ICCL Summer School 2004

Game Semantics and its Applications

Luke Ong

Oxford University Computing Laboratory

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

Outline of the Course: Five Parts

1. Introduction and Overview
2. Categories of Games and Strategies
3. PCF and the Full Abstraction Problem
4. Modelling Higher-order Procedural Languages: Idealized Algol
5. Algorithmic Game Semantics and Software Model Checking

Course material

Slides are fairly detailed, contain full definitions, examples and key theorems.

Pointers to proofs in the literature. Bibliography.

Open problems and research directions.

[Detailed LaTeX-ed lecture notes from my home page – in the fullness of time!]

1 Introduction and Overview

Outline of Part 1

1. Background: precursors, a bit of history.
2. What is game semantics? An informal treatment.
3. Overview of game semantics by examples, and a quick survey.
4. Comparison with standard semantic styles.

Precursors of Game Semantics

Logic

- Higher-Type Recursion Theory: Is there a canonical notion of sequential computation at higher types? - A higher-order analogue of Church's Thesis.

  Kleene [Kle59, Kle78], Platek [Pla66], Gandy + Pani [Gan67]

- Proof Theory: Dialogue Games for Intuitionistic Logic by Lorenz and Lorenzen, see Felscher's survey paper [Fel86]. No interpretaiton of cut - aim was to characterize provability.

Computer Science

Research motivated by the Full Abstraction Problem for PCF: Sequential Algorithms on Concrete Data Structures, Kahn and Plotkin [KP93], Berry and Curien [BC82]

Game Theory

(as studied by sociol scientists including economists). The formal study of conflict and cooperation between goal-oriented agents.

Game Semantics for Proofs and Computation: Brief Historical Notes

Cambridge-Imperial Peripatetic Seminars on Game Semantics 1992-1993

Game models of MLL (Multiplicative Linear Logic) proofs

Aim: Find good model of proofs (not just provability). Blass' game model for MLL [Bla92] not associative.

- Abramsky + Jagadeesan: Fully complete model of MLL + MIX (1992) [AJ94]. Definition of composition of strategies and full completeness.
- Hyland + Ong: Fair games and fully complete model of MLL (without MIX) [HO93]

First syntax-independent fully abstract model for PCF (July 1993)

- Abramsky, Jagadeesan + Malacaria [AJM94, AJM00]: via history-free strategies which are models of linear logic. "AJM-games"
- Hyland + Ong [HO95, HO00]: direct construction of a cartesian closed category of innocent strategies. (Further details in Parts 2 and 3.) "HO-games"
- Nickau [Nic96].

Outline of Part 1

1. Background: precursors, a bit of history
2. **What is game semantics? An informal treatment.**
3. Overview of game semantics by examples.
4. Comparison with standard semantic styles.

Game Theory: Basic Notions

A game is played by n players, where n is usually, but can be greater than, 2.

Players are rational, goal-directed agents.

Various kinds of goals. E.g.

- winning (e.g. when modelling proofs)
- optimizing certain quantity, subject to constraints
- "non-competitive" - aim is just to complete (when modelling computation).

Strategies are methods to achieve these goals.

We consider perfect-information games, and strategies in extensive form.

We use games and strategies to construct models of programs. Thus seek ways of constructing games, and classes of strategies that compose w.r.t. these constructions. We study what happens during a play - the dynamics, not so much the outcome.

Game Semantics: Perspectives

Game semantics is a way to giving meanings to computation (and to proofs) using simple and intuitively ideas of game playing.

Two players:

P "Proponent" (or "Player")
O "Opponent"

(Terminology from Logic: P asserts a thesis, O seeks to demolish it.)

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.

Game Semantics: Basic Ingredients

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

Types are modelled by arenas (or games).

The game semantics of a term (P) is given in terms of its potential interaction with its context (O), i.e. specified by P's actions in response to all possible actions by O. Thus

- terms $M : A$ are modelled by P-strategies
- program contexts $X : A \vdash C[X] : \mathtt{nat}$ are modelled by O-strategies

for playing in the arena denoting type A.

An evaluation of a term (P) in a given program context (O) corresponds to a play between P and O in the arena.

A play is a sequence of moves satisfying certain rules, tracing out a dialogue of questions and answers between the two players.

Outline of Part 1

1. Background: precursors, a bit of history
2. What is game semantics? An informal treatment.
3. **Overview of game semantics by examples.**
   - Ground-type values
   - First-order functions
   - Higher-order programs
   - Multiple and nested use of arguments
   - Are pointers necessary?
   - A survey
4. Comparison with standard semantic styles.

Modelling values (ground type)

Natural numbers (0, 1, 2, etc.) can be modelled by simple interactions.

Shorthand:

OQ = O-question	PQ = P-question
OA = O-answer	PA = P-answer

Example The value 1 is denoted by the trivial P-strategy:

- OQ: "What is the number?"
- PA: "The number is 1".

Note: The decomposition of the "atomic" value 1 is a key step.

Example: A first-order function

To model a first-order function, we have:

	Input	Output
System (P)	*consumes*	*produces*
Environment (O)	*produces*	*consumes*

Example. A typical interaction of the successor function:

$$\mathtt{succ} : \mathbb{N} \to \mathbb{N}$$

1. OQ: "What is the output of this function?"
2. PQ: "What is the input to this function?"
3. OA: "The input is 5."
4. PA: "The output is 6."

This is a play resulting from P playing succ versus O playing $C[\,] = [\,]5$, which corresponds to the evaluation of $C[\mathtt{succ}]$.

Modelling higher-order computation

When confronted with a question, a player may answer it directly, or he may respond with a subsidiary question (usually with the intention of answering it eventually).

Example. Oral examination in a doctoral thesis defense.

Principles of Civil Conversation

1. Justification:
   - A question is asked only if the dialogue warrants it at that point.
   - An answer is proferred only if a question expecting it is pending.
2. Priority: "Last asked first answered."

   Equivalent to Well-Bracketing, and to Gandy's "No dangling question mark" condition.

Outcome of play:

- No question of winning (we model computatons here, not proofs).
- Dialogue ends when the opening question is answered.

Example: higher-order program

The question-answer dialogue reading extends to higher-order programs. Take

$$f : \mathbb{N} \to \mathbb{N} \vdash \text{if } (f(5) = 6) \text{ then } 7 \text{ else } 0 : \mathbb{N}$$

1. OQ: "What is the value (of type `nat`) of this program?"
2. PQ: "What is the output of the function argument f?"
3. OQ: "What is the input to f."
4. PA: "The input to f is 5."
5. OA: "The output of f is 6." (Suppose O interprets f as the successor function.)
6. PA: "The value of the program is 7."

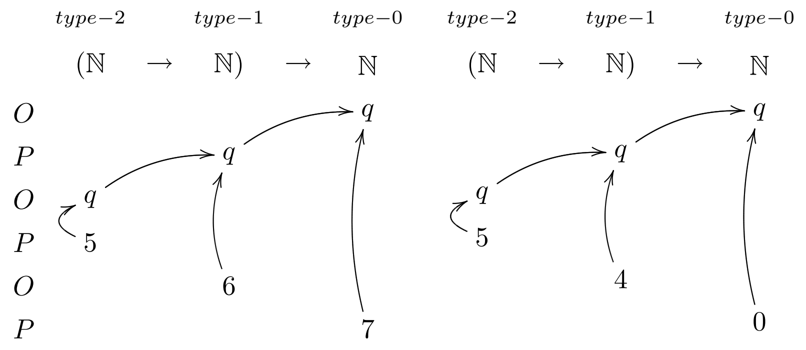
Observations

A play is opened by an OQ; thereafter it alternates between P and O.

Each non-opening move (whether question or answer) is made because it is warranted i.e. there is a justification for it.

Thus move 2 is justified (or prompted by) 1, 3 by 2, 4 by 3, 5 by 2, and 6 by 1).

All the moves for higher-type functions are built from the simple ones as for basic data types.



O plays successor O plays predecessor

How to read the picture. Time flows downwards.

Moves are written in the same colomn as the part of the type to which they correspond.

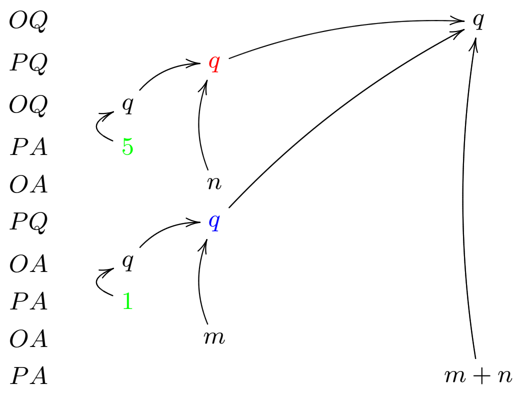
$\left\{ \begin{array}{l} \text{O} \\ \text{P} \end{array} \right.$ questions are in $\left\{ \begin{array}{l} \text{positive} \\ \text{negative} \end{array} \right.$ occurrences of $\mathbb{N}$.

The justification of a move is indicated by a pointer.

Multiple use of arguments

Programs may use their arguments more than once:

$$f : \mathbb{N} \to \mathbb{N} \vdash f(5) + f(1) : \mathbb{N}$$

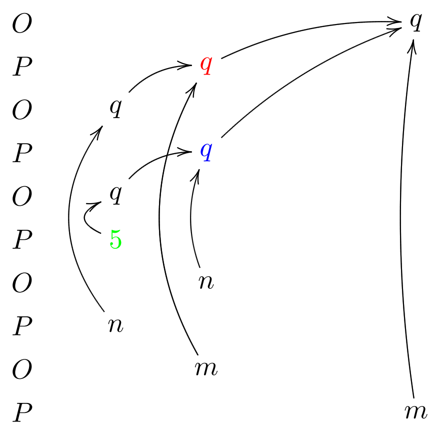


Observations. (i) Although both arguments of $+$ are needed, the algorithm evaluates the left argument first. (ii) P-questions (resp. P-answers) correspond to variables (resp. constants) occurring in the program.

Nested use of arguments

Programs may nest uses of their arguments:

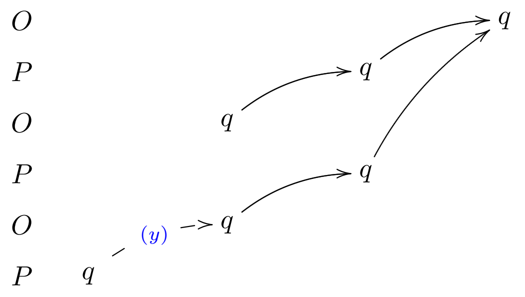
$$f : \mathbb{N} \to \mathbb{N} \vdash f(f(5)) : \mathbb{N}$$



Are pointers really necessary? (Yes, at orders higher than 2.)

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

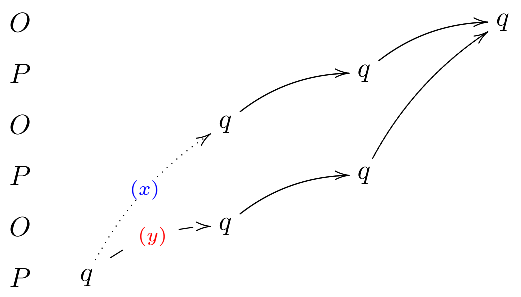
$$\begin{array}{ccccccc} type{-}3 & & type{-}2 & & type{-}1 & & type{-}0 \\ ((\mathbb{N} & \to & \mathbb{N}) & \to & \mathbb{N}) & \to & \mathbb{N} \end{array}$$



Are pointer really necessary? (Yes, at orders higher than 2.)

