A Type Assignment System for Game Semantics ⋆

Pietro Di Gianantonio, Furio Honsell, Marina Lenisa [a]

[a] *Dipartimento di Matematica e Informatica, Università di Udine*
via delle Scienze 206, 33100 Udine, Italy.
{digianantonio,honsell,lenisa}@dimi.uniud.it

dedicated to Mariangiola, Mario and Simona,
on the occasion of their 60^{th} *birthdays*

Abstract

We present a *type assignment system* that provides a *finitary* interpretation of lambda terms in a *game semantics* model. Traditionally, type assignment systems describe the semantic interpretation of terms in domain theoretic models. Quite surprisingly, the type assignment system presented in this paper is very similar to the traditional ones, the main difference being the omission of the subtyping rules.

Key words: Lambda Calculus, Game Semantics, Type Assignment System

1 Introduction

About twentyfive years ago, Mario Coppo, Mariangiola Dezani, and their group, started to provide *logical descriptions* of models of λ-calculus, in terms of *intersection type assignment systems*, [8,10,7,15]. This logical approach was related explicitly to Scott Information Systems in [9], and put on firm categorical grounds by Abramsky in [1]. In this paper, we present a logical analysis of game models in the style of intersection types. We feel that it provides new insights both in the semantics of λ-calculus and in the fine structure of game semantics. Thus we show that the idea underpinning intersection types is an outstanding contribution to Theoretical Computer Science, which allows to reap fruitful results in any semantical framework.

⋆ Work partially supported by UE Project IST-CA-510996 Types, and by Italian MIUR Project PRIN 2005015824 ART.

The intersection type approach can be outlined as follows. The semantics of a programming language can be given in two forms: a term can be interpreted either *denotationally* by a point in a particular domain, or *logically* by a set of *properties*. *Stone-duality*, as presented in [1], establishes an equivalence between these two alternate descriptions for suitable categories of domains. In this approach, properties of terms are normally called "types". The logical semantics consists of the set of rules, called "type assignment system", which allow to derive the properties satisfied by a term. Type assignment systems can be seen to provide concrete, *finitary* approximations of the semantics of a term.

Differently from the standard case, in type assignment systems for game semantics, a type cannot describe simply the input-output behavior of a term, but it needs to describe a more detailed *interaction* of the term with the environment. In particular, a type t for a term M describes a set of moves that the Proponent and the Opponent may exchange in some phases of the interaction of the term M with the environment. Quite surprisingly, the syntax for standard intersection types is used to describe sets of moves. The game-theoretic perspective is achieved by removing all structural and congruence rules from standard assignment systems. In our framework, no form of weakening rule is present and the types $(t_0 \wedge t_1) \wedge t_2$, $t_0 \wedge (t_1 \wedge t_2)$, $t_0 \wedge (t_2 \wedge t_0)$ are all distinct.

In this paper, we consider the game semantics framework presented by Abramsky, Jagadeesan and Malacaria in [4], and known as AJM-games. The strategies that game intersection types generate are naturally *history-free*, the $\wedge$ operator is used to model the exponential construction, the lack of associativity and commutativity rules for $\wedge$ is connected to the use of indexes to distinguish different instances of moves in a exponential type. As for AJM-games, it is necessary to introduce a partial equivalence relation on interpretations to recover subject reduction, due to the arbitrariness in the use of indexes.

In this paper, we focus on simply typed λ-calculus. We define a game λ-model in the standard way in a category of games and history-free strategies, and we introduce an intersection-like type system for describing such a game model. Our approach to game intersection types is "typed", i.e. intersection types are built inductively over games. The usual untyped intersection semantics can be recovered as a special case of the typed case. As already mentioned, in our setting, types on a game A represent sets of Opponent and Proponent moves on A. The intended meaning of a judgment in our typing system is that a set of equal number of Opponent and Proponent moves appear in the history-free strategy interpreting the term in the given environment. Moreover, the moves in this set may be exchanged during the interaction between that term and that environment. The main point which allows to establish a bridge between the intersection-like types that we introduce and game semantics is that history-free strategies induce *partial functions* from Opponent to

Proponent moves, in a *Geometry of Interaction* (GoI) fashion, [13,2,3]. Under this perspective, the most informative judgments are those involving *step types*, where exactly two moves appear, an Opponent move and the Proponent reaction move in the graph of the partial function defining this strategy. The main result of this paper amounts to the fact that the intersection type semantics of a given term in context induces a partial function from Opponent to Proponent moves, which defines the strategy interpreting the term in the game model.

Our approach to game intersection types is quite general. In particular, type assignment systems for GoI combinatory algebras in "particle-style" [3] can be easily derived from game type assignment systems, simply by forgetting the distinction between Opponent and Proponent moves.

Type assignment systems like the one presented in this paper are quite useful in the context of game semantics, since they provide a more concrete and intuitive account of the interpretation of terms w.r.t. categorical game models. In fact, deriving a concrete definition from a categorical one can be a heavy task.

The problem of giving a concrete and finitary description of game models has been also investigated in [11], where a type assignment system describing the game model of the untyped λ-calculus of [12] has been presented. However, the approach of [11] is different and more directly connected to the representation of strategies as sets of plays and to the categorical combinators involved in the game semantics.

Synopsis. In Section 2, we recall basic notions on games and strategies, we present a new alternative definition of the exponential game, and we discuss the representation of history-free strategies as partial functions. In Section 3, we present syntax and game semantics of the simply typed λ-calculus, which we use as target language. In Section 4, we introduce and study a type assignment system giving a finitary description of the game model of Section 3. In Section 5, we establish the connection between the type assignment system and the game model for the simply typed λ-calculus of Section 3. Finally, in Section 6, we discuss further developments.

2 Game Categories

In this section, first we recall basic notions and constructions on games and strategies in the style of [4]. Then, in Section 2.1, we present an alternative construction of the exponential game, which will be useful in order to study

the connections between our typing semantics and the game semantics. To the same purpose, in Section 2.2, we discuss the alternative representation of history-free strategies as *partial functions* from Opponent to Proponent moves.

The following are the usual definitions of game and strategy in the style of [4]:

Definition 2.1 (Games) *A* game *has two participants: the* Proponent *and the* Opponent. *A game A is a quadruple $(M_A, \lambda_A, P_A, \approx_A)$ where*

- *M_A is the set of* moves *of the game.*
- *$\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ is the* labeling *function: it tells us if a move is taken by the Opponent or by the Proponent, and if it is a Question or an Answer. We can decompose λ_A into $\lambda_A^{OP} : M_A \to \{O, P\}$ and $\lambda_A^{QA} : M_A \to \{Q, A\}$ and put $\lambda_A = \langle \lambda_A^{OP}, \lambda_A^{QA} \rangle$. We denote by $^-$ the function which exchanges Proponent and Opponent, i.e. $\overline{O} = P$ and $\overline{P} = O$. We also denote with $\overline{\lambda_A^{OP}}$ the function defined by $\overline{\lambda_A^{OP}}(a) = \overline{\lambda_A^{OP}(a)}$. Finally, we denote with $\overline{\lambda_A}$ the function $\langle \overline{\lambda_A^{OP}}, \lambda_A^{QA} \rangle$.*
- *P_A is a non-empty and prefix-closed subset of the set $M_A^{\circledast}$ (written as $P_A \subseteq^{nepref} M_A^{\circledast}$), where $M_A^{\circledast}$ is the set of all sequences of moves which satisfy the following conditions:*
  - *$s = at \Rightarrow \lambda_A(a) = OQ$*
  - *$(\forall i : 1 \leq i \leq |s|)[\lambda_A^{OP}(s_{i+1}) = \overline{\lambda_A^{OP}(s_i)}]$*
  - *$(\forall\ t \sqsubseteq s)[|t \upharpoonright M_A^A| \leq |t \upharpoonright M_A^Q|]$*

  where M_A^A and M_A^Q denote the subsets of game moves labeled respectively as Answers and as Questions, $s \upharpoonright M$ denotes the set of moves of M which appear in s and $\sqsubseteq$ is the substring relation. P_A denotes the set of positions *of the game A.*
- *$\approx_A$ is an equivalence relation on P_A which satisfies the following properties:*
  - *$s \approx_A s' \Rightarrow |s| = |s'|$*
  - *$sa \approx_A s'a' \Rightarrow s \approx_A s'$*
  - *$s \approx_A s'$ & $sa \in P_A \Rightarrow (\exists a')[sa \approx_A s'a']$*

In the above s, s', t and t' range over sequences of moves, while a, a', b and b' range over moves. The empty sequence is written ϵ.

In a position, questions and answers match together like open and closed parentheses in an algebraic expression.

Definition 2.2 (History-free Strategies) *A* strategy *for the Proponent in a game A is a non-empty set $\sigma \subseteq P_A^{even}$ of positions of even length such that $\overline{\sigma} = \sigma \cup dom(\sigma)$ is prefix-closed, where $dom(\sigma) = \{t \in P_A^{odd} \mid (\exists a)[ta \in \sigma]\}$, and P_A^{odd} and P_A^{even} denote the sets of positions of odd and even length respectively. A strategy σ for a game A is* history-free *if it satisfies the following properties:*

(1) $sab, tac \in \sigma \Rightarrow b = c$
(2) $sab, t \in \sigma, ta \in P_A \Rightarrow tab \in \sigma$

A strategy can be seen as a set of rules which tells the Proponent which move to take after the last move by the Opponent. History-free strategies are strategies which depend only on the *last* move by the Opponent.

The equivalence relation on positions $\approx_A$ can be extended to strategies in the following way.

Definition 2.3 *Let σ, τ be strategies, $\sigma \approx \tau$ if and only if*

(1) $sab \in \sigma, s'a'b' \in \tau, sa \approx_A s'a' \Rightarrow sab \approx_A s'a'b'$
(2) $s \in \sigma, s' \in \tau, sa \approx_A s'a' \Rightarrow (\exists b)[sab \in \sigma]$ *iff* $(\exists b')[s'a'b' \in \tau]$

Such an extension is not in general an equivalence relation since it might lack reflexivity. If σ is a strategy for a game A such that $\sigma \approx \sigma$, we write $\sigma : A$.

Game Constructions.

Definition 2.4 (Tensor product) *Given games A and B, the tensor product $A \otimes B$ is the game defined as follows:*

- $M_{A\otimes B} = M_A + M_B$
- $\lambda_{A\otimes B} = [\lambda_A, \lambda_B]$
- $P_{A\otimes B} \subseteq M^{\circledast}_{A\otimes B}$ *is the set of positions, s, which satisfy the following conditions:*

(1) the projections on each component (written as $s \upharpoonright A$ or $s \upharpoonright B$) are positions for the games A and B respectively;
(2) every answer in s must be in the same component game as the matching question.

