such that $\langle M_A, \vdash_A \rangle$ is a directed graph which is a forest, whose nodes (i.e. moves) are partitioned by

$$\lambda_A : M_A \longrightarrow \{ OQ, PA, PQ, OA \}$$

into P-questions, P-answers, O-questions and O-answers, that satisfies the following rules:

(1) Roots (moves at level 0), called initial, are O-questions.

(2) Levels 0, 2, 4, etc. are O-moves; levels 1, 3, 5, etc. are P-moves.

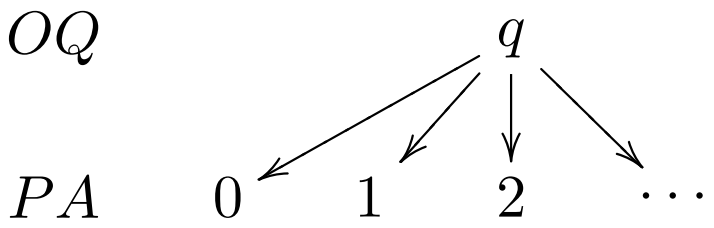
(3) Whenever $m \vdash_A m'$ (read "m justifies m'") then m is a question. □

Note. The arena is not the game tree. The game tree over an arena A is generated from A, subject to a number of rules.

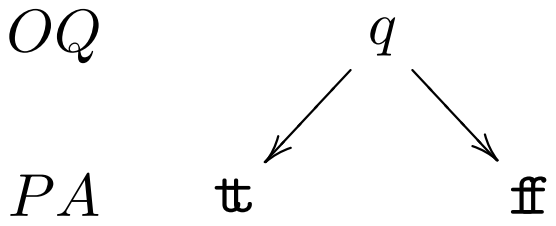
Example arenas

Empty arena $\mathbf{1} = \langle \varnothing, \varnothing, \varnothing \rangle$.

Natural numbers arena:



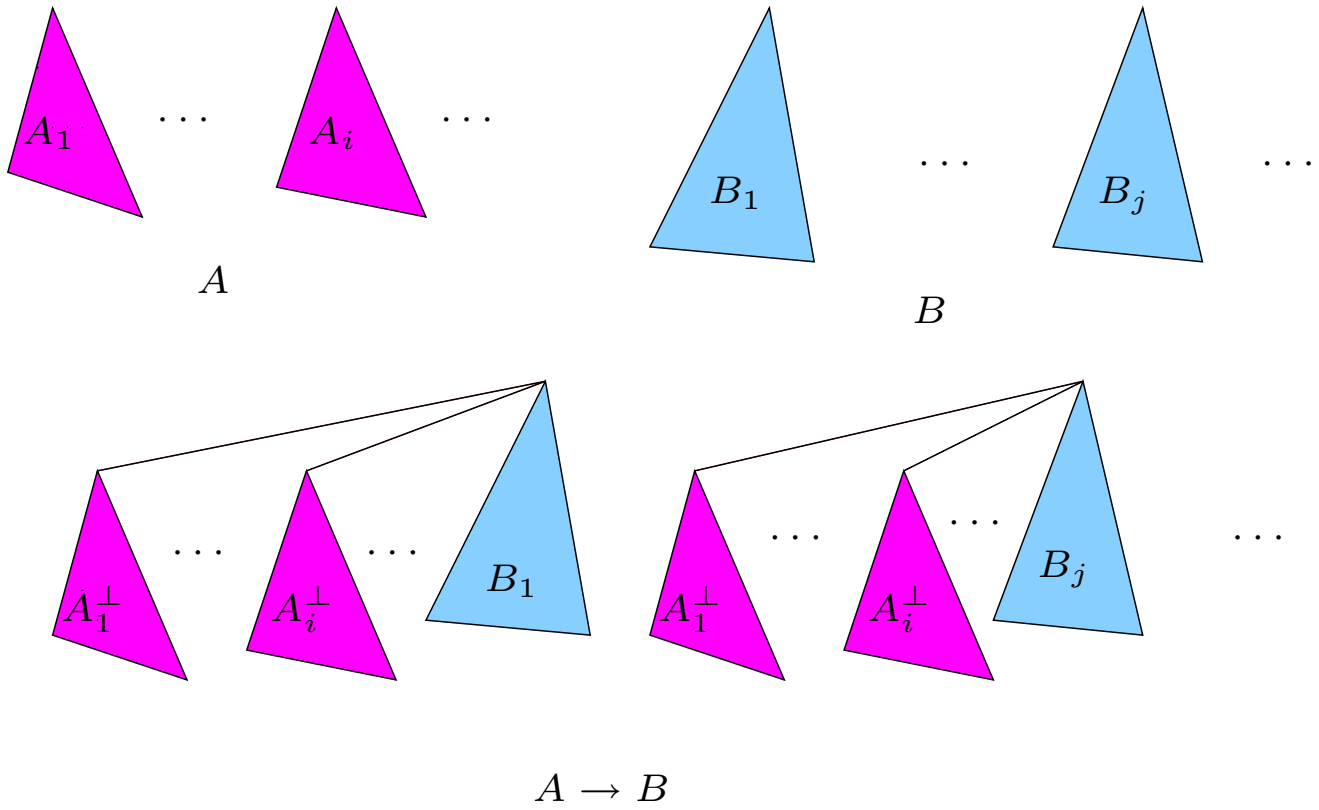
Boolean arena:



