What is Game Semantics?

Background: Where to place the subject?

Applied Logic.

Logic in Computer Science / Logical Methods in Computer Science.

Two branches of Theoretical Computer Science:

- Structures: Semantics and Logics
- Algorithmics and Complexity

Game Semantics is a relatively recent development in Semantics of Programming Languages - a branch of Semantics of Computation.

Game semantics is a way of giving meanings to computation (and to proofs) using the intuitive idea of game playing.
Synopsis of the Course

Game Semantics and its Algorithmic Applications

Part 1. Overview and Basics of Game Semantics

Part 2. Application 1: Decidability of MSO Theories of Trees generated by Higher-Order Recursion Schemes

Part 3. Application 2: Algorithmic Game Semantics and Software Model Checking
Part 1: Overview and Basics of Game Semantics

1. Introduction: Modelling higher-order computation by dialogue game.

2. Scott and Plotkin’s PCF and the Full Abstraction Problem.

Kleene’s Problem (1959). Find “a class of functions which shall coincide with all the partial functions which are ‘effectively computable’, so that Church’s 1936 Thesis will apply with the higher types included”

Kleene’s approach (in 4 papers from ’59 through to 80s): Capture the class of functions by a protocol that governs a two-player dialogue game.

3. CA – a cartesian closed category of arenas and innocent strategies.

Composing strategies by “parallel composition plus hiding”.

4. Definability Theorem.

Theorem [Hyland-O.]. CA gives rise to a (syntax-independent) fully abstract, extensional model of PCF.
A Problem in Verification: Find finitely-presentable infinite-state systems with decidable (MSO, modal mu-calculus, etc.) model-checking problem.

Structures with decidable MSO theories: Some Milestones

1. Rabin 1969: Regular trees. “Mother of all decidability results”
4. Knapik et al. (TLCA 2001, FOSSACS 2002): \(\bigSigma\)-trees generated by safe recursion schemes of all finite orders (= higher-order pushdown trees). All subsumed by Cauca\textit{cal Tree and Graph Hierarchies}.

Question. Do \(\bigSigma\)-trees generated by higher-order recursion schemes, whether safe or not, have decidable MSO theories? If so, at which levels?
**Theorem.** The modal mu-calculus model checking problem for trees generated by order-$n$ recursion schemes is $n$-EXPTIME complete, for each $n \geq 0$.

**Three major ingredients:**

1. A **transference principle** from tree generated from the scheme – value tree – to an auxiliary **computation tree**.

   A strong correspondence between paths in the value tree and traversals in the computation tree – proof is by game semantics.

   Thus an alternating parity tree automaton (APT) has an accepting run-tree over the value tree iff it has an accepting **traversal-tree** over the computation tree.

2. **Simulation** of an (accepting) traversal-tree by a certain set of annotated paths over the computation tree; the latter recognised by traversal-simulating APT.

3. Application of Jurdziński’s complexity bound for solving acceptance parity game to a graph generator and the traversal-simulating APT.
Part 3: Algorithmic Game Semantics and Software Model Checking

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

**Algorithmic game semantics:** Extraction of algorithms for program verification from (representations of) game semantics.

**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.
Challenges of the Approach. To carry over methods of model checking to much more structured, modern programming situations.

A case study: Idealized Algol (IA) – a higher-order procedural language

Let $\mathcal{L}$ range over sublanguages of IA.

$\text{OBS}_\mathcal{L}\text{EQUIV}$: Given two $\mathcal{L}$-terms (in normal form), are they observationally equivalent?

We use game semantics to study decidability of $\text{OBS}_\mathcal{L}\text{EQUIV}$.

A Complete Classification of Decidable Fragments of IA

<table>
<thead>
<tr>
<th></th>
<th>pure</th>
<th>$+\text{while}$</th>
<th>$+Y_0$</th>
<th>$+Y_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{IA}_1$</td>
<td>CO$\text{NP}$</td>
<td>PSPACE</td>
<td>DPDA EQUIV</td>
<td>undecidable</td>
</tr>
<tr>
<td>$\text{IA}_2$</td>
<td>PSPACE</td>
<td>PSPACE</td>
<td>DPDA EQUIV</td>
<td>undecidable</td>
</tr>
<tr>
<td>$\text{IA}_3$</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
<td>DPDA EQUIV</td>
<td>undecidable</td>
</tr>
<tr>
<td>$\text{IA}_i, i \geq 4$</td>
<td>undecidable</td>
<td>undecidable</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>
1 Overview and Basics of Game Semantics

1. Introduction: Modelling higher-order computation by dialogue game.

2. Scott and Plotkin’s PCF and the Full Abstraction Problem.

3. $\mathcal{CA}$ – a cartesian closed category of arenas and innocent strategies.
   Composing strategies by “parallel composition plus hiding”.

4. Definability Theorem.

**Theorem** [Hyland-O.]. $\mathcal{CA}$ gives rise to a (syntax-independent) fully abstract, extensional model of PCF.
Precursors of Game Semantics

**Logic**

- **Higher-type recursion theory**: Is there a canonical notion of sequential computation at higher types? - higher-order analogue of Church’s Thesis. Kleene [Kle59, Kle78], Platek [Pla66], Gandy + Pani [Gan67]

- **Proof theory**: Dialogue games for modelling intuitionistic provability by Lorenz and Lorenzen, see Felscher’s survey paper [Fel86]. No interpretation of the cut rule – compositionality of strategies was not addressed.

**Computer Science**

**The Full Abstraction Problem for PCF**. Sequential algorithms on concrete data structures, Kahn and Plotkin [KP93], Berry and Curien [BC82]
Game Semantics: Perspectives

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

Two players:

P  “Proponent”  “Verifier”  0
O  “Opponent”  “Refuter”  1

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

<table>
<thead>
<tr>
<th>Players</th>
<th>Point-of-view</th>
<th>Functional</th>
<th>Imperative</th>
<th>Concurrent</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>System</td>
<td>term</td>
<td>procedure</td>
<td>process</td>
</tr>
<tr>
<td>O</td>
<td>Environment</td>
<td>program context</td>
<td>context &amp; store</td>
<td>rest-of-system</td>
</tr>
</tbody>
</table>

Basic idea of game semantics:

The meaning of a system is given in terms of all 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 a term $M$ of type $A$ is modelled by a P-strategy $\left[ M \right]$ for playing in the arena $\left[ A \right]$ that models the type $A$.

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

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>System (P)</td>
<td>consumes</td>
<td>produces</td>
</tr>
<tr>
<td>Environment (O)</td>
<td>produces</td>
<td>consumes</td>
</tr>
</tbody>
</table>

Example. A typical interaction of the successor function:

\[
\text{succ} : \mathbb{N} \rightarrow \mathbb{N}
\]

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.”
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** (or **Well-Bracketing**): “Last asked first answered.”

**Outcome of play:** We don’t care about winning. Dialogue ends when the opening question is answered.
**Order of a Type**

Types are ranged over by $A, B, \cdots$.

$$A ::= o \mid (A \to B)$$

Every type can be written uniquely as

$$A_1 \to \cdots \to A_n \to o, \quad n \geq 0$$

(-arrows associate to the right), which is then abbreviated to

$$(A_1, \cdots, A_n, o).$$

The order of a type measures how nested it is in the LHS of the arrow.

\begin{align*}
\text{order}(o) &= 0 \\
\text{order}(A \to B) &= \max(\text{order}(A) + 1, \text{order}(B))
\end{align*}
Example: higher-order program

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

\[ f : \mathbb{N} \rightarrow \mathbb{N} \vdash \text{if } (f(5) = 6) \text{ then } 7 \text{ else } 0 : \mathbb{N} \]

1. OQ: “What is the value (of type nat) of this program?”

2. PQ: “What is the output of the function variable \( f \)?”

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.”
Example: Multiple use of argument

\[ f : \mathbb{N} \rightarrow \mathbb{N} \vdash f(5) + f(6) : \mathbb{N} \]

\[ f(5) = 6 \]

\[ f(6) = 7 \]

\[ q \]

\[ 13 \]
Example: Nested use of arguments

Programs may nest uses of their arguments:

\[ f : \mathbb{N} \rightarrow \mathbb{N} \quad \vdash f(f(5)) : \mathbb{N} \]

\[ f(5) = 6 \]

\[ f(6) = 7 \]
Game models are accurate

Criteria of goodness-of-fit

- A denotational model of a programming language is **fully abstract** if the equational theory induced by the model coincides with observational (or contextual) equivalence.
  
  I.e. \( \mathcal{M} \) **fully abstract** means \( \mathcal{M} \models M = N \iff M \approx N \).

- A model of (the proofs of) a logic is **fully complete** just if two proofs are identified by the model iff they are equal.
  
  (Equivalently the functor from the classifying category of the logic to the model is full and faithful.)
Part 1: Overview and Basics of Game Semantics

1. Introduction: Modelling higher-order computation by dialogue game.

2. **Scott and Plotkin’s PCF and the Full Abstraction Problem.**

3. $\mathcal{CA}$ – a cartesian closed category of arenas and innocent strategies. Composing strategies by “parallel composition plus hiding”.

4. **Definability Theorem.**

   **Theorem** [Hyland-O.]. $\mathcal{CA}$ gives rise to a (syntax-independent) fully abstract, extensional model of PCF.
Brief history of PCF, Programmable Computable functions

PCF = “simply-typed \( \lambda \)-calculus + basic arithmetic + definition-by-cases + fixpoint operators”

Gödel’s System T: primitive recursive functionals of finite types

Kleene: full blown generalization of recursion theory to higher types. Recursion introduced by a computation scheme.

Platek (66’ thesis): 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. \( A ::= \text{nat} \mid \text{bool} \mid A \rightarrow A \)

PCF terms.

\[ n : \text{nat} \quad \text{numerals} \]
\[ \text{tt, ff} : \text{bool} \quad \text{booleans} \]
\[ \text{succ} : \text{nat} \rightarrow \text{nat} \quad \text{successor} \]
\[ \text{pred} : \text{nat} \rightarrow \text{nat} \quad \text{predecessor} \]
\[ \text{zero?} : \text{nat} \rightarrow \text{bool} \quad \text{test for zero} \]
\[ \text{cond}^{\text{nat}} : \text{bool} \rightarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \quad \text{natural number conditional} \]
\[ \text{cond}^{\text{bool}} : \text{bool} \rightarrow \text{bool} \rightarrow \text{bool} \rightarrow \text{bool} \rightarrow \text{bool} \quad \text{boolean conditional.} \]

\[
\frac{s : A \rightarrow A}{Y^A(s) : A} \quad \frac{s : A \rightarrow B \quad t : A}{(s \ t) : B} \quad \frac{s : B}{(\lambda x : A. s) : A \rightarrow B}
\]

PCF term-in-context. \( x_1 : A_1, \ldots, x_n : A_n \vdash M : B \)
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 $v$”.

$$
\begin{align*}
    v & \Downarrow v \\
    u[t/x] & \Downarrow v \\
    (\lambda x. u)t & \Downarrow v \\
    st & \Downarrow v' \\
    sY^A(s) & \Downarrow v \\
    s \Downarrow 0 \\
    \text{zero? } s & \Downarrow \text{tt} \\
    \text{zero? } s & \Downarrow \text{ff} \\
    s \Downarrow \text{tt} \\
    u & \Downarrow v \\
    \text{cond}^\beta suu' & \Downarrow v \\
    s \Downarrow \text{ff} \\
    u' & \Downarrow v \\
    s \Downarrow n \\
    \text{succ } s & \Downarrow n + 1 \\
    s \Downarrow n + 1 \\
    \text{pred } s & \Downarrow n \\
    s \Downarrow 0 \\
    \text{pred } s & \Downarrow 0
\end{align*}
$$
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 hard to reason about.

Examples

1. $(\lambda x : A. M)N \approx M[N/x]$
2. $Y^A(M) \approx M(Y^A(M))$
3. Assuming $x \notin \text{FV}(B)$,
   $$\lambda x. \text{if } B \text{ then } M \text{ else } N \approx \text{if } B \text{ then } \lambda x. M \text{ else } \lambda x. N.$$
What is a (denotational) model of PCF?

Two key ideas:

- Models of simply-typed \( \lambda \)-calculus are cartesian closed categories.
- Tarski-Knaster interpretation of fixpoints.

CPO-enriched CCCs are models of PCF.

Interpret PCF types as objects: \([ A \rightarrow B ] = [ A ] \rightarrow [ B ]\).

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

\[
[ \Gamma \vdash x_i : A_i ] = \pi_i : [ A_1 ] \times \cdots \times [ A_n ] \rightarrow [ A_i ]
\]
\[
[ \Gamma \vdash \lambda x : A.M : B ] = \Lambda([\Gamma, x : A \vdash M : B]) : [\Gamma] \rightarrow ([A] \rightarrow [B])
\]
\[
[ \Gamma \vdash MN : B ] = \langle [\Gamma \vdash M], [\Gamma \vdash N] \rangle ; ev_{A,B} : [\Gamma] \rightarrow [B]
\]
\[
[ \Gamma \vdash Y^A(M) : A ] = \text{lfp}([\Gamma \vdash M : A \rightarrow A]) : [A]
\]
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

$$\llbracket \Gamma \vdash M : A \rrbracket = \llbracket \Gamma \vdash N : A \rrbracket \iff M \approx N$$

**Theorem.** [Sco93, Plo77] The Scott model (of domains and continuous functions) is not fully abstract for PCF, but it is for PCF + parallel-or.

<table>
<thead>
<tr>
<th>p—or</th>
<th>⊥</th>
<th>tt</th>
<th>ff</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>tt</td>
<td>⊥</td>
</tr>
<tr>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
</tr>
<tr>
<td>ff</td>
<td>⊥</td>
<td>tt</td>
<td>ff</td>
</tr>
</tbody>
</table>
The Full Abstraction Problem for PCF

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

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

**Want:** A abstract, syntax-independent and “synthetic” characterization of Milner’s model.

**Syntax-independent fully abstract models of PCF**

**Theorem.** [HO00] $G_{ib}$ gives rise to an order-extensional, fully abstract model for PCF.

Another solution was obtained by Abramsky, Jagadeesan and Malacaria [AJM00] independently. Their model was based on history-free strategies, and via models of intuitionistic fragments of multiplicative linear logic.
1. Introduction: Modelling higher-order computation by dialogue game.

2. Scott and Plotkin’s PCF and the Full Abstraction Problem.

3. \( \mathcal{CA} \) – a cartesian closed category of arenas and innocent strategies. Composing strategies by “parallel composition plus hiding”.

4. Definability Theorem.

**Theorem** [Hyland-O.]. \( \mathcal{CA} \) gives rise to a (syntax-independent) fully abstract, extensional model of PCF.
References


