Discovering new languages from the study of games semantics

Alexis Goyet

Discovering new languages from the study of games semantics

Alexis GOYET
under the supervision of Russ Harmer (PPS) and Pierre Louis Curien (PPS)

23rd of August, 2009

In compliance with the rules set by the MPRI (Master Parisien de Recherche en Informatique[1]), this master thesis starts with a 4 pages summary in French, followed by a 2 pages "syntetic overview". The main body of the thesis starts on page 7.

Summary

La sémantique des jeux fut créée pour donner un modèle du langage PCF ([2] et [3]). Cette démarche a été adaptée depuis à d'autres cas, partant d'autres langages et cherchant à les munir de nouvelles sémantiques.

La motivation de notre approche est différente : nous nous intéressons à un objet sémantique préexistant - l'ensemble des stratégies *cellulaires*, décrite par Russ Harmer dans [4] - et cherchons son interprétation en termes de langages. Parce que cet objet n'est pas en relation directe avec l'ensemble des stratégies *innocentes*, qui correspondent à la sémantique de PCF, le langage obtenu est entièrement nouveau.

Nous donnons en fait deux langages pour ces stratégies cellulaires : une syntaxe spécifique, qui provient d'une première approche directe ; et une syntaxe générique, obtenue comme conséquence d'une démarche de généralisation menée sur la sémantique des jeux. Cette généralisation va au delà de son but initial ; elle introduit des notions générales d'*enregistrements* et d'*opérateurs à historique* sur les jeux. Ceci rend la syntaxe générale commune à un vaste ensemble de classes de stratégies (y compris les stratégies innocentes et cellulaires), tout en rendant visibles, sur les termes, une symétrie nouvelle entre fonctions et environnements.

[1] http ://mpri.master.univ-paris7.fr

La syntaxe spécifique

Seule une présentation informelle de cette syntaxe est donnée (y compris dans le rapport). Le but est uniquement la comparaison avec la syntaxe générique, qui constitue un meilleur cadre pour donner formellement un language pour les stratégies cellulaires. La syntaxe est la suivante :

$$\begin{aligned} t ::= & \mid x \mid \Omega \\ & \mid \mathrm{record}_{r_1 \dots r_m}(\lambda x_1 \dots x_n.t) \\ & \mid \mathrm{record}_{r_1 \dots r_m}((t)t_1 \dots t_n) \\ & \mid \varphi(r)t_1 \dots t_n \end{aligned}$$

Où $x, x_1 \dots x_n$ sont des noms de variables, $r, r_1 \dots r_n$ des noms d'*enregistrements*[2], et φ est un nom de *test*.

Lorsque $\mathrm{record}_r(s')$ apparaît comme sous-terme d'un terme s_1, cela correspond à la requête, de la part de s_1, que la première étape de l'exécution [3] de s' soit "enregistrée" dans r. Lors de l'interaction de s_1 avec un autre terme s_2, au moment de l'exécution de s', un objet r est créé pour répondre à cette requête. Cet enregistrement r est créé en tant qu'objet de s_1 ; différentes copies de sous-termes de s_1 peuvent alors y accéder, incrémentant l'enregistrement. Le terme s_1 peut ensuite faire un branchement conditionnel en fonction de la valeur de l'enregistrement r, avec un sous-terme de la forme $\varphi(r)t_1 \dots t_n$ (le prochain terme exécuté dépend de la valeur de r et du test φ).

Un terme de cette syntaxe spécifique peut donc avoir accès à plus d'information qu'un terme de PCF, grâce aux enregistrements. Ceci correspond à une modification de la condition de *dépendance* (la somme des informations dont les actions d'un terme peuvent dépendre).

Mais, pour correspondre aux stratégies cellulaires, la condition de *visibilité* (spécifiant les actions que le terme peut effectuer) doit également être modifiée. Cette fois, il suffit d'appliquer une restriction par rapport à PCF, qui s'effectue simplement au niveau du typage. Ce typage utilise une structure de pile pour l'environnement ; appeler une variable liée peut ainsi dépiler, et donc perdre, l'accès à d'autres variables.