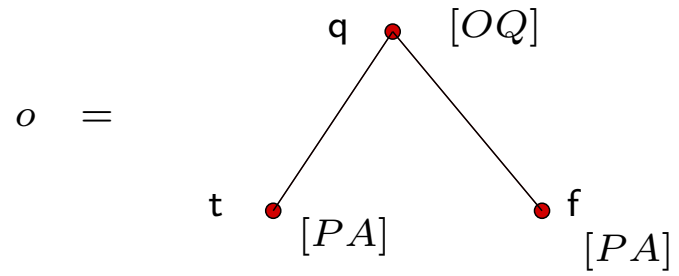
Arena constructions

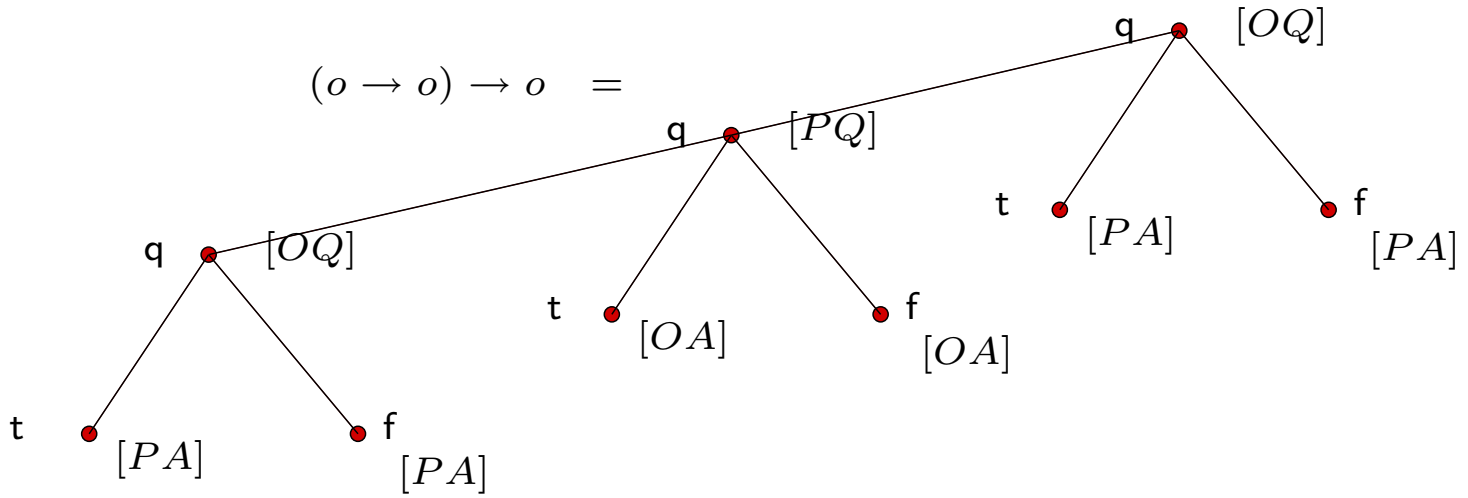
Product arena $A \times B$: Disjoint union of (the underlying forest of) A and B.

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



More examples of arenas





What are the plays (= legal positions)?

Intuitively, a play or legal position is a record of the moves made by O and P in the course of a game.

Basic rules of the game (for HOPL computation):

1. Alternation. O starts, thereafter P and O alternate strictly.
2. Justification. At any point (after the opening move), the playable moves are exactly those whose justifier moves have already been played.