- $s \approx_{A\otimes B} s' \iff s \upharpoonright A \approx_A s' \upharpoonright A, s \upharpoonright B \approx_B s' \upharpoonright B, (\forall i)[s_i \in M_A \Leftrightarrow s'_i \in M_A]$

Here $+$ denotes disjoint union of sets, that is $A+B = \{(l, a) \mid a \in A\} \cup \{(r, b) \mid b \in B\}$, and $[-,-]$ is the usual (unique) decomposition of a function defined on disjoint unions.

Definition 2.5 (Linear implication) *Given games A and B, the compound game $A \multimap B$ is defined as follows:*

- $M_{A\multimap B} = M_A + M_B$
- $\lambda_{A\multimap B} = [\overline{\lambda_A}, \lambda_B]$
- $P_{A\multimap B} \subseteq M^{\circledast}_{A\multimap B}$ *is the set of positions, s, which satisfy the following conditions:*

(1) the projections on each component (written as $s \upharpoonright A$ or $s \upharpoonright B$) are positions for the games A and B respectively;
(2) every answer in s must be in the same component game as the matching question.
- $s \approx_{A \multimap B} s' \iff s \upharpoonright A \approx_A s' \upharpoonright A, s \upharpoonright B \approx_B s' \upharpoonright B, (\forall i)[s_i \in M_A \Leftrightarrow s'_i \in M_A]$

Definition 2.6 (Exponential) *Given a game A, the game $!A$ is defined by:*

- $M_{!A} = \omega \times M_A = \sum_{i \in \omega} M_A$
- $\lambda_{!A}(\langle i, a \rangle) = \lambda_A(a)$
- $P_{!A} \subseteq M_{!A}^{\circledast}$ *is the set of positions, s, which satisfy the following conditions:*

(1) $(\forall i \in \omega)[s \upharpoonright A_i \in P_{A_i}]$;
(2) every answer in s is in the same index as the matching question.

- $s \approx_{!A} s' \iff \exists$ *a permutation of indexes $\alpha \in S(\omega)$ such that:*
  - $\pi_1^*(s) = \alpha^*(\pi_1^*(s'))$
  - $(\forall i \in \omega)[\pi_2^*(s \upharpoonright \alpha(i)) \approx \pi_2^*(s' \upharpoonright i)]$

  where π_1 and π_2 are the projections of $\omega \times M_A$ and $s \upharpoonright i$ is an abbreviation of $s \upharpoonright A_i$.

The Game Category $\mathcal{G}$. We define a monoidal closed category $\mathcal{G}$.

Objects: games.

Morphisms: a morphism between games A and B is an equivalence class w.r.t. the relation $\approx_{A \multimap B}$ of history-free strategies $\sigma : A \multimap B$. We denote the equivalence class of σ by $[\sigma]$.

Composition: the composition is given by the extension on equivalence classes of the following composition of strategies. Given strategies $\sigma : A \multimap B$ and $\tau : B \multimap C$, $\tau \circ \sigma : A \multimap C$ is defined by

$$\sigma || \tau = \{s \in (M_A + M_B + M_C)^* \mid s \upharpoonright (A, B) \in \overline{\sigma} \ \& \ s \upharpoonright (B, C) \in \overline{\tau}\}$$
$$\tau \circ \sigma = \{s \upharpoonright (A, C) \mid s \in \sigma || \tau\}^{even}$$

Identity: the identity $id_A : A \multimap A$ is defined by

$$id_A = \{s \in P_A^{even} \mid s \upharpoonright 1 = s \upharpoonright 2\} \ .$$

The game constructions of tensor product and linear implication can be made functorial, in such a way that:

Proposition 2.1 ([4]) *The category $\mathcal{G}$ is monoidal closed.*

However, as it is well-known, $\mathcal{G}$ is not cartesian closed.

The Game Category $K_!(\mathcal{G})$. The exponential game construction of Definition 2.6 can be made functorial, by defining, for any strategy $\sigma : A \multimap B$, the strategy $!\sigma : !A \multimap !B$ by
$!\sigma = \{s \in P_{!A \multimap !B} \mid \forall i\, \exists s' \in \sigma.\, (\forall s_1, s_1' \text{ prefixes of } s, s' \text{ of the same even length.}\ (s_1 \upharpoonright (A)_i = s_1' \upharpoonright A \ \&\ s_1 \upharpoonright (B)_i = s_1' \upharpoonright B))\}$.
Moreover, the exponential can be endowed with a comonad structure $(!, der, \delta)$ [4], where for each game A the morphisms $der_A : !A \multimap A$ and $\delta_A : !A \multimap !!A$ are defined as follows:

- $der_A = [\{s \in P^{even}_{!A \multimap A} \mid \forall s' \text{ even length prefix of } s.\ (s' \upharpoonright (!A)_0 = s' \upharpoonright A \ \&\ \forall i \neq 0.\ s' \upharpoonright (!A)_i = \epsilon)\}]$
- $\delta_A = [\{s \in P^{even}_{!A \multimap\, !!A} \mid \forall s' \text{ even length prefix of } s.\ (\forall i, j.\ s' \upharpoonright (!A)_{c(i,j)} = s' \upharpoonright (!(!A)_i)_j \ \&\ \forall k \notin codom(c).\ s' \upharpoonright (!A)_k = \epsilon)\}]$, where c is a *pairing function*, i.e. an injective map $c : \omega \times \omega \to \omega$.

Let $K_!(\mathcal{G})$ be the co-Kleisli category over the comonad $(!, der, \delta)$, i.e.:

Objects of $K_!(\mathcal{G})$: games.

Morphisms of $K_!(\mathcal{G})$: a morphism between games A and B is an equivalence class of history-free strategies for the game $!A \multimap B$.

Composition on $K_!(\mathcal{G})$: given strategies $\sigma : A \to B$ and $\tau : B \to C$, the strategy $\tau \circ \sigma : A \to C$ is given by the composition in the category $\mathcal{G}$ of the strategies $\sigma^\dagger : !A \multimap !B$ and $\tau : !B \multimap C$, where $\sigma^\dagger$ is defined by $(!\sigma) \circ \delta_A$.

The following strategies give a commutative comonoid structure on $!A$, [4]:

- the empty strategy $weak_A : !A \multimap I$ (*weakening*), where $I = (\emptyset, \emptyset, \{\epsilon\}, \{(\epsilon, \epsilon)\})$ is the empty game;
- the *contraction* strategy $con_A : !A \multimap !A \otimes !A$,
  $con_A = [\{s \in P^{even}_{!A \multimap\, !A \otimes !A} \mid \forall s' \text{ even length prefix of } s.\, \forall i\ (s' \upharpoonright (!A)_{d(l,i)} = s' \upharpoonright ((!A)_l)_i \ \&\ s' \upharpoonright (!A)_{d(r,i)} = s' \upharpoonright ((!A)_r)_i) \ \&\ \forall j \notin codom(d).\ (s' \upharpoonright (A)_j = \epsilon)\}]$,
  where d is a *tagging function*, i.e. an injective map $d : \omega + \omega \to \omega$.

Identity on $K_!(\mathcal{G})$: the identity $id_A : !A \multimap A$ is der_A.

Using the above structure, one can define a cartesian product on $K_!(\mathcal{G})$, see [4] for more details:

Proposition 2.2 ([4]) *The category $K_!(\mathcal{G})$ is cartesian closed.*

Finally, we point out that Propositions 2.1 and 2.2 hold also if, in the definition of games, we abandon the machinery of "questions and answers", *i.e.* the bracketing condition. Thus, since for our purposes the bracketing condition is not relevant, in the sequel we will simply focus on games with no questions/answers. This corresponds to consider games where all moves are labeled as questions. In this section we have chosen to present the questions/answers machinery, because this is standard, and also in view of possible extensions of the present work.

2.1 An Alternative Construction of the Exponential Game

In this section, we present an exponential game construction alternative to the standard one of Definition 2.6 above. The new exponential will turn out to be naturally isomorphic to the old one. This alternative definition is motivated by the fact that it makes the connection between moves on games and intersection types more direct.

The new exponential game is built using, in place of ω, a set of indexes I defined by:

Definition 2.7 *Let I be the set of all indexes represented by (possibly empty) lists of symbols in $\{0, 1\}$.*

In the sequel, it will be useful to view indexes in I as paths of a binary tree. We will denote by $!^I A$ the new exponential game. The main difference between the standard exponential game and the new one lies in the fact that the set of legal positions over the game $!^I A$ is a proper subset of the positions over $!A$. Namely, we consider as legal only those positions which use a subset of *compatible* indexes, in the following sense:

Definition 2.8 (Compatible Subsets of Indexes) *Two indexes $i, j \in I$ are* compatible *if neither i is a prefix of j nor j is a prefix of i. A set of indexes $J \subseteq I$ is* compatible *if, for each $i, j \in J$, i and j are compatible.*

Notice that the set of indexes with the compatible relation is a *web* and the compatible subsets of an index set I form the corresponding *coherent space* in

the sense of Girard.

For any pair of indexes $i, j \in I$, we can define the *composition* operation $c^I : I \times I \to I$ simply as list concatenation. Viewing indexes as paths, the composition operation yields the index corresponding to the path obtained by appending the second path to the first one.

Coherent subsets of indexes have the following relevant property w.r.t. composition:

Lemma 2.1 *For any compatible set J of indexes, and any family of compatible sets $\{K_j\}_{j\in J}$, the set $\{c^I(j,k) \mid j \in J \;\&\; k \in K_j\}$ is compatible.*

The above lemma, together with the definition of the legal positions on the game $!^I A$, will allow us to use the composition operation c^I as pairing function for defining the comonad structure on $!^I$. Namely, even if c^I is *not* injective in general, it is injective on compatible sets of indexes. Formally:

Definition 2.9 (Alternative Exponential) *Given a game A, the game $!^I A$ is defined by:*

- $M_{!^I A} = I \times M_A = \sum_{i\in I} M_A$
- $\lambda_{!^I A}(\langle i, a\rangle) = \lambda_A(a)$
- $P_{!^I A} \subseteq M^{\circledast}_{!^I A}$ *is the set of positions, s, which satisfy the following conditions:*

(1) $(\forall i \in I)[s \upharpoonright A_i \in P_{A_i}]$;
(2) (every answer in s is in the same index as the matching question;)
(3) the set of indexes appearing in the moves of s is compatible.

- $s \approx_{!^I A} s' \iff \exists$ *a permutation of indexes $\alpha \in S(I)$ such that:*
  - $\pi_1^*(s) = \alpha^*(\pi_1^*(s'))$
  - $(\forall i \in I)[\pi_2^*(s \upharpoonright \alpha(i)) \approx \pi_2^*(s' \upharpoonright i)]$

The exponential game construction $!^I$ can be naturally lifted to a functor such that:

Proposition 2.3 *The exponential functor $!^I$ is naturally isomorphic to the standard exponential functor $!$.*

Proof. Let $\iota : I \to \omega$ be any injective function, e.g.

$$\iota(i) = \begin{cases} 0 & \text{if } i = \epsilon \\ 2 * \iota(i') + 1 & \text{if } i = 0i' \\ 2 * \iota(i') + 2 & \text{if } i = 1i' \end{cases}$$

Let $\iota' : \omega \to I$ be any injective function whose codomain is a compatible set of indexes, e.g. $\iota'(i) = \underbrace{1 \ldots 1}_{i} 0$.