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

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



The plays played out by the two terms have the same underlying move-sequence, but are distinct *justified sequences*.

A Survey of Game Semantics (of Computation)

Fully abstract (or universal) game models have been constructed for:

Recursively defined types: FPC [McC98]

Higher-order Procedural Languages: Idealized Algol [AM97]

Non-local control: “PCF + catch” [Lai97]

Call-by-value games: [HY99, AM98]

Pure untyped λ-calculus: characterizing Bøhm- and Levy-Longo tree theories [DGFH99, DG01, KNO02, KNO03, ODG04]

General reference: IA + references [AHM98]

Polymorphism: [Hug97, MO01] etc.

Concurrency: Concurrent Idealized Algol [GM04]; Syntactic Control of Interference [GMO04]; Erratic choice [HM99]; Probabilistic choice [DH00], etc.

Dynamically generated local names: ν-calculus [AGM$^+$04]

Note: This is a very incomplete list.

Outline of Part 1

1. Background: precursors, a bit of history
2. What is game semantics? An informal treatment.
3. Overview of game semantics by examples.
4. **Comparison with standard semantic styles.**

What kind of semantics is game semantics?

Operational semantics

- Game semantics has clear algorithmic content but given independently of syntax.
- Game semantics admits concrete, automata-theoretic representations yet gives rise to a mathematically well-structured semantic universe.
- Strategies are a form of highly constrainted processes.

Denotational semantics

- Game semantics is a form of denotational semantics in that it supports compositional reasoning, but not based on some “abstract” category of domains and functions.
- Game semantics is both more abstract (in the technical sense that it gives rise to fully abstract models) and more concrete than traditional denotational semantics.
- Intensional / extensional: misleading beyond the functional paradigm. Rather it gives a dynamical (or interactional) view of computation.

2 Categories of Games and Strategies

Outline of Part 2

1. Arenas
2. Plays, Views and Strategies
3. Uncovering and Composition
4. Four Categories of Games
5. Intrinsic Preorder

Arenas

An arena is a description of a game, specifying its moves and the justification relation between moves.

Definition. An arena is a triple $A = \langle\, M_A, \lambda_A, \vdash_A \,\rangle$ where M_A is a set of moves,

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

is a labelling function, and $\vdash_A \subseteq (M_A \cup \{\, * \,\}) \times M_A$, where $* \notin M_A$, is called the justification relation satisfying: let m and m' range over M_A

1. For each $m \in M_A$ there is a unique $x \in M_A \cup \{\, * \,\}$ such that $x \vdash_A m$; in case $* \vdash_A m$, we call m an initial move.
2. Every initial move is an O-question.
3. If $m \vdash_A m'$ then m and m' are moves by different players.
4. If $m \vdash_A m'$ then m is a question.

Arenas

Note.

An arena is just a forest (where M_A and $\vdash_A$ are the node- and edge-sets respectively) whose nodes are labelled by λ_A satisfying the conditions 2, 3 and 4.

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 (sometimes written nat or ι):

$$\langle \underbrace{\{\, q \,\} \cup \mathbb{N}}_{\text{move}-\text{set}}, \underbrace{\{\, (*, q) \,\} \cup \{\, (q, n) : n \in \mathbb{N} \,\}}_{\text{justification relation}}, \underbrace{\{\, (q, OQ) \,\} \cup \{\, (n, PA) : n \in \mathbb{N} \,\}}_{\text{labelling function}} \rangle$$

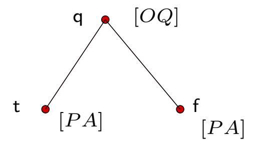
$$\begin{tikzcd}
OQ & & & q \arrow[dlll] \arrow[dll] \arrow[d] \arrow[dr] & \\
PA & 0 & 1 & 2 & \cdots
\end{tikzcd}$$

Boolean arena (sometimes written bool or o):

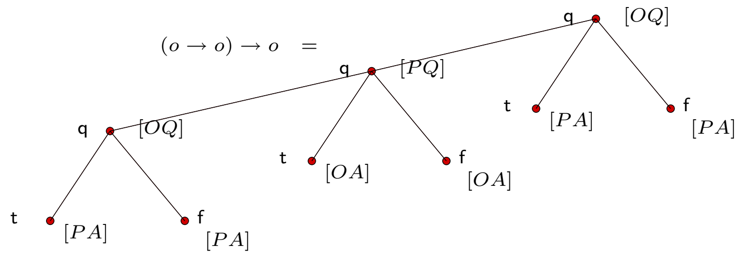
$$\begin{tikzcd}
OQ & & q \arrow[dl] \arrow[dr] & \\
PA & \mathtt{tt} & & \mathtt{ff}
\end{tikzcd}$$

More example arenas

$o \;=$



$(o \to o) \to o \;=$



Arena constructions

The product arena $A \times B$ is just the disjoint union of A and B (*qua* labelled graphs).

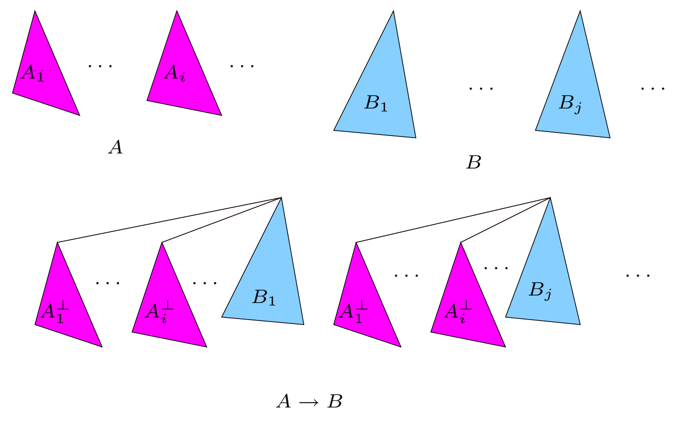
Formally we have

$$\begin{array}{rcl} M_{A\times B} & = & M_A + M_B \\ \lambda_{A\times B} & = & [\lambda_A, \lambda_B] \\ * \vdash_{A\times B} m & \iff & * \vdash_A m \ \vee\ * \vdash_B m \\ m \vdash_{A\times B} n & \iff & m \vdash_A n \ \vee\ m \vdash_B n. \end{array}$$

Function space arena $A \longrightarrow B$

Informally, to construct $A \longrightarrow B$ from arenas A and B:

"First invert the P and O moves of A – call it $A^\perp$, then graft one such copy of $A^\perp$ just below every root of B."



Function space arena $A \longrightarrow B$

Formally we have

$$\begin{array}{rcl} M_{A\to B} & = & (M_A \times M_B^{\mathrm{Init}}) + M_B \\ \lambda_{A\to B} & = & [\overline{\pi_1 \,;\, \lambda_A}, \lambda_B] \end{array}$$

writing M_B^{Init} for the set of initial moves of B, where $\overline{PQ}, \overline{PA}, \overline{OQ}, \overline{OA}$ are defined to be OQ, OA, PQ, PA respectively, and $\vdash_{A\to B}$ is defined by:

$$\begin{array}{rcl} * \vdash_{A\to B} b & \iff & * \vdash_B b \\ b \vdash_{A\to B} (a, b') & \iff & b = b' \ \wedge\ * \vdash_A a \\ (a, b) \vdash_{A\to B} (a', b') & \iff & b = b' \ \wedge\ a \vdash_A a' \\ b \vdash_{A\to B} b' & \iff & b \vdash_B b'. \end{array}$$

PCF-type arenas

In this course we shall only concern ourselves with arenas given by PCF types:

$$A \ ::= \ \mathsf{bool} \mid \mathsf{nat} \mid A \longrightarrow A$$

PCF-type arenas are trees: they have unique roots.

- 1-1 correspondence between $\left\{ \begin{array}{l} \text{positive} \\ \text{negative} \end{array} \right.$ occurrences of base types in A and $\left\{ \begin{array}{l} \text{O-} \\ \text{P-} \end{array} \right.$ questions in arenas determined by A.
- to each question-occurrence is associated a set of answers which are values of the corresponding base type.

Order of a type: measures the complexity on the lhs of $\longrightarrow$

$$\begin{array}{l} \mathsf{order}(\mathsf{bool}) = \mathsf{order}(\mathsf{nat}) = 0 \\ \mathsf{order}(A \longrightarrow B) = \max\left(\mathsf{order}(A) + 1, \mathsf{order}(B)\right) \end{array}$$

E.g. $((\mathsf{bool} \longrightarrow \mathsf{bool}) \longrightarrow \mathsf{bool}) \longrightarrow (\mathsf{bool} \longrightarrow \mathsf{bool}) \longrightarrow \mathsf{bool}$.

Outline of Part 2

1. Arenas
2. **Plays, Views and Strategies**
3. Uncovering and Composition
4. Four Categories of Games and Strategies
5. Intrinsic Preorder

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:

1. Alternation. O starts, thereafter P and O alternate.
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. These considerations motivate the definition of justified sequence.

Justified sequence

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

$$\begin{tikzcd}
s & = & a & b \arrow[l, bend right] & c \arrow[l, bend right] & d \arrow[l, bend right] & e \arrow[lll, bend right] & f \arrow[lllll, bend right] & g \arrow[l, bend right] & h \arrow[l, bend right] & i \arrow[lll, bend right] \\
 & & O & P & O & P & O & P & O & P & O
\end{tikzcd}$$

If there is a pointer from a move(-occurrence) m to some early move(-occurrence) m' in a justified sequence, we say that m is explicitly justified by m'.

P-views and O-views

Intuitively 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. We define:

$\ulcorner s\, m \urcorner$	$=$	$\ulcorner s \urcorner m$	if m is a P-move
$\ulcorner m \urcorner$	$=$	m	if m is initial
$\ulcorner s\, m_0\, u\, m \urcorner$	$=$	$\ulcorner s \urcorner m_0\, m$	if the O-move m is explicitly justified by m_0

In $\ulcorner s\, m_0\, u\, m \urcorner$ the pointer from m to m_0 is retained, similarly for the pointer from m in $\ulcorner s\, m \urcorner$ in case m is a P-move. (Similarly for O-view.)

Example

$$\begin{tikzcd}
s & = & a & b \arrow[l, bend right] & c \arrow[l, bend right] & d \arrow[l, bend right] & e \arrow[lll, bend right] & f \arrow[lllll, bend right] & g \arrow[l, bend right] & h \arrow[l, bend right] & i \arrow[lll, bend right] \\
 & & O & P & O & P & O & P & O & P & O
\end{tikzcd}$$

$$\begin{tikzcd}
\ulcorner s \urcorner & = & a & b \arrow[l, bend right] & e \arrow[l, bend right] & f \arrow[lll, bend right] & i \arrow[l, bend right]
\end{tikzcd}$$

Plays

Definition. A play (over A) is a justified sequence satisfying

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

Write P_A for the set of plays over A.

Example

$$\begin{tikzcd}
s & = & a & b \arrow[l, bend right] & c \arrow[l, bend right] & d \arrow[l, bend right] & e \arrow[lll, bend right] & f \arrow[lllll, bend right] & g \arrow[l, bend right] & h \arrow[lllll, bend right] \\
 & & O & P & O & P & O & P & O & P
\end{tikzcd}$$

violates Visibility.

Lemma. The P-view and O-view of a play satisfy Visibility. $\square$

Thanks to Visibility, the pointer from a P-move (resp. O-move) can be encoded as a numeric offset relative to the P-view (resp. O-view) at that point.

Strategies

A strategy is a rule that tells a player which move he should make at any given position where he is play. Formally a P-strategy σ (over A) is a non-empty prefix-closed set of plays satisfying: for any s, a and b

(i) (Determinacy). If even-length $s\,a, s\,b \in \sigma$, then $a = b$ (as justified move).

(ii) If even-length $s \in \sigma$ and $s\,a$ is a play, then $s\,a \in \sigma$.

We write $\sigma : A$ to mean σ is a strategy over arena A.

Example. Identity strategy $\mathsf{id}_A : A \to A$

$$\mathsf{id}_A \;=\; \{\, s : P_{A \to A} : \forall u \leq^{\text{even}} s \,.\, u \upharpoonright A^- = u \upharpoonright A^+ \,\}$$

$u \leq^{\text{even}} s$ means "u is an even-prefix of s".

Write $u \upharpoonright A^-$ for the subsequence of u consisting of moves (with pointers) from (the copies of) the A on the left of $\to$.

Outline of Part 2

Composing strategies

Given strategies $\sigma : A \to B$ and $\tau : B \to C$, construct a new strategy $\sigma\,;\tau : A \to C$ by composition:

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

Cutting sequents:

$$\frac{A \vdash B \qquad B \vdash C}{A \vdash C}$$

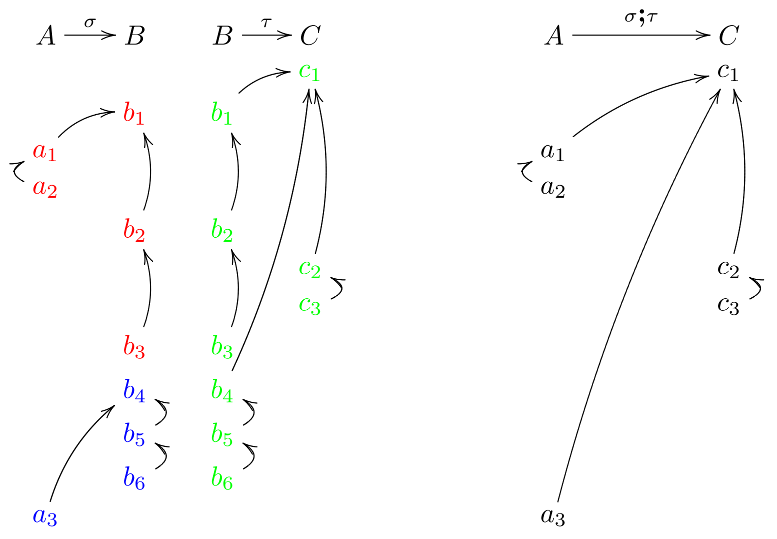
Substituting terms:

$$\frac{x : A \vdash M : B \qquad y : B \vdash N : C}{x : A \vdash N[M/y]C}$$

Curry-Howard Correspondence

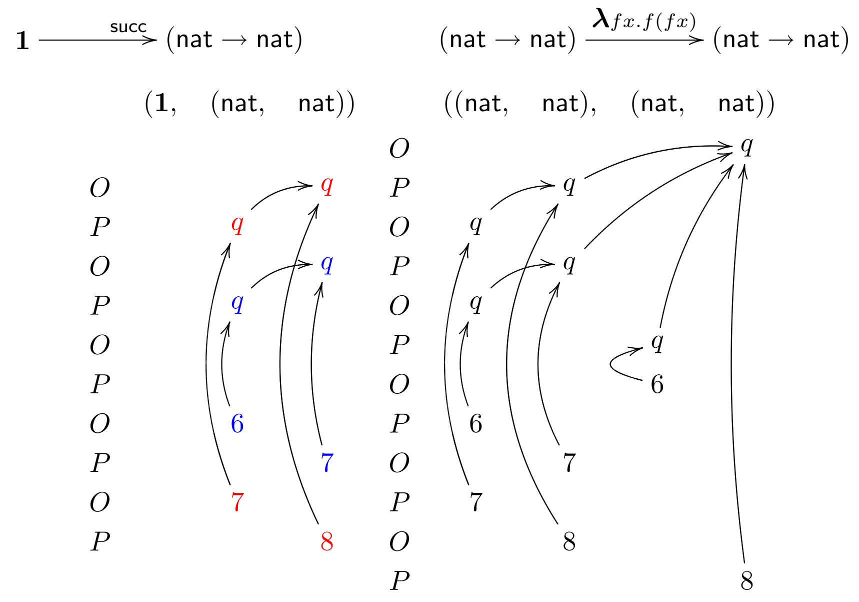
"Composing (strategies) = Cutting (sequents) = Substituting (terms)"

“Composition = CSP-style parallel composition + hiding”


$A \xrightarrow{\sigma} B$
$B \xrightarrow{\tau} C$
$A \xrightarrow{\sigma;\tau} C$
a_1 a_2 a_3
b_1 b_2 b_3 b_4 b_5 b_6
c_1 c_2 c_3

Take $u = c_1\, b_1\, a_1\, a_2\, b_2\, c_2\, c_3\, b_3\, b_4\, b_5\, b_6\, a_3$. We have $u \upharpoonright (B, C)$ (green) is in τ, $u \upharpoonright (A, B, b_1)$ (red) and $u \upharpoonright (A, B, b_4)$ (blue) are in σ

Example


$\mathbf{1} \xrightarrow{\text{succ}} (\text{nat} \to \text{nat})$
$(\text{nat} \to \text{nat}) \xrightarrow{\boldsymbol{\lambda} fx.f(fx)} (\text{nat} \to \text{nat})$
$(\mathbf{1}, \quad (\text{nat}, \quad \text{nat}))$
$((\text{nat}, \quad \text{nat}), \quad (\text{nat}, \quad \text{nat}))$
O P O P O P O P O P
O P O P O P O P O P O P
q q q q 6 7 7 8
q q q q q q q 6 6 7 7 8 8

Composition of strategies 1: Uncovering

Informally $\mathbf{u}(s, \sigma, \tau)$ is the longest justified sequence that is an appropriate interleaving of a play of $B \to C$ (from τ) and a finite number of plays of $A \to B$ (from σ), such that its projection on $A \to C$ is (a prefix of) s.

Suppose $\sigma : A \to B$ and $\tau : B \to C$. Take $s \in P_{A \to C}$. Define the uncovering of s in accord with σ and τ, written $\mathbf{u}(s, \sigma, \tau)$, to be the unique maximal justified sequence of moves of A, B and C satisfying:

$$\begin{array}{rcl} u \upharpoonright (A, C) & \leq & s \\ u \upharpoonright (B, C) & \in & \tau \\ u \upharpoonright (A, B, b) & \in & \sigma \end{array}$$

for each occurrence b of initial B-move in u.

Definition The composite $\sigma\,;\tau = \{\, \mathbf{u}(s, \sigma, \tau) \upharpoonright (A, C) : s \in P_{A \to C} \,\}$.

Composition = “CSP-style parallel composition plus hiding”

Composition of strategies 2

$u \upharpoonright (B, C)$ is the subsequence of u consisting of moves (with pointers) from B or C.

$u \upharpoonright (A, B, b)$ is the subsequence of u consisting of moves (with pointers) from A or B that are hereditarily justified by b.

$u \upharpoonright (A, C)$ is the subsequence of u consisting of moves (with pointers) from A or C, except that initial A-moves are to point to the opening C-move.

Exercise. Prove that for any $\sigma : A \to B$, $\text{id}_A\,; \sigma = \sigma = \sigma\,; \text{id}_B$.

Theorem. Composition is associative. I.e. for any $\sigma : A \to B, \tau : B \to C$ and $\rho : C \to D$, we have $(\sigma\,;\tau)\,;\rho = \sigma\,;(\tau\,;\rho)$.

Innocent strategies

A strategy σ is innocent just if for any s, t, a and b, if even-length $s\,a\,b \in \sigma$ and odd-length $t\,a \in \sigma$ and $\ulcorner s\,a \urcorner = \ulcorner t\,a \urcorner$, then $t\,a\,b \in \sigma$ such that (the pointer from b is governed by) $\ulcorner s\,a\,b \urcorner = \ulcorner t\,a\,b \urcorner$.

Equivalently an innocent strategy σ is generated by a view function f from odd-length P-views to justified P-moves in the sense that for any odd-length $s \in \sigma$, we have $s\,a \in \sigma$ iff $f(\ulcorner s \urcorner) = a$.

We define the view function of σ to be the least generating view function i.e. partial function f from P-views to justified P-moves satisfying

$$f(v) = b \iff \exists s, a \,.\, s\,a\,b \in \sigma \wedge \ulcorner s\,a \urcorner = v$$

history-sensitive – depends on the entire history

history-free – depends only on the last move

Innocent strategies are neither history-free nor history-sensitive, but view-dependent.

Example: View Function

$$[\![\, \boldsymbol{\lambda} fx.f(fx) \,]\!] : (\mathbb{N}^{\langle 11 \rangle} \to \mathbb{N}^{\langle 1 \rangle}) \to (\mathbb{N}^{\langle 2 \rangle} \to \mathbb{N})$$

is generated by the view function:

$$\left\{ \begin{array}{rcl} \text{Odd-length P-views} & & \text{Justified P-moves} \\ q & \mapsto & (q^{\langle 1 \rangle}, 0) \\ q\,q^{\langle 1 \rangle}\,q^{\langle 11 \rangle} & \mapsto & (q^{\langle 1 \rangle}, 1) \\ n \geq 0, \quad q\,q^{\langle 1 \rangle}\,n^{\langle 1 \rangle} & \mapsto & n \\ q\,q^{\langle 1 \rangle}\,q^{\langle 11 \rangle}\,q^{\langle 1 \rangle}\,q^{\langle 11 \rangle} & \mapsto & (q^{\langle 2 \rangle}, 2) \\ n \geq 0, \quad q\,q^{\langle 1 \rangle}\,q^{\langle 11 \rangle}\,q^{\langle 1 \rangle}\,n^{\langle 1 \rangle} & \mapsto & n^{\langle 11 \rangle} \\ n \geq 0, \quad q\,q^{\langle 1 \rangle}\,q^{\langle 11 \rangle}\,q^{\langle 1 \rangle}\,q^{\langle 11 \rangle}\,q^{\langle 2 \rangle}\,n^{\langle 2 \rangle} & \mapsto & n^{\langle 11 \rangle} \end{array} \right.$$

P-view offset. The offset n is the number of O-questions strictly between the source and target of the pointer.

Well-Bracketed Strategies

A strategy is well-bracketed just if whenever P makes an answer move, it is explicitly justified by the last pending question (i.e. the answer move is in answer to the most recent unanswered question).

