A TYPE ASSIGNMENT SYSTEM FOR THE GAME SEMANTICS

PIETRO DI GIANANTONIO
Dipartimento di Matematica e Informatica
Università di Udine
ITALY
E-mail: digianantonio@dimi.uniud.it

GIANLUCA FRANCO
Dipartimento di Matematica e Informatica
Università di Udine
ITALY
E-mail: gfranco@dimi.uniud.it

In this paper an alternative description of the game semantics for the untyped lambda calculus is given. More precisely, we introduce a finitary description of lambda terms. This description turns out to be equivalent to a particular game denotational semantics of the lambda calculus.

Introduction

The aim of this paper is to give an alternative description of the game semantics ([2,3]). The main idea is to mimic, in the game semantics setting, a duality existing in domain theory. In domain theory, the denotational semantics of a language can be given in two forms: a term can be interpreted by some point in a particular domain, or by a set of properties. The Stone-duality theorems, for domain theory [1], tell us that these two alternative descriptions are equivalent. In this setting, the properties of a term are normally called "types" and a set of rules, that allow to derive the properties satisfied by a term, is called "type assignment system". In the literature type assignment systems are mainly used in the semantics of lambda calculi ([5,6]). But, in principle, they can be used as an alternative description of any kind of denotational semantics, based on domain theory. Type assignment systems provide a concrete finitary way for defining the semantics of a term.

In this paper we introduce a finitary description of λ-terms in a particular game model. We think that type assignment systems are useful in the context of game semantics since they provide a more concrete and intuitive account of the interpretation of terms given by game models. In fact, classically game semantics is given using a categorical definition. To derive the concrete definition from the categorical one can be a tiring task.

In Section 1 a general class of extensional game models for the untyped λ-calculus is introduced with the categorical semantics of the language. All these models, as proved in [7], induce the same equational theory on λ-terms. Hence a particular element of this class is chosen as the canonical model and presented in Section 2. In Section 3 the main purpose of the paper is realized. An alternative description of the interpretation of λ-terms in the game model is given by means of a type assignment system. The equivalence between this description and the categorical one is then stated in Section 4. Final remarks and hints for future work are given in Section 5.

1 Game models for the λ-calculus

In this paper we will not give the basic definitions of the game semantics. We refer to [3,2] for an exhaustive presentation of the subject.

We will use the categories of games $\mathcal{G}$ and $K_!(\mathcal{G})$, introduced in [3]. These categories allow for the construction of recursive objects, *i.e.* objects which are fixed points of functors. A first method to construct reflexive objects is introduced in [4]. The aim of this method is to construct initial fixed points of functors. In [7] the above method is extended to deal also with non-initial fixed points, and, hence, allowing for the construction of game models of untyped λ-calculus.

In the first method a fixed point D of a functor is obtained as the limit of a chain of approximant games $D_0, D_1, D_2, \ldots$, where each D_n is a subgame ([4]) of the game D_{n+1} (written $D_n \trianglelefteq D_{n+1}$). The extended method differs from the first one in requiring a weaker relation between D_n and D_{n+1}: it suffices that D_n is isomorphic to a subgame of D_{n+1}.

We briefly recall the extended method presented in [7]. To do this we introduce a new category of games: the category $\mathcal{G}^e$.

Definition 1 *Given games A and B an* embedding $f : A \hookrightarrow B$ *is a total injective function $f : M_A \to M_B$ such that:*

- $\lambda_A = \lambda_B \circ f$
- $f^\star(P_A) = P_B \cap (f(A))^\star$
- $s \approx_A s'$ *iff* $f^\star(s) \approx_B f^\star(s')$

In the above we have used the notation $f^\star$ to denote the natural extension of f both to sequences and sets of sequences.

Definition 2 *The category of games $\mathcal{G}^e$ has as objects games and as morphisms embeddings.*

Proposition 3 *The category $\mathcal{G}^e$ is ω-cocomplete.*

Proof. Given an ω-chain $\langle D_n, f_n\rangle$ with $f_n : D_n \hookrightarrow D_{n+1}$, its colimit is the cocone $\langle D_\infty, \mu_n\rangle_{n\in\omega}$, where D_∞ is the game:

- $M_{D_\infty} = (\bigcup_{n\in\omega} M_{D_n})/_{\equiv}$
  where $\equiv$ is the least equivalence relation such that
  $$\forall n \in \omega\ \forall a \in D_n\ \forall b \in D_{n+1}.\ f_n(a) = b\ \Rightarrow\ a \equiv b.$$
- $\lambda_{D_\infty}([a]_\equiv) = \lambda_{D_n}(a)$ if $a \in D_n$
- $P_{D_\infty} = \bigcup_{n\in\omega}\{[a_1]_\equiv[a_2]_\equiv \dots [a_p]_\equiv \mid a_1a_2\dots a_p \in P_{D_n}\}$
- $\approx_{D_\infty} = \bigcup_{n\in\omega} \{([a_1]_\equiv[a_2]_\equiv \dots [a_p]_\equiv, [a'_1]_\equiv[a'_2]_\equiv \dots [a'_p]_\equiv) \mid (a_1a_2\dots a_p, a'_1a'_2\dots a'_p) \in \approx_{D_n}\}$

The colimit functions $\mu_n : D_n \hookrightarrow D_\infty$ are defined by $\mu_n(a) = [a]_\equiv$. □

Several functors in $\mathcal{G}$ can be also defined in $\mathcal{G}^e$. A sufficient condition for this to happen is the following.

Definition 4 *A functor $F : \mathcal{G} \to \mathcal{G}$ is* e-extensible *if for each pair of isomorphic objects A and B in $\mathcal{G}^e$, $F(A)$ and $F(B)$ are isomorphic objects in $\mathcal{G}^e$.*

Definition 5 *$\mathcal{G}^i$ is the subcategory of $\mathcal{G}^e$ defined as follows:*

- *the objects of $\mathcal{G}^i$ are the same of $\mathcal{G}^e$;*
- *the morphisms $f : A \to B$ of $\mathcal{G}^i$ are the isomorphisms $f : A \hookrightarrow B$ of $\mathcal{G}^e$.*

Lemma 6 *Let $F : \mathcal{G} \to \mathcal{G}$ be an e-extensible functor. We can define a derived functor $F^i : \mathcal{G}^i \to \mathcal{G}^i$.*

Proof. The action of F^i on objects is the same of F. The action on morphisms is defined in two steps.

1. First choose for each pair of objects A and B in $\mathcal{G}^i$ a "canonical" isomorphism $f_{AB} : A \to B$, in such a way that $f_{AC} = f_{BC} \circ f_{AB}$ for every A, B, C.
2. For each isomorphism $g \in \mathcal{G}^i(A, B)$,‘ let $F^i(g) : F(A) \to F(B)$ be the canonical isomorphism $f_{F(A)F(B)}$. □

From the above we can state the following proposition.

Proposition 7 *Let $F : \mathcal{G} \to \mathcal{G}$ be an e-extensible functor which is monotonic w.r.t. $\trianglelefteq$. We can define a functor $F^e : \mathcal{G}^e \to \mathcal{G}^e$ as follows:*

- $F^e(A) = F(A)$
- *given an embedding $f : A \hookrightarrow B$, let $f' : A \hookrightarrow img\ f$ be the isomorphism obtained restricting f to the image of f, and $i : F(img\ f) \to F(B)$ be the inclusion function; then put $F^e(f) = i \circ F^i(f')$.*

It is not difficult to prove that

Proposition 8 *Every e-extensible functor $F : \mathcal{G} \to \mathcal{G}$, which is continuous w.r.t $\trianglelefteq$, induces a continuous functor F^e in $\mathcal{G}^e$.*

One can easily see that the functors &, $\otimes$, $\multimap$, ! are e-extensible. Notice that the functor $(\multimap)^e$ is covariant in each of its arguments. In particular, given games A and B with move sets M_A and M_B, the action of these functors on the move sets is the following:

- $M_{A \otimes B} = M_{A \multimap B} = M_{A \& B} = M_A + M_B = \{(0, a) \mid a \in M_A\} \cup \{(1, b) \mid b \in M_B\}$;
- $M_{!A} = \sum_\omega M_A = \omega \times M_A = \{(n, a) \mid n \in \omega,\ a \in M_A\}$.

while the action on the embeddings is defined as follows. Let $f : A \hookrightarrow B$ and $g : A' \hookrightarrow B'$ be two embeddings. Then:

- $f \otimes g = f\ \&\ g = f \multimap g = f + g$
- $!f = id \times f$

where $\times$ and $+$ are respectively the product and coproduct functors in $\mathcal{SET}$.

Each embedding $f : A \hookrightarrow B$ in $\mathcal{G}^e$ induces two morphisms $f^+ : A \multimap B$ and $f_- : B \multimap A$ in $\mathcal{G}$ defined as follows.

