Semantics of Interaction: an Introduction to Game Semantics

Samson Abramsky

Contents

1 Introduction

The "classical" paradigm for denotational semantics models data types as *domains*, *i.e.* structured sets of some kind, and programs as (suitable) *functions* between domains. The semantic universe in which the denotational modelling is carried out is thus a category with domains as objects, functions as morphisms, and composition of morphisms given by function composition. A sharp distinction is then drawn between denotational and operational semantics. Denotational semantics is often referred to as "mathematical semantics" because it exhibits a high degree of mathematical structure; this is in part achieved by the fact that denotational semantics abstracts away from the dynamics of computation—from time. By contrast, operational semantics is formulated in terms of the syntax of the language being modelled; it is highly intensional in character; and it is capable of expressing the dynamical aspects of computation.

The classical denotational paradigm has been very successful, but has some definite limitations. Firstly, fine-structural features of computation, such as sequentiality, computational complexity, and optimality of reduction strategies, have either not been captured at all denotationally, or not in a fully satisfactory fashion. Moreover, once languages with features beyond the purely functional are considered, the appropriateness of modelling programs by functions is increasingly open to question. Neither concurrency nor "advanced" imperative features such as local references have been captured denotationally in a fully convincing fashion.

This analysis suggests a desideratum of *Intensional Semantics*, interpolating between denotational and operational semantics as traditionally conceived. This should combine the good mathematical structural properties of denotational semantics with the ability to capture dynamical aspects and to embody computational intuitions of operational semantics. Thus we may think of Intensional semantics as "Denotational semantics + time (dynamics)", or as "Syntax-free operational semantics".

A number of recent developments (and, with hindsight, some older ones) can be seen as contributing to this goal of Intensional Semantics. We will focus on the recent work on Game semantics, which has led to some striking advances in the Full Abstraction problem for PCF and other programming languages (Abramsky *et al.* 1995) (Abramsky and McCusker 1995) (Hyland and Ong 1995) (McCusker 1996a) (Ong 1996). Our aim is to give a genuinely elementary first introduction; we therefore present a simplified version of game semantics, which nonetheless contains most of the essential concepts. The more complex game semantics in (Abramsky *et al.* 1995) (Hyland and Ong 1995) can be seen as refinements of what we present. Some background in category theory, type theory and linear logic would be helpful in reading these notes; suitable references are (Crole 1994), (Girard *et al.* 1989), (Girard 1995) (which contain much more than we will actually need).

Acknowledgements I would like to thank the Edinburgh "interaction group" (Kohei Honda, Paul-André Melliès, Julo Chroboczek, Jim Laird and Nobuko Yoshida) for their help in preparing these notes for publication, Peter Dybjer for his comments on a draft version, and Peter Dybjer and Andy Pitts for their efforts in organizing the CLiCS summer school and editing the present volume.

Notation

If X is a set, X^* is the set of finite sequences (words, strings) over X. We use s, t, u, v to denote sequences, and a, b, c, d, m, n to denote elements of these sequences. Concatenation of sequences is indicated by juxtaposition, and we won't distinguish notationally between an element and the corresponding unit sequence. Thus as denotes the sequence with first element a and tail s.

If $f : X \longrightarrow Y$ then $f^* : X^* \longrightarrow Y^*$ is the unique monoid homomorphism extending f. We write $|s|$ for the length of a finite sequence, and s_i for the ith element of s, $1 \leq i \leq |s|$.

Given a set S of sequences, we write S^{even}, S^{odd} for the subsets of even- and odd-length sequences respectively.

We write $X + Y$ for the disjoint union of sets X, Y.

If $Y \subseteq X$ and $s \in X^*$, we write $s \restriction Y$ for the sequence obtained by deleting all elements not in Y from s. In practice, we use this notation in the context where

$X = Y + Z$, and by abuse of notation we take $s \restriction Y \in Y^*$, *i.e.* we elide the use of injection functions.

We write $s \sqsubseteq t$ if s is a prefix of t, *i.e.* $t = su$ for some u.

$\mathrm{Pref}(S)$ is the set of prefixes of elements of $S \subseteq X^*$. S is *prefix-closed* if $S = \mathrm{Pref}(S)$.

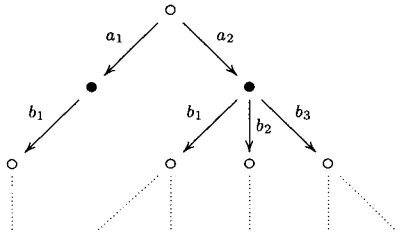
2 Game Semantics

We give a first introduction to game semantics. We will be concerned with two-person games. Why the magic number 2? The key feature of games, by comparison with the many extant models of computation (labelled transition systems, event structures, etc. etc.) is that they provide an *explicit representation of the environment*, and hence model interaction in an intrinsic fashion. By contrast, interaction is modelled in, say, labelled transition systems using some additional structure, typically a "synchronization algebra" on the labels. One-person games would degenerate to transition systems; it seems that multi-party interaction can be adequately modeled by two-person games, in much the same way that functions with multiple arguments can be reduced to one-place functions and tupling. We will use such games to model interactions between a System and its Environment. One of the players in the game is taken to represent the System, and is referred to as Player or Proponent; the other represents the Environment and is referred to as Opponent. Note that the distinction between System and Environment and the corresponding designation as Player or Opponent depend on *point of view*:

> If Tom, Tim and Trevor converse in a room, then from Tom's point of view, he is the System, and Tim and Trevor form the Environment; while from Tim's point of view, he is the System, and Tom and Trevor form the Environment.

A single 'computation' or 'run' involving interaction between Player and Opponent will be represented by a sequence of *moves*, made alternately by Player and Opponent. We shall adopt the convention that *Opponent always makes the first move*. This avoids a number of technical problems which would otherwise arise, but limits what we can successfully model with games to the *negative fragment* of Intuitionistic Linear Logic. (This is the $\otimes$, $\multimap$, $\&$, $!$, $\forall$ fragment).

A game specifies the set of possible runs (or 'plays'). It can be thought of as a tree



where hollow nodes ○ represent positions where Opponent is to move; solid nodes • positions where Player is to move; and the arcs issuing from a node are labelled with the moves which can be made in the position represented by that node.

Formally, we define a game G to be a structure (M_G, λ_G, P_G), where

- M_G is the set of *moves* of the game;
- $\lambda_G : M_G \longrightarrow \{P, O\}$ is a labelling function designating each move as by Player or Opponent;
- $P_G \subseteq^{\text{nepref}} M_G^{\text{alt}}$, *i.e.* P_G is a non-empty, prefix-closed subset of M_G^{alt}, the set of alternating sequences of moves in M_G.

More formally, M_G^{alt} is the set of all $s \in M_G^*$ such that

$$\begin{aligned} \forall i : 1 \leq i \leq |s| \qquad & \text{even}(i) \Longrightarrow \lambda_G(s_i) = P \\ \wedge \quad & \text{odd}(i) \Longrightarrow \lambda_G(s_i) = O \end{aligned}$$

i.e.

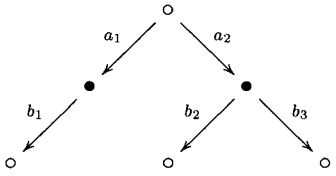
$$\begin{array}{cccccccc} s & = & a_1 & a_2 & \cdots & a_{2k+1} & a_{2k+2} & \cdots \\ \lambda_G & & \downarrow & \downarrow & & \downarrow & \downarrow & \\ & & O & P & & O & P & \end{array}$$

Thus P_G represents the game tree by the prefix-closed language of strings labelling paths from the root. Note that the tree can have infinite branches, corresponding to the fact that there can be infinite plays in the game. In terms of the representation by strings, this would mean that all the finite prefixes of some infinite sequence of moves would be valid plays.

For example, the game

$$\begin{array}{lcccccccccccc} (\{a_1, a_2, b_1, b_2, b_3\}, & \{ & a_1 & , & a_2 & , & b_1 & , & b_2 & , & b_3 & \}, & \{\epsilon, a_1, a_1b_1, a_2, a_2b_2, a_2b_3\}) \\ & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\ & & O & & O & & P & & P & & P & & \end{array}$$

represents the tree



We are using games to represents *types* (objects in the semantic category). A game can be seen as specifying the possible interactions between a System and its Environment. In the traditional interpretation of types as structured sets of some kind, types are used to classify *values*. By contrast, games classify *behaviours*. *Programs* will be modelled by *strategies*, *i.e.* rules specifying how the System should actually play.

Formally, we define a (deterministic) strategy σ on a game G to be a subset $\sigma \subseteq P_G^{\text{even}}$ of the game tree, satisfying:

- $\epsilon \in \sigma$
- $sab \in \sigma \implies s \in \sigma$
- $sab, sac \in \sigma \implies b = c$

To understand this definition, think of

$$s = a_1 b_1 \cdots a_k b_k \in \sigma$$

as a record of repeated interactions with the Environment following σ. It can be read as follows:

If the Environment initially does a_1,
then respond with b_1;
If the Environment then does a_2,
then respond with b_2;
$\vdots$
If the Environment finally does a_k,
then respond with b_k.

The first two conditions on σ say that it is a sub-tree of P_G of even-length paths. The third is a determinacy condition.

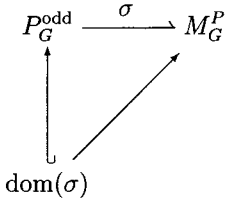
This can be seen as generalizing the notion of graph of a relation, *i.e.* of a set of ordered pairs, which can be read as a set of stimulus-response instructions. The generalization is that ordinary relations describe a single stimulus-response event only (giving rules for what the response to any given stimulus may be), whereas strategies describe repeated interactions between the System and the Environment. We can regard $sab \in \sigma$ as saying: 'when given the stimulus a in the context s,

respond with b'. Note that, with this reading, the determinacy condition generalizes the usual single-valuedness condition for (the graphs of) partial functions. Thus a useful slogan is:

"Strategies are (partial) functions extended in time."

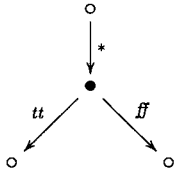
(*cf.* interaction categories (Abramsky, Gay and Nagarajan 1996b)).

Notation 2.1 We write $\mathrm{dom}(\sigma) = \{sa \mid \exists b.sab \in \sigma\}$, and then by the determinacy condition we have a well-defined partial function, which we shall also write as σ $(M_G^P = \lambda_G^{-1}(P))$:



Example 2.1 Let $\mathbb{B}$ be the game

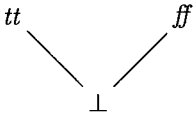
$$(\{*, tt, ff\}, \{* \mapsto O, tt \mapsto P, ff \mapsto P\}, \{\epsilon, *, *tt, *ff\})$$



This game can be seen as representing the data type of booleans. The opening move $*$ is a request by Opponent for the data, which can be answered by either tt or ff by Player. The strategies on $\mathbb{B}$ are as follows:

$$\{\epsilon\} \quad \mathrm{Pref}\{*tt\} \quad \mathrm{Pref}\{*ff\}$$

The first of these is the undefined strategy ('$\perp$'), the second and third correspond to the boolean values tt and ff. Taken with the inclusion ordering, this "space of strategies" corresponds to the usual flat domain of booleans:



Constructions on games

We will now describe some fundamental constructions on games.

Tensor Product

Given games A, B, we describe the tensor product $A \otimes B$.

$$\begin{array}{lcl} M_{A\otimes B} & = & M_A + M_B \\ \lambda_{A\otimes B} & = & [\lambda_A, \lambda_B] \\ P_{A\otimes B} & = & \{s \in M^{\mathrm{alt}}_{A\otimes B} \mid s{\restriction}M_A \in P_A \wedge s{\restriction}M_B \in P_B\} \end{array}$$

We can think of $A \otimes B$ as allowing play to proceed in *both* the subgames A and B in an interleaved fashion. It is a form of 'disjoint (*i.e.* non-communicating or interacting) parallel composition'.

A first hint of the additional subtleties introduced by the explicit representation of both System and Environment is given by the following result.

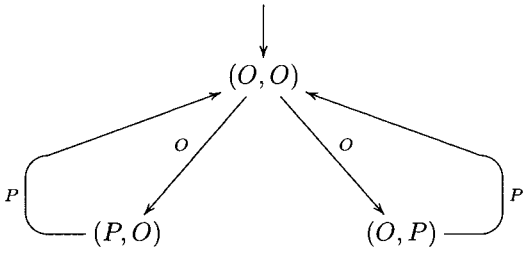
Proposition 2.1 *(Switching condition)*
In any play $s \in P_{A\otimes B}$, if successive moves s_i, s_{i+1} are in different subgames (i.e. *one is in A and the other in B), then $\lambda_{A\otimes B}(s_i) = P$, $\lambda_{A\otimes B}(s_{i+1}) = O$.*

In other words, only Opponent can switch from one subgame to another; Player must always respond in the same subgame that Opponent just moved in.

To prove this, consider for each $s \in P_{A\otimes B}$ the 'state'

$$\ulcorner s \urcorner = (\mathrm{parity}(s \restriction A), \mathrm{parity}(s \restriction B))$$

We will write O for even parity, and P for odd parity, since *e.g.* after a play of even parity, it is Opponent's turn to move. Initially, the state is $\ulcorner \epsilon \urcorner = (O, O)$. Note that O can move in either sub-game in this state. If O moves in A, then the state changes to (P, O). P can now only move in the first component. After he does so, the state is back to (O, O). Thus we obtain the following 'state transition diagram':



We see immediately from this that the switching condition holds; and also that the state (P, P) can never be reached (*i.e.* for no $s \in P_{A\otimes B}$ is $\ulcorner s \urcorner = (P, P)$).

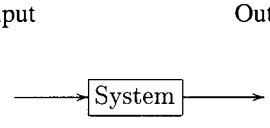
Linear Implication

Given games A, B, we define the game $A \multimap B$ as follows:

$$\begin{array}{lcll} M_{A\multimap B} & = & M_A + M_B & \\ \lambda_{A\otimes B} & = & [\overline{\lambda_A}, \lambda_B] & \text{where } \overline{\lambda}_A(m) = \begin{cases} P \text{ when } \lambda_A(m) = O \\ O \text{ when } \lambda_A(m) = P \end{cases} \\ P_{A\multimap B} & = & \{s \in M^{\mathrm{alt}}_{A\multimap B} \mid s \upharpoonright M_A \in P_A \wedge s \upharpoonright M_B \in P_B\} & \end{array}$$

This definition is *almost* the same as that of $A \otimes B$. The crucial difference is the inversion of the labelling function on the moves of A, corresponding to the idea that on the left of the arrow the rôles of Player and Opponent are interchanged.

If we think of 'function boxes', this is clear enough:



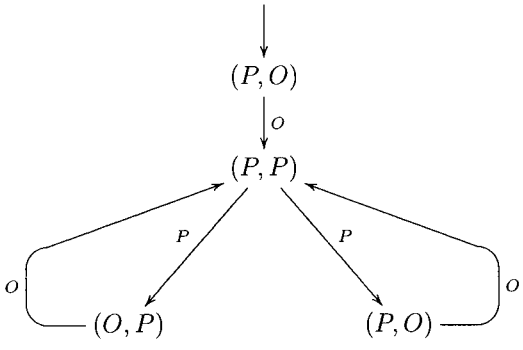
On the output side, the System is the producer and the Environment is the consumer; these rôles are reversed on the input side.

Note that $M^{\mathrm{alt}}_{A\multimap B}$, and hence $P_{A\multimap B}$, are in general quite different to $M^{\mathrm{alt}}_{A\otimes B}$, $P_{A\otimes B}$ respectively. In particular, the first move in $P_{A\multimap B}$ must always be in B, since the first move must be by Opponent, and all opening moves in A are labelled P by $\overline{\lambda_A}$.

We obtain the following switching condition for $A \multimap B$:

> If two consecutive moves are in different components, the first was by Opponent and the second by Player; so only Player can switch components.

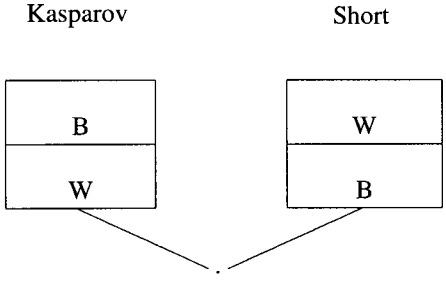
This is supported by the following state-transition diagram:



Example 2.2 The copy-cat strategy.

For any game A, we define a strategy on $A \multimap A$. This will provide the identity morphisms in our category, and the interpretation of logical axioms $A \vdash A$.

To illustrate this strategy, we undertake by the power of pure logic to beat either Kasparov or Short in chess. To do this, we play two games, one against, say, Kasparov, as White, and one against Short as Black. The situation is as follows:



We begin with the game against Short. He plays his opening move, and we play his move in our game against Kasparov. After Kasparov responds, we play his move as our response to Short. In this way, we *play the same game twice*, but *once as White* and *once as Black*. Thus, whoever wins, we win one game. Otherwise put, we act as a buffer process, indirectly playing Kasparov off against Short.

This copy-cat process can be seen as a 'dynamic tautology', by contrast with classical propositional tautologies, which are vacuous static descriptions of states of affairs. The logical aspect of this process is a certain 'conservation of flow of information' (which ensures that we win one game).

In general, a copy-cat strategy on A proceeds as follows:

	A	$\multimap$	A	
Time				
1			a_1	O
2	a_1			P
3	a_2			O
4			a_2	P
$\vdots$		$\vdots$		$\vdots$

$$\mathtt{id}_A = \{s \in P^{\mathrm{even}}_{A_1 \multimap A_2} \mid \forall t \text{ even-length prefix of } s : \; t{\restriction}A_1 = t{\restriction}A_2\}$$

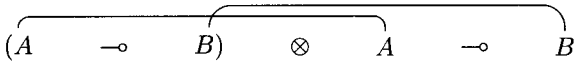
(Here, we write A_1, A_2 to index the two occurrences of A in $A \multimap A$ for ease of reference. Note also that we write $s \restriction A_1$ rather than $s{\restriction}M_{A_1}$. We will continue with both these notational "abuses").

We indicate such a strategy briefly by $\overbrace{A \multimap A}$, alluding to axiom links in the proof nets of Linear Logic.

Example 2.3 Application (*Modus Ponens*).

$$\mathtt{Ap}_{A,B} : (A \multimap B) \otimes A \multimap B$$

This is the conjunction of two copy-cat strategies



Note that A and B each occur once positively and once negatively in this formula; we simply connect up the positive and negative occurrences by 'copy-cats'.

$$\mathtt{Ap}_{A,B} = \{s \in P^{\text{even}}_{(A_1 \multimap B_1) \otimes A_2 \multimap B_2} \mid \forall t \text{ even-length prefix of } s : \; t\restriction A_1 = t\restriction A_2 \;\wedge\; t\restriction B_1 = t\restriction B_2\}$$

To understand this strategy as a protocol for function application, consider the following play:

	$(\ A \multimap B\ ) \otimes A \multimap B$
O	ro
P	ro
O	ri
P	ri
O	id
P	id
O	od
P	od

ro — request for output
ri — request for input
id — input data
od — output data

The request for output to the application function is copied to the output side of the function argument; the function argument's request for input is copied to the other argument; the input data provided at the second argument is copied back to the function argument; the output from the function argument is copied back to answer the original request. It is a protocol for *linear* function application since the state of both the function and the argument will change as we interact with them; we have no way of returning to the original state. Thus we "consume" our "resources" as we produce the output. In this way there is a natural match between game semantics and linear logic.

The Category of Games $\mathcal{G}$

- Objects: Games
- Morphisms: $\sigma : A \longrightarrow B$ are strategies σ on $A \multimap B$.
- Composition: interaction between strategies.