Writing questions as open and answers as close brackets, the condition is exactly the standard well-bracketting condition.

Exercise. The identity strategy is both innocent and well-bracketed.

Outline of Part 2

1. Arenas
2. Plays, Views and Strategies
3. Uncovering and Composition
4. **Four Categories of Games and Strategies**
5. Intrinsic Preorder

Four categories of arenas and strategies

We define four categories $\mathbb{G}, \mathbb{G}_i, \mathbb{G}_b$ and $\mathbb{G}_{ib}$. All four have arenas as objects.

- $\mathbb{G}$-maps from A to B are strategies over $A \to B$.
- $\mathbb{G}_i$-maps from A to B are innocent strategies over $A \to B$.
- $\mathbb{G}_b$-maps from A to B are well-bracketed strategies over $A \to B$.
- $\mathbb{G}_{ib}$-maps from A to B are innocent and well-bracketed strategies over $A \to B$.

To prove that $\mathbb{G}_{ib}$ is a category, we need to show that strategy composition preserve well-bracketing and innocence.

Theorem. If $\sigma : A \to B$ and $\tau : B \to C$ are innocent (and well-bracketed), so is $\sigma\,;\tau : A \to C$.

To prove it, given view functions f and g that generate σ and τ respectively, we need to construct a view function h of type $A \to C$ that generates $\sigma\,;\tau$. See [HO00] for a proof.

Definition: Category

A category $\mathbb{C}$ is given by a set of objects, and for A, B, C, D ranging over objects:

- a set $\mathbb{C}(A, B)$ of maps
- an identity map $\mathrm{id}_A \in \mathbb{C}(A, A)$
- a composition function $; : \mathbb{C}(A, B) \times \mathbb{C}(B, C) \longrightarrow \mathbb{C}(A, C)$

such that for each $f \in \mathbb{C}(A, B), g \in \mathbb{C}(B, C)$ and $h \in \mathbb{C}(C, D)$, we have

Identity	$\mathrm{id}_A\,; f = f = f\,; \mathrm{id}_B$
Associativity	$(f\,; g)\,; h = f\,; (g\,; h).$

Definition: Cartesian closed category

An object A is terminal just in case for any C, there is a unique map $C \longrightarrow A$.

A category has binary products if for any objects A and B, there are projection maps $\pi_1 : A \times B \longrightarrow A$ and $\pi_1 : A \times B \longrightarrow B$ such that for any $f : C \longrightarrow A$ and $g : C \longrightarrow B$, there is a unique map $h : C \longrightarrow A \times B$ satisfying

$$\begin{tikzcd}
& C \arrow[dl, "f"'] \arrow[d, "h"] \arrow[dr, "g"] & \\
A & A \times B \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & B
\end{tikzcd}$$

A category has exponential if (in addition) for any A, B there is a map $\mathrm{ev}_{A,B} : (A \to B) \times A \longrightarrow B$ such that for any $f : C \times A \longrightarrow B$, there is a unique $\Lambda(f) : C \longrightarrow (A \to B)$ satisfying

$$\begin{tikzcd}
C \times A \arrow[r, "f"] \arrow[d, "\Lambda(f)\times\mathrm{id}_A"'] & B \\
(A \to B) \times A \arrow[ur, "\mathrm{ev}_{A,B}"'] &
\end{tikzcd}$$

A category is cartesian closed just if it has terminal object, binary products, and exponentials.

Four cartesian closed categories of arenas and strategies

Theorem. $\mathbb{G}$ is a CCC, with sub-CCCs $\mathbb{G}_i$ and $\mathbb{G}_b$; and $\mathbb{G}_{ib}$ is a sub-CCC of each of $\mathbb{G}, \mathbb{G}_i$ and $\mathbb{G}_b$.

Proof is now easy - the hard work has already been done in compositionality.

Terminal object is the empty arena.

Binary product is given by the arena product construction - the projection maps are copycat strategies.

Exponentiation is given by the function space arena construction. Note the underlying arena forests of $(C \times A) \to B$ and $C \to (A \to B)$ are isomorphic. The eval map ev$_{A,B}$: $(A \to B) \times A \longrightarrow B$ is a copycat strategy.

Outline of Part 2

1. Arenas
2. Plays, Views and Strategies
3. Uncovering and Composition
4. Four Categories of Games and Strategies
5. **Intrinsic Preorder**

Order-enrichment and compactness

Strategies for an arena A form a directed-complete CPO under inclusion, with lubs given by unions. Moreover composition, product and currying are continuous w.r.t. the order.

Lemma. $\mathbb{G}$ is a CPO-enriched CCC.

For any innocent $\sigma : A$, define the view function of σ to be the partial function f from P-views to justified P-moves satisfying

$$f(v) = b \iff \exists s, a \,.\, s\,a\,b \in \sigma \wedge \ulcorner s\,a \urcorner = v$$

Proposition. The innocent (and well-bracketed) strategies over an arena form a dI-domain. The compact elements are those whose view functions are finite.

Intrinsic preorder

Our PCF full abstraction result holds in $\mathbb{G}_{ib}$ quotiented by a preorder (defined intrinsically as opposed to syntactically).

Let Σ be the arena with a single O-question q and a P-answer a. There are only two strategies for Σ: $\bot = \{\,\epsilon, q\,\}$ and $\top = \{\,\epsilon, q, q\,a\,\}$.

The intrinsic preorder for strategies over A is defined by:

$$\sigma \lesssim \tau \iff \forall \gamma : A \longrightarrow \Sigma \,.\, \sigma\,;\gamma = \top \Longrightarrow \tau\,;\gamma = \top$$

Note that this defines four different preorder, one on each of $\mathbb{G}, \mathbb{G}_i, \mathbb{G}_b$; and $\mathbb{G}_{ib}$. E.g. we write $\lesssim_{ib}$ etc. to distinguish them.

Prop. For each of $\mathbb{G}, \mathbb{G}_i, \mathbb{G}_b$; and $\mathbb{G}_{ib}$, the relation $\lesssim$ defined above is a preorder on each hom-set and the quotient category is a poset-enriched CCC.

Open Problem. Are the quotiented hom-sets (of $\mathbb{G}_{ib}$) complete partial orders?

3 PCF and the Full Abstraction Problem

Outline of Part 3

1. Introduction to PCF: History, Syntax and Operational Semantics
2. The Full Abstraction Problem
3. Game semantics of PCF
4. Definability and Full Abstraction
5. PCF constraints: Determinacy, Well-Bracketing, Innocence and Visibility

Origins of PCF

PCF = "simply-typed λ-calculus + basic arithmetic + definition-by-cases + fixpoint operators"

Gødel's System T (primitive functionals of finite types) in Dialectica Interpretation – contribution to a liberalized version of Hilbert's Programme.

Kleene: full blown generalization of recursion theory to higher types. Over a total type structure, partial recursive functions not closed under substitution, and first recursion theorem fails.

Platek (thesis 66): recursion theory over hereditarily order-preserving partial functions over $\mathbb{N}$; studied a programming syntax essentially a precursor of PCF.

Scott: "A type-theoretical alternative to CUCH, ISWIM, OWHY" 69 (93) [Sco93]. LCF as logical calculus (or algebra) for computability using type theory; higher-order types used to study the first-order and ground types.

Plotkin introduced PCF as a programming language in [Plo77].

Types and terms of PCF

PCF types. $A ::= \mathsf{nat} \mid \mathsf{bool} \mid A \to A$

PCF terms.

n	:	nat	numerals
$\mathtt{tt}, \mathtt{ff}$	:	bool	booleans
succ	:	nat $\to$ nat	successor
pred	:	nat $\to$ nat	predecessor
zero?	:	nat $\to$ bool	test for zero
$\mathsf{cond}^{\mathsf{nat}}$	:	bool $\to$ nat $\to$ nat $\to$ nat	natural number conditional
$\mathsf{cond}^{\mathsf{bool}}$	:	bool $\to$ bool $\to$ bool $\to$ bool	boolean conditional.

$$\frac{s : A \to A}{\mathbf{Y}^A(s) : A} \qquad \frac{s : A \to B \quad t : A}{(s\,t) : B} \qquad \frac{s : B}{(\lambda x : A.s) : A \to B}$$

PCF term-in-context. $x_1 : A_1, \cdots, x_n : A_n \vdash M : B$

Operational semantics

Programs are closed terms of ground type. Values are abstractions and constants.

Evaluation relation. Call-by-name, normal order.

Read $s \Downarrow v$ as "closed term s evaluates to value v".

$$v \Downarrow v \qquad \frac{u[t/x] \Downarrow v}{(\lambda x.u)t \Downarrow v} \qquad \frac{s \Downarrow v \quad vt \Downarrow v'}{st \Downarrow v'} \qquad \frac{s\mathbf{Y}^A(s) \Downarrow v}{\mathbf{Y}^A(s) \Downarrow v}$$

$$\frac{s \Downarrow 0}{\mathsf{zero?}\, s \Downarrow \mathtt{tt}} \qquad \frac{s \Downarrow n+1}{\mathsf{zero?}\, s \Downarrow \mathtt{ff}} \qquad \frac{s \Downarrow \mathtt{tt} \quad u \Downarrow v}{\mathsf{cond}^\beta suu' \Downarrow v} \qquad \frac{s \Downarrow \mathtt{ff} \quad u' \Downarrow v}{\mathsf{cond}^\beta suu' \Downarrow v}$$

$$\frac{s \Downarrow n}{\mathsf{succ}\, s \Downarrow n+1} \qquad \frac{s \Downarrow n+1}{\mathsf{pred}\, s \Downarrow n} \qquad \frac{s \Downarrow 0}{\mathsf{pred}\, s \Downarrow 0}$$

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 any context $C[\,]$ such that $C[M]$ and $C[N]$ are programs, for any value V

$$C[M] \Downarrow V \iff C[N] \Downarrow V.$$

$\approx$ is an intuitively compelling notion of program equivalence, but very hard to reason about.

Examples

1. $(\boldsymbol{\lambda} x : A.M)N \approx M[N/x]$
2. $\mathbf{Y}^A(M) \approx M(\mathbf{Y}^A(M))$
3. Assuming $x \notin \mathsf{FV}(B)$

$$\boldsymbol{\lambda} x.\text{if } B \text{ then } M \text{ else } N \approx \text{if } B \text{ then } \boldsymbol{\lambda} x.M \text{ else } \boldsymbol{\lambda} x.N$$

What is a (denotational) model of PCF?

Two key ideas:

- Models of simply-typed λ-calculus are cartesian closed categories. (Lambek)
- Tarski-Knaster reading of fixpoints.

Thus models of PCF are CPO-enriched CCCs.

Interpret PCF types as objects: $[\![A \to B]\!] = [\![A]\!] \to [\![B]\!]$.