Then ι, ι' induce families of strategies
– $\sigma^{\iota} = \{\sigma^{\iota}_A : !^I A \multimap !A\}_A$, where $\sigma^{\iota}_A = \{s \in P_{!^I A \multimap\, !A} \mid \forall s'$ even length prefix of $s.$ $s' \upharpoonright (!^I A)_{\iota(i)} = s' \upharpoonright (!A)_i\}$;
– $\sigma^{\iota'} = \{\sigma^{\iota'}_A : !A \multimap !^I A\}_A$, where $\sigma^{\iota'}_A = \{s \in P_{!A \multimap\, !^I A} \mid \forall s'$ even length prefix of $s.$ $s' \upharpoonright (!A)_{\iota'(i)} = s' \upharpoonright (!^I A)_i\}$.
One can show that $[\sigma^{\iota}]$ and $[\sigma^{\iota'}]$ are well-defined natural isomorphisms; moreover, one is the inverse of the other. $\square$

As a consequence of the above proposition, the exponential functor $!^I$ can be endowed with a comonad structure $(!^I, der^I, \delta^I)$, and, for any game A, the game $!^I A$ can be endowed with a commutative comonoid structure $(!^I A, con^I_A, weak^I_A)$. For our purposes, it is useful to give explicit definitions of the morphisms der^I_A, δ^I_A, con^I_A, as equivalence classes of special strategies:

- $der^I_A = [\{s \in P^{even}_{!^I A \multimap A} \mid \forall s'$ even length prefix of $s.$ $(s' \upharpoonright (!^I A)_\epsilon = s' \upharpoonright A$ & $\forall i \neq \epsilon.\ s' \upharpoonright (!^I A)_i = \epsilon)\}]$
- $\delta^I_A = [\{s \in P^{even}_{!^I A \multimap\, !^I !^I A} \mid \forall s'$ even length prefix of $s.$ $(\forall i, j.\ s' \upharpoonright (!^I A)_{c^I(i,j)} = s' \upharpoonright (!^I (!^I A)_j)_i$ & $\forall k \notin codom(c^I).\ s' \upharpoonright (!^I A)_k = \epsilon)\}]$, where $c^I : I \times I \to I$ is the composition function defined above;
- $con^I_A = [\{s \in P^{even}_{!^I A \multimap\, !^I A \otimes !^I A} \mid \forall s'$ even length prefix of $s.$ $\forall i.$ $(s' \upharpoonright (!^I A)_{0i} = s' \upharpoonright ((!^I A)_l)_i$ & $s' \upharpoonright (!A)_{1i} = s' \upharpoonright ((!^I A)_r)_i$ & $s' \upharpoonright (!^I A)_\epsilon = \epsilon)\}]$.

Notice that δ^I_A is well defined by Lemma 2.1, and the tagging function $d^I : I + I \to I$ is implicitly defined by $d^I(l, i) = 0i$ and $d^I(r, i) = 1i$.

From now on, we will use the symbol ! to refer to the standard or to the new exponential, indifferently. We will state it explicitly, when the new exponential comes into play.

2.2 History-free Strategies as Partial Functions

Following Definition 2.2 of Section 2, strategies are usually represented as trees, where each path corresponds to a position of the strategy. As shown in [4], history-free strategies admit also an alternative presentation as *partial functions* from Opponent to Proponent moves, which will be quite useful in the sequel. In this section, we study in detail such presentation. The representation of history-free strategies as partial functions will be exploited in the sequel of this paper, where a type assignment system is introduced and its connections with the game semantics are studied.

Following [4]:

Definition 2.10 *Let σ be a history-free strategy. We define a partial function*

$f_\sigma : M_O^A \rightharpoonup M_P^A$ *by*

$$f_\sigma(a) = b \quad \textit{iff} \quad \exists s \in P_A.\ sab \in \sigma\ .$$

Vice versa, let $f_\sigma : M_O^A \rightharpoonup M_P^A$*, we define inductively the set* $traces(f)$ *as follows:*
$\epsilon \in traces(f)$
$s \in traces(f)\ \&\ sa \in P_A\ \&\ f(a) = b \implies sab \in traces(f)$.
We say that f *induces the strategy* $\sigma_f = traces(f)$*, if* $traces(f) \subseteq P_A$.

Proposition 2.4 *If* $f : M_O^A \rightharpoonup M_P^A$ *is a partial function inducing a strategy* σ_f *on* A*, then* σ_f *is history-free.*

Notice that, for any partial function f, we have $f_{\sigma_f} \subseteq f$, while for any strategy τ, we have $\sigma_{f_\tau} = \tau$. Thus there is always a *least partial function* on moves canonically inducing a history-free strategy.

Using the representation of strategies as partial functions, the morphisms from A to B on the category $\mathcal{G}$ are represented by partial functions $f : M_A^P + M_B^O \rightharpoonup M_A^O + M_B^P$. Such functions can be written as matrixes:

$$f = \begin{pmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{pmatrix}$$

where

$$f_{11} : M_A^P \rightharpoonup M_A^O \quad f_{12} : M_B^O \rightharpoonup M_A^O \quad f_{21} : M_A^P \rightharpoonup M_B^P \quad f_{22} : M_B^O \rightharpoonup M_B^P$$

Functions such as f above are amenable of a useful geometrical description in terms of "boxes and wires", [2,3], as in Fig. 1(i).

The composition on the category $\mathcal{G}$ can be equivalently expressed in terms of the representation of strategies as partial functions as follows. Let $f : M_A^P + M_B^O \rightharpoonup M_A^O + M_B^P$, $g : M_B^P + M_C^O \rightharpoonup M_B^O + M_C^P$ representing strategies, then f, g are composed in such a way that Proponent moves in B under σ get turned into Opponent moves in B for τ, and vice versa. Geometrically, we have the picture in Fig. 1(ii).

Algebraically, the composition of f and g is obtained via a Girard's Execution Formula, see [4] for more details.

The application morphism $app_{A,B} : (A \multimap B) \otimes A \multimap B$ determined by the monoidal closed structure on $\mathcal{G}$ is induced by the isomorphism

$$((M_A^O + M_B^P) + M_A^P) + M_B^O \simeq ((M_A^P + M_B^O) + M_A^O) + M_B^P\ .$$

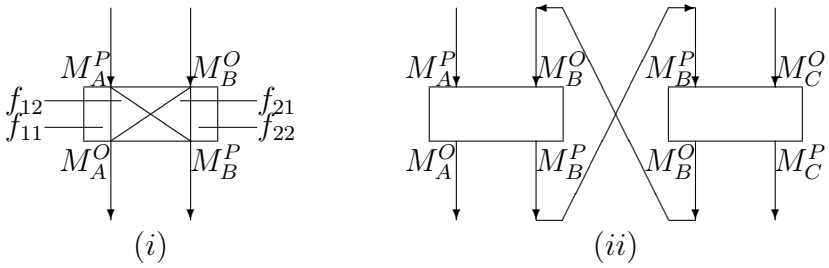

Fig. 1. Geometrical description of strategies and strategy composition.

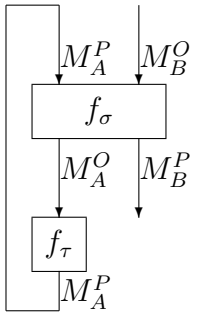

Fig. 2. Geometrical description of application on $\mathcal{G}$.

Geometrically, the application of two strategies $\sigma : A \multimap B$ and $\tau : A$, i.e. $app_{A,B} \circ (\sigma \otimes \tau)$ is represented as in Fig. 2.

In view of studying the connections between our type semantics and the game semantics, it is useful to give an explicit description of the application of two strategies, $\sigma : !C \multimap (!A \multimap B)$ and $\tau : !C \multimap A$, in the category $K_!(\mathcal{G})$. The application of σ, τ in $K_!(\mathcal{G})$, i.e. $ev \circ \langle \sigma, \tau \rangle$, coincides (up-to $\approx$) with the strategy obtained by the following composition on the category $\mathcal{G}$: $app_{A,B} \circ (\sigma \otimes \tau^\dagger) \circ con_C$ (see [4] for more details).

In Fig. 3 appears the geometrical description of the strategy resulting from the application of strategies σ, τ, represented by partial functions $f_\sigma : M_{!C}^P + (M_{!A}^P + M_B^O) \to M_{!C}^O + (M_{!A}^O + M_B^P)$ and $f_\tau : M_{!C}^P + M_{!A}^O \to M_{!C}^O + M_{!A}^P$.

The final box (dash box in figure) represents a two-input/two-output function $f : M_{!C}^P + M_B^O \to M_{!C}^O + M_B^P$. If the input enters through the wire M_B^O, then it is directly sent to f_σ, otherwise, if the input enters through the wire $M_{!C}^P$, then the contraction con_C acts where the $\bullet$ appears, by sending the token either to f_σ or to $f_{\tau\dagger}$, depending on the index of the move. Once the token is sent to f_σ or $f_{\tau\dagger}$, it can possibly cycle along the wires $M_{!A}^O$ and $M_{!A}^P$, and finally it exists either from the box f_σ (through the wire $M_{!C}^O$ or M_B^P) or from the box f_τ (through the wire $M_{!C}^O$). The contraction merges the outputs coming from the wires $M_{!C}^O$ of f_σ and $f_{\tau\dagger}$ where indicated.

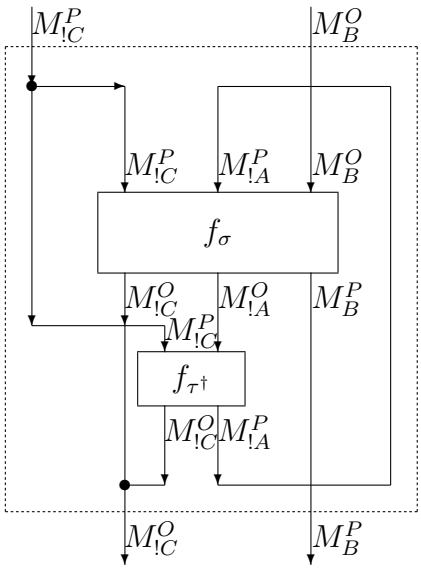

Fig. 3. Geometrical description of application on $K_!(\mathcal{G})$.

3 The Simply Typed λ-calculus

In this section we recall the syntax of the simply typed λ-calculus with two ground constants, $\bot, \top$, and we introduce a game model for such calculus. In Section 4, we will introduce a finitary description of this game model, based on a typed assignment system.

Definition 3.1 *The class SimType of simple types over a ground type o is defined by:*

$$(SimType \ni)\ A \ ::= \ o \ \mid \ A \to A \ .$$

Raw Terms are defined as follows:

$$\Lambda \ni \ M \ ::= \bot \ \mid \ \top \ \mid \ x \ \mid \ \lambda x^A.M \ \mid \ MM \ ,$$

where $\bot, \top \in Const$ are ground constants, $x \in Var$. We denote by Λ^0 the set of closed λ-terms.

Well-typed terms. *We introduce a* proof system *for deriving* typing judgements *of the form $\Gamma \vdash M : A$, where Γ is a* type environment*, i.e. a finite list $x_1 : A_1, \ldots, x_k : A_k$. The rules of the proof system are the following:*

$$\frac{}{\Gamma \vdash C : o} \qquad \frac{}{\Gamma, x : A, \Gamma' \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

where $C \in \{\bot, \top\}$.

β-conversion. *β-conversion between well-typed terms is the least relation generated by the following rule and the rules for congruence closure (which we omit):*
$\Gamma \vdash (\lambda x^A.M)N = M[N/x] \ : \ B$, *where* $\Gamma, x : A \vdash M : B$, *and* $\Gamma \vdash N : A$.

3.1 A Game Model

We define a game model for the λ-calculus of Definition 3.1 in the cartesian closed category $K_!(\mathcal{G})$. Simple types are interpreted by the hierarchy of games over the following *Sierpinski Game* (without questions/answers):

Definition 3.2 (Sierpinski Game) *The game $\mathcal{O}$ is defined as follows:*

- $M_{\mathcal{O}} = \{*, a\}$
- $\lambda_{\mathcal{O}}(*) = O \quad \lambda_{\mathcal{O}}(a) = P$
- $P_{\mathcal{O}} = \{\epsilon, *, *a\}$
- $\approx_{\mathcal{O}} = id_{P_{\mathcal{O}}}$

The only two strategies on the Sierpinski Game are the empty strategy, which we denote by $\bot_{\mathcal{O}}$, and the strategy $\top_{\mathcal{O}}$ induced by the partial function $f_{\top_{\mathcal{O}}}(*) = a$. More in general, we denote by $\bot_{!A_1 \otimes \ldots \otimes !A_k \multimap \mathcal{O}}$ the empty strategy on $!A_1 \otimes \ldots \otimes !A_k \multimap \mathcal{O}$, and by $\top_{!A_1 \otimes \ldots \otimes !A_k \multimap \mathcal{O}}$ the strategy induced by $f_{\top_{!A_1 \otimes \ldots \otimes !A_k \multimap \mathcal{O}}}(\langle r, * \rangle) = \langle r, a \rangle$.

Types are interpreted by games over the hierarchy on the Sierpinski game. Terms in contexts are interpreted as strategies in the usual way, i.e. $x_1 : A_1, \ldots, x_k : A_k \vdash M : A$ is interpreted as a strategy on the game $!\llbracket A_1 \rrbracket^{\mathcal{G}} \otimes \ldots \otimes !\llbracket A_k \rrbracket^{\mathcal{G}} \multimap \llbracket A \rrbracket^{\mathcal{G}}$ using standard categorical combinators as follows:

Definition 3.3 (Term Interpretation)

- $\llbracket x_1 : A_1, \ldots, x_k : A_k \vdash \bot : o \rrbracket^{\mathcal{G}} = \bot_{!\llbracket A_1 \rrbracket^{\mathcal{G}} \otimes \ldots \otimes !\llbracket A_k \rrbracket^{\mathcal{G}} \multimap \llbracket A \rrbracket^{\mathcal{G}}}$
- $\llbracket x_1 : A_1, \ldots, x_k : A_k \vdash \top : o \rrbracket^{\mathcal{G}} = \top_{!\llbracket A_1 \rrbracket^{\mathcal{G}} \otimes \ldots \otimes !\llbracket A_k \rrbracket^{\mathcal{G}} \multimap \llbracket A \rrbracket^{\mathcal{G}}}$
- $\llbracket x_1 : A_1, \ldots, x_k : A_k \vdash x_i : A_i \rrbracket^{\mathcal{G}} = \pi_i : !\llbracket A_1 \rrbracket^{\mathcal{G}} \otimes \ldots \otimes !\llbracket A_k \rrbracket^{\mathcal{G}} \multimap \llbracket A \rrbracket^{\mathcal{G}}$

- $[\![\Gamma \vdash \lambda x^A.M : A \to B]\!]^{\mathcal{G}} = \Lambda([\![\Gamma, x : A \vdash M : B]\!]^{\mathcal{G}})$
- $[\![\Gamma \vdash MN : B]\!]^{\mathcal{G}} = ev \circ \langle [\![\Gamma \vdash M : A \to B]\!]^{\mathcal{G}}, [\![\Gamma \vdash N : A]\!]^{\mathcal{G}} \rangle$

where π_i denotes the i-th projection.

Notice that, by abuse of notation, we have used the same symbols $A, B, \ldots$ to denote simple types and the games interpreting them.

4 The Type Assignment System

In this section, we introduce and study a type assignment system, which gives a finitary description of the game model of Section 3.1. The types involved are essentially the standard intersection types, where the intuitionistic arrow is substituted by the linear arrow type constructor, and the structural rules are missing. Our approach to intersection types is "typed", i.e. intersection types are built inductively over games. The usual untyped intersection semantics can be recovered as a special case of the typed case (see Section 6 below for more details). In our setting, types on a game A represent sets of Opponent and Proponent moves on A. The judgments derivable in our typing system are of the shape $x_1 : t^{!A_1}, \ldots, x_k : t^{!A_k} \vdash M : t^{!A}$, whose intended meaning is to represent a set of equal number of Opponent and Proponent moves. We will show that, for each Opponent move in such a set, there will be a corresponding Proponent answer such that the pair of moves belongs to the graph of the strategy interpreting the term.

4.1 Types and Environments

For each game A, we define the set of corresponding intersection types. At this stage, a type on A simply represents a set of moves on the game A. The intersection type constructor is used to represent sets of moves on exponential games, i.e. the moves appearing in each $\wedge$-component correspond to moves in different components of the exponential game. This is why the $\wedge$ constructor is not commutative neither associative nor idempotent. In Section 5, the exact correspondence between types and games is established.

Definition 4.1 (Types) *We define a family of intesection type sets $IntType^A$, by induction on the structure of the game A via the following abstract syntax:*

- Types on Sierpinski game*:*

$$t^{\mathcal{O}} ::= c^{\mathcal{O}}_{\emptyset} \;\mid\; c^{\mathcal{O}}_{\{*\}} \;\mid\; c^{\mathcal{O}}_{\{a\}} \;\mid\; c^{\mathcal{O}}_{\{*,a\}}$$

- Types on linear arrow games*:*

$$t^{A \multimap B} ::= t^A \multimap t^B$$

- Types on exponential games*:*

$$t^{!A} ::= t^A \mid t^{!A} \wedge t^{!A}$$

In what follows, we use the symbols t^A, u^A, v^A to denote elements in IntType^A, and we simply write t in place of t^A, when the game is irrelevant.
We use the symbols $c_\emptyset^{A \multimap B}$ and $c_\emptyset{}^{!A}$ to denote, respectively, the types $c_\emptyset^A \multimap c_\emptyset^B$ and $c_\emptyset^A$ (which, in particular, is a type on $!A$). Moreover, we endow the set of types with the equivalence relation induced by $c_\emptyset^A = c_\emptyset^A \wedge c_\emptyset^A$.

When related to game semantics, types represent sets of moves, in the sense presented by the following definitions. First, we define a subclass of types representing a single move. Informally, a single-move type is a type whose term structure contains a single instance of one of the two constant $c_{\{a\}}^{\mathcal{O}}, c_{\{*\}}^{\mathcal{O}}$, while all the other instances of basic constants are in the form $c_\emptyset^A$. We mark single-move types as Proponent or Opponent types, mimicking the usual game semantic definitions.

Definition 4.2 (Single-move Types) *We distinguish between types where the only move is a Proponent move (p^A) and types where the only move is an Opponent move (o^A). The definition of the family of sets SingleType^A,* single-move types *on A, is by induction on the game A:*

$$p^{\mathcal{O}} ::= c_{\{a\}}^{\mathcal{O}}$$
$$o^{\mathcal{O}} ::= c_{\{*\}}^{\mathcal{O}}$$

$$p^{A \multimap B} ::= c_\emptyset^A \multimap p^B \;\mid\; o^A \multimap c_\emptyset^B$$
$$o^{A \multimap B} ::= c_\emptyset^A \multimap o^B \;\mid\; p^A \multimap c_\emptyset^B$$

$$p^{!A} ::= p^A \mid p^{!A} \wedge c_\emptyset^A \mid c_\emptyset^A \wedge p^{!A}$$
$$o^{!A} ::= o^A \mid o^{!A} \wedge c_\emptyset^A \mid c_\emptyset^A \wedge o^{!A}$$

In what follows, we will use the symbol m^A to denote a single-move type p^A or o^A, indifferently.

Any intersection type can be seen as the union of single-move types by the following definition:

Definition 4.3 *The family of functions $S^A : IntType^A \to \wp(SingleType^A)$ are defined, with some abuse of notation, by induction as follows:*

$$S^A(c^A_\emptyset) = \emptyset$$

$$S^A(t^A) = \{t^A\} \quad \text{if } t^A \in SingleType^A$$

$$S^{\mathcal{O}}(c^{\mathcal{O}}_{\{*,a\}}) = \{c^{\mathcal{O}}_{\{*\}}, c^{\mathcal{O}}_{\{a\}}\}$$

$$S^{A \multimap B}(t^A \multimap u^B) = (c^A_\emptyset \multimap S^B(u^B)) \cup (S^A(t^A) \multimap c^B_\emptyset)$$

$$S^{!A}(t^A) = S^A(t^A)$$

$$S^{!A}(t^{!A} \wedge u^{!A}) = (c^A_\emptyset \wedge S^{!A}(u^{!A})) \cup (S(t^{!A}) \wedge c^A_\emptyset)$$

where the symbols $\multimap$ and $\wedge$ on the righthand side of the equations denote the pointwise application of the corresponding constructors to a set.

As a curiosity, notice that, for any game A, the set $\{S(t^A) | t^A \in IntType^A\}$ is the set of finite elements of a coherent space built on a web having $SingleType^A$ as set of elements. The coherence relation on single-move types is related and similar to the compatible relation on indexes presented in Definition 2.8. Loosely speaking, two single-move types are coherent if the corresponding expression trees are not included one into the other.

Given the above correspondence between types and sets of moves, it is natural, and for our purpose useful, to introduce on types some of the basic notions on sets.

Definition 4.4

- *We say that a type t^A* contains *the single-move type u^A if $u^A \in S^A(t^A)$.*
- *We define the* cardinality *of a type t^A as the cardinality of the set $S^A(t^A)$.*
- *We say that two types t^A and u^A are* disjoint*, if the sets $S^A(t^A)$ and $S^A(u^A)$ are disjoint.*
- *We define a family of partial union operations on types, $\{\uplus_A\}_A$, $\uplus_A : IntType^A \to IntType^A$, as follows:*

$$t^A_1 \uplus t^A_2 = \begin{cases} u^A & \text{if } S^A(t^A_1) \cup S^A(t^A_2) = S^A(u^A) \\ \text{undefined} & \text{if there is no such a } u^A \end{cases}$$

If $t \uplus u$ is defined, we say that t and u are compatible*.*

Notice that the union operations $\uplus_A$ are well-defined, since it is easy to check that, if the union of two single-move types correspond to an intersection type u^A, then u^A is unique. Alternatively, the partial union operations can be more explicitly characterized as follows:

Lemma 4.1 *The operations $\uplus_A : IntType^A \to IntType^A$ are the least partially defined functions satisfying:*

$$c^{\mathcal{O}}_X \uplus_{\mathcal{O}} c^{\mathcal{O}}_Y = c^{\mathcal{O}}_{X \cup Y}$$

$$(t^A_1 \multimap t^B_2) \uplus_{A \multimap B} (u^A_1 \multimap u^B_2) = t^A_1 \uplus_A u^A_1 \multimap t^B_2 \uplus_B u^B_2$$
$$(t^{!A}_1 \wedge t^{!A}_2) \uplus_{!A} (u^{!A}_1 \wedge u^{!A}_2) = (t^{!A}_1 \uplus_{!A} u^{!A}_1) \wedge (t^{!A}_2 \uplus_{!A} u^{!A}_2)$$

Notice that, for any A, $c^A_\emptyset \uplus t^A = t^A$.

To recover the history-free strategy corresponding to a term, it is useful to introduce a subclass of types consisting of exactly two moves, an Opponent move and a Proponent one. Namely, such types will represent pairs in the graph of a partial function describing a history-free strategy.

Definition 4.5 *The* Step types *on the game A ($StepType^A$) are the types that can be obtained as union of a Proponent and an Opponent single-move type:*

$$s^A ::= p^A \uplus o^A \ .$$

Lemma 4.2 *Step types can be characterized by induction on games as follows:*

$$s^{\mathcal{O}} ::= c^{\mathcal{O}}_{\{*,a\}}$$

$$s^{A \multimap B} ::= p^A \multimap p^B \ \mid \ o^A \multimap o^B \ \mid \ c^A_\emptyset \multimap s^B \ \mid \ s^A \multimap c^B_\emptyset$$

$$s^{!A} ::= s^A \mid s^{!A} \wedge c^A_\emptyset \mid c^A_\emptyset \wedge s^{!A} \mid p^{!A} \wedge o^{!A} \mid o^{!A} \wedge p^{!A}$$

Definition 4.6 (Environments) • *Let $x^{A_1}, x^{A_2}, \ldots$ be a list of variables with domains $A_1, A_2, \ldots$, ranging over simple types.*
Environments *are lists defined by:*

$$\Gamma, \Delta ::= \epsilon \mid x^A : t^{!A}, \Gamma_1$$

where x^A does not appear in Γ_1 and, by abuse of notation, $x^A : t^{!A}$ is used in place of $x^A : t^{![\![A]\!]^{\mathcal{G}}}$.
We will simply write x in place of x^A, when the game is irrelevant.

- *Let $dom(\Gamma)$ denote the list of variables in the domain of Γ, i.e., if $\Gamma = [x^{A_1} : t^{!A_1}, \ldots, x^{A_k} : t^{!A_k}]$, then $dom(\Gamma) = [x^{A_1}, \ldots, x^{A_k}]$.*

- *Let $\Gamma_\emptyset$ denote a generic environment, where all types are $c_\emptyset$.*
- *Let Γ, Γ' be contexts such that $dom(\Gamma) = dom(\Gamma')$. We define the* disjoint union context $\Gamma \uplus \Gamma'$ *(the* intersection context $\Gamma \wedge \Gamma'$*) as the pointwise application of the $\uplus$ ($\wedge$) operation to the types in the contexts.*

4.2 The Typing System

We introduce a typing system for deriving judgments of the shape $x_1 : t^{!A_1}, \ldots, x_k : t^{!A_k} \vdash M : t^{!A}$, whose intended meaning is to represent a set of equal number of Opponent and Proponent moves. If the main connective of $t^{!A}$ is not $\wedge$, then, for each Opponent move in such a set, there is a corresponding Proponent answer such that the pair of moves belongs to the graph of the strategy interpreting the term. The most informative judgments are those involving step types, where exactly two moves appear, an Opponent move and the Proponent answer. When the type contains more than two moves, we lose the exact matching between Opponent and Proponent moves. In principle, step types would be sufficient to recover the strategy interpreting the term in the game model, however, it is useful to consider general types in the type assignment system, because this simplifies the presentation of the rule for application.

Definition 4.7 (Typing System) *The* typing rules *for deriving judgments $x^{A_1} : t^{!A_1}, \ldots, x^{A_k} : t^{!A_k} \vdash M : t^{!A}$ are almost the standard ones, i.e.:*

$$\frac{}{\Gamma_\emptyset \vdash C : c^{\mathcal{O}}_\emptyset} \qquad (\emptyset)$$

$$\frac{}{\Gamma_\emptyset \vdash \top : c^{\mathcal{O}}_{\{*,a\}}} \qquad (\top)$$

$$\frac{}{\Gamma_\emptyset, x^A : t^A, \Delta_\emptyset \vdash x^A : t^A} \qquad (\text{var})$$

$$\frac{\Gamma \vdash M : t_1^A \quad \Delta \vdash M : t_2^A}{\Gamma \wedge \Delta \vdash M : t_1^A \wedge t_2^A} \qquad (\text{intersection})$$

$$\frac{\Gamma, x^A : u^{!A}, \Delta \vdash M : t^B \quad t^B \text{ not a } \wedge\text{-type}}{\Gamma, \Delta \vdash \lambda x^A.M : u^{!A} \multimap t^B} \qquad (\text{abs})$$

$$\frac{\Gamma \vdash M : u^A \multimap t^B \quad \Delta \vdash N : u^A}{\Gamma \wedge \Delta \vdash MN : t^B} \qquad (\text{app})$$

where $C \in \{\bot, \top\}$ and a $\wedge$-type is a type whose main constructor is $\wedge$.

In what follows, we will simply drop game tags from variables and types in the judgments, when the game is not relevant.

In the rest of this section, we study basic properties of the typing system.

The following definition will be useful:

Definition 4.8

- *The type* associated *to a judgement* $x^{A_1} : t^{!A_1}, \ldots, x^{A_k} : t^{!A_k} \vdash M : t$ *is the type of its curryfication i.e. the type* $t^{!A_1} \multimap (\ldots \multimap (t^{!A_k} \multimap t)\ldots)$
- *The* cardinality *of a judgment is the cardinality of the associated type.*
- *A* step judgment *is a judgment whose associated type is a step type.*
- *A* $\wedge$-judgment *is a judgment* $\Gamma \vdash M : t^A$ *where* A *is a* $\wedge$*-type, i.e. the main constructor of* t^A *is* $\wedge$*.*
- *A* non-$\wedge$ judgment *is a judgment that is not a* $\wedge$*-judgment.*
- *Two judgments* $\Gamma \vdash M : t$ *and* $\Gamma' \vdash M : u$ *are* compatible *(*disjoint*) if their associated types are compatible (disjoint).*
- *We say that a judgment* contains *a single-move type* t^A *if the associated type contains* t^A*.*

The single move types contained in the conclusion of a typing rule are inherited from the single-move types contained in the premises. However, when moving from a premise to the conclusion, a single-move type partly changes its term structure. For example, when applying the (app) rule to premises $x : c_\emptyset^C \vdash N : u^A$ and $x : t^C \vdash M : u^A \multimap t^B$, where the latter contains the single-move type $m^C \multimap c_\emptyset^{A \multimap B}$, we obtain a conclusion containing the single-move type $m^C \wedge c_\emptyset^C \multimap c_\emptyset^B$. In the sequel of this paper, to avoid irrelevant details, we will be a little sloppy in the notation, and we denote with the same symbols single-move types appearing in premises and the corresponding single-move types in the conclusions.

Lemma 4.3 *All judgments derivable in the typing system have even cardinality. All derivable judgments contain equal number of Proponent and Opponent single-move types.*

Proof. Straightforward, by induction on derivations. □

The following lemma collects a number of technical properties of the typing system. In particular, items (ii) and (iii) will be useful to prove Proposition 4.1 below, which expresses the fact that all derivable judgments can be "decomposed" in step judgments. The most technical part of Lemma 4.4 below is item (v), which amounts to the counterpart in the typing system of the application between strategies as described in Section 2.2, Fig. 3.

Lemma 4.4

(i) If $x^{A_1} : t_1^{!A_1}, \ldots, x^{A_k} : t_1^{!A_k} \vdash M : t_1$ *and* $x^{A_1} : t_2^{!A_1}, \ldots, x^{A_k} : t_2^{!A_k} \vdash M : t_2$ *are derivable non-*$\wedge$ *step judgments, containing the single-move types* m_1, m_1' *and* m_2, m_2', *respectively, then*

- $m_1 = m_2$ *iff* $m_1' = m_2'$.
- m_1 *is compatible with* m_2 *if and only if* m_1' *is compatible with* m_2'.

(ii) For any set $\{\Gamma_i \vdash M : t_i | i \in I\}$ *of pairwise compatible non-*$\wedge$ *step judgments, the judgment* $\uplus_{i\in I} \Gamma_i \vdash M : \uplus_{i\in I} t_i$ *is derivable.*

(iii) For any derivable non-$\wedge$ *judgment* $\Gamma \vdash M : t$, *there exist decompositions of* t *and* Γ *in types* $t_1, \ldots t_n$ *and environments* $\Gamma_1, \ldots, \Gamma_n$ *such that* $t = \uplus_i t_i$, $\Gamma = \uplus_i \Gamma_i$, *and* $\Gamma_i \vdash M : t_i$ *are derivable step judgments, for all* $i = 1, \ldots, n$.

(iv) Items (i)–(iii) hold also for $\wedge$*-judgments.*

(v) Given any pair of judgments $\Gamma \vdash M : u \multimap t$ *and* $\Delta \vdash N : u$, *and any single-move type* m *contained in the judgment* $\Gamma \wedge \Delta \vdash MN : t$, *there exist a single-move type* m' *contained in* $\Gamma \wedge \Delta \vdash MN : t$ *and a chain of single-move types* $\langle u_j \rangle_{j\in J}$ *contained in* u *such that*

- J *is an interval of integers.*
- *If the chain is empty, then there exists a judgement either in the form* $\Gamma \vdash M : c_\emptyset \multimap t$, *or* $\Delta \vdash N : c_\emptyset$, *containing the single-move types* m *and* m'.
- *If the chain is non-empty, then*
  - *either there exists a step judgment* $\Gamma_1 \vdash M : u_1 \multimap t_1$ *containing* m, *and the first element in* J *is* 1 *or there exists a step judgement* $\Delta_0 \vdash N : u_0$ *containing* m, *and the first element in* J *is* 0.
  - $\Gamma_\emptyset \vdash M : (u_{2k} \multimap c_\emptyset) \uplus (u_{2k+1} \multimap c_\emptyset)$, *for all* $2k, 2k+1 \in J$, $k \geq 0$.
  - $\Delta_\emptyset \vdash N : u_{2k+1} \uplus u_{2k+2}$, *for all* $2k+1, 2k+2 \in J$, $k \geq 0$.
  - *Either there exists a step judgment* $\Gamma_{2k} \vdash M : u_{2k} \multimap t'_{2k}$ *containing* m' *and the last element in* J *is* $2k$ *or there exists a step judgement* $\Delta_{2k+1} \vdash N : u_{2k+1}$ *containing* m' *and the last element in* J *is* $2k+1$.

Proof. First we define the complexity of a term M as the number of constructors appearing in it. The proof of items (i)–(v) is by induction on the complexity of the terms M and N.
Basic cases: M and N have complexity 1.

- The term M is a constant: items (i)–(iii) and (v) are trivial, since the only rules usable in the derivations are the rules ($\emptyset$) and ($\top$).

For item (iv), we present a uniform proof not depending on the structure of the two terms M, N. This proof can therefore be used also for the other base case and for the induction step.
Item (iv): (i). Assume $\Gamma_1 \vdash M : t_1$, $\Gamma_2 \vdash M : t_2$ are derivable step judgments containing single-move types m_1, m_1' and m_2, m_2', respectively. Then, by the shape of the rules in the typing system, the two step judgments are derivable from a non-$\wedge$ step judgment and a set of non-$\wedge$ empty judgments, combined by a series of applications of the (intersection) rule. In order to prove item (i) for general judgments, we proceed by induction on the number of applications of the (intersection) rule. In the base case, the final judgments are both non-$\wedge$ and the thesis follows from item (i). Induction step: if $m_1 = m_2$ (or m_1 is compatible with m_2), both judgments must be $\wedge$-judgments obtained from a step judgment and an empty judgment through an (intersection) rule, and the premise step judgments must contain a common (or compatible) single-move type. Thus, by induction hypothesis, we get the thesis.
Item (iv): (ii). We proceed as for item (iv): (i), by induction on the number of (intersection) rules used in the last parts of the derivations of the judgments $\{\Gamma_i \vdash M : t_i\}_{i \in I}$. The base case follows immediately from item (ii). Induction step: Since the judgments are all pairwise compatible, all derivations must end with an (intersection) rule whose premises are an empty judgment and a step judgment. Moreover, all left-hand (right-hand) premises must be compatible. By induction hypothesis, the unions of all left-hand (right-hand) premises are derivable. Thus, by an application of the (intersection) rule, also the judgment $\biguplus_{i \in I} \Gamma_i \vdash M : \biguplus_{i \in I} t_i$ is derivable.
Item (iv): (iii). The proof is similar to the ones for items (iv): (i) and (iv): (ii).

- The term M is a variable: items (i), (ii), (iii) follows from the fact that the only rule deriving a judgement in the form $\Gamma \vdash x : t$ is the (var) rule.

  For item (v), we present a uniform proof not depending on the structure of the two terms M, N, that can therefore be used also for the induction step.

  By items (iii) and (iv), both judgments $\Gamma \vdash M : u \multimap t$ and $\Delta \vdash N : u$ can be decomposed in a set of step judgments. The chain $\langle u_j \rangle_{j \in J}$ is built as follows: select the step judgment in the decomposition containing the single-move type m, let us suppose it is a judgment of the form $\Gamma_1 \vdash M : u_1 \multimap t_1$, we need to distinguish two cases. If $u_1 = c_\emptyset$, the judgment will contain a second single-move type m', that, together with the empty chain, satisfies item (v). If u_1 is not the empty type, then it is a single-move type and it will be contained in a step judgment belonging to the decomposition of $\Delta \vdash M : u \multimap t$ and having form $\Delta_2 \vdash M : u_2' \multimap t_2'$. Here again we distinguish two cases, if $u_2' = u_1$, then the judgment will contain a second single-move type m', that, together with the chain $\langle u_1 \rangle$, satisfies item (v). If $u_2' = u_1 \uplus u_2$, then the single-move type $u_2 \multimap c_\emptyset$ will be contained in a step judgment belonging to the decomposition of $\Gamma \vdash M : u \multimap t$ and in

the form $\Gamma_3 \vdash M : u'_3 \multimap t'_3$. Repeating the argument used for the judgment $\Gamma_1 \vdash M : u_1 \multimap t_1$, here again we can either stop our construction, or make another step in the construction of the chain. By item (i), the chain must contain single-move types that are all different, and since any type contains a finite number of single-move types, we eventually produce the element m'.

Induction case: M has complexity $n+1$ and N has complexity less than or equal to $i+1$.

- If M is a lambda abstraction, $\lambda x.M'$, items (i), (ii) and (iii) follows immediately from the induction hypothesis and from the fact that any non-$\wedge$ judgment relative to M can be derived only by an application of the (abs) rule .
- Let M be an application, $M'N'$.
  - Item (i). Let $x^{A_1} : t_1^{!A_1}, \ldots, x^{A_k} : t_1^{!A_k} \vdash M'N' : t_1$ and $x^{A_1} : t_2^{!A_1}, \ldots, x^{A_k} : t_2^{!A_k} \vdash M'N' : t_2$ be two non-$\wedge$ step judgments containing single-move types m_1, m'_1 and m_2, m'_2 respectively. The two judgements can be derived only by application of the (app) rule from judgments of the form $\Gamma \vdash M' : u \multimap t$ and $\Delta \vdash N' : u$. Let us fix a single-move type in each original judgment, say m_1 and m_2. By induction hypothesis, item (v), there are two chains of moves, $\langle u_{i_1} \rangle_{i_1 \in I_1}$ and $\langle u_{i_2} \rangle_{i_2 \in I_2}$, satisfying the conditions listed in item (v).

    If $m_1 = m_2$, then, by applying the induction hypothesis, items (i) and (iv), we get that the two chains must coincide, and hence $m'_1 = m'_2$.

    If m_1 is not compatible with m_2, then both moves should be inherited from premises of the same kind, i.e. either both are inherited from premises of the shape $\Gamma \vdash M' : u \multimap t$ or from premises of the shape $\Delta \vdash N' : u$. Then one can check that the chains $\langle u_{i_1} \rangle_{i_1 \in I_1}$ and $\langle u_{i_2} \rangle_{i_2 \in I_2}$ must have the same length and contain pairs of non compatible elements, thus in particular m'_1 is not compatible with m'_2.
  - Item (ii). The derivations of all the step judgments $\Gamma_i \vdash M'N' : t_i$ must end with an (app) rule, with premises having form $\Gamma'_i \vdash M' : u_i \multimap t_i$ and $\Delta_i \vdash N' : u_i$.

    It is easy to check that all Γ'_i, all Δ_i and all t_i are pairwise compatible. However, it is not guaranteed that the types u_i are pairwise compatible. Let m_i and m'_i be the single-move types contained in the step judgment $\Gamma_i \vdash M'N' : t_i$. Fix m_i as starting move, then, by induction hypothesis, there exist chains $\langle u_{j,i} \rangle_{j \in J_i}$ satisfying the conditions listed in item (v). Moreover, by induction hypothesis, the judgments $\Gamma'_i \vdash M' : \uplus_{j \in J_i} u_{j,i} \multimap t_i$ and $\Delta_i \vdash N' : \uplus_{j \in J_i} u_{j,i}$ are derivable. We need to prove that the families of types $\uplus_{j \in J_i} u_{j,i}$, with $i \in I$, are compatible. Suppose by contradiction that there exist two non compatible moves $u_{i',j'}$, $u_{i'',j''}$. Immediately we have that $j' \neq j''$, moreover, by induction hypothesis, and repeating the argument used for item (i) above, it is possible to prove that also the elements consecutive to $u_{i',j'}$, $u_{i'',j''}$ in the respective chains must be pairwise

not compatible, and moreover that elements $m'_{i'}$ and $m'_{i''}$ are not compatible. This contradicts the fact that all Γ'_i, all Δ_i, and all t_i are pairwise compatible.
By induction hypothesis, items (ii) and (iv), $\uplus_{i\in I}\Gamma'_i \vdash M' : \uplus_{i\in I, j\in J_i} u_{j,i} \multimap \uplus_{i\in I} t_i$. and $\uplus_{i\in I}\Delta_i \vdash N' : \uplus_{i\in I, j\in J_i} u_{j,i}$. A final application of the (app) rule concludes the proof of this item.

· Item (iii). Let $\Gamma \vdash M'N' : t$ be a derivable non-$\wedge$ judgment. The last rule in the derivation of the judgment must be the (app)-rule:

$$\frac{\Gamma' \vdash M' : u \multimap t \quad \Delta \vdash N' : u}{\Gamma' \wedge \Delta \vdash M'N' : t}$$

By induction hypothesis, item (v), it is possible to partition all the single-move types contained in the judgment $\Gamma \vdash M'N' : t$ in a set of pairs $\{(m_i, m'_i) | i \in I\}$, and to define a set of chains of single-move types $\{\langle u_{j,i}\rangle_{j\in J_i} \mid i \in I\}$ such that, for each i, the chain $\langle u_{j,i}\rangle_{j\in J_i}$ satisfies the conditions listed in item (v). Thus, by induction hypothesis, there is a family of step judgments $\Gamma'_i \wedge \Delta_i \vdash M'N' : t_i$, containing the single-move types m_i, m'_i, that are derivable, using the (app) rule, from judgments of the shape $\Gamma' \vdash M' : \uplus_{j\in J_i} u_{j,i} \multimap t_i$ and $\Delta_i \vdash N' : \uplus_{j\in J_i} u_{j,i}$. The families $\{\Gamma'_i \wedge \Delta_i\}_{i\in I}$ and $\{t_i\}_{i\in I}$ define a decomposition of the environment Γ and of the type t.

□

The following proposition summarizes the main results in Lemma 4.4 and clarifies the intended meaning of judgments.

Proposition 4.1 *A judgment $\Gamma \vdash M : t$ is derivable if and only if there exist unique decompositions of t and Γ in types $t_1, \ldots t_n$ and environments $\Gamma_1, \ldots, \Gamma_n$ such that $t = \uplus_i t_i$, $\Gamma = \uplus_i \Gamma_i$, and $\Gamma_i \vdash M : t_i$ are derivable step judgments, for all $i = 1, \ldots, n$.*

Remark. By Proposition 4.1 above, it is sufficient to focus on step judgments. The question naturally arises why we do not have considered a typing system for deriving only step judgments. The answer is that, in such case, the rule (app) would have an unbounded number of step judgments in the premises, forming a chaining as described in Lemma 4.4(v) above.