Definition 9 *Given an embedding $f : A \hookrightarrow B$, put*

$$f^+ = \{t \in P_{A \multimap circ B} \mid t \in s_f\}$$

$$f_- = \{t' \in P_{B \multimap A} \mid t' \in s_f\}$$

where s_f is the least set satisfying:

$$s_f = \{t\, a\, f(a) \mid t \in s_f,\ a \in M_A\} \cup \{t' f(a)\, a \mid t' \in s_f,\ a \in M_A\} \cup \{\epsilon\}.$$

Since $(f \circ g)^+ = f^+ \circ g^+$ and $(f \circ g)^- = g^- \circ f^-$ the category $\mathcal{G}^e$ is indeed isomorphic to a subcategory of $\mathcal{G}$ and to a subcategory of $\mathcal{G}^{op}$.

Now, using the well-known machinery, we can obtain fixed points of an e-extensible functor F, continuous w.r.t. $\trianglelefteq$, and hence solutions of the equation $D \simeq F(D)$ in $\mathcal{G}$.

Theorem 10 *Given a game D and an embedding $f : D \hookrightarrow F(D)$, let $\langle D_\infty, \mu_n \rangle_{n\in\omega}$ be the colimit of the chain $\langle (F^e)^n(D), (F^e)^n(f) \rangle_{n\in\omega}$. Then, the game D_∞ is the fixed point of the functor F^e. The isomorphic embeddings $\varphi : D_\infty \hookrightarrow F^e(D_\infty)$ and $\psi : F^e(D_\infty) \hookrightarrow D_\infty$ are given by $\varphi = \bigsqcup_{n\in\omega} F^e(\mu_n) \circ \mu_n^{-1}$ and $\psi = \bigsqcup_{n\in\omega} \mu_n \circ F^e(\mu_n)^{-1}$ respectively, where the lubs are taken in the category of* partial *embeddings.*

2 A game model for the $\lambda\beta\eta$-calculus

An extensional model for the untyped λ-calculus is an *extensional reflexive object* in a cartesian closed category, that is an object D s.t. $D \simeq D \to D$. The category $K_!(\mathcal{G})$ is cartesian closed, the product functor is & and the "functional space" $A \to B$ is internalized by the game $!A \multimap B$.

In $K_!(\mathcal{G})$ an extensional model for the untyped λ-calculus can be build as follows:

Definition 11 *Let D_0^* be the game: $D_0^* = (\{*\}, \lambda(*) = OQ, \{\epsilon, *\}, id)$ and let $f_* : D_0^* \hookrightarrow (!D_0^* \multimap D_0^*)$ be the embedding defined by $f_*(*) = (1, *)$. The extensional reflexive object D^* is defined to be the limit of the chain generated by the embedding f_* and the functor $G(D) = (!)^e D (\multimap)^e D$.*

We now give the standard denotational semantics for pure λ-terms.

Definition 12 *Given a reflexive object D in the cartesian closed category $K_!(\mathcal{G})$, and given a λ-term M, whose free variables are among the list $\Delta = \{x_1, \ldots, x_n\}$, its interpretation in D is the strategy $[\![M]\!]^D_\Delta : !(\overbrace{D \;\&\; \ldots \;\&\; D}^{n}) \multimap D$ defined inductively as follows:*

$$[\![x_i]\!]^D_\Delta = \pi_i^\Delta;$$
$$[\![MN]\!]^D_\Delta = ev \circ \langle (\varphi \circ [\![M]\!]^D_\Delta) \;,\; [\![N]\!]^D_\Delta \rangle;$$
$$[\![\lambda x.M]\!]^D_\Delta = \psi \circ \Lambda([\![M]\!]^D_{\Delta,x});$$

where π_i^Δ are the canonical projection morphisms, $(\varphi, \psi) : D \to (!D \multimap D)$ are the isomorphisms characterizing the reflexive object D, and ev, Λ denote "evaluation" and "abstraction" in the cartesian closed category $K_!(\mathcal{G})$.

3 The type assignment system

In the game semantics, positions of games are particular instances of a computation. If a position s belongs to the strategy interpreting the term M, it means that the term M, in a suitable environment, can perform the computation steps described by s.

The underlying idea of the type assignment system we present, is to consider positions, that is the instances of computation, as the properties (types) to be used in describing programs. This approach is possible because the set of instances of the computation that a program M can perform, describe completely the strategy that interprets M.

In the specific case that we consider, we define a set of rules that allow to derive judgments in the form

$$\vdash_\Delta M : s$$

where M is a λ-term whose free variables are in the list $\Delta = \{x_1, \ldots x_n\}$ and s is a position in the game $!(\overbrace{D^* \ \& \ \ldots \& \ D^*}^{n}) \multimap D^*$.

It is worth noticing that the environment never appears in a judgment, since it is included in the position itself. In fact, a position in the game $!(\overbrace{D^* \ \& \ \ldots \& \ D^*}^{n}) \multimap D^*$ can contain also the moves describing the interaction of a term with the environment. The traditional form of the judgment for the type assignment system, that is $\Gamma \vdash M : s$, with Γ a description of the environment, cannot be used in our case. This because, using the above traditional form of judgment, a term is described by an extensional function from the values of the free variables of the term to the value of the term itself. Since the category of games is not concrete, this extensional description is not adequate.

In order to give a type assignment system, it is convenient to define a suitable notation for the moves of the game model, *i.e.* D^*. We choose to indicate moves by sequences of labels, and in this case, we use juxtaposition to indicate sequence concatenation.

Notation Let a, b be moves of games A and B respectively, and $i \in \omega$. Then we indicate with la and rb the moves $(0, a)$ and $(1, b)$ in the game $A \multimap B$; with ia the move (i, a) of the game $!A$; if $i < n$, then, we indicate with ia also the obvious move in the game $(\overbrace{A \ \& \ \ldots \ \& \ A}^{n})$.

Using the construction presented in the proof of Proposition 3, moves in

the game D^* are equivalence classes of moves in the games D^*_i, $i \in \omega$. Using the above notation, each move in D^*_i can be denoted by a finite sequence of labels ending with $*$. Moreover the move $a*$ in D^*_i belongs to the same equivalence class of the move $ar*$ in D^*_{i+1}. In order to have a unique representation for each move in D^*, we denote the equivalence class, containing the moves $a*$, $ar*$, $arr*$, ... in D^*_i, D^*_{i+1}, D^*_{i+2}, ... respectively, by the infinite sequence ar^ω; that is we denote moves in D^* by infinite sequences of labels in the set $\{l, r\} \cup \omega$, having just a finite set of labels different from r. Notice that, as a consequence, a move $a \in D^*$ and the corresponding move $\varphi(a) \in !D^* \multimap D^*$ are denoted in the same way.

For what concerns positions, we will use the symbol $\cdot$ to indicate concatenation of positions. The use of two different symbolisms for concatenation, one for moves and one for positions, allow us to omit parenthesis.

In order to present the type assignment system we need to define the following (partial) functions on sequences of moves.

Definition 13 *1. Let s be a sequence of moves of the game D^* (not necessarily giving rise to a valid position), let $\Delta = \{x_1, \dots x_n\}$ be a list of variables with $n \geq 1$, and let $0 < i \leq n$ be a natural number. The sequences $\mathtt{cc}^i_\Delta(s)$ and $\overline{\mathtt{cc}}^i_\Delta(s)$ of moves of the game $!(\overbrace{D^* \;\&\; \dots \;\&\; D^*}^{n}) \multimap D^*$, are defined as follows:*

(a) $\mathtt{cc}^i_\Delta(\epsilon) = \overline{\mathtt{cc}}^i_\Delta(\epsilon) = \epsilon$

(b) $\mathtt{cc}^i_\Delta(a \cdot s') = ra \cdot l0ia \cdot \overline{\mathtt{cc}}^i_\Delta(s')$

(c) $\overline{\mathtt{cc}}^i_\Delta(a \cdot s') = l0ia \cdot ra \cdot \mathtt{cc}^i_\Delta(s')$

2. Let λ be the function from moves in the game $!(\overbrace{D^* \;\&\; \dots \;\&\; D^*}^{n+1}) \multimap D^*$ to moves in $!(\overbrace{D^* \;\&\; \dots \;\&\; D^*}^{n}) \multimap (!D^* \multimap D^*)$ defined as follows:

$$\lambda(a) = \begin{cases} a & \text{if } a = ljia' \text{ and } i \leq n \\ rlja' & \text{if } a = lj(n+1)a' \\ ra & \text{if } a = ra' \end{cases}$$

Let $\lambda^\star$ be the extension of λ to positions.

3. *Let s be a sequence of moves in the game $!(\overbrace{D^* \,\&\, \ldots \,\&\, D^*}^{n}) \multimap (!D^* \multimap D^*)$, $t_1, \ldots, t_m$ be sequences of moves in the game $!(\overbrace{D^* \,\&\, \ldots \,\&\, D^*}^{n}) \multimap D^*$ and $j \leq n$, we indicate with $(s|t_1|\ldots|t_m, j)$ a sequence of the game $!(\overbrace{D^* \,\&\, \ldots \,\&\, D^*}^{n}) \multimap D^*$, defined as follows:*

   (a) $(\epsilon|\epsilon|\ldots|\epsilon, i) = \epsilon$;

   (b) $(rra \cdot s|t_1|\ldots|t_m, 0) = ra \cdot (s|t_1|\ldots t_m, 0)$;

   (c) $(rlia \cdot s|t_1|\ldots|ra \cdot t'_i|\ldots|t_m, 0) = (s|t_1|\ldots|t'_i|\ldots|t_m, i)$;

   (d) $(rlia \cdot s|t_1|\ldots|ra \cdot t'_i|\ldots|t_m, i) = (s|t_1|\ldots|t'_i|\ldots|t_m, 0)$;

   (e) $(lhja \cdot s|t_1|\ldots|t_m, 0) = l(2h+1)ja \cdot (s|t_1|\ldots|t_m, 0)$;

   (f) $(s|t_1|\ldots|lhja \cdot t'_i|\ldots|t_m, i) = lh(2p(i,j))a \cdot (s|t_1|\ldots|t'_i|\ldots|t_m, i)$

   where $p : \omega \times \omega \to \omega$ is a pairing function, that is a bijection between $\omega \times \omega$ and ω. Take, for example, the function $\lambda x.\lambda y.(2x+1)2^y$. Observe that the sequence $(s|t_1|\ldots|t_m, j)$ is not always defined. If it is defined we write $(s|t_1|\ldots|t_m, j)\downarrow$.

Some explanation for the definitions above. Observe that if s is a valid position in the game D^*, then $\mathtt{cc}^i_\Delta(s)$ is a valid position in the game $!(\overbrace{D^* \,\&\, \ldots \,\&\, D^*}^{n}) \multimap D^*$. Moreover one can easily prove that $\mathtt{cc}^i_\Delta(s)$ is a position of the "copy-cat" strategy on the i^{th} argument ([2]), and that every position of the copy-cat strategy on the i^{th} argument can be obtained in this way.

It is not difficult to observe that the function λ defines the "currying" isomorphism between the games $!(\overbrace{D^* \,\&\, \ldots \,\&\, D^*}^{n+1}) \multimap D^*$ and the game $!(\overbrace{D^* \,\&\, \ldots \,\&\, D^*}^{n}) \multimap (!D^* \multimap D^*)$.

The position $(s|t_1|\ldots|t_m, 0)$ is the result, as defined by game semantics, of the interaction between the position s, seen as the instance of computation of a function, and the positions $t_1, \ldots, t_m$, seen as several instances of computation of the argument of s. Since a function can interrogate its argument several times, more than one instance of computation for the argument is necessary. The instance of computation $s, t_1, \ldots, t_m$ can also interact with an

environment represented by the game $!(\overbrace{D^* \;\&\; \ldots \;\&\; D^*}^{n})$. In the expression $(s|t_1|\ldots|t_m, i)$ the index i is necessary to indicate which instance of computation has to move.

Rule (b) considers the case where s has to move and makes a move a on the result; in this case a appears also on the result of the interaction. Rule (c) considers the case where s has to move and makes a move a on the i^{th} copy of the argument. If t_i can make the same move a, then there is an interaction between function and arguments, the two instances of a are canceled and the control is passed to the i^{th} copy of the argument. If s and t_i do not agree on the move to make, it means that there cannot be any interaction between these particular instances of computation. Rule (d) is the dual of rule (c); it considers the case where t_i has to move and makes a move a. If s can make the same move a then there is an interaction between function and arguments, the two instances of a are cancelled and the control is given to the function. Rule (e) considers the case where s has to move and makes a move a in correspondence of the h^{th} copy of the j^{th} variable of the environment. In this case, the move a appears also on the result of the interaction, but in a different copy of the environment. Rule (f) is similar to the rule (e), and considers the case where the i^{th} copy of the argument is in control.

Definition 14 *Let M, N be λ-terms and let Δ be a list of variables. We give the following type assignment rules in a natural deduction system style:*

$$\frac{s \in P_{D^*}}{\vdash_\Delta x_i : \mathsf{cc}^i_\Delta(s)} \quad (var)$$

$$\frac{\vdash_{\Delta,x} M : s}{\vdash_\Delta \lambda x.M : \lambda^\star s} \quad (\lambda)$$

$$\frac{\vdash_\Delta M : s \quad \vdash_\Delta N : t_1 \ldots \vdash_\Delta N : t_m}{\vdash_\Delta MN : (s|t_1 \ldots |t_m, 0)} \quad (app)$$

the side condition of the rule (app) being that $(s|t_1|\ldots|t_m, 0)\downarrow$.

4 Equivalence of semantics

We shall show now that game denotational semantics and the semantics given by the type assignment system are equivalent.

Theorem 15 *For every λ-term M whose free variables are among the list*

$\Delta = \{x_1, \ldots, x_n\}$, *we have:* $[\![M]\!]^{D^*}_{\Delta} \approx \{s \mid \vdash_{\Delta} M : s\}$.
Proof. By induction on the structure of M.

- $M \equiv x_i$. In this case $[\![x_i]\!]^{D^*}_{\Delta} = \pi^{\Delta}_i$. By explicating the categorical definition of $\pi^{\Delta}_i : !(\overbrace{D^* \,\&\, \ldots \,\&\, D^*}^{n}) \multimap D^*$, one can observe that π^{Δ}_i is the "copy-cat" strategy on the i^{th} argument, that is the same strategy defined by the type assignment system.

- $M \equiv \lambda x.P$. In this case $[\![\lambda x.P]\!]^{D^*}_{\Delta} = \psi \circ \Lambda([\![P]\!]^{D^*}_{\Delta,x})$.

  In our sequence notation for moves, we use the same sequence to denote a move d in D^* and the equivalent (unfolded) move in $!D^* \multimap D^*$. It follows that, in our notation, the strategies $\psi \circ \Lambda([\![P]\!]^{D}_{\Delta,x})$ and $\Lambda([\![P]\!]^{D}_{\Delta,x})$ are denoted in the same way.

  It is not difficult to prove that the isomorphism between the game $!(\overbrace{D^* \,\&\, \ldots \,\&\, D^*}^{n+1}) \multimap D^*$ and the game $!(\overbrace{D^* \,\&\, \ldots \,\&\, D^*}^{n}) \multimap (!D^* \multimap D^*)$ defined by the function Λ coincides with the one defined by the function $\lambda^\star$.

- $M \equiv PQ$. In this case we have: $[\![PQ]\!]^{D^*}_{\Delta} = ev \circ \langle (\varphi \circ [\![P]\!]^{D^*}_{\Delta}) \,,\, [\![Q]\!]^{D^*}_{\Delta} \rangle$.

  By the same arguments used in the previous point, in our notation the strategies $\varphi \circ [\![P]\!]^{D^*}_{\Delta}$ and $[\![P]\!]^{D^*}_{\Delta}$ are denoted in the same way. The remaining part of the proof can be obtained by explicating the categorical definition of ev. □

5 Final remarks

This case study, shows that also in the game setting it is possible to give a finitary description of the denotational semantics of the untyped λ-calculus. It is immediate to extend the methodology developed in this paper to all the models in the extensional class described in Section 1. It would be interesting to extend it also to other contexts, *e.g.* the language PCF, the lazy λ-calculus.

This purpose is the natural development of this work and we plan to achieve it in future papers.

References

1. S. Abramsky. Domain theory in logical form. In *Annals of Pure and Applied Logic*, volume 51, pages 1-77, 1991.

2. S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543-574, June 1994.
3. S. Abramsky and R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. Ftp-available at `http://www.dcs.ed.ac.uk/home/samson`, 1995.
4. S. Abramsky and G. McCusker. Games for recursive types. In I.C.Mackie C.L. Hankin and R. Nagarajan, editors, *Theory and Formal Methods of Computing 1994: Proceedings of the Second Imperial College Department of Computing Workshop on Theory and Formal Methods*. Imperial College Press, October 1995.
5. H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931-940, 1983.
6. M. Coppo, M. Dezani-Ciancaglini, F. Honsell, and G. Longo. Extended type structures and filter λ-models. In G. Lolli G. Longo and A. Marcja, editors, *Logic Colloquium '82*. Elsevier Science Publishers, 1984.
7. P. Di Gianantonio, G. Franco, and F. Honsell. Game semantics for the untyped lambda calculus. Ftp-available at `http://www.dimi.uniud.it/~pietro/Papers/paper_arg.html`, 1998.