Interpret PCF terms-in-context $\Gamma \vdash M : B$, where $\Gamma = x_1 : A_1, \cdots, x_n : A_n$, as maps $[\![A_1]\!] \times \cdots \times [\![A_n]\!] \longrightarrow [\![B]\!]$.

$$\begin{array}{rcl} [\![\Gamma \vdash x_i : A_i]\!] & = & \pi_i : [\![A_1]\!] \times \cdots \times [\![A_n]\!] \longrightarrow [\![A_i]\!] \\ [\![\Gamma \vdash \boldsymbol{\lambda} x : A.M : B]\!] & = & \Lambda([\![\Gamma, x : A \vdash M : B]\!]) : [\![\Gamma]\!] \longrightarrow ([\![A]\!] \to [\![B]\!]) \\ [\![\Gamma \vdash MN : B]\!] & = & \langle [\![\Gamma \vdash M]\!], [\![\Gamma \vdash N]\!] \rangle \,; \mathsf{ev}_{A,B} : [\![\Gamma]\!] \longrightarrow [\![B]\!] \\ [\![\Gamma \vdash \mathbf{Y}^A(M) : A]\!] & = & \mathsf{lfp}([\![\Gamma \vdash M : A \to A]\!]) : [\![A]\!] \end{array}$$

For simplicity we shall often omit the semantic brackets and confuse $[\![A]\!]$ with A.

How accurate is the model?

Definition. Fix a model of PCF. We say that a model is (equationally) fully abstract just if for all M, N, if $\Gamma \vdash M, N : A$ then

$$[\![\Gamma \vdash M]\!] = [\![\Gamma \vdash N]\!] \iff M \approx N$$

Theorem. [Plo77] The Scott model (of domains and continuous functions) is not fully abstract for PCF (parallel-or is not definable), but it is for PCF + parallel-or.

p−or	$\bot$	$\mathtt{tt}$	$\mathtt{ff}$
$\bot$	$\bot$	$\mathtt{tt}$	$\bot$
$\mathtt{tt}$	$\mathtt{tt}$	$\mathtt{tt}$	$\mathtt{tt}$
$\mathtt{ff}$	$\bot$	$\mathtt{tt}$	$\mathtt{ff}$

Theorem. [Mil77] There is a unique continuous, order-extensional fully abstract model for PCF (obtained by a term model construction).

The Full Abstraction Problem for PCF

In view of Milner's result, what then is the Full Abstraction problem?

It is about the search for a "good" denotational model that is fully abstract.

Possible meaning of "good": syntax-independent, abstract.

A good model should yield an abstract, synthetic characterization of higher-type sequentially computable functions. (cf. Kleene's Problem).

Theorem. [HO00] $\mathbb{G}_{ib}$ gives rise to an order-extensional fully abstract model for PCF.

Outline of Part 3

1. Introduction to PCF: History, Syntax and Operational Semantics
2. The Full Abstraction Problem
3. **Game semantics of PCF**
4. Definability and Full Abstraction
5. PCF constraints: Determinacy, Well-Bracketing, Innocence and Visibility

Game semantics of PCF

All four categories $\mathbb{G}, \mathbb{G}_i, \mathbb{G}_b$ and $\mathbb{G}_{ib}$ are cartesian closed.

Interpreting PCF types. `nat` is (interpreted by) the flat arena

$$\begin{tikzcd}
OQ & & & q \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] & \\
PA & 0 & 1 & 2 & \cdots
\end{tikzcd}$$

Interpreting PCF constants. E.g. $[\![\, 1 \,]\!] = \{\, \epsilon, q, q\,1 \,\}$.

$[\![\, \mathsf{succ} \,]\!]$ is the prefix-closure of $\{\, q\, q^{\langle 1\rangle}\, n^{\langle 1\rangle}\, n+1 : n \in \mathbb{N} \,\}$.

$$\begin{tikzcd}
 & \mathsf{nat}^{\langle 1\rangle} & \longrightarrow & \mathsf{nat} \\
O & & & q \\
P & q^{\langle 1\rangle} \arrow[urr, bend left] & & \\
O & n^{\langle 1\rangle} \arrow[u, bend left] & & \\
P & & & n+1 \arrow[uuu, bend left]
\end{tikzcd}$$

$\mathsf{cond}^{\mathsf{nat}} : \mathsf{bool} \to \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$

$[\![\, \mathsf{cond}^{\mathsf{nat}} \,]\!] : \mathsf{bool} \to \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$ is the prefix-closure of

$$\{\, q\, q^{\langle 1\rangle}\, \mathtt{tt}^{\langle 1\rangle}\, q^{\langle 2\rangle}\, n^{\langle 2\rangle}\, n : n \in \mathbb{N} \,\} \cup \{\, q\, q^{\langle 1\rangle}\, \mathtt{ff}^{\langle 1\rangle}\, q^{\langle 3\rangle}\, n^{\langle 3\rangle}\, n : n \in \mathbb{N} \,\}.$$

$$\begin{tikzcd}
 & (\mathsf{bool}, & (\mathsf{nat}, & (\mathsf{nat}, & \mathsf{nat}))) & (\mathsf{bool}, & \mathsf{nat}, & \mathsf{nat}, & \mathsf{nat}))) \\
O & & & & q & & & & q \\
P & q^{\langle 1\rangle} \arrow[urrr, bend left] & & & & q^{\langle 1\rangle} \arrow[urrr, bend left] & & & \\
O & \mathtt{tt}^{\langle 1\rangle} \arrow[u] & & & & \mathtt{ff}^{\langle 1\rangle} \arrow[u] & & & \\
P & & q^{\langle 2\rangle} \arrow[uuurr, bend left] & & & & & q^{\langle 3\rangle} \arrow[uuur, bend left] & \\
O & & n^{\langle 2\rangle} \arrow[u] & & & & & m^{\langle 3\rangle} \arrow[u] & \\
P & & & & n \arrow[uuuuu, bend left] & & & & m \arrow[uuuuu, bend left]
\end{tikzcd}$$

Notation. Moves that are from the subarena corresponding to the i-th argument are annotated with superscript $\langle\, i \,\rangle$

View function of $[\![\, \mathsf{cond}^{\mathsf{nat}} \,]\!]$

$[\![\, \mathsf{cond}^{\mathsf{nat}} \,]\!]$ is innocent with view function:

$$\left\{ \begin{array}{rcll}
 & & & \text{view offset} \\
q & \mapsto & q^{\langle 1\rangle} & 0 \\
q\, q^{\langle 1\rangle}\, \mathtt{tt} & \mapsto & q^{\langle 2\rangle} & 0 \\
\text{each } n \geq 0, \quad q\, q^{\langle 1\rangle}\, \mathtt{tt}\, q^{\langle 2\rangle}\, n^{\langle 2\rangle} & \mapsto & n & 0 \\
q\, q^{\langle 1\rangle}\, \mathtt{ff} & \mapsto & q^{\langle 3\rangle} & 0 \\
\text{each } n \geq 0, \quad q\, q^{\langle 1\rangle}\, \mathtt{ff}\, q^{\langle 3\rangle}\, n^{\langle 3\rangle} & \mapsto & n & 0
\end{array} \right.$$

P-view offset. The offset n is the number of O-questions strictly between the source and target of the pointer.

Fact. All PCF constants are interpreted by strategies that are both well-bracketed and innocent. Thus all four categories admit the same model of PCF.

Definability

Theorem. Every compact well-bracketed and innocent strategy (or ib-strategy) $\sigma : A = (A_1, \cdots, A_n, \mathtt{nat})$ (i.e. generated by a finite view function f, say) is the denotation of some PCF-term $\boldsymbol{\lambda} x_1^{A_1} \cdots x_n^{A_n}.M_f$.

Proof. Three cases of f:

(1) f is empty (i.e. everywhere undefined), and $M_f = \Omega$

(2) f maps initial A-question q_0 to some answer, say, 5; $M_f = \boldsymbol{\lambda} x_1^{A_1} \cdots x_n^{A_n}.5$

(3) f maps initial A-question q_0 to initial A_i-question q'.

Suppose $A_i = (C_1, \cdots, C_m, \mathtt{nat})$ where $C_j = (D_{j1}, \cdots, D_{jr_j}, \mathtt{nat})$. Let q_j be the initial question of C_j.

For each $1 \leq j \leq m$, consider view function $g_j = \{\, (q''\,\alpha, m) : (q_0\, q'\, q_j\, \alpha, m) \in f \,\}$, which generates a ib-strategy of $(\overline{A}, D_{j1}, \cdots, D_{jr_j}, \mathtt{nat})$ with initial question q''. Since g_j is smaller than f, by the IH, the strategy is denoted by $\boldsymbol{\lambda} \overline{x} y_{j1} \cdots y_{jr_j}.M_{g_j}$.

Next, since f is finite, there are only finitely many numbers i such that f is defined on $q_0\, q'\, i$. Suppose these numbers are $n_1, \cdots, n_l$. Then the view function $h_k = \{\, (q_0\, \alpha, m) : (q_0\, q'\, n_k\, \alpha, m) \in f \,\}$ generates a ib-strategy of A. By the IH, the strategy is denoted by $\boldsymbol{\lambda} \overline{x}.M_{h_k}$.

Putting it together, σ is denoted by $\boldsymbol{\lambda} \overline{x}.M_f$ where

$$M_f = \mathsf{case}\, (x_i\, (\boldsymbol{\lambda} \overline{y_1}.M_{g_1}) \cdots (\boldsymbol{\lambda} \overline{y_m}.M_{g_m}))[n_1 : M_{h_1}] \cdots [n_l : M_{h_l}]$$

$\square$

Full abstraction

Theorem. For any closed PCF terms M and N, if $\Gamma \vdash M, N : A$ then

$$[\![\, \Gamma \vdash M \,]\!] \lesssim_{ib} [\![\, \Gamma \vdash N \,]\!] \iff M \sqsubseteq\!\!\!\sim N.$$

Proof. $\Rightarrow$: Suppose $[\![\, M \,]\!] \lesssim_{ib} [\![\, N \,]\!]$, and $C[M] \Downarrow$. $C[\,]$ determines a map $\gamma : A \longrightarrow \mathbb{N}$ such that $[\![\, C[P] \,]\!] = [\![\, P \,]\!] \,;\, \gamma$ for all closed P of type A. Since $C[M] \Downarrow$, it follows from Adequacy that $[\![\, M \,]\!] \,;\, \gamma \neq \bot$. Since $[\![\, M \,]\!] \lesssim_{ib} [\![\, N \,]\!]$, we have $[\![\, N \,]\!] \,;\, \gamma \neq \bot$.

$\Leftarrow$: Suppose $[\![\, M \,]\!] \not\lesssim_{ib} [\![\, N \,]\!]$. Then for some γ, $[\![\, M \,]\!] \,;\, \gamma \neq \bot$ and $[\![\, N \,]\!] \,;\, \gamma = \bot$. We can insist that γ be generated by a compact view function. By Definability, there is some PCF term-in-context $x : A \vdash C[x] : \mathtt{nat}$ whose denotation is γ. It follows from Adequacy that $C[M] \Downarrow$ and $C[N] \Uparrow$, as required. $\square$

Limitations on the game model for PCF

Game models must be quotiented to get the fully abstract model.

The quotient is defined “intrinsically” i.e. purely in terms of the model.

Nevertheless we understand the game model but not the quotient.

Can we do any better?

Theorem. [Loa01] Observational equivalence in Finitary PCF is undecidable. $\square$

Thus the above situation is the best that we can hope to do.

Outline of Part 3

1. Introduction to PCF: History, Syntax and Operational Semantics
2. The Full Abstraction Problem
3. Game semantics of PCF
4. Definability and Full Abstraction
5. **PCF constraints: Determinacy, Well-Bracketing, Innocence and Visibility**

Constraints that characterize PCF

An aim of semantics is to characterize the "universe of discourse" implicit in a programming language or a logic. E.g. "What does it mean to be a PCF-definable functional?"

Kleene's Question:

> Find the protocol governing a two-player dialogue game that characterizes higher-type sequentially computable functionals.

A modern paraphrase: Characterize the computational processes that arise in computing these functionals.

An inherent difficulty is to understand the precise interplay between extensional and intensional properties of these processes.

Game semantics offers a solution: Four constraints identified:

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

Classifying Programming Features

Strategies can be classified according to the behavioural constraints identified in the fully abstract game model for PCF: D + B + I + V.

Each constraint corresponds precisely to the absence (or presence) of certain programming features.

1. Determinacy: Erratic choice (Harmer + McCusker 1998)
2. Well-Bracketing: Non-local control operators. $\mathbb{G}_i$ gives rise to a fully abstract model for "PCF + control" (Laird 1998).
3. Innocence: Mutable store. $\mathbb{G}_b$ gives rise to a fully abstract model for "PCF + states" or Idealized Algol (Abramsky + McCusker 1996).
4. Visibility: Higher-order store (reference types) (Abramsky, Honda + McCusker 1999)

Related work: A π-calculus representation of the fully abstract model of PCF [HO95].

Subsequently Honda *et al* (TLCA'04 paper) have sought to give characterizations of functional computation within the setting of π-calculus.

Innocence (= view-dependence) illustrated

PCF functionals interact extensionally with their arguments

> **Lemma** (Curien). The monotone function $F : (\mathsf{bool} \rightarrow \mathsf{bool} \rightarrow \mathsf{bool}) \longrightarrow \mathsf{bool}$ satisfying
>
> $$F : \begin{cases} \mathsf{l\!-\!or} & \mapsto \quad \mathtt{tt} \\ \mathsf{r\!-\!or} & \mapsto \quad \mathtt{ff} \end{cases}$$
>
> is not PCF-definable. $\square$

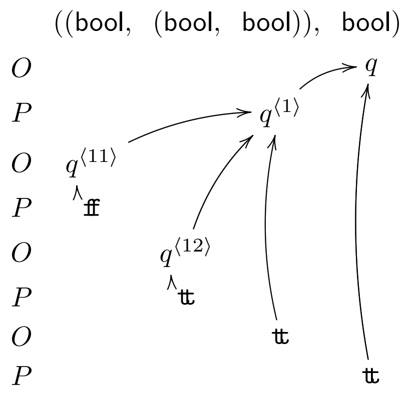
l−or	$\bot$	tt	ff
$\bot$	$\bot$	$\bot$	$\bot$
tt	tt	tt	tt
ff	$\bot$	tt	ff

r−or	$\bot$	tt	ff
$\bot$	$\bot$	tt	$\bot$
tt	$\bot$	tt	tt
ff	$\bot$	tt	ff

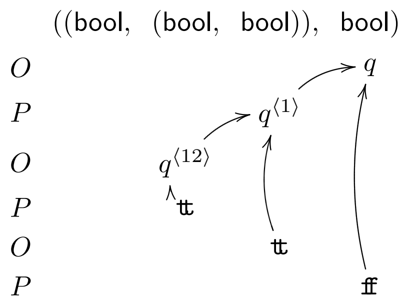
l−or is PCF-definable by $\lambda xy.\mathsf{if}\ x\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ y$; similarly for r−or.

F is not definable in $\mathbb{G}_{ib}$

Claim. Any strategy that computes F is not innocent. Suppose σ computes F.



O plays l−or



O plays r−or

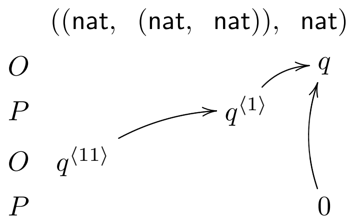
Now $\ulcorner q\,q^{\langle 1\rangle}\,q^{\langle 11\rangle}\,\mathtt{ff}\,q^{\langle 12\rangle}\,\mathtt{tt}\,\mathtt{tt}\urcorner = \ulcorner q\,q^{\langle 1\rangle}\,q^{\langle 12\rangle}\,\mathtt{tt}\,\mathtt{tt}\urcorner = q\,q^{\langle 1\rangle}\,\mathtt{tt}$. The P-view of the two plays are the same, hence σ's response must be the same.

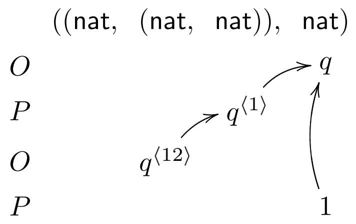
Well-Bracketing: Non-local control

No strategy satisfying Determinacy, Well-Bracketing and Innocence can compute catch : (nat → nat → nat) ⟶ nat satisfying

$$\text{catch } f = \begin{cases} 0 & f \text{ calls 1st argument first} \\ 1 & f \text{ calls 2nd argument first} \\ n+2 & f \text{ returns } n \text{ immediately} \end{cases}$$

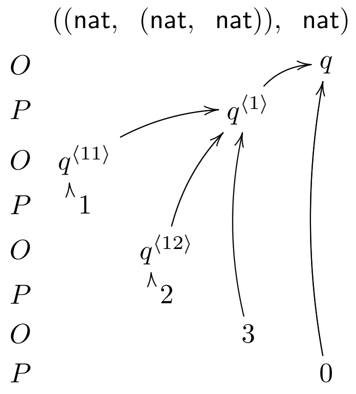
Suppose for a contradiction σ does.



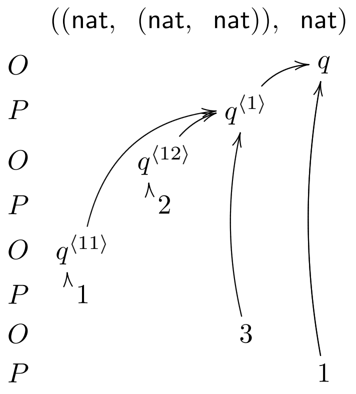


This violates the Bracketing condition.

Now suppose σ tries to observe Bracketing.



O plays $+_l$



O plays $+_r$

$\ulcorner q\,q^{\langle 1\rangle}\,q^{\langle 11\rangle}\,1\,q^{\langle 12\rangle}\,2\,3\urcorner = \ulcorner q\,q^{\langle 1\rangle}\,q^{\langle 12\rangle}\,2\,q^{\langle 11\rangle}\,1\,3\urcorner = q\,q^{\langle 1\rangle}\,3$. The P-view of the two plays are the same, hence any *innocent* strategy's response must be the same.

4 Modelling Higher-Order Procedural Languages

Outline of Part 4

1. Idealized Algol (IA): Syntax and Operational Semantics
2. Observational Equivalence
3. Game semantics of IA

Motivation

Recall: (i) The monotone function $F : (\mathsf{bool} \times \mathsf{bool} \to \mathsf{bool}) \longrightarrow \mathsf{bool}$ satisfying

$$F \ : \ \begin{cases} \mathsf{l{-}or} & \mapsto \ \mathtt{tt} \\ \mathsf{r{-}or} & \mapsto \ \mathtt{ff} \end{cases}$$

is not PCF-definable. (ii) No innocent strategy can compute F. □

- There are strategies in $\mathbb{G}_b$ that compute F.
- F is definable in "PCF + state" (e.g. ML) by

$$\boldsymbol{\lambda} f.\mathsf{new}\, l, r := 0, 0 \,\mathsf{in}\ \boxed{\mathtt{if}\ f\,(l^+\,\mathbf{;}\,\mathtt{tt})\,(r^+\,\mathbf{;}\,\mathtt{tt})\ \mathtt{then}\,(\mathsf{if}\ !r = 0\ \mathsf{then}\ \mathtt{tt}\ \mathsf{else}\ \mathtt{ff})}$$

where $!r$ is explicit dereferencing, l^+ is $l := !l + 1$, and $l^+\,\mathbf{;}\,\mathtt{tt}$ is an active expression.

Question Is "PCF + state" the programming counterpart of $\mathbb{G}_b$?

Idealized Algol (IA) [Reynolds 80]

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

IA Types

T	$::=$	int	numbers-valued expressions
	$\mid$	com	commands
	$\mid$	loc	locations (or assignable variables)
	$\mid$	$T \to T$	

IA Terms

PCF + imperative constructs + block-allocated local variables.

Imperative constructs of IA

1. Null command. skip : com
2. Command sequencing. seq : com $\to$ com $\to$ com and seq : com $\to$ int $\to$ int (expressions may have side-effects). Write seq $C\,C'$ as $C\,\mathbf{;}\,C'$.
3. Assignment. assign : loc $\to$ int $\to$ com
   Write "assign $M\,N$" as $M := N$.
4. Dereferencing (explicit in the syntax). deref : loc $\to$ int
   Write "deref M" as $!M$.
5. Block-allocated local (assignable) variables: $n \geq 0$, $\beta =$ com or int

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

   Declare a new assignable variable x, initialized to n, whose scope is M.

IA is expressive: using the fixpoint operator, iteration and recursively defined procedures are expressible.

Examples

1. "$x := 1\,\mathbf{;}\,x := !x + 1$" becomes

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

2. We can express the while-loop "while B do C" as

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

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

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

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

$$\frac{M, S \Downarrow \mathtt{mkloc}\, P\, Q, S' \quad P, S' \Downarrow n, S''}{!M, S \Downarrow n, S''} \qquad \frac{N, S \Downarrow n, S' \quad M, S' \Downarrow l, S''}{M := N, S \Downarrow \mathsf{skip}, S''[l \mapsto n]}$$

$$\frac{N, S \Downarrow n, S' \qquad M, S' \Downarrow \mathtt{mkloc}\, P\, Q, S'' \qquad Q\, n, S'' \Downarrow \mathsf{skip}, S'''}{M := N, S \Downarrow \mathsf{skip}, S'''}$$

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

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

Observational equivalence $\quad M \approx N$

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)

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

- 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

$$\mathsf{Snapback} \;:\; \mathsf{com} \rightarrow \mathsf{com}$$

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

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

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

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

More examples

(iii) 3rd-order examples of type $((\mathsf{bool} \rightarrow \mathsf{bool}) \rightarrow \mathsf{bool}) \rightarrow \mathsf{bool}$:

$$\begin{array}{rcl} \Theta_1 & = & \boldsymbol{\lambda} F.\mathsf{new}\, x := \mathsf{f} \,\mathsf{in}\, F(\boldsymbol{\lambda} y.\Xi_{x,y}) \\ \Theta_2 & = & \boldsymbol{\lambda} F.F(\boldsymbol{\lambda} y.\mathsf{new}\, x := \mathsf{f} \,\mathsf{in}\, \Xi_{x,y}) \\ \Theta_3 & = & \boldsymbol{\lambda} F.F(\boldsymbol{\lambda} x.x) \end{array}$$

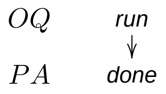
where $\Xi_{x,y} = (\mathsf{if}\, !x = \mathsf{f} \,\mathsf{then}\, x := y \,\mathsf{else}\, x := \neg y) \mathbin{;} !x$.