La syntaxe générique

Dans le λ-calcul habituel, les noms de variables représentent des fonctions que le terme peut appeler ; nous les appelons donc des *noms de fonctions.*

[2] Nous n'utiliserons pas le terme d'"enregistement" en son sens habituel, ce qui devrait éviter toute confusion. Cette ambiguïté n'est pas présente en anglais, où le terme "recording" (et non "record") est utilisé.

[3] Au cours d'une β-réduction avec une stratégie linéaire de tête, la réduction se fait suivant un certain ordre sur les sous-termes, qui correspond à l'ordre des coups en sémantique des jeux.

La syntaxe générique introduit, symétriquement, des *noms d'arguments* :

$$\begin{aligned} t ::= & \mid x \mid \Omega \\ & \mid \lambda x_1 \dots x_n.t \\ & \mid t^{u_1 \dots u_n}(v_1 : t) \dots (v_m : t) \end{aligned}$$

Où les $x_1 \dots x_n$ sont des noms de fonctions, et les $u_1 \dots u_n, v_1 \dots v_m$ sont des noms d'arguments.

Si le terme f attend n arguments, la construction $f^{u_1 \dots u_n}$ leur associe les noms d'arguments $u_1 \dots u_n$. Pour passer un argument aà f, il faut ensuite utiliser $(u_i : a)$, ce qui précise quel argument est transmis. Cette précision serait superflue dans PCF, puisque l'ordre de passage des arguments suffit à lever toute ambiguïté. Mais ici, l'application est plus générale : il est possible de "passer" au terme t l'expression $(v_j : t)$, où v_j n'est pas l'un des u_i. Le nom v_j désigne alors l'un des arguments demandé auparavant, lors d'un autre appel de fonction de la forme $g^{v_1 \dots v_m)}$. Le nom d'argument v_j est le symétrique du nom de fonction que g a attribué à cet argument (disons x). Lors de sa première action, f peut appeler x ; dans ce cas, c'est le terme t qui sera passé comme valeur pour x, indépendamment de toute autre occurrence $(v_j : t')$ (notamment si une autre valeur a déjà été passée pour x). Une description formelle de la réduction est donnée dans le rapport.

Lors de la β-réduction usuelle d'un terme de PCF, le deuxième appel d'une variable x brise une certaine symétrie entre fonction et environnement. La valeur initiale de x est dupliquée, et passée à nouveau. Cette valeur peut être vue comme une continuation ; la dupliquer force donc l'environnement à revenir sur ses pas, lui faisant perdre la mémoire de l'interaction qui s'est déroulée depuis le premier passage de la valeur de x. La possibilité pour l'environnement de nommer aussi x rétablie cette symétrie.

La démarche sémantique

Dans la sémantique des jeux, l'interaction entre une fonction et son environnement est représentée par une partie entre deux joueurs, dans un jeu dont les règles sont déterminées par les types. Lorsqu'une fonction est appelée, elle doit indiquer la valeur de son résultat, ce qu'elle peut faire en appelant une fonction de l'environnement ; la main est alors passée au second joueur (représentant l'environnement). Les fonctions sont ainsi des stratégies sur le jeu.

La présentation usuelle définit un état du jeu comme l'historique complet de l'interaction passée. Chaque nouveau coup doit indiquer un coup qui le *justifie* dans l'historique, de manière compatible avec le type. Cette seule

condition autoriserait des comportements ne correspondant à aucun terme de PCF ; ces termes correspondent en fait à l'ensemble des stratégies vérifiant également la condition d'*innocence*. L'innocence restreint la "vue" qu'une stratégie a de l'historique de l'exécution. La vue est une sous séquence de l'historique, correspondant exactement à la branche du terme qui a déjà été exécutée. Il s'agit des seules informations dont dispose la stratégie pour choisir son prochain coup (condition de dépendance), et, en même temps, des seuls coups disponibles pour justifier le suivant (condition de visibilité). Ainsi, l'appel d'une variable, qui est justifié par le coup correspondant à son abstraction, doit apparaître dans la branche du terme déjà exécutée (c'est-à-dire dans la vue).

Dans le cas des stratégies cellulaires, les conditions de dépendance et de visibilité utilisent des notions de vue dont l'une est plus permissive, et l'autre plus restrictive que la vue des stratégies innocentes. Il s'agit donc d'un type de contrôle d'accès plus fin, imposé aux stratégies.