5 From Types to Strategies

In this section, we study the relationship between the type assignment system introduced in Section 4 and the game model of Section 3.1. To this aim, it

is convenient to consider the alternative exponential defined in Section 2.1, in order to have a more direct correspondence with the $\wedge$-type constructor in the typing system. Moreover, it is useful to consider the global type associated to a given judgment in its uncurried form. Thus, we extend the grammar of (single-move, step) types with the type constructor $\otimes$ for denoting types in the tensor product game:

$$\begin{array}{lll}
\text{Types on } A \otimes B & : & t^{A\otimes B} ::= t^A \otimes t^B \\
\text{Single-move types on } A \otimes B & : & p^{A\otimes B} ::= p^A \otimes c_\emptyset \mid c_\emptyset \otimes p^B \\
 & & o^{A\otimes B} ::= o^A \otimes c_\emptyset \mid c_\emptyset \otimes o^B \\
\text{Step types on } A \otimes B & : & s^{A\otimes B} ::= p^A \otimes o^B \mid o^A \otimes p^B \mid s^A \otimes c_\emptyset \mid c_\emptyset \otimes s^B
\end{array}$$

In order to recover, from the type assignment system, the strategy corresponding to a given term in context $\Gamma \vdash M : A$, it is sufficient to consider judgments of the shape $x_1 : t^{!A_1}, \ldots, x_k : t^{!A_k} \vdash M : t^A$, where the global type $t^{!A_1} \otimes \ldots \otimes t^{!A_k} \multimap t^A$ is a step type. Namely, a step type on $!A_1 \otimes \ldots \otimes !A_k \multimap A$ can be read as a pair in the graph of a history-free strategy on $!A_1 \otimes \ldots \otimes !A_k \multimap A$.

Formally, we define a mapping from single move types to moves in the corresponding game and a mapping from step types to pairs in the graph of a strategy on the corresponding game:

Definition 5.1 *Let $\mathcal{M}^A$: $SingleMoveType^A \to M_A$ be a map from single move types on A to moves on the game A defined by induction on A:*

$$\mathcal{M}^{\mathcal{O}}(c_{\{m\}}) = m$$
$$\mathcal{M}^{A\otimes B}(m^A \otimes c_\emptyset) = \langle l, \mathcal{M}^A(m^A)\rangle \qquad \mathcal{M}^{A\otimes B}(c_\emptyset \otimes m^B) = \langle r, \mathcal{M}^B(m^B)\rangle$$
$$\mathcal{M}^{A\multimap B}(c_\emptyset \multimap m^B) = \langle r, \mathcal{M}^B(m^B)\rangle \qquad \mathcal{M}^{A\multimap B}(m^A \multimap c_\emptyset) = \langle l, \mathcal{M}^A(m^A)\rangle$$
$$\mathcal{M}^{!A}(m^A) = \langle \epsilon, \mathcal{M}^A(m^A)\rangle$$
$$\mathcal{M}^{!A}(m^{!A} \wedge c_\emptyset) = \langle 0\pi_1(\mathcal{M}^{!A}(m^{!A})), \pi_2(\mathcal{M}^{!A}(m^{!A}))\rangle$$
$$\mathcal{M}^{!A}(c_\emptyset \wedge m^{!A}) = \langle 1\pi_1(\mathcal{M}^{!A}(m^{!A})), \pi_2(\mathcal{M}^{!A}(m^{!A}))\rangle$$

Let $\mathcal{T}$: $StepType^A \to M_A^O \times M_A^P$ be the map from step types to pairs of moves defined by:

$$\mathcal{T}^A(p^A \uplus o^A) = (\mathcal{M}^A(o^A), \mathcal{M}^A(p^A)) \ .$$

For example:
– $\mathcal{M}((c_\emptyset^{\mathcal{O}} \wedge (c_{\{*\}}^{\mathcal{O}} \wedge c_\emptyset^{\mathcal{O}})) \multimap c_\emptyset^{\mathcal{O}}) = \langle l, \langle 10, * \rangle\rangle$;
– $\mathcal{M}(c_\emptyset^{\mathcal{O}} \multimap (((c_\emptyset^{\mathcal{O}} \wedge c_{\{a\}}^{\mathcal{O}}) \wedge c_\emptyset^{\mathcal{O}}) \multimap c_\emptyset^{\mathcal{O}})) = \langle r, \langle l, \langle 01, a\rangle\rangle\rangle$.

Definition 5.2 (Type Semantics) *Let $[\![\ ]\!]^{\mathcal{T}}$ be the interpretation function*

defined by
$$[\![x_1 : A_1, \ldots, x_k : A_k \vdash M : A]\!]^{\mathcal{T}} = \{\mathcal{T}^{!A_1 \otimes \ldots \otimes !A_k \multimap A}(t^{!A_1} \otimes \ldots \otimes t^{!A_k} \multimap t^A) \mid x_1 : t^{!A_1}, \ldots, x_k : t^{!A_k} \vdash M : t^A \text{ is a derivable step judgment } \}$$

Notice that, by Lemma 4.4(i), the type semantics is an injective function from Opponent to Proponent moves on the game $!A_1 \otimes \ldots \otimes !A_k \multimap A$.

The following theorem establishes the connection between the type semantics and the game semantics. The type semantics yields a partial function representing a member in the equivalence class of the strategy interpreting the term in the game model:

Theorem 5.1 $[\![\Gamma \vdash M : A]\!]^{\mathcal{T}} \in [\![\Gamma \vdash M : A]\!]^{\mathcal{G}}$.

Proof. We prove by induction on M that the function $[\![\Gamma \vdash M : A]\!]^{\mathcal{T}}$ represents the strategy $[\![\Gamma \vdash M : A]\!]^{\mathcal{G}}$.
• If M is the constant $\bot$ or $\top$, then $[\![\Gamma \vdash M : A]\!]^{\mathcal{T}}$ induces the strategy $\bot$ or $\top$, respectively.
• If M is a variable, then $[\![\Gamma \vdash M : A]\!]^{\mathcal{T}}$ represents the projection strategy.
• If M is an abstraction $\lambda x.M'$, then all derivations of the shape $\Gamma \vdash \lambda x.M' : t^{A \multimap B}$ end with an (abs)-rule. Thus, by induction hypothesis, and by definition of Λ in the game interpretation, we get the thesis.
• If M is an application $M'N$, then $\dfrac{\Gamma \vdash M : B \to A \quad \Gamma \vdash N : B}{\Gamma \vdash M'N : A}$, and the step types in $[\![\Gamma \vdash M : A]\!]^{\mathcal{T}}$ are obtained from derivations whose last rule is the (app)-rule, i.e.: $\dfrac{\Gamma' \vdash M' : u \multimap t \quad \Delta \vdash N : u}{\Gamma' \wedge \Delta \vdash M'N : t}$. Now, let $f_\sigma : !C \multimap (B \multimap A)$, $f_\sigma = [\![\Gamma \vdash M' : B \to A]\!]^{\mathcal{T}}$, and $f_\tau : !C \multimap B$, $f_\tau = [\![\Gamma \vdash N : B]\!]^{\mathcal{T}}$, where, by abuse of notation, we use the same symbols for types and their game interpretations, and $!C$ is, up-to isomorphism, the game interpreting the types of the variables in the environment Γ. In order to prove the thesis, we show that:
a) each step judgment for $M'N$ induces a pair in the graph of the partial function f obtained by composing f_σ and f_τ, according to Figure 3 of Section 2.2;
b) for each pair in the graph of f, there exists a derivable step judgment which induces such pair of moves.
Proof of a). By Proposition 4.1, a step judgment $\Gamma' \wedge \Delta \vdash M'N : t$ is derivable if and only if there are sets of step judgments for M', $\{\Gamma'_i \vdash M' : u_i^{!B} \multimap t_i^A\}_{i \in I}$, and for N, $\{\Delta_j \vdash N : v_j^{!B}\}_{j \in J}$, satisfying $\uplus_{i \in I} u_i = \uplus_{j \in J} v_j$, $\uplus_{i \in I} \Gamma'_i = \Gamma'$, $\uplus_{j \in J} \Delta_j = \Delta$, $\uplus_{i \in I} t_i = t$. By induction hypothesis, the step judgments for N whose global type is on the game $!C \multimap B$ correspond to the graph of f_τ. Moreover, one can prove that any step judgment $\Delta_j \vdash N : v_j^{!B}$ with global type on the game $!C \multimap !B$ is equal to $c_\emptyset[\Delta'_j] \vdash N : c_\emptyset[v_j^B]$, where $\Delta'_j] \vdash N : v_j^B$ is a step judgment on the game $!C \multimap B$, and $c_\emptyset[\,]$ is an empty $\wedge$-type context, i.e. a context built over basic contexts $[\,], c_\emptyset$ only using the $\wedge$-type constructor. Using the definition of pairing function c^I as given in Section 2.1, one can check that the step judgments $\Delta_j \vdash N : v_j^{!B}$ determine the function $f_{\tau\dagger}$,

which induces the strategy $\tau^\dagger$. Now the fact that the pair of moves induced by $\Gamma' \wedge \Delta \vdash M'N : t$ is in the graph of of f follows by Lemma 4.4(v) (case $|I| = 1$), and by the definition of f (see Figure 3 of Section 2.2), using the induction hypothesis.
Proof of b). Let $f(m) = m'$. There are various cases, according to the domains of the moves m, m'. We only deal with one case, the others being dealt with similarly. Assume $m \in M^P_{!C}$, $m' \in M^P_B$, and $m = \langle 0 i_0, m_0 \rangle$ Then, by definition of $con_{!C}$ (see Section 2), $\langle i_0, m_0 \rangle$ is sent as input to f_σ, and we have, for $k \geq 0$:

$$f_\sigma(\langle i_0, m_0 \rangle) = \langle i_1, m_1 \rangle \in M^O_{!A} \qquad f_{\tau^\dagger}(\langle i_1, m_1 \rangle) = \langle i_1, m_2 \rangle \in M^P_{!A}$$

$$f_\sigma(\langle i_1, m_2 \rangle) = \langle i_2, m_3 \rangle \in M^O_{!A} \qquad f_{\tau^\dagger}(\langle i_2, m_3 \rangle) = \langle i_2, m_4 \rangle \in M^P_{!A}$$

$$\ldots \qquad \ldots$$

$$f_\sigma(\langle i_{\frac{k-2}{2}}, m_{k-2} \rangle) = \langle i_{\frac{k}{2}}, m_{k-1} \rangle \in M^O_{!A} \qquad f_{\tau^\dagger}(\langle i_{\frac{k}{2}}, m_{k-1} \rangle) = \langle i_{\frac{k}{2}}, m_k \rangle \in M^P_{!A}$$

$$f_\sigma(\langle i_{\frac{k}{2}}, m_k \rangle) = m \in M^P_B$$

By induction hypothesis, there exist step judgments $\{\Gamma_q \vdash M' : u_q \multimap t_q\}_{q=0}^{\frac{k}{2}}$ and $\{\Delta_r \vdash N : v_r\}_{r=1}^{\frac{k}{2}}$ such that $\Gamma_q \vdash M' : u_q \multimap t_q$ induces the pair of moves $(\langle i_q, m_{2q} \rangle, \langle i_{q+1}, m_{2q+1} \rangle)$, and $\Gamma_r \vdash N : v_r$ induces the pair $(\langle i_r, m_{2r-1} \rangle, \langle i_r, m_{2r} \rangle)$. Moreover, $\uplus_q u_q = \uplus_r v_r$. Thus, using Proposition 4.1, $\uplus_q \Gamma_q \wedge \uplus_r \Delta_r \vdash M'N : \uplus_q t_q$ is derivable, and it induces the pair (m, m'). $\square$

6 Further developments

Essentially, in this work we have shown as a type assignment system can be used to determine the interpretation of λ-terms in a game model. However, there are several other aspects in game semantics that arguably can be expressed in terms of intersection types. Game semantics is a quite sophisticated theory and so far we have formulated in the intesection types approach, just one part of it. It is therefore natural to investigate what will be a suitable translation of the other game semantics concepts.