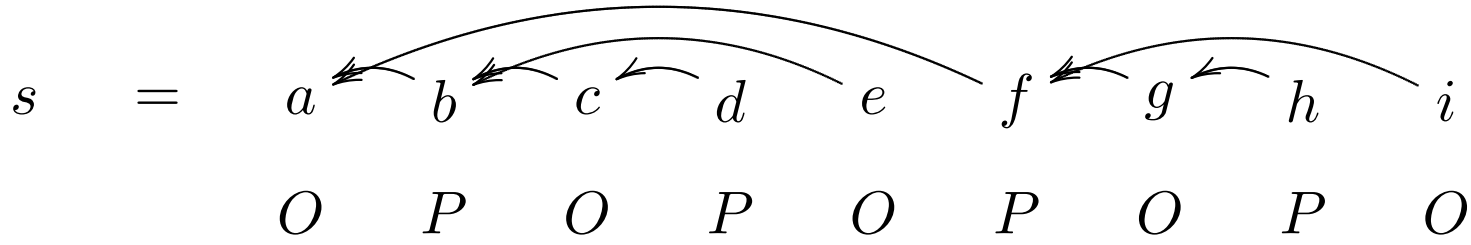
   [The justifier move of an non-initial m is the unique m' such that $m' \vdash_A m$.]
3. Whenever a player makes a move, he must also declare the justification for it (by way of a pointer from the move to some occurrence of its justifier move).

Thus every move in a play is explicitly justified.

What are the plays (i.e. legal positions)?

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 $m' \vdash_A m$.

Example

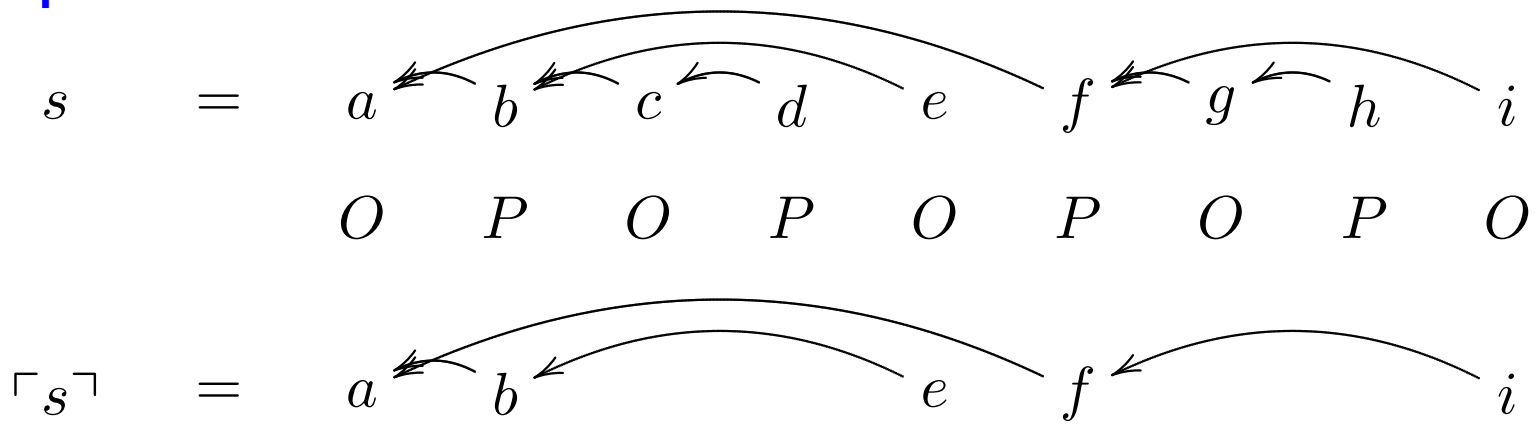


What are the plays?

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). Intuitively these are "O-moves that directly respond to moves that P has made (in a hereditarily fashion)".

Example



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 $m' \vdash_A m$.

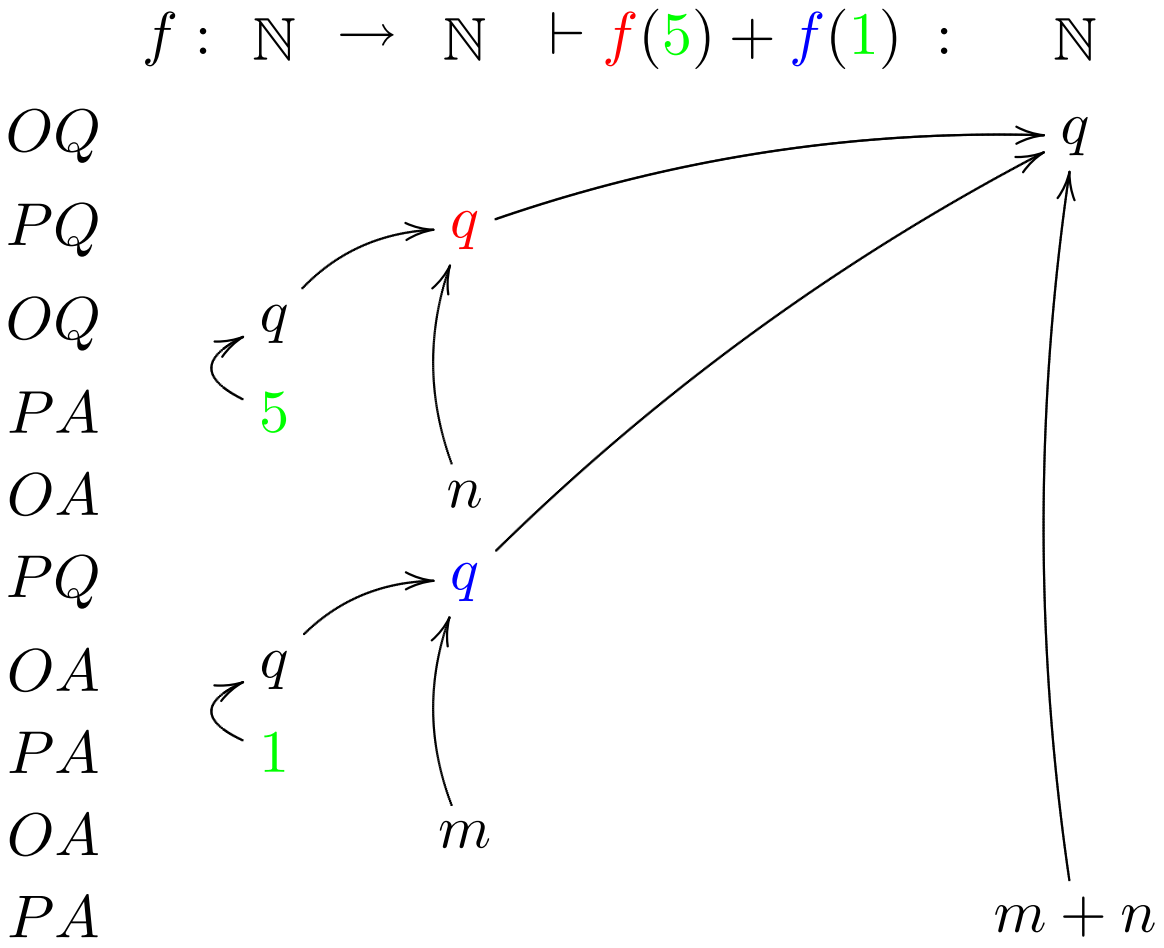
The **P-view** $\ulcorner s \urcorner$ of a justified sequence s is a subsequence consisting only of moves which P considers relevant (for determining his response). (Similarly for O-view.)

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

(1) **Visibility**: Each P-move is explicitly justified by (i.e. points to) some move that appears in the P-view at that point; similarly for O.

(2) **Well-Bracketing**: Each answer is explicitly justified by the last *pending* question.

Example

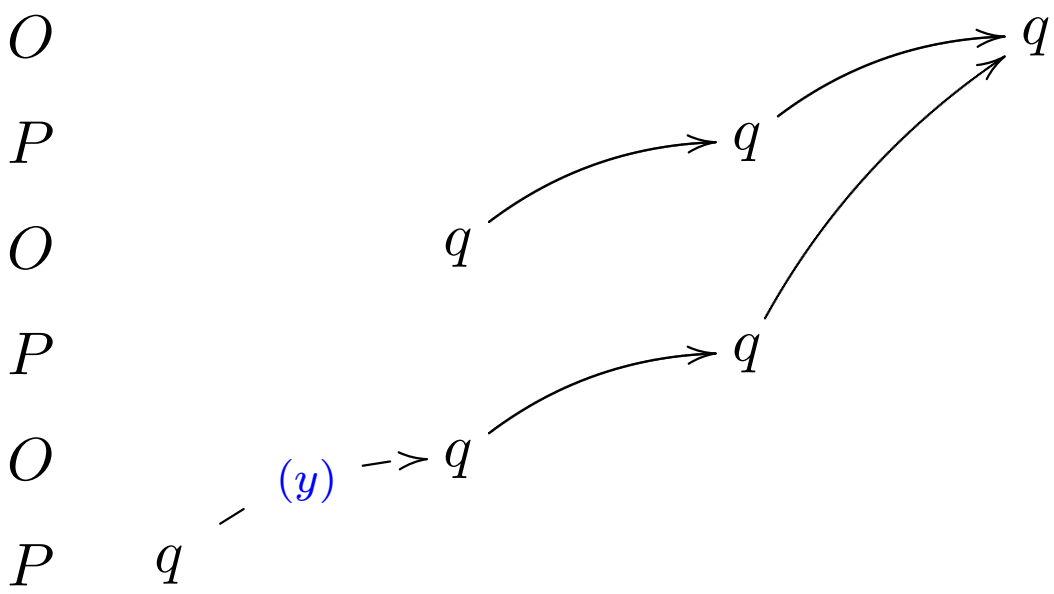

$f : \mathbb{N} \to \mathbb{N} \vdash f(5) + f(1) : \mathbb{N}$
OQ
PQ
OQ
PA
OA
PQ
OA
PA
OA
PA
q
q
5
n
q
q
1
m
q
m + n

Example: What's the point of pointers?

Kierstead terms

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

$$\begin{array}{cccccccc} type{-}3 & & type{-}2 & & type{-}1 & & type{-}0 \\ ((\mathbb{N} & \to & \mathbb{N}) & \to & \mathbb{N}) & \to & \mathbb{N} \end{array}$$

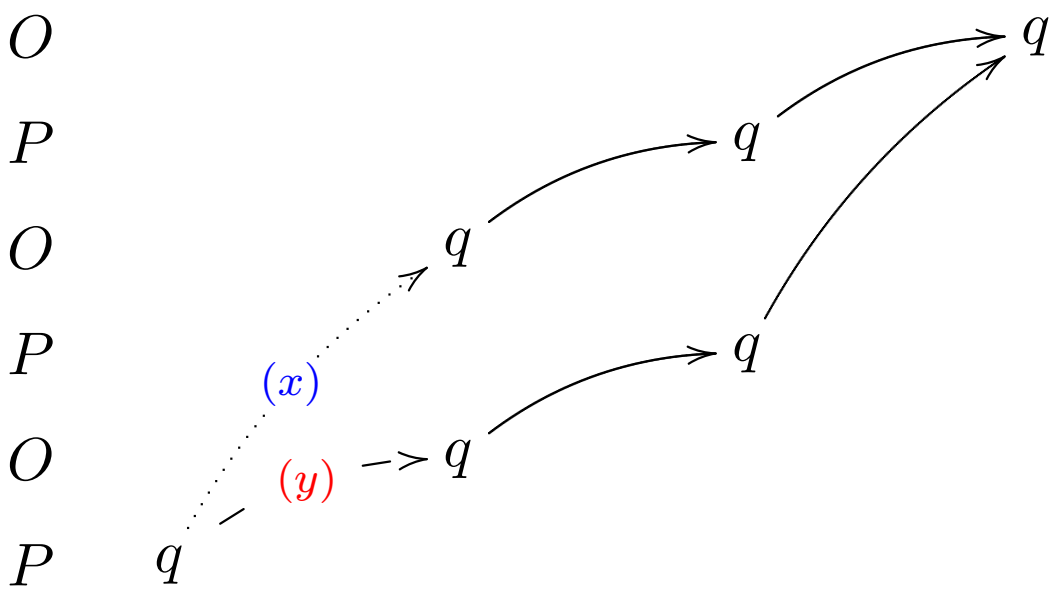


Example: What's the point of pointers?

Kierstead terms

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

$$\begin{array}{cccccccc} type{-}3 & & type{-}2 & & type{-}1 & & type{-}0 \\ ((\mathbb{N} & \to & \mathbb{N}) & \to & \mathbb{N}) & \to & \mathbb{N} \end{array}$$



A fully abstract game model of PCF [Hyland-O. 2000]

A P-strategy σ is said to be **innocent** iff it is (deterministic) and view-dependent i.e. it is generated from a partial function from P-views to justified P-moves.

The category:

- Objects are arenas
- Maps $\sigma : A \longrightarrow B$ are innocent P-strategies over the arena $A \to B$.

is cartesian closed, and gives rise to the first (syntax-independent) fully abstract model of PCF – the functional part of IA.

[Recall: $\mathcal{M}$ fully abstract means $\mathcal{M} \models M = N \iff M \approx N$.]

Classifying Programming Features

Strategies can be classified according to the behavioural constraints they satisfy. Four constraints identified in the fully abstract game model for PCF.

1. Determinacy
2. Well-Bracketing
3. Innocence
4. Visibility

Classifying Programming Features

Strategies can be classified according to the behavioural constraints they satisfy. Four constraints identified in the fully abstract game model for PCF.

1. Determinacy: Erratic choice (Harmer + McCusker 1998)
2. Well-Bracketing: Non-local control operators (Laird 1998)
3. Innocence: Mutable store (Abramsky + McCusker 1996)
4. Visibility: Higher-order store (reference types) (Abramsky, Honda + McCusker 1999)

Later work showed that each corresponds precisely to the absence/presence of certain programming features.

Game semantics gives accurate models of advanced high-level programming languages.

III. Some results in Algorithmic Game Semantics

Putting Game Semantics to work!

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

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

Theorem. (Ghica-McCusker). For finitary 2nd-order IA terms M and N

$$M \approx N \iff \underbrace{[\![\Gamma \vdash M : A]\!]^{\mathrm{reg}}}_{R} = \underbrace{[\![\Gamma \vdash N : A]\!]^{\mathrm{reg}}}_{S}$$

Moreover $R = S$, equivalence of regular expressions, is decidable. □

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

$$M \approx N \iff \textbf{cplays}[\![\Gamma \vdash M : A]\!]^{\mathrm{Kn}} = \textbf{cplays}[\![\Gamma \vdash N : A]\!]^{\mathrm{Kn}}$$

Lemma. The set of complete plays in $[\![\Gamma \vdash M : A]\!]^{\mathrm{Kn}}$ is regular.

Note. Up to order 2, justification pointers may be ignored; so plays are just words over alphabet of moves.

Decidability and undecidability results

Two orthogonal directions of extension:

Fragments of finitary IA	**Is observational equivalence decidable?**
2nd-order	Yes. (Ghica+McCusker 00)

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

Decidability and undecidability results

Two orthogonal directions of extension:

Fragments of finitary IA	**Is observational equivalence decidable?**
2nd-order	Yes. (Ghica+McCusker 00)
2nd-order + iteration	Yes (GM 00); PSPACE-complete (Murawski 03)
2nd-order + recursion	No. (Ong LICS'02)

Decidability and undecidability results

Two orthogonal directions of extension:

Fragments of finitary IA	**Is observational equivalence decidable?**
2nd-order	Yes. (Ghica+McCusker 00)
2nd-order + iteration	Yes (GM 00); PSPACE-complete (Murawski 03)
2nd-order + recursion	No. (Ong LICS'02)
3rd-order	Yes: reduction to DPDA Equivalence. (Ong 02)
4th-order or higher	No. (Murawski LICS 03)

Decidability and undecidability results

Two orthogonal directions of extension:

Fragments of finitary IA	**Is observational equivalence decidable?**
2nd-order	Yes. (Ghica+McCusker 00)
2nd-order + iteration	Yes (GM 00); PSPACE-complete (Murawski 03)
2nd-order + recursion	No. (Ong LICS'02)
3rd-order	Yes: reduction to DPDA Equivalence. (Ong 02)
4th-order or higher	No. (Murawski LICS 03)
3rd-order + iteration	?

Decidability and undecidability results

Two orthogonal directions of extension:

Fragments of finitary IA	**Is observational equivalence decidable?**
2nd-order	Yes. (Ghica+McCusker 00)
2nd-order + iteration	Yes (GM 00); PSPACE-complete (Murawski 03)
2nd-order + recursion	No. (Ong LICS'02)
3rd-order	Yes: reduction to DPDA Equivalence. (Ong 02)
4th-order or higher	No. (Murawski LICS 03)
3rd-order + iteration	Yes. Rationally innocent strategies.

A rationally innocent strategy is generated by a view function whose domain is a disjoint union of a finite number of regular sets such that the function acts uniformly on each set.

Type-theoretic orders and expressiveness

Consider finitary IA generated from base types $\Sigma = \{ x_1, \cdots, x_n \}$, com and loc. For a language $L \subseteq \Sigma^*$, we say that $\Gamma \vdash M : A$ represents L just in case

$$L \;=\; \textbf{cplays}[\![\Gamma \vdash M : A]\!] \restriction \Sigma$$

Theorem (Murawski 03) Let $L \subseteq \Sigma^*$.

(i) L representable by a 2nd-order IA-term-in-context iff L is regular.

(ii) L representable by a 3rd-order IA-term-in-context iff L is context-free.

(iii) L representable by a 4th-order IA-term-in-context iff L is r.e.

Obs. Equiv of 3rd-order IA is decidable: Proof Idea

At 3rd and higher orders, pointers can no longer be ignored. We use view offset to encode pointers.

We give an intensional, state-explicit representation of the fully abstract game semantics $[\![\Gamma \vdash M : A]\!]^{\mathrm{state}}$ (so that each move in a play is annotated with a state).

Lemma. (i) $[\![\Gamma \vdash M : A]\!]^{\mathrm{state}}$ is compactly innocent i.e. generated by a finite (view) function f_M. (ii) Further f_M determines an associated DPDA P_{f_M}.

Theorem. The language recognized by the DPDA P_{f_M} is precisely the complete plays of $[\![\Gamma \vdash M : A]\!]^{\mathrm{Kn}}$.

Thus we can decide observational equivalence of 3rd-order IA, provided there is a procedure to decide: DPDA Equivalence.

DPDA Equivalence: Given two DPDAs, do they recognize the same language?

First posed in 1966, it was proved to be decidable by Seńizergues in 1998/2001 (Gödel Prize 2002). Primitive recursive bound by Stirling (*TCS* **255**:1-31, 2001.

Theorem. There is an algorithm for deciding

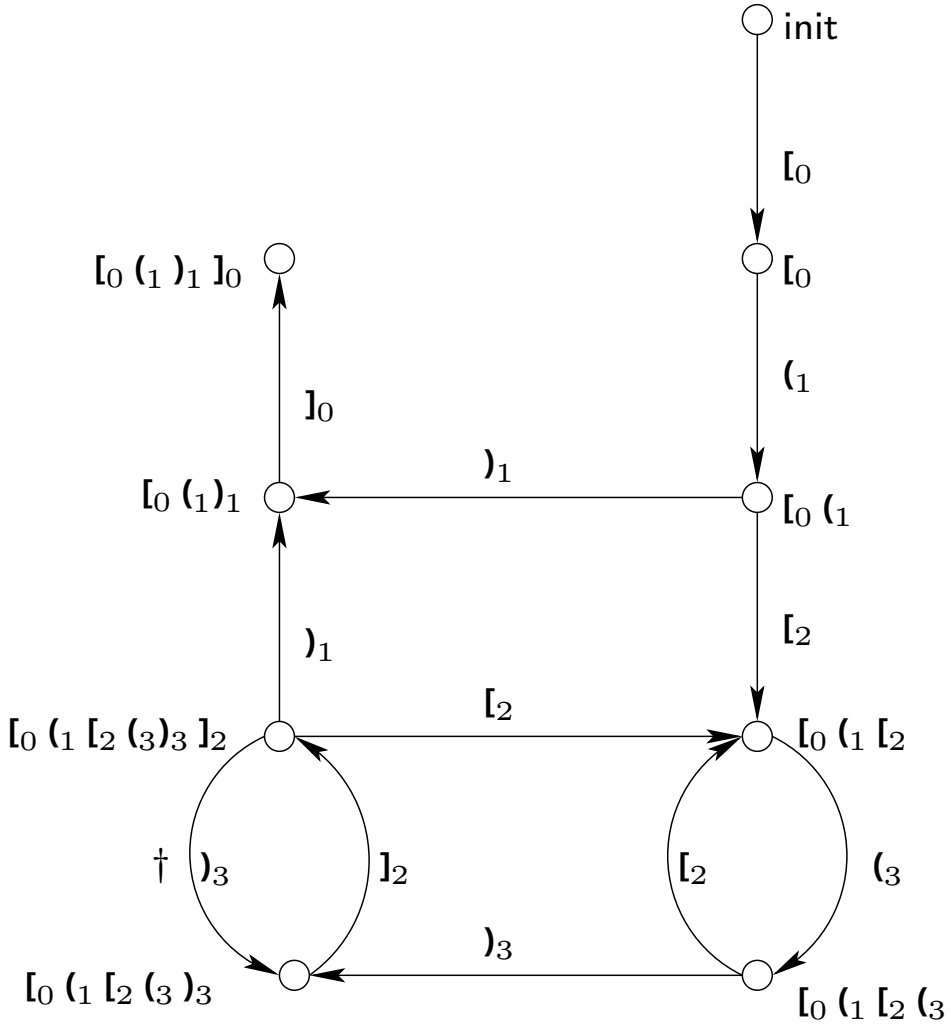
Given two finitary 3rd-order IA terms M and N, does M observationally approximate N?

in $O(2^{2^n})$.

Proof: superdeterministic PDAs: subclass with a decidable inclusion problem.

Or use Visibly pushdown automata [AM04]!

Example: DPDA defined by $\boldsymbol{\lambda} f.f(\boldsymbol{\lambda} x.x) : ((o \to o) \to o) \to o$



IV. Applications to Software Model Checking

Prototype Tool (developed by Dan Ghica)

A compiler transforming an open procedural program into finite automaton representing its fully abstract game semantics. See [AGMO04] at TACAS '04.

Parser + type inference + back-end processing in CAML. Back-end heavy-duty regular language processing uses AT&T FSM Library.

A case study: Bubble sort

Why sorting? Well-known pathological case involving complicating interactions between data and control control.

"... 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.

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

The DFA model is fully abstract.

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

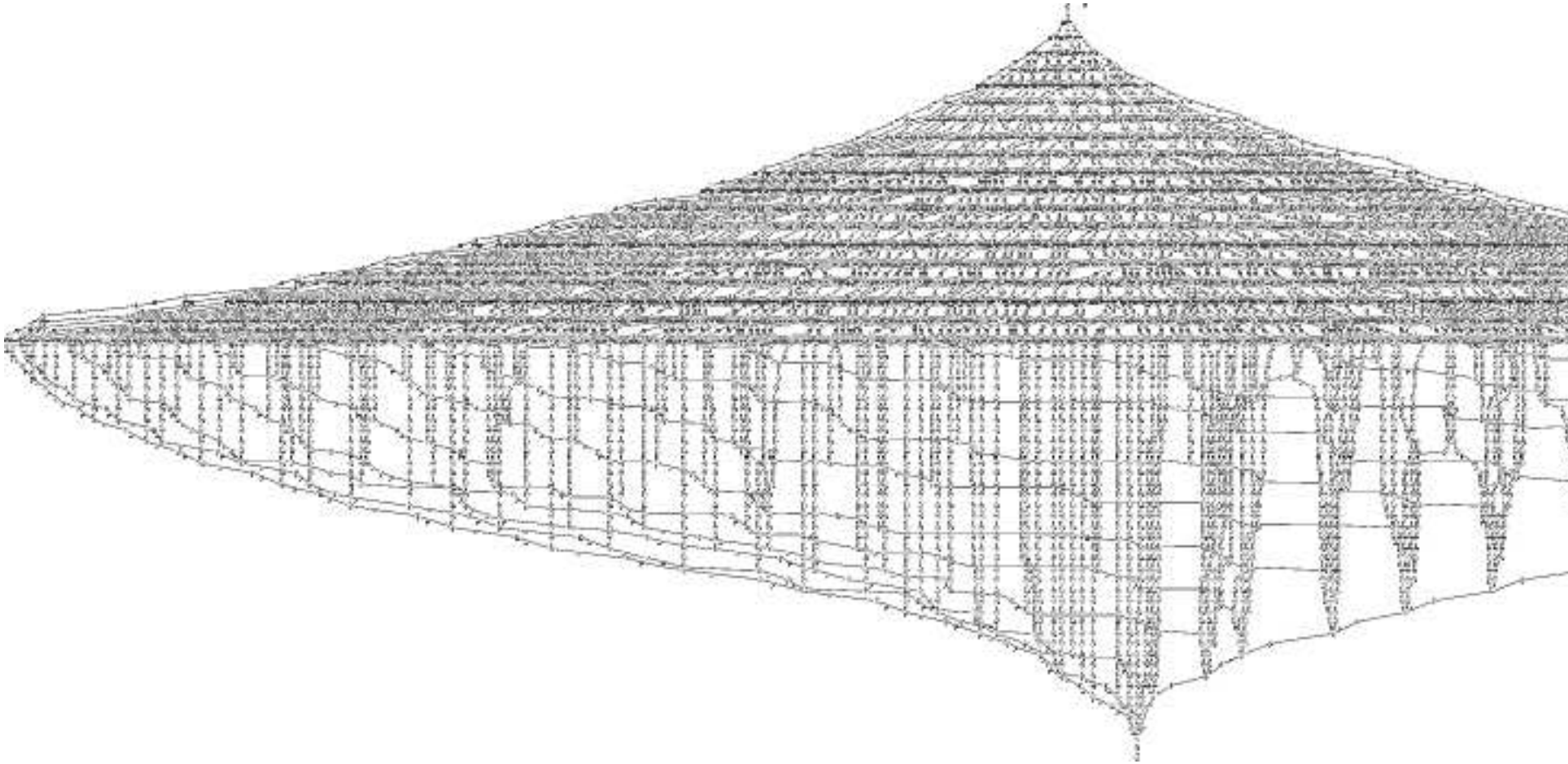
array size n	model construction time
2	5 mins
5	10 mins
10	15 mins
20	4 hours
25	10 hours

An array of size 20 (over integers MOD 3) has *circa* 3^{20} states.

Key features of our Model:

- representation of the fully abstract semantics.
- highly compact: it has only about 5500 states!
- extensional: tranforms of all (correct) sorting algorithms are isomorphic.

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



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 $\Xi = p : \mathsf{int} \to \mathsf{int}, x : \mathsf{loc} \vdash M : \mathsf{com}$, and

> ϕ = "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

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

A new approach to Software Model Checking (con'd)

Fact. Definability by regular expressions is equivalent to definability in QPLTL.

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.

We intend to:

- 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.

Further directions: Verification of recursive HOPL Programs

Verifying infinitary computation against infinitary properties

Finitary computation	Finitary properties	Yes
Infinitary computation	Finitary properties	Yes
Infinitary computation	Infinitary properties	?

E.g. Model Checking. Given a order-n recursively-defined IA term M and a μ-calculus formula ϕ, does $[\![M]\!] \models \phi$?

General problem is hard: Don't have a sensible model of computation for generating $[\![M]\!]$ for $n > 3$.

First step: restrict to Pure functional fragment of IA - already very rich; e.g. it contains the hierarchy of safe higher-order recursion scheme Damm / Knapik-Niwinski-Urzyczyn / Caucal etc. Many intriguing and challenging problems.