NOMINAL GAME SEMANTICS

PART II

Andrzej Murawski
UNIVERSITY OF OXFORD

FULL ABSTRACTION

$$[\![M_1]\!] = [\![M_2]\!] \quad \text{if and only if} \quad M_1 \cong M_2$$

$$\vdash \lambda f^{\mathsf{int}\to\mathsf{int}}.f(0)+1 : (\mathsf{int}\to\mathsf{int})\to\mathsf{int}$$

$$\begin{array}{cccccc} \star & \dagger & \dagger' & 0_2 & 3_1 & 4_0 \\ O & P & O & P & O & P \end{array}$$

		$\vdash$	$(\mathsf{int}_2$	$\to$	$\mathsf{int}_1)$	$\to$	int_0
O	$\star$						
P						$\dagger$	
O				$\dagger'$			
P			0_2				
O					3_1		
P							4_0

$$\mathsf{let}\; g = [\;]\;\mathsf{in}\; g(\lambda x^{\mathsf{int}}.x+3)$$

ARENAS

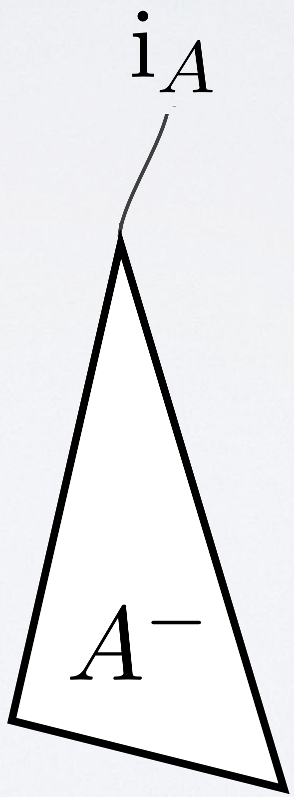
An *arena* $A = \langle M_A, I_A, \lambda_A, \vdash_A \rangle$ is given by:

- a set of moves M_A and a subset $I_A \subseteq M_A$ of initial ones,
- a labelling function $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$,
- an enabling relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$;

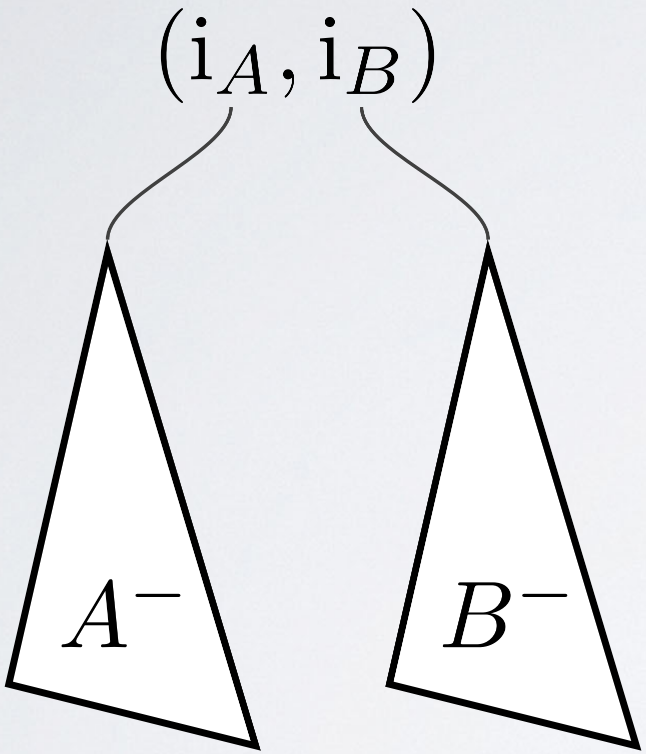
satisfying, for each $m, m' \in M_A$, the conditions:

- $m \in I_A \implies \lambda_A(m) = (P, A)$,
- $m \vdash_A m' \wedge \lambda_A^{QA}(m) = A \implies \lambda_A^{QA}(m') = Q$,
- $m \vdash_A m' \implies \lambda_A^{OP}(m) \neq \lambda_A^{OP}(m')$.

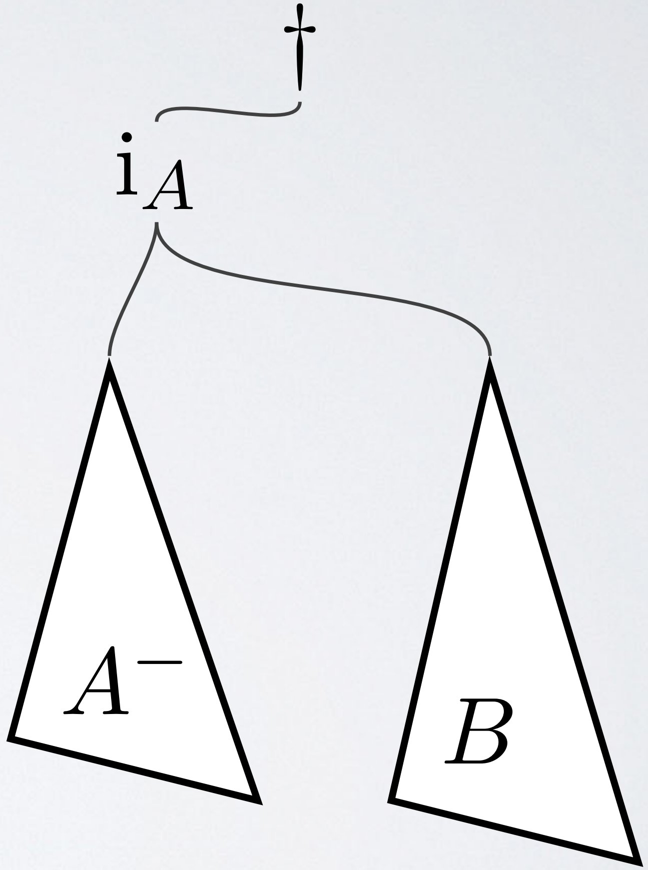
ARENA (INITIAL+REST)



ARENA CONSTRUCTIONS



$A \otimes B$



$A \Rightarrow B$

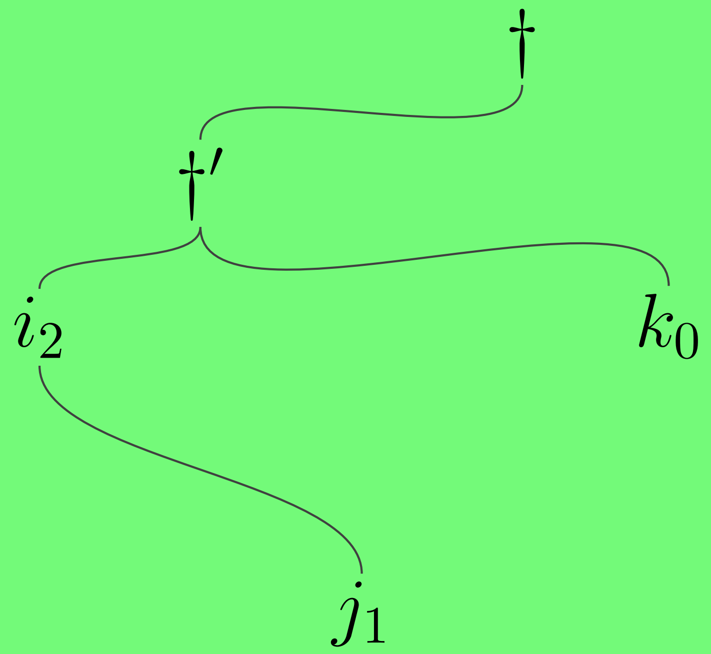
ARENA EXAMPLES

$$
\begin{aligned}
[\![\textsf{unit}]\!] &= \langle \{\star\}, \{\star\}, \emptyset, \emptyset \rangle \\
[\![\textsf{int}]\!] &= \langle \mathbb{Z}, \mathbb{Z}, \emptyset, \emptyset \rangle \\
[\![\theta \to \theta']\!] &= [\![\theta]\!] \Rightarrow [\![\theta']\!] \\
[\![\textsf{ref}\theta]\!] &= [\![\textsf{unit} \to \theta]\!] \otimes [\![\theta \to \textsf{unit}]\!]
\end{aligned}
$$

EXAMPLE (ARENA)

$$[\![(\mathsf{int} \to \mathsf{int}) \to \mathsf{int}]\!]$$

$$(\mathsf{int}_2 \;\to\; \mathsf{int}_1) \;\to\; \mathsf{int}_0$$



INTERPRETATION

- Although types are interpreted by arenas, the actual games will be played in *prearenas*, which are defined in the same way as arenas with the exception that initial moves are O-questions.

- Typed terms

$$x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M : \theta$$

are interpreted using the (pre)arena

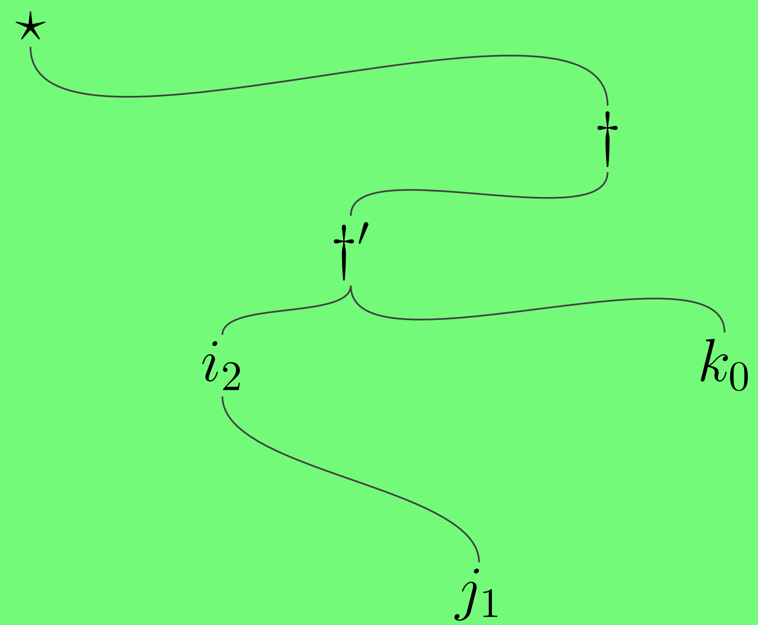
$$[\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_n]\!] \to [\![\theta]\!]$$

where $\to$ is the same way as $\Rightarrow$ but without $\dagger$.

EXAMPLE (PREARENA)

$$[\![\vdash (\mathsf{int} \to \mathsf{int}) \to \mathsf{int}]\!]$$

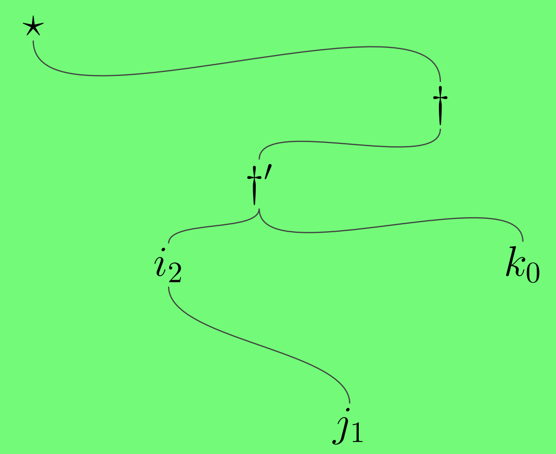
$$\vdash \; (\mathsf{int}_2 \; \to \; \mathsf{int}_1) \; \to \; \mathsf{int}_0$$

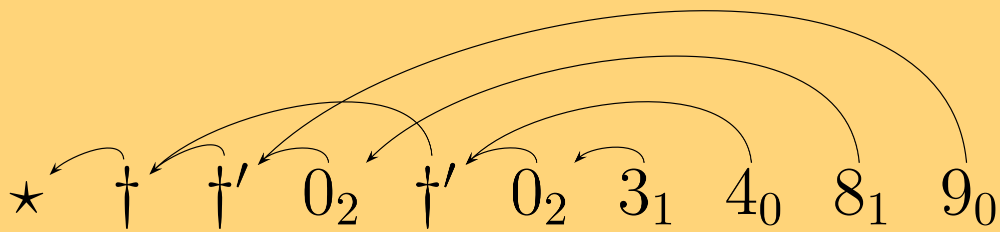


JUSTIFIED SEQUENCES

A *justified sequence* on a prearena A is a sequence of moves from M_A such that

- the first move must be from I_A,
- any other move n is equipped with a pointer to an earlier move m such that $m \vdash_A n$.

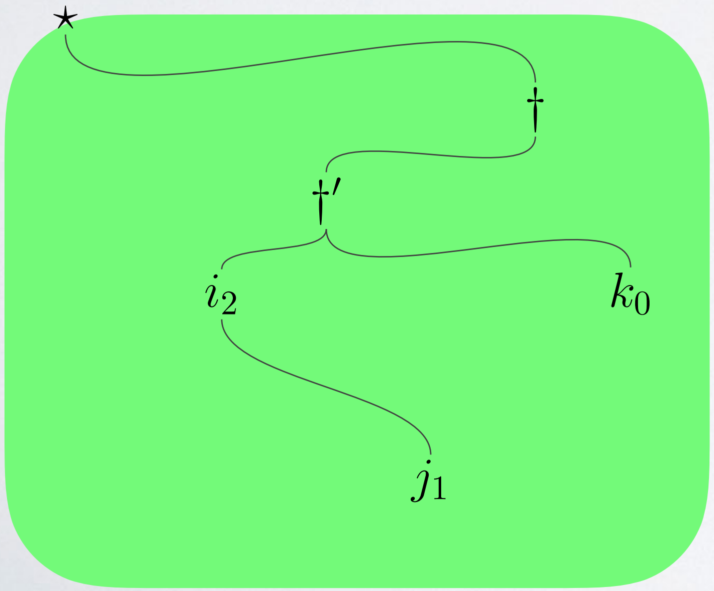


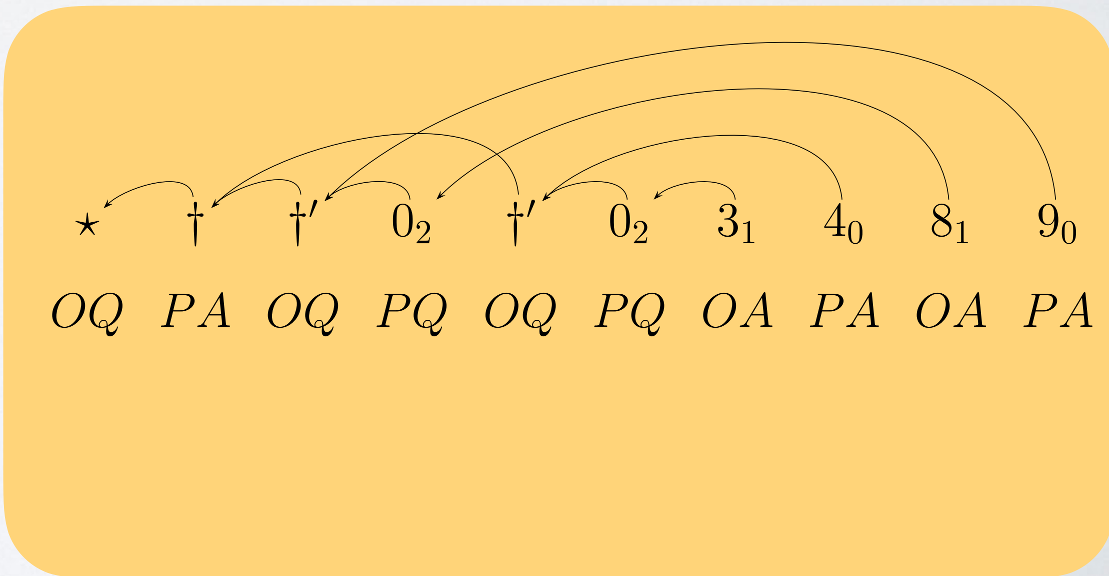


PLAYS

A *play* is a justified sequence satisfying

- alternation,
- bracketing.





STRATEGIES

A (deterministic) ***strategy*** σ on a prearena A, written $\sigma : A$, is a set of even-length plays of A satisfying

- even-prefix closure: if $sop \in \sigma$ then $s \in \sigma$,
- determinacy: if $sp_1, sp_2 \in \sigma$ then $p_1 = p_2$.

even-length prefixes of

$$\begin{array}{cccccccccc} \star & \dagger & \dagger' & 0_2 & \dagger' & 0_2 & 3_1 & 4_0 & 8_1 & 9_0 \\ OQ & PA & OQ & PQ & OQ & PQ & OA & PA & OA & PA \end{array}$$

STRATEGY COMPOSITION

$$\sigma : A \to B \qquad \tau : B \to C$$

$$\begin{array}{lccc}
 & \vdash & \mathsf{int}_2 & \to & \mathsf{int}_1 \\
O & \star & & & \\
P & & & \dagger & \\
O & & 7_2 & & \\
P & & & & 8_1 \\
O & & 10_2 & & \\
P & & & & 11_1
\end{array}$$

TOWARDS STRATEGY COMPOSITION

$$g : \mathsf{int} \to \mathsf{int} \vdash g(g(7) + 2) + 3 : \mathsf{int}$$

$$
\begin{array}{cccccc}
 & \mathsf{int}_2 & \to & \mathsf{int}_1 & \vdash & \mathsf{int}_0 \\
O & & \dagger & & & \\
P & 7_2 & & & & \\
O & & & 7_1 & & \\
P & 9_2 & & & & \\
O & & & 9_1 & & \\
P & & & & & 12_0
\end{array}
\qquad
\begin{array}{cccccc}
 & \mathsf{int}_2 & \to & \mathsf{int}_1 & \vdash & \mathsf{int}_0 \\
O & & \dagger & & & \\
P & 7_2 & & & & \\
O & & & 8_1 & & \\
P & 10_2 & & & & \\
O & & & 11_1 & & \\
P & & & & & 14_0
\end{array}
$$

INTERACTION

$$
\begin{array}{cccccc}
 & \vdash & \mathsf{int}_2 & \rightarrow & \mathsf{int}_1 & \\
O & \star & & & & \\
P & & & \dagger & & \\
O & & 7_2 & & & \\
P & & & & 8_1 & \\
O & & 10_2 & & & \\
P & & & & 11_1 &
\end{array}
$$

$$
\begin{array}{cccccc}
 & \mathsf{int}_2 & \rightarrow & \mathsf{int}_1 & \vdash & \mathsf{int}_0 \\
O & & \dagger & & & \\
P & 7_2 & & & & \\
O & & & 8_1 & & \\
P & 10_2 & & & & \\
O & & & 11_1 & & \\
P & & & & & 14_0
\end{array}
$$

INTERACTION SEQUENCE

$$\vdash \ \mathsf{int}_2 \ \rightarrow \ \mathsf{int}_1 \ \vdash \ \mathsf{int}_0$$

O	$\star$					
P			$\dagger$			O
O		7_2				P
P				8_1		O
O		10_2				P
P				11_1		O
					14_0	P

HIDING

$$\vdash \quad \mathsf{int}_0$$

$$O \quad \star$$

$$14_0 \quad P$$

STRATEGY COMPOSITION

- **Composition** = synchronised parallel composition (interaction sequence) followed by hiding

- It is non-trivial to establish associativity.

COMPOSITIONAL INTERPRETATION

- Types interpreted by games between O and P.
- Terms interpreted by strategies for P.
- Each syntactic construct interpreted through special strategies, constructions on strategies and composition.
- Categories of games (arenas) and strategies.

LICS'98

A fully abstract game semantics for general references

Samson Abramsky Kohei Honda Guy McCusker*
LFCS, University of Edinburgh St John's College, Oxford

Abstract

A games model of a programming language with higher-order store in the style of ML-references is introduced. The category used for the model is obtained by relaxing certain behavioural conditions on a category of games previously used to provide fully abstract models of pure functional languages. The model is shown to be fully abstract by means of factorization arguments which reduce the question of definability for the language with higher-order store to that for its purely functional fragment.

created object to another object.

The key idea behind our model is to represent a reference by a certain form of *information flow*: a reference is modelled not as a static entity, but as a dynamic behaviour which mediates the flow of information between readers and writers, connecting them in an appropriate fashion. In the presence of higher-order references, these connections have to be made dynamically, and the computations of the multiple readers and writers may be interleaved in arbitrarily complex ways. The technical apparatus of game semantics provides exactly the right setting in which to formalize this idea. The "dynamic connections" used to in-

REFERENCES

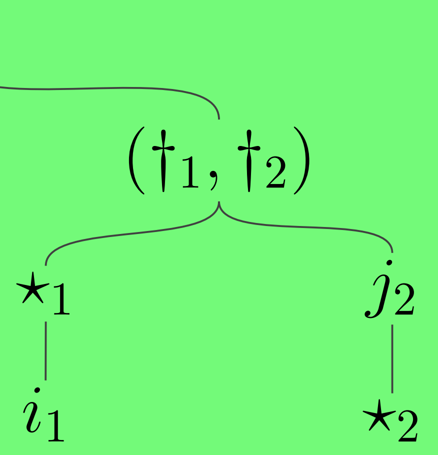
- Operational semantics uses **names** to manage resources via references.
- They come from an infinite set, can be compared for equality and generated afresh.
- Game models of references from the 1990s were name-free, though, e.g. Abramsky, Honda, McCusker [LICS'98].

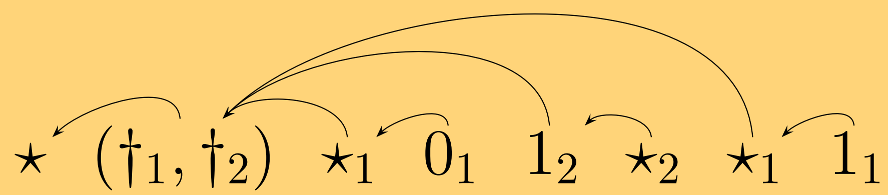
$$[\![\mathsf{ref}\,\theta]\!] = [\![\mathsf{unit} \to \theta]\!] \otimes [\![\theta \to \mathsf{unit}]\!]$$

NAME-FREE GAMES

$\vdash \mathsf{ref}_{\mathsf{int}}(0) : \mathsf{ref\ int}$

$[\![\mathsf{ref\ int}]\!] = [\![\mathsf{unit} \to \mathsf{int}]\!] \otimes [\![\mathsf{int} \to \mathsf{unit}]\!]$





BAD VARIABLES

- The model can detect the act of reading and writing.
- Full abstraction results from 1990s had to rely on syntax augmented with bad variables (and no name equality).

$$\frac{\Gamma \vdash M : \mathsf{unit} \to \theta \qquad \Gamma \vdash M : \theta \to \mathsf{unit}}{\Gamma \vdash \mathsf{mkvar}(M, N) : \mathsf{ref}\,\theta}$$

CONSEQUENCES

$$x := 1 \quad \not\cong \quad x := 1; x := 1$$

$x : \mathsf{ref\ int} \vdash x := 1 : \mathsf{unit}$

$$(\dagger_1, \dagger_2) \quad 1_2 \quad \star_2$$

$x : \mathsf{ref\ int} \vdash x := 1; x := 1 : \mathsf{unit}$

$$(\dagger_1, \dagger_2) \quad 1_2 \quad \star_2 \quad 1_2 \quad \star_2$$

FULL ABSTRACTION BY COMPLETE PLAYS

- A play is *complete* if all questions have been answered.
- Let $\mathsf{comp}(\sigma)$ be the set of complete plays in σ.

- **Full Abstraction**:

$$\Gamma \vdash M_1 \cong M_2$$

$$\text{if and only if}$$

$$\mathsf{comp}(\llbracket \Gamma \vdash M_1 \rrbracket) = \mathsf{comp}(\llbracket \Gamma \vdash M_2 \rrbracket)$$

VISIBILITY

- Without higher-order references, the patterns created by justification pointers are more restrictive.
- The target of a pointer must be present in the view of a play (**visibility**).

$$\ulcorner \varepsilon \urcorner = \varepsilon$$

$$\ulcorner s\, m t\, n \urcorner = \ulcorner s \urcorner\, m\, n$$

INNOCENCE

- Without references, strategies turn out to depend only on a fragment of play.

- **Innocence**: P's responses are determined by the view.

$$\begin{array}{cccccc} \star & \dagger & 3_1 & 4_0 & 3_1 & 4_0 \\ O & P & O & P & O & P \end{array}$$

OTHER PROPERTIES

- **Lack of alternation** (concurrency)
- **Lack of bracketing** (control)
- General theme in game semantics: capture programming language features by conditions on plays/strategies!

NOMINAL GAMES

- Dialogue between the environment (O) and the program (P).
- Technically, plays are moves that involve names drawn from an infinite set (stable under name invariance, i.e. nominal sets).
- Moves are accompanied by evolving stores.

$$\mathbb{A} = \biguplus_{\theta} \mathbb{A}_{\theta}$$

NOMINAL GAMES

$$[\![\mathsf{ref}\ \theta]\!] = \langle \mathbb{A}_\theta, \mathbb{A}_\theta, \emptyset, \emptyset \rangle$$

- Moves may contain names.
- Moves carry a store: once a new name is played, it is added to the domain of the store.

$$\star \qquad n^{(n,0)} \qquad\qquad\qquad n^{(n,i)} \qquad \star^{(n,1)}$$

EXAMPLE

$$\vdash \mathsf{let}\, n = \mathsf{ref}_{\mathsf{int}}(0)\ \mathsf{in}\ \lambda x^{\mathsf{unit}}.n : \mathsf{unit} \to \mathsf{ref\ int}$$

$$\begin{array}{cccccccc}
\star & \dagger & \star_1 & n^{(n,0)} & \star_1^{(n,5)} & n^{(n,5)} & \star_1^{(n,12)} & n^{(n,12)} \\
O & P & O & P & O & P & O & P
\end{array}$$

EXAMPLE

$$\vdash \lambda x^{\mathsf{unit}}.\mathsf{ref}_{\mathsf{int}}(0) : \mathsf{unit} \to \mathsf{ref\ int}$$

$$\begin{array}{cccccccc}
\star & \dagger & \star_1 & n_1^{(n_1,0)} & \star_1^{(n_1,5)} & n_2^{(n_1,5)(n_2,0)} & \star_1^{(n_1,7)(n_2,12)} & n_3^{(n_1,7)(n_2,12)(n_3,0)} \\
O & P & O & P & O & P & O & P
\end{array}$$

strong nominal

NOMINAL ARENA

nominal

An ***arena*** $A = (M_A, I_A, \vdash_A, \lambda_A)$ is given by:

- a set M_A of moves,
- a subset $I_A \subseteq M_A$ of initial moves,
- a relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$,
- a function $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$,

nominal

nominal

satisfying, for each $m, m' \in M_A$, the conditions:

- $m \in I_A \implies \lambda_A(m) = (P, A)$,
- $m \vdash_A m' \wedge \lambda_A^{QA}(m) = A \implies \lambda_A^{QA}(m') = Q$,
- $m \vdash_A m' \implies \lambda_A^{OP}(m) \neq \lambda_A^{OP}(m')$.

We call $\vdash_A$ the *justification relation* of A, and λ_A its *labelling function*.

STRATEGIES

A ***strategy*** σ on a prearena A is a non-empty set of even-length plays of A satisfying:

- If $so^S p^{S'} \in \sigma$ then $s \in \sigma$ (*Even-prefix closure*).
- If $s \in \sigma$ then, for all permutations π, $\pi \cdot s \in \sigma$ (*Equivariance*).
- If $sp_1^{S_1}, sp_2^{S_2} \in \sigma$ then $sp_1^{S_1} = \pi \cdot sp_2^{S_2}$ for some permutation π (*Determinacy*).

STRONG SUPPORT

- For any nominal set X, any $x \in X$ and any $S \subseteq \mathbb{A}$, S *strongly supports* x if, for any permutation π,

$$(\forall a \in S.\, \pi(a) = a) \iff \pi x = x.$$

- $\{a, b\}$ strongly supports (a, b) but not $\{a, b\}$.
- If one makes $[(a, b)\{a, b\}]$ interact with $[\{a, b\}\, a] = [\{a, b\}\, b]$ via $\{a, b\}$ one gets

$$\text{both } (a, b)\, a \text{ and } (a, b)\, b.$$

- Strong support is necessary/sufficient to preserve determinacy [Tzevelekos, LMCS'09].

HIGHER-ORDER STATE

- We cannot reveal higher-order values in the store. This would jeopardize full abstraction!

- The properties of stored values will be revealed during play thanks to the use of special pointers to the store (in previous game models, pointers could only point at other moves).

$$m^{(a,\dagger)} \quad \ldots \quad n^{(\cdots)}$$

EXAMPLE

$$x : \mathsf{ref}\,(\mathsf{int} \to \mathsf{int}) \vdash\ !x : \mathsf{int} \to \mathsf{int}$$

$$n^{(n,\dagger)}\ \ \dagger^{(n,\dagger)}\ \ 1^{(n,\dagger)}\ \ 1^{(n,\dagger)}\ \ 3^{(n,\dagger)}\ \ 3^{(n,\dagger)}$$

$$x : \mathsf{ref}\,(\mathsf{int} \to \mathsf{int}) \vdash \lambda h^{\mathsf{int}}.(!x)h : \mathsf{int} \to \mathsf{int}$$

$$n^{(n,\dagger)}\ \ \star^{(n,\dagger)}\ \ 1^{(n,\dagger)}\ \ 1^{(n,\dagger)}\ \ 3^{(n,\dagger)}\ \ 3^{(n,\dagger)}$$

COMPOSITION

- Move ownership (O-name vs P-name)
- **Interaction**: enforce disjointness of P-names, propagate foreign names
- **Hiding**: P-names cannot become O-names.

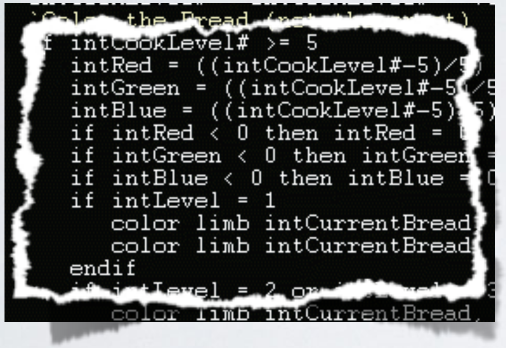
NOMINAL GAMES BIBLIOGRAPHY

- $\lambda\nu!$ (Laird; FOSSACS'04)
- ν (Abramsky, Ghica, M., Ong, Stark; LICS'04)
- Concurrent ML (Laird; FOSSACS'06)
- Reduced ML (M.,Tzevelekos; FOSSACS'09)
- RefML (M., Tzevelekos; LICS'11)
- Interface Middleweight Java (M.,Tzevelekos; POPL'14)
- ExML (M., Tzevelekos; FOSSACS 2014)

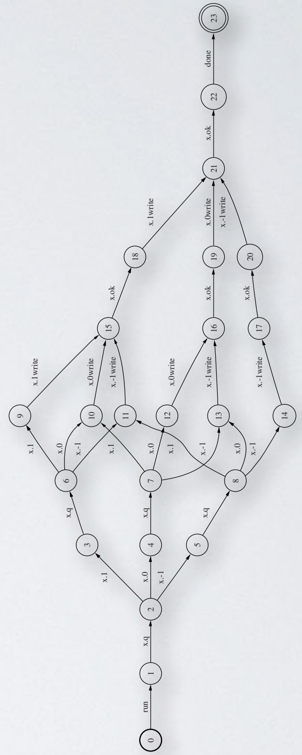
ALGORITHMIC GAME SEMANTICS

- Design of algorithms based on game semantics.
- Because of full abstraction, the most immediate application is equivalence testing.
- Numerous relationships between classes of automata and classes of strategies (obtained for restricted finitary fragments).
- Source of the first and only decidability routines for contextual equivalence.

ALGORITHMIC GAME SEMANTICS



strategy



M_1, M_2 contextually equivalent $\iff$ $[\![M_1]\!] = [\![M_2]\!]$ $\iff$ $\mathcal{A}_{M_1} \approx \mathcal{A}_{M_2}$

ALGORITHMIC NOMINAL

- The use of names means that the alphabet has to be infinite.
- Automata theory over infinite alphabets
- Lots of automata to choose from: RA, PDRA, CMA, …
- Freshness is not a major concern in XML research, but can be integrated within existing frameworks.

FINITARY GROUND ML
(FINITE INT, LOOPING, NO RECURSION)

ref int   ref (ref int)   ref (ref (ref int)) $\cdots$

$$\cdots, \theta_L, \cdots \vdash \theta_R$$

θ_R	decidability
unit	☺
unit $\to$ unit	☺
(unit $\to$ unit) $\to$ unit	☺
((unit $\to$ unit) $\to$ unit) $\to$ unit	☹
unit $\to$ unit $\to$ unit	☹

(M., Tzevelekos; ICALP'12)

TWO REASONS FOR INFINITE ALPHABETS

- **resource creation**

$$q \quad \star \quad q \quad n_1^{(n_1,\mathsf{true})} \quad q^{(n_1,\mathsf{false})} \quad n_2^{(n_1,\mathsf{false}),(n_2,\mathsf{true})}$$

- **binding structure**

$$\begin{array}{cccccccccc} q_0 & a_0 & q_1 & a_1 & q_1 & a_1 & q_1 & a_1 & q_2 & a_2 \\ n_0 & n_0 & n_1 & n_1 & n_2 & n_2 & n_3 & n_3 & n_2 & n_2 \end{array}$$

FINITARY REDUCED ML
(FINITE INT, LOOPING, NO RECURSION)

ref int $\qquad\qquad \vdash \text{unit} \rightarrow \text{unit} \rightarrow \text{unit}$

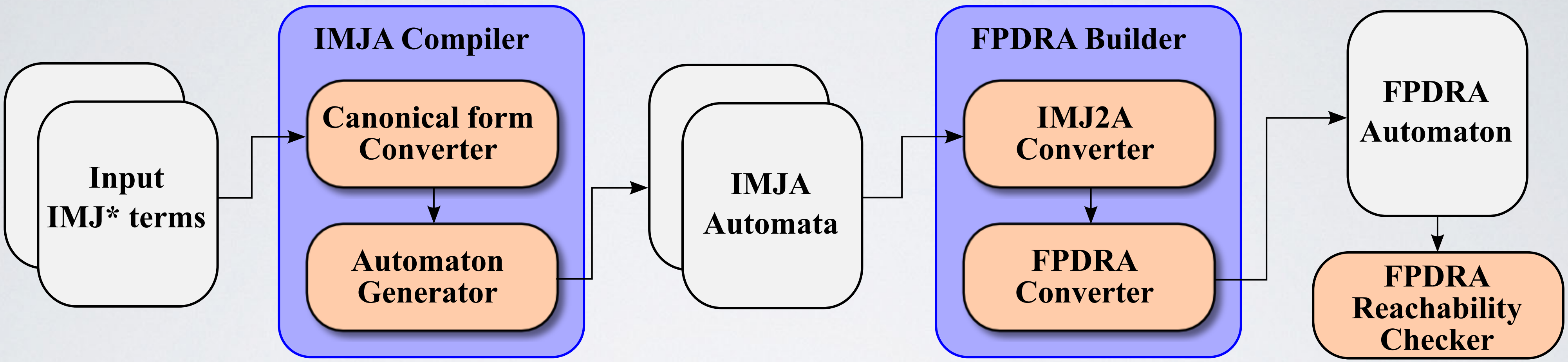
☹	Ground ML
☺	Reduced ML

- Names used to encode pointers
- Connections with (nested) Petri nets

(C.-Barratt, Hopkins, M., Ong; FOSSACS'15)

EPSRC
Engineering and Physical Sciences Research Council

CONEQCT



[POPL'14]
[ATVA'15]

[MFCS'14]
[ATVA'15]

[ICALP'12]
[MFCS'14]

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

OPERATIONAL GAME SEMANTICS

A Fully Abstract Trace Semantics for General References

J. Laird*

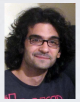
Dept. of Informatics, University of Sussex, UK
jiml@sussex.ac.uk

Abstract. We describe a fully abstract trace semantics for a functional language with locally declared general references (a fragment of Standard ML). It is based on a bipartite LTS in which states alternate between program and environment configurations and labels carry only (sets of) basic values, location and pointer names. Interaction between programs and environments is either direct (initiating or terminating subprocedures) or indirect (by the overwriting of shared locations): actions reflect this by carrying updates to the shared part of the store.

The trace-sets of programs and contexts may be viewed as deterministic strategies and counter-strategies in the sense of game semantics: we prove soundness of the semantics by showing that the evaluation of a program in an environment tracks the interaction between the corresponding strategies. We establish full abstraction by proving a definability result: every bounded deterministic strategy of a given type is the trace-set of a configuration of that type.

TUTORIALS

An invitation to game semantics



Andrzej S. Murawski
Department of Computer Science
University of Warwick, UK

Nikos Tzevelekos
School of Electronic Engineering and Computer Science
Queen Mary University of London, UK

Game semantics is a flexible semantic theory that has led in recent years to an unprecedented number of full abstraction results for various programming paradigms. We present a gentle introduction to the subject, focussing on high-level ideas and examples with a view to providing a bridge to more technical literature.

1. INTRODUCTION

Denotational semantics aims at finding meaningful compositional interpretations (denotations) of programs, couched in a variety of mathematical universes. The quality of such interpretations can then be measured by understanding which programs are interpreted in the same way, i.e. by the same elements of the model. For example, injective interpretations will be faithful models of the syntax. In contrast to that, if the modelling objective is to characterise program behaviour then one would like the interpretations of two programs to coincide if and only if the two programs are equivalent. This criterion of modelling accuracy was introduced in the 1970s [Milner 1977], under the name *full abstraction*. It has ever since become the highest prize for the practising semanticist.

However, the quest for fully abstract models was not to be an easy one. Despite advances in domain theory, which fuelled early semantic research, the construction of fully abstract models turned out elusive, even though the techniques were ripe enough to provide many informative models for numerous complicated programming features. The efforts of the semantic community in the 1990s, focussed on the purely functional language PCF, have generated a wealth of results. Among them was the emergence of a new modelling approach, referred to as game semantics, which uses the metaphor of game playing as a foundation for building models.

2. GAMES

Game semantics views computation as a two-player dialogue between a program and the context (or environment) in which it was deployed. The interlocutors, or *players*, are traditionally called O (Opponent) and P (Proponent). The former represents the context, the latter corresponds to the program. Accordingly, a program is interpreted by a strategy for P that tells P how to conduct the dialogue. Game semantics is not about winning. Rather, the challenge is to design games in such a way that strategies express the *observable* behaviour of code interacting with its computational environment.

Foundations and Trends® in Programming Languages
Vol. 2, No. 4 (2015) 191–269

DOI: 10.1561/2500000017

now
the essence of knowledge

Nominal Game Semantics

Andrzej S. Murawski
University of Warwick

Nikos Tzevelekos
Queen Mary University of London

ALGORITHMIC GAME SEMANTICS

A Tutorial Introduction

SAMSON ABRAMSKY (samson@comlab.ox.ac.uk)
Oxford University Computing Laboratory

1. Introduction

Game Semantics has emerged as a powerful paradigm for giving semantics to a variety of programming languages and logical systems. It has been used to construct the first syntax-independent fully abstract models for a spectrum of programming languages ranging from purely functional languages to languages with non-functional features such as control operators and locally-scoped references [4, 21, 5, 19, 2, 22, 17, 11]. A substantial survey of the state of the art of Game Semantics *circa* 1997 was given in a previous Marktoberdorf volume [6].

Our aim in this tutorial presentation is to give a first indication of how Game Semantics can be developed in a new, algorithmic direction, with a view to applications in computer-assisted verification and program analysis. Some promising steps have already been taken in this direction. Hankin and Malacaria have applied Game Semantics to program analysis, e.g. to certifying secure information fl
in programs [25]. A particularly striking development was the work by Ghica
McCusker [15] which captures the game semantics of a fragment of Ideal
Algol in a remarkably simple form as regular expressions. This leads to a deci
procedure for observation equivalence on this fragment. Ghica has subseque
extended the approach to a call-by-value language with arrays [14], and to mo
checking Hoare-style program correctness assertions [13].

We believe the time is ripe for a systematic development of this algorith
approach to game semantics. Game Semantics has several features which m
it very promising from this point of view. It provides a very *concrete* way
building *fully abstract* models. It has a clear operational content, while admit
compositional methods in the style of denotational semantics. The basic obj
studied in Game Semantics are games, and strategies on games. Strategies ca
seen as certain kinds of highly-constrained processes, hence they admit the sa
kind of automata-theoretic representations central to model checking and al

paper.tex; 29/11/2001; 14:08; p

Game Semantics

Samson Abramsky
University of Edinburgh
Department of Computer Science
James Clerk Maxwell Building
Edinburgh EH9 3JZ
Scotland
email: samson@dcs.ed.ac.uk

Guy McCusker
St John's College
Oxford OX1 3JP
England
email: mccusker@comlab.ox.ac.uk

1 Introduction

The aim of this chapter is to give an introduction to some recent work on the application of game semantics to the study of programming languages.

An initial success for game semantics was its use in giving the first syntax-free descriptions of the fully abstract model for the functional programming language **PCF** [1,14,31].

One goal of semantics is to characterize the "universe of discourse" implicit in a programming language or a logic. Thus for a typed, higher-order functional programming language such as **PCF**, one may try to characterize "what it is to be a **PCF**-definable functional". Well established domain-theoretic models [12,35] provide sufficiently rich universes of functionals to interpret languages such as **PCF**, but in fact they are *too* rich; they include functionals, even "finitary" ones (defined over the booleans, say), which are *not* definable in **PCF**. Moreover, by a remarkable recent result of Ralph Loader [25], this is not an accident; this result (technically the undecidability of observation equivalence on finitary **PCF**) implies that *no* effective characterization of which functionals are definable in **PCF** (even in finitary **PCF**) can exist. Thus in particular a model containing all and only the **PCF**-definable functionals cannot be effectively presentable.

However, rather than focussing on the functionals *in extenso*, we may instead seek to characterize those *computational processes* which arise in computing the functionals. For a sequential, deterministic language such as **PCF** (and most func-