The main aspects that need to be investigated are briefly discussed below.

The Equivalence Relation on Strategies. In AJM-games, the semantical objects are equivalence classes of strategies. The equivalence its defined in terms of an equivalence relation between positions and it uses the definition of strategies as sets of positions. In our approach, we look at strategies as partial functions on moves. As far as we know, there is no direct and simple definition of equivalence relation on strategies given in terms of partial functions. In other words, to determine if two partial funcitions on moves define

equivalent strategies, one is essentialy forced to go through representation of strategies as sets of positions and to use the equivalence defined on them. It will be interesting to find a simple direct definition stating when two sets of types, obtained by the interpretation of two different terms, induce equivalent strategies. We conjecture that this can be obtained by introducing the associativity and commutativity rules for the intersection operator $\wedge$.

It is worth noticing that, without the equivalence relation, the subject reduction property is lost. For example, in our semantics, the interpretation of the term $\lambda y^{o\to o}.(\lambda x^{o\to o}.xy\bot)y$ is only equivalent but not equal to the interpretation of the term $\lambda y^{o\to o}.yy\bot$. Namely, the semantic interpretation of the first term contains the type $(c_\emptyset \wedge c_{\{*\}}) \multimap c_{\{*\}}$, while the second does not contain such type, but instead the type $(c_{\{*\}} \wedge c_\emptyset) \multimap c_{\{*\}}$.

Characterization of Semantical Objects. In game semantics, terms are interpreted by (equivalence classes of) history free strategies, that is a subset of position satisfying some extra properties. Similarly to what happens for the equivalence relation, the notion of strategy refers to positions and there is no direct and simple definition in terms of function on moves.

In our approach, we are interested in determining which property characterizes the sets of types that are interpretations of λ-terms, that is, to define a suitable class of sets of types to be considered as semantical objects.

So far, Proposition 4.1 gives a first characterization of the sets of types obtainable as interpretations of terms. This characterization justifies Definition 5.2, that limits the interpretation of terms to step types. A finer analysis and a more precise characterization will be the object of future investigation.

In game semantics, where a full definability result holds, we have a precise characterization of the strategies obtainable as interpretations of λ-terms (programs). In general, in order to achieve this exact characterization, several notions are introduced on games: the answer-question labeling, the bracketing condition, the partial equivalence relation on strategies. A goal for a possible research is to find the corresponding, analogous, notions on types.

The Untyped λ-calculus and Solutions of Recursive Domain Equations. Intersection types are traditionally used in the semantics of the untyped λ-calculus. The set of intersection types interpreting untyped λ-terms are obtained through a limit process, that can be repeated also for in the present setting. This limit process gives a semantics for λ terms equivalent to the game semantics for the untyped λ-calculus presented in [12,11]. In more detail the limit construction is the following. One starts with a basic game A_0,

and the corresponding set of types T_0, then one builds a hierarchy of set of types by the construction

$$T_{k+1} ::= T_k \to T_k \ = !T_k \multimap T_k$$

moreover one need to define a suitable injection function ι from T_0 to T_1. The set of types interpreting the untyped λ-terms is defined as the union of the hierarchy, i.e. $T = \bigcup_{k\in\omega} T_i$, quotient by the congruence relation generated by the set of equations $\{t_0 = \iota(t_0) \mid t_0 \in T_0\}$. In this construction one require that the injective function ι maps single move types to single move types, preserves the Proponent/Opponent labeling, and preserves the union operation. It is an open question to check what happens if one considers more liberal conditions on the injective function. A simple instance of this construction is obtained by taking $A_0 = \mathcal{O}$ and ι as the function mapping $c_{\{*\}}$ in $c_\emptyset \multimap c_{\{*\}}$ and $c_{\{a\}}$ in $c_\emptyset \multimap c_{\{a\}}$.

Type Assignment Systems for GoI Combinatory Algebras. Type assignment systems describing the interpretation of λ-terms on GoI *Linear Combinatory Algebras* (LCAs) in "wave-style" [14] are essentially standard type assignment systems. On the contrary, type assignment systems for GoI LCAs in "particle-style" [3,5] can be easily derived from game type assignment systems, simply by forgetting the distinction between Proponent and Opponent moves. More precisely, for a given LCA $([M \rightharpoonup M], \bullet)$, types will represent sets of moves in M, and a step type naturally describes two pairs of moves, (a, b) and (b, a). Thus, the type semantics of a term will define a *partial involution* on $[M \rightharpoonup M]$. For the typed λ-calculus, such partial involution should represent the interpretation of the term in a model of *Partial Equivalence Relations* (PERs) over the LCA, [5].

In this paper, we have worked in the setting of games, rather than working directly in the GoI setting, because games are more widely used and, being more "structured", the pure GoI case can be seen as a simplification.

Finally, we point out that the interpretation of λ-terms, when seen as partial functions on moves, is essentially the same in the two models, the main difference being the equivalence relation defined on partial functions.

Towards a Stone Duality for Game Types. In the simple set-theoretic interpretation of intersection types [6], types can be viewed as sets of points over an applicative structure. Building on this interpretation, a suitable *Stone duality* between types and terms can be set up, [1]. Furthermore, type constructors can be interpreted as operators over this space. Can this programme be carried out also for the notion of type in the present paper?

In the game setting, the set of strategies over the Sierpinski hierarchy of games, with application between strategies defined as in Section 2.2, Fig. 3, form a (partial) applicative structure. Over this structure, we could give an interpretation of the set *EvenType* of *even types*, i.e. the types with equal number of Opponent and Proponent moves, which are those involved in the judgments derivable in the typing system. Namely, for a given type $t^A = \biguplus_{i\in I} p_i^A \uplus \biguplus_{j\in j} o_j^A$ such that $|\biguplus_{i\in I} p_i^A| = |\biguplus_{j\in j} o_j^A|$, one can consider all the strategies on the game A, which, viewed as partial functions, extend t^A, i.e.:

$[\![\biguplus_{i\in I} p_i^A \uplus \biguplus_{i\in I} o_i^A]\!]^{\mathcal{S}} = \{f : M_A^O \rightharpoonup M_A^P \mid f$ represents a strategy &
$\exists f' \subseteq f.\ dom(f') = \{\mathcal{M}^A(o_i^A) \mid i \in I\}$ & $codom(f') = \{\mathcal{M}^A(p_i^A) \mid i \in I\}$ $\}$.

Using Theorem 5.1, one can prove the following soundness and completeness result for the interpretation of types w.r.t. the game model in question:

$$[\![x_1 : A_1, \ldots, x_k : A_k \vdash M : A]\!]^{\mathcal{T}} =$$
$$min \bigcap \{[\![t^{!A_1} \otimes \ldots \otimes t^{!A_k} \multimap t^A]\!]^{\mathcal{S}} \mid x_1 : t^{!A_1}, \ldots, x_k : t^{!A_k} \vdash M : t^A \text{ is derivable } \}.$$

The interpretation of $\multimap$ is *not* logical, i.e. it is not the case that $f \in [\![u^A \multimap t^B]\!]^{\mathcal{S}}$ if and only if $\forall g \in [\![u^A]\!]^{\mathcal{S}}$, $f \bullet g \in [\![t^B]\!]^{\mathcal{S}}$ (*). Namely, there are constant strategies satisfying condition (*), which are not in $[\![u^A \multimap t^B]\!]^{\mathcal{S}}$. There is a mismatch between the intensional interpretation of types as sets of graphs, and the extensional applicative behaviour of strategies. According to the interpretation $[\![\ ]\!]^{\mathcal{S}}$, all the moves in the left-hand part of a $\multimap$ type *must* be used. Nevertheless, the interpretation is *prelogical*, namely the "only if" part holds. Building out of this type interpretation a satisfactory duality à la Stone deserves further study.

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. Retracing some paths in process algebra. In *Concur '96*, volume 1119 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[3] S. Abramsky, E. Haghverdi, and P. Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.

[4] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.

[5] S. Abramsky and M. Lenisa. Linear realizability and full completeness for typed lambda calculi. *Annals of Pure and Applied Logic*, 134:122–168, 2005.

[6] F. Alessi, M. Dezani-Ciancaglini, and F. Honsell. A complete characterization of complete intersection-type preorders. *ACM TOCL*, 4:120–147, 2003.

[7] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type-assignment. *J. Symbolic Logic*, 48:931–940, 1983.

[8] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Logic*, 21(4):685–693, 1980.

[9] M. Coppo, M. Dezani-Ciancaglini, F. Honsell, and G. Longo. Extended type structure and filter lambda models. In G. Lolli, G. Longo, and A. Marcja, editors, *Logic Colloquium '82*, pages 241–262. Elsevier Science Publishers B.V. (North-Holland), 1984.

[10] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. *Z. Math. Logik. Grundlagen*, 27:44–58, 1981.

[11] G. Franco and P. Di Gianantonio. The fine structure of game models. In *Foundation of Software Technology and Theoretical Computer Science (FSTTS '00)*, volume 1974 of *Lecture Notes in Computer Science*, pages 429–441. Springer-Verlag, 2000.

[12] 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.

[13] J.-Y. Girard. Towards a geometry of interaction. In S. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, pages 69–108. American Mathemtical Society, 1989.

[14] F. Honsell and M. Lenisa. Wave-style geometry of interaction models in rel are graph-like lambda models. In *Types'2003*, volume 3085. Springer-Verlag, 2004. Lecture Notes in Computer Science.

[15] Simona Ronchi Della Rocca and B. Venneri. Principal type scheme for an extended type theory. *Theoretical Computer Science*, 28:151–169, 1984.