We have $\Theta_1 \not\approx \Theta_2 \approx \Theta_3$. Scope extrusion fails!

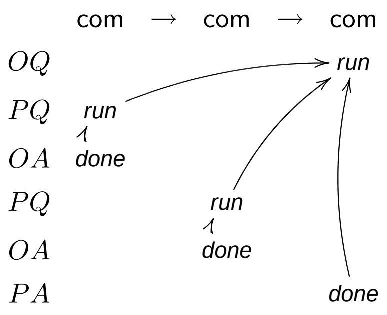
Question. Is observational equivalence decidable(assuming that terms are built up from finite base types e.g. booleans)?
Game semantics can help to answer the question.

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

We simply interpret com by the two-element arena:



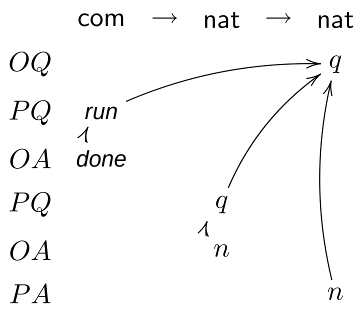
Interpreting sequential composition. $;$: com $\longrightarrow$ com $\longrightarrow$ com is modelled by the prefix-closure of:



The strategy is innocent (it is a $\mathbb{G}_{ib}$-map).

Interpreting sequential composition - active version

$;$: com $\longrightarrow$ nat $\longrightarrow$ nat is modelled by the prefix-closure of plays where n ranges over $\mathbb{N}$:



The strategy is innocent (it is a $\mathbb{G}_{ib}$-map).

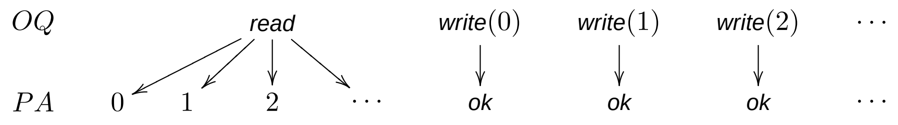
Interpreting loc

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

$$\text{loc} = \text{nat} \times \text{com}^{\omega}$$

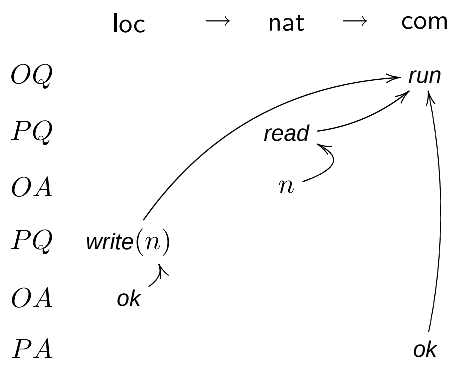
- first component is value held at that location
- second component contains countably many commands to write 0, 1, 2, etc. to that location.

Thus arena loc is the product arena nat $\times$ com$^{\omega}$:



Interpreting assignment

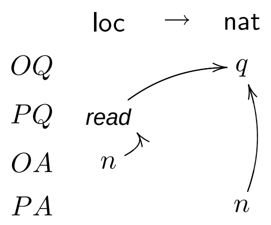
The strategy (interpreting) assign : loc $\longrightarrow$ nat $\longrightarrow$ com is the prefix-closure of (complete) plays of the form (with n ranging over $\mathbb{N}$)



The strategy is innocent (it is a $\mathbb{G}_{ib}$-map).

Interpreting dereferencing

The strategy (interpreting) $\mathsf{deref} : \mathsf{loc} \rightarrow \mathsf{nat}$ is the prefix-closure of (complete) plays of the form (with n ranging over $\mathbb{N}$)



The strategy is innocent (it is a $\mathbb{G}_{ib}$-map).

Interpreting new

We decompose the formation

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

into two constructions:

1. currying $\Gamma \vdash \boldsymbol{\lambda} x : \mathsf{loc}.M : \mathsf{loc} \rightarrow \mathsf{com}$
2. application by a constant $\mathsf{new}_n : (\mathsf{loc} \rightarrow \mathsf{com}) \rightarrow \mathsf{com}$.

I.e.

$$\Gamma \vdash \mathsf{new}\, x := n \,\mathsf{in}\, M \;=\; \mathsf{new}_c(\boldsymbol{\lambda} x : \mathsf{loc}.M)$$

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

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

Question. What is the strategy new_n?

The strategy $\mathsf{new}_n : (\mathsf{loc} \rightarrow \mathsf{com}) \rightarrow \mathsf{com}$

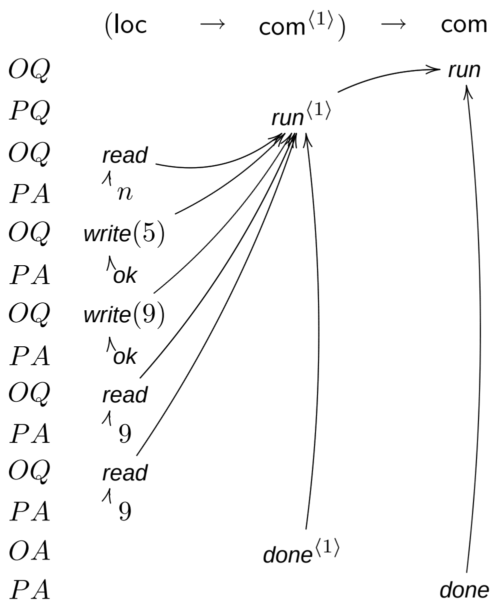
The plays in new_n should correspond to the behaviour of a *prima facie* variable (or location) initialized to n, namely, whenever the variable is read, it should yield the value last written to. Informally new_n is the prefix-closure of complete plays of the form shown on following page.

Equivalently the maximal plays are words matching the regular expression:

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

- The (infinite) alphabet is the move-set of $(\mathsf{loc} \rightarrow \mathsf{com}^{\langle 1 \rangle}) \rightarrow \mathsf{com}$ (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.

Note: new_n is not innocent!



$\mathbb{G}_b$ gives rise to a fully abstract model for IA

We say that a play is complete if it has no unanswered question.

Theorem [AM97] For any IA terms-in-context $\Gamma \vdash M, N : A$, we have

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

5 Algorithmic Games Semantics and Software Model Checking

Outline of Part 5

1. Introduction to Software Model Checking
2. Some results in Algorithmic Game Semantics: putting Game Semantics to work
3. Applications to Software Model Checking: A Prototype Tool
4. Further directions

What is (Software) Model Checking?

Problem: Given a system M and a property ϕ, does $M \vDash \phi$ hold?

The Model Checking approach:

1. Construct an abstraction – the "model" – of the system.
2. Formalize the property as a formula of some (decidable) modal logic.
3. Exhaustively check the model for violation of the formula.

Extremely successful in verifying relatively flat "unstructured" finite-state processes (e.g. digital circuits and communication protocols).

Less effective when applied to software. Why?

1. Programs are potentially infinite-state systems.
2. Modern programs can only be accurately modelled using rich, highly-structured models, as studied in Semantics.

Software Model Checking: Some Challenges

"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.

Several symptoms of the "Semantic Gap":

1. Issues of soundness and completeness.
2. Unable to handle many key high-level programming features.
3. Typically non-compositional and hence non-modularizable.

Semantics can and should help to address the Problem!

A new approach to Software Model Checking: 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 Part 5

1. Introduction to Software Model Checking
2. **Some results in Algorithmic Game Semantics: putting Game Semantics to work**
3. Applications to Software Model Checking: A Prototype Tool
4. Further directions

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.

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

Note. Up to order 2, justification pointers 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. (McCusker + Ghica) The set of complete plays in $[\![\Gamma \vdash M : A]\!]^{\mathrm{Kn}}$ is regular.

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

$$L \ = \ \textbf{cplays}(\llbracket \Gamma \vdash M : A \rrbracket) \upharpoonright \Sigma$$

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.

Theorem. Observational equivalence of 2nd-order IA with recursion (thus admitting recursively-defined first-order functions) is undecidable.

Jones + Muchnick 1978

Queue System: Fix alphabet Σ. Machine has one memory cell X, and a unbounded FIFO queue data structure.

A *queue program* is a finite sequence of the form $1 : I_1, \cdots, n : I_n$ where each I_i is one of

1. `enqueue` a where $a \in \Sigma$
2. `dequeue` X
3. `if` $X = a$ `goto` i
4. `halt`.

Fact. "Given a queue program, will it terminate eventually?" is undecidable.

Proof of theorem is by an idea of Jones and Muchnick (1978): simulate Queue programs in IA. I.e. $Q \mapsto \ulcorner Q \urcorner$: com such that Q terminates iff $\ulcorner Q \urcorner$ terminates.

$$Q \text{ terminates} \iff \exists Q \,.\, \ulcorner Q \urcorner, \varnothing \Downarrow \mathsf{skip}, S \iff \ulcorner Q \urcorner \approx \mathsf{skip}.$$

Deterministic Pushdown Automata (DPDA)

$$P = \langle Q, \mathsf{init}, \Sigma, \Gamma, \bot, \delta \rangle$$

finite set Q of *states*, an initial state init, finite set Σ of *input symbols*, finite set Γ of *stack symbols*, $\bot$ is bottom-of-stack symbol; δ is a finite set of *transitions*:

$$p, Z \xrightarrow{a} q, \alpha$$

where $p, q \in Q, Z \in \Gamma, \alpha \in \Gamma^*, a \in \Sigma \cup \{ \epsilon \}$

- Determinary: If $p, Z \xrightarrow{a} q, \alpha$ and $p, Z \xrightarrow{a} r, \beta$ and $a \in \Sigma \cup \{ \epsilon \}$ then $q = r$ and $\alpha = \beta$.
- Disjointness of a- and ϵ-transitions: If $p, Z \xrightarrow{\epsilon} q, \alpha$ and $p, Z \xrightarrow{a} r, \beta$ then $a = \epsilon$.

We say P is *real-time* if it has no ϵ-transition.

Configuration: a pair q, α

Transitions of configurations defined by *Prefix rule*: If $p, Z \xrightarrow{a} q, \alpha \in \delta$ then for any $\gamma \in \Gamma^*$, we have $p, \gamma Z \xrightarrow{a} q, \gamma\alpha$.

$\xrightarrow{a}$ between configurations, where $a \in \Sigma$, is then standardly extended to words $\xrightarrow{w}, w \in \Sigma^*$

Acceptance is by empty stack (or by final states).

Language recognized by P

$$\mathcal{L}(P) = \{ w \in \Sigma^* : \exists q \neq \mathsf{init} \,.\, \mathsf{init}, \bot \xrightarrow{w} q, \bot \}$$

Aim: Represent justified sequences as words of an alphabet

Encode pointers by exploiting Visibility:

We represent the pointer of a P-move by its *P-view offset*.

For any even-length legal position sm, code the pointer of m to q, which must appear in $\ulcorner s \urcorner$ by Visibility, by a numeric offset n defined to be the number of pending O-questions that occur *after* q in $\ulcorner s \urcorner$.

We can represent the move-with-pointer m as "$o^n \; m$".

E.g.

Representation has the force of Visibility; it works for all finite types.

Simplifications of the view-offset encoding of legal positions

1. Omit pointers from answer-moves. Ensure *Well-Bracketing* by a pushdown stack.
2. In 3rd-order case: another simplification

   Fact. In any 3rd-order arena, every O-view has at most one pending 1st-order P-question.

   Lemma. The pointer of any 2nd-order O-question is to the *unique* 1st-order pending P-question that appears in the O-view, which is guaranteed to appear in the P-view, at that pointer.

   Thus no need to consider O-views explicitly in the representation!

Every finite view functions f defines a (real-time) deterministic pushdown automaton (DPDA) $\mathsf{P}_f = \langle\, Q_f, \mathsf{init}, \Sigma_f, \Gamma_f, \bot, \delta_f \,\rangle$

$\Sigma_f = M_A$ augmented by a new symbol for each 3rd-order P-question $(_3$, with view-offset i, that appears in the range of f

$Q_f \;=\; \{\, \mathsf{init} \,\} \cup \mathrm{dom}(f) \cup \{\, p\,m : f(p) = m \,\}$

$\Gamma_f = \{\, f$*-reachable questions* $\}$

Four groups of transitions in δ_f:

Group	State Before (P-views)	Input Symbol	State After (P-views)	Stack Action
1	odd	P-question	even	push
2	odd	P-answer	even	pop
3	even (or init)	O-question	odd	push
4	even	O-answer	odd	pop

Hardest part: how to build the verification of O-visibility into the transition rules. Our method only works for up to 3rd order.

Order of Questions	Is explicit coding of pointers necessary?
0th (O)	Opening move has no pointer!
1st (P)	No. ∵ target of pointer is unique
2nd (O)	No. ∵ Lemma
3rd (P)	Yes

Case of *3rd-order compact innocent* strategies: *further* simplification!

For each 3rd-order q, for each n, simply introduce a *new* symbol $q^{[n]}$ (say) for the move-with-pointer "$o^n\;q$".

No harm: numeric offset n bounded by $\frac{1}{2}\max\{\, |p| : p \in \mathrm{dom}(f) \,\}$, where f is the generating view function.

Game semantics of 3rd-order is *not* regular

Take $\boldsymbol{\lambda} F.F(\boldsymbol{\lambda} x.x) : ((o \to o) \to o) \to o.$

Moves: $[_0,]_0, (_1,)_1, [_2,]_2, (_3,)_3$

For each $n \geq 2$ set $z_n \;=\; [_0\, (_1\; \underbrace{[_2\, (_3 \cdots [_2\, (_3}_{2(\lfloor n/2 \rfloor)\ \text{moves}}\;)_3\,]_2 \cdots)_3\,]_2\,)_1\,]_0$

For any $u, v, w \in \Sigma^*$ such that $z_n = u\,v\,w \in$ **cplays**$((\sigma))$, and $|v| \geq 1$, and $u\,v^i\,w \in$ **cplays**$((\sigma))$ for each $i \geq 0$, it follows from Well-Bracketing that v must have the form

$$\underbrace{[_2\, (_3 \cdots [_2\, (_3}_{j\ \text{moves}}\;)_3\,]_2 \cdots)_3\,]_2$$

such that $j \leq 2(\lfloor n/2 \rfloor)$, which implies that $|u\,v| > n$.

By the Pumping Lemma, $[\![\, \boldsymbol{\lambda} F.F(\boldsymbol{\lambda} x.x) \,]\!]^{\mathrm{Kn}}$ is not regular.

Recursion (fixpoint operator) can express standard iteration constructs and recursively-defined procedures. "Core C" embeds into (CBV) IA.

IA is essentially a CBN variant of Core ML; it is prototypical for languages combining state, block structure with higher-order features.

IA is surprisingly expressive! Scoped arrays with dynamically computed bounds, classes, objects and methods can be introduced by definitional extensions.

See O'Hearn + Tennent (ed). *Algol-like Languages: Vol I & II*. Biekhauser, 1977.

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?

Obs. Equiv of 3rd-order IA is decidable (con'd)

The problem "Is DPDA EQUIVALENCE decidable", first posed in 1966, was proved decidable by Sénizergues in 1998/2001 (awarded Gödel Prize 2002). (For a primitive recursive bound, see Stirling (*TCS* **255**:1-31, 2001.)

Complexity: Decision procedure is dominated by:

1. computing view-function generator (essentially normalization of IA)
2. deciding equivalence of the two derived DPDAs.

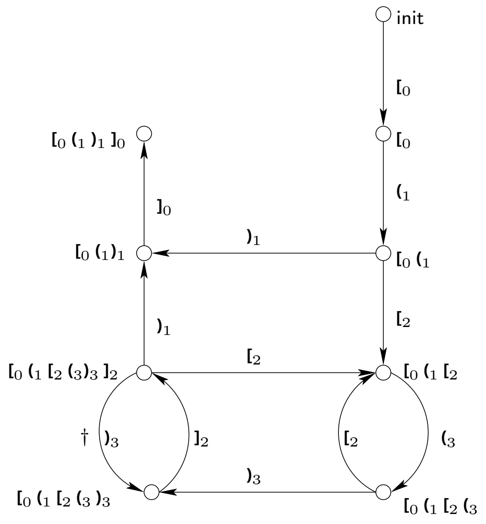
Theorem. There is an algorithm for deciding

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.

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



Outline of Part 5

1. Introduction to Software Model Checking
2. Some results in Algorithmic Game Semantics: putting Game Semantics to work
3. **Applications to Software Model Checking: A Prototype Tool**
4. Further directions

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

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.

[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 ($\mathbb{Z}$ 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.

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 $[\![M]\!] \vDash \phi$?

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.

References

[AGM$^+$04] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS'04)*, pages 150–159. IEEE Computer Society Press, 2004.

[AHM98] S. Abramsky, K. Honda, and G. McCusker. Fully abstract game semantics for general reference. In *Proceedings of IEEE Symposium on Logic in Computer Science, 1998*. Computer Society Press, 1998.

[AJ94] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *J. Symb. Logic*, 59:543–574, 1994.

[AJM94] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In *Theoretical Aspects of Computer Software: TACS'94, Sendai, Japan*, pages 1–15. Springer-Verlag, 1994. LNCS Vol. 789.

[AJM00] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163, 2000.

[AM97] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In P. W. O'Hearn and R. D.

Tennent, editors, *Algol-like languages*. Birkhauser, 1997.

[AM98] S. Abramsky and G. McCusker. Call-by-value games. In *Proceedings of CSL '97*. Springer-Verlag, 1998. Lecture Notes in Computer Science.

[BC82] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.

[Bla92] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220, 1992.

[DG01] P. Di Gianantonio. Game semantics of the Pure Lazy λ-Calculus. In *Proceedings of Typed Lambda Calculus and Applications*. Springer-Verlag, 2001. LNCS, To appear.

[DGFH99] P. Di Gianantonio, G. Franco, and F. Honsell. Game semantics for the untyped $\lambda\beta\eta$-calculus. In *Proceedings of the International Conference on Typed Lambda Calculus and Applications*, pages 114–128. Springer-Verlag, 1999. LNCS Vol. 1591.

[DH00] V. Danos and R. Harmer. Probabilistic game semantics. In *Proc. IEEE Symposium on Logic in Computer Science, Santa Barbara, June, 2000*. Computer Science Society, 2000.

[Fel86] W. Felscher. Dialogues as a foundation for intuitionistic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Vol. III*, pages 341–372. D. Reidel Publishing Company, 1986.

[Gan67] R. O. Gandy. Computable functionals of finite type I. In J. N. Crossley, editor, *Sets, Models and Recursion Theory*. North-Holland, 1967.

[GM04] D. R. Ghica and A. S. Murawski. Angelic semantics of fine-grained concurrency. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'04)*, pages 211–225, 2004. LNCS Volume 2987.

[GMO04] D. R. Ghica, A. S. Murawski, and C.-H. L. Ong. Syntactic control of concurrency. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, pages 683–694, 2004. LNCS Volume 3142.

[HM99] R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. In *Proceedings of Fourteenth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1999.

[HO93] J. M. E. Hyland and C.-H. L. Ong. Fair games and full completeness for Multiplicative Linear Logic without the MIX-rule. preprint, 1993.

[HO95] J. M. E. Hyland and C.-H. L. Ong. Pi-calculus, dialogue games and PCF. In *Proc. 7th* ACM *Conf. Functional Prog. Lang. Comp. Architecture*, pages 96 – 107. ACM Press, 1995.

[HO00] J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I. Models,

observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation*, 163:285–408, 2000.

[Hug97] D. Hughes. Games and definability for System F. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science*. IEEE Computer Science Society, 1997.

[HY99] K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation. *Theoretical Computer Science*, 221:393–456, 1999.

[Kle59] S. C. Kleene. Recursive functionals and quantifiers of finite types I. *Trans. American Mathematical Society*, 91:1–52, 1959.

[Kle78] S. C. Kleene. Recursive functionals and quantifiers of finite types revisited I. In J. E. Fenstad, R. O. Gandy, and G. E. Sacks, editors, *General Recursion Theory II, Proceedings of the 1977 Oslo Symposium*, pages 185–222. North-Holland, 1978.

[KNO02] A. D. Ker, H. Nickau, and C.-H. L. Ong. Innocent game models of untyped λ-calculus. *Theoretical Computer Science*, 272:247–292, 2002.

[KNO03] A. D. Ker, H. Nickau, and C.-H. L. Ong. Adapting innocent game model for the bəhm tree lambda theory. *Theoretical Computer Science*, 308:333–366, 2003.

[KP93] G. Kahn and G. D. Plotkin. Concrete domains. *Thereotical Computer Science*,

1993. in Bəhm Feschrift Special Issue. Article first appeared (in French) as technical report 338 of INRIA-LABORIA, 1978.

[Lai97] J. D. Laird. Full abstraction for functional languages with control. In *Proceedings of the Twelfth International Symposium on Logic in Computer Science*, pages 58–67. IEEE Computer Society, 1997.

[Loa01] R. Loader. Finitary PCF is not decidable. *Theoretical Computer Science*, 266:342–364, 2001.

[McC98] G. McCusker. *Games for recursive types*. BCS Distinguished Dissertation. Cambridge University Press, 1998.

[Mil77] R. Milner. Fully abstract models of typed lambda-calculus. *Theoretical Computer Science*, 4:1–22, 1977.

[MO01] A. S. Murawski and C.-H. L. Ong. Evolving games and essential nets for affine polymorphism. In *Typed Lambda Calculi and Applications: Proc. 5th Int. Conf. TLCA2001, Krakow, Poland, May 2001*, pages 360–375. Springer-Verlag, 2001. LNCS Vol. 2044.

[Nic96] H. Nickau. *Hereditarily Sequential Functionals: A Game-Theoretic Approach to Sequentiality*. PhD thesis, Universitat-Gesamthochschule-Siegen, 1996.

[ODG04] C.-H. L. Ong and P. Di Gianantonio. Games characterizing Levy-Longo trees.

Theoretical Computer Science, 312:121–142, 2004.

[Pla66] R. A. Platek. *Foundations of Recursion Theory*. PhD thesis, Standford University, 1966.

[Plo77] G. D. Plotkin. LCF as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Sco93] D. S. Scott. A type-theoretical alternative to CUCH, ISWIM and OWHY. *Theoretical Computer Science*, 121:411–440, 1993.