La syntaxe générique permet de voir ces deux différents types de contrôles d'accès simplement comme des conditions statiques sur les termes (c'est-à-dire des méthode de typage). Le point le plus frappant est que, pour les stratégies cellulaires, la condition de visibilité (qui fait appel à la notion, peu naturelle, d'enregistrement) correspond à appliquer aux noms d'arguments la condition de visibilité donnée pour les noms de fonctions dans la syntaxe spécifique (typage avec pile). La syntaxe générique exhibe cette régularité nouvelle car elle est le produit d'une démarche de généralisation menée sur la présentation de la sémantique des jeux.

Cette généralisation était initialement motivée par la recherche d'une structure catégorique riche sur les stratégies cellulaires, comme celle présentée dans [5] par Harmer, Hyland et Melliès, pour les stratégies innocentes. Dans [5], la condition d'innocence correspond à un foncteur, et la composition des stratégies est rendue possible par des propriétés de ce foncteur.

Notre généralisation permet d'obtenir l'équivalent de ce foncteur pour la notion de cellularité ; ceci permet de voir que les stratégies cellulaires n'ont malheureusement pas de structure équivalente à celle montrée dans [5]. Néanmoins, la généralisation fait apparaître une condition plus faible, qui est suffisante pour décrire la composition. Il s'agit de la notion d'*opérateur à historique*, qui englobe l'innocence et la cellularité. Cette approche se base sur des définitions plus générales pour les jeux eux-mêmes, faisant également apparaître une notion de *plongement* entre jeux.

Cette notion de plongement, qui caractérise de manière locale un rapport entre deux jeux, est un concept central dans notre démarche. Il constitue en premier lieu un outil essentiel dans les preuves ; les opérateurs à historique garantissent par exemple un certain nombre de plongements. Mais un plongement est également, en lui-même, l'expression d'un certain contrôle

d'accès : un plongement de A dans B envoie toute stratégie de A vers une stratégie de B dont les actions, mais aussi les informations, sont restreintes.

La syntaxe générique donne alors un terme pour chaque stratégie de l'opérateur à historique maximal \$, dans lequel plongent tous les autres ; la correspondance est exacte. Grace à ces mêmes plongements, ce langage est commun à tous les opérateurs à historique, et en particulier aux stratégies cellulaires. Mieux, chaque opérateur à historique correspond à une condition statique sur les termes. Tous ces termes peuvent donc interagir à travers la β-réduction de la syntaxe générique.

Synthetic overview ("Fiche synthétique")

The general context

Game semantics was first presented in [2] by Hyland and Ong (and independently in [3], with another presentation), to give a fully abstract model of PCF. Each step of the (head linear) β-reduction corresponds to a move in a game ; different terms of the same type corresponding to strategies on the same game.

In the original presentation, the state of the game is defined as the full history of the interaction up to the current point (along with additional informations). With this definition, not all strategies correspond to PCF terms ; the model is given by the *innocent* strategies. Innocence is a condition which restricts the part of the history which may be used by the strategy. This subsequence of the history is called *P-view* (player view) ; it corresponds exactly to the branch of the PCF term already executed.

Other kinds of views have been defined, to give models of other languages in the same hierarchy as PCF. For example, enforcing no condition corresponds to the λ-calculus with (arbitrary type) references, which contains PCF.

The problem studied

The initial problem was the study of a set of strategies, called *cellular*, recently described by Russ Harmer in [4], which is yet unpublished. Contrarily to innocence, cellularity uses two different kind of views for *visibility* (which actions can be performed) and *dependency* (what information is available). The motivation was to strengthen the visibility condition of innocent strategies (through a translation) ; but the translated strategies broke the dependency condition, which had to be weakened.

The resulting condition is thus neither stronger, nor weaker than innocence. For example, a cellular function can count the number of times it has been called, which an innocent function cannot. Even more unusual is the fact that this object comes purely from the study of the semantics (cellular strategies, with the initial definition, play a central role in proofs of decidability of contextual equivalences) : it did not correspond to any language. Finding such a language was thus the first task of this internship.

The proposed contribution

We first obtained a specific syntax for cellular strategