OPLSS22: Game Semantics

Speaker: Pierre-Louis Curien

June 21, 2022

1 Preliminaries

There are a number of duals in the world of programming, e.g. memory cell vs. value, input vs. output, sending vs. receiving messages, program and its context, CBN vs. CBV, etc. Game semantics introduces a dual between *player* vs. *opponent*. Broadly, a player can be understood as a program and the opponent can be thought of as a (potentially adversarial) environment.

1.1 Programs as strategies

In this context, proofs are imbued with a dialogue-game interpretation where the player specifies a proof and the opponent challenges some formulas. It is then the player's responsibility to justify the subsequent parts of their proof step-by-step for each of the opponent's challenges.

An *arena*, or game, is defined as a set of moves by either player or opponent, some relational to others. Player and opponent each play in turns, thus generating a sequence of moves $m_1 \dots m_n$ which we refer to as a *play*. A *strategy* is defined as a set of plays that can be generated via some rules. Strategies are given by the notation $\{\epsilon, qn, qnqn, qnqnqn, \dots\}$ where each qn is a move and each element of the set is a play.

1.2 Untyped lambda calculus

A term of the untyped lambda calculus can be a variable, an abstraction or an application

$$M ::= x \mid \lambda x.M \mid M\ M$$

Every term can be represented in exactly one of the following forms

- **Head normal form:** For $n \geq 1, p \geq 1$,

  $$\lambda x_1 \dots x_n.x\ M_1 \dots M_p$$

  where x is called the *head variable*.

- **Head redex form:** For $n \geq 0, p \geq 1$,

  $$\lambda x_1 \dots x_n.(\lambda x.M)\ M_1 \dots M_p$$

2 Böhm trees

From dialogue-game interpretation naturally arise *Böhm trees*. In particular, the game begins with the opponent challenging the initial term. To this challenge, the player must provide a proof, i.e. the corresponding Böhm tree. Continuing, the opponent may challenge a subterm, so long as its head normal form exists. The player is then tasked with responding to this challenge. Böhm trees are defined as follows:

$$BT(M) = \begin{cases} \Omega & \text{if } M = \lambda x_1 \dots x_n.(\lambda x.M')M_1 \dots M_p \\ \lambda x_1 \dots x_n.x\omega(M_1) \dots \omega(M_p) & \text{if } M = \lambda x_1 \dots x_n.xM_1 \dots M_p \end{cases}$$

Terms Ω or $\lambda x_1 \dots x_n.xM_1 \dots M_p$ (for $n \geq 0$, $p \geq 0$) can be ordered using a prefix ordering:

$$\frac{}{\Omega \leq M} \qquad \frac{M_1 \leq N_1 \dots M_p \leq N_p}{\lambda x_1 \dots x_n.xM_1 \dots M_p \leq \lambda x_1 \dots x_n.xN_1 \dots N_p}$$

The Böhm tree of a term is the (possibly infinite) least upper bound of all $\omega(N)$ for all N such that $M \to^* N$ where $\omega(N)$ is a normal form. Note that there exists a Böhm tree for any given λ term.

2.1 η-long Böhm trees

For the moment, we temporarily restrict ourselves to finitely-long Böhm trees, i.e. *η-long Böhm trees*. An η-long Böhm tree is a Böhm tree that satisfies two criteria:

- Each occurrence of a variable must appear in a context where it is applied to all its arguments.
- In each sequence of abstractions $\lambda\vec{x}.M$, the number of parameters $x_1, \dots, x_n$ is exactly the number of arguments $N_1, \dots, N_n$ which the term $x_1, \dots, x_n$ can accept according to its type.

2.2 The language PCF

The seminal work in the area of game semantics brought two models of a paradigmatic core pure functional language PCF, where types are interpreted by games and terms by strategies. The complete syntax of PCF is given as follows:

$$M ::= x \mid \lambda x.M \mid MM \mid n \mid T \mid F \mid pred \mid succ \mid zero? \mid \texttt{case}\ M[..., c \to M, ...] \mid Y$$

PCF is a simple typed λ-calculus with the following constants:

$$
\begin{aligned}
n &: Nat && (n \in \omega)\\
T, F &: Bool\\
succ, pred &: Nat \to Nat\\
zero? &: Nat \to Bool\\
\texttt{if then else} &: Bool \to Nat \to Nat \to Nat\\
\texttt{if then else} &: Bool \to Bool \to Bool \to Bool\\
\Omega &: \sigma && \forall\sigma\\
Y &: (\sigma \to \sigma) \to \sigma && \forall\sigma
\end{aligned}
$$

2.2.1 PCF Böhm trees

The key difference between PCF Böhm trees and normal Böhm trees is that terms may be wrapped in *case* statements over y whose value is a continuation.

$$M ::= \lambda\vec{x} : \vec{\sigma}.W$$
$$W ::= v \mid \texttt{case}\ \ yM_1 \dots M_n\,[v_1 \mapsto W_1, \dots, v_k \mapsto W_k]$$

$$\frac{v : C}{\Gamma \vdash v : C} \qquad \frac{\Gamma, \vec{x} : \vec{\sigma} \vdash W : C}{\Gamma \vdash (\lambda\vec{x} : \vec{\sigma}.W) : \vec{\sigma} \to C}$$

$$\frac{\begin{array}{c}\Gamma \cdot y = \sigma_1 \to \dots \sigma_n \to C\\ v_1, \dots, v_k : C \qquad \Gamma \vdash M_1 : \sigma_1 \dots \Gamma \vdash M_n : \sigma_n \qquad \Gamma \vdash W_1 : C_1 \dots \Gamma \vdash W_k : C_1\end{array}}{\Gamma \vdash \texttt{case}\ \ yM_1 \dots M_n\,[v_1 \to W_1, \dots, v_k \to W_k] : C_1}$$

3 Abstract machine

Böhm trees are functional values evaluated by applying the rules within a framework of an abstract machine, where the expression regularly associated with the body of a value is being replaced by a variable called body. Environment is now a list of stacked frames, a frame a vector or tuple of bindings to variables, while the head variable can take a value relative to each environment. Every strategy is an interpretation of the initial program.

- Terms: $M ::= (\lambda\vec{x}.W)$
- Code: $W ::= y\vec{M}$
- Environments: $e ::= nil \mid B \cdot e$
- Frames: $B ::= \langle\vec{z} \leftarrow \vec{M}\rangle\,[e]$

Observe that the term M is split in this way even if $\vec{x}$ is empty, in which case we write $M = (W)$ and $W = y\vec{N}$ while retaining a trace of the λ. We call W the body. A frame is a list of bindings with terms. An environment is a list of frames.

Computation proceeds by application of the following rule:

$$(K) \qquad (x\vec{M})[e] \to W[\langle \vec{z} \leftarrow \vec{M} \rangle [e] \cdot e_i]$$

where

$$e = B_0 \cdot \cdots \cdot B_n \qquad B_i = \langle \vec{x}_i \leftarrow \vec{N}_i \rangle [e_i]$$
$$x = x_{ij} \qquad N_{ij} = (\lambda \vec{z}.W)$$

4 HO Games

Introduced by Hyland and Ong in the early 1990s, HO Games present a denotational semantics for PCF in which each datatype is interpreted as an *arena*.

4.1 Natural numbers

The arena `nat` for natural numbers is composed by only one opponent move called q, that asks the player for a number, and an infinite number of Player moves that return a natural number, one for each possible output.

For example, a program that just returns the number 3 can be interpreted as the play $q3$.

The denotation of an inhabitant of a datatype is a strategy (in the datatype's arena). In later lectures we will formalize the notion of a strategy and define special properties that hold for some strategies (e.g. innocence, determinism).

We diagram the arena `nat` as follows:

$$\begin{cases} \{\epsilon\} \\ \{\epsilon, qn\} \end{cases} \qquad \begin{cases} \bot \\ n \end{cases}$$

The diagram above has two parts: on the left we have the HO game semantics and on the right we have a traditional Scott denotational semantics. Each line shows how an inhabitant of `nat` is represented in each semantics. So the strategy $\{\epsilon\}$ is the HO game semantic denotation for the undefined value commonly denoted $\bot$ and $\{\epsilon, qn\}$ is the HO game semantic denotation of any $n \in \mathbb{N}$.

4.2 Causality between moves

Apart from the set of player and opponent moves, an arena must define an enabling relation among moves (i.e., when a move is a valid answer to an opposing move).

We write $m \vdash m'$ to note that the move m enables the use of the move m'. Since plays consist of a sequence of alternating opponent and player moves, the moves in an enabling relation must have opposite polarities (i.e., they can't be both player or opponent moves).

4.3 Product types

The arena for $\texttt{nat} \times \texttt{nat}$ is given by two copies of $\texttt{nat}$, that we will call $\texttt{nat}_1$ and $\texttt{nat}_2$.

The opponent can request the first or second projection from the pair (moves q_1 and q_2, respectively) and the player can answer with the request number (noted m_1 or n_2). The enablings are inherited from the components of the product:

$$q_1 \vdash m_1 \quad q_2 \vdash n_2$$

Notice that $q_1 n_2$ is not a valid play. The initial moves are q_1 and q_2.

We diagram the arena $\texttt{nat}_1 \times \texttt{nat}_2$ as follows:

$$\begin{cases} \{\epsilon\} \\ \{\epsilon, q_1 m_1\} \\ \{\epsilon, q_2 n_2\} \\ \{\epsilon, q_1 m_1, q_2 n_2\} \end{cases} \qquad \begin{cases} (\bot,\bot) \\ (m,\bot) \\ (\bot,n) \\ (m,n) \end{cases}$$

4.4 Function types

Apart from the enablings inherited from its components, the arena for a function type needs $q_\epsilon \vdash q_1$. The only initial move is q_ϵ.

We diagram the arena $\texttt{nat}_1 \to \texttt{nat}_\epsilon$ as follows:

$$q_\epsilon q_1 \begin{cases} m_{11} n_{\epsilon 1} \\ m_{12} n_{\epsilon 2} \\ \vdots \end{cases} \qquad \lambda x.\texttt{case}\ x \begin{cases} m_1 \to n_1 \\ m_2 \to n_2 \\ \vdots \end{cases}$$

The sequence $q_\epsilon q_1$ at the beginning states that a function looks at its argument before outputting anything.

As an example, lets look at the following program:

$$\lambda x.\texttt{case}\ x \begin{cases} 0 \to 3 \\ 1 \to 4 \end{cases}$$

We can write the Böhm tree as,

$$q_\epsilon q_1 \begin{cases} 0_1 3_\epsilon \\ 1_1 4_\epsilon \end{cases}$$

The play $q_\epsilon q_1 0_1 3_\epsilon$ can be interpreted as follows:

1. The opponent starts by asking about the behaviour of the function.
2. The player asks for an specific input.
3. The opponent asks for the behaviour on input 0.
4. The player returns 3, stating that the function returns 3 on input 0.

References

Pierre-Louis Curien. Notes on game semantics, February 2019. URL `https://curien.galene.org/papers/Game-semantics.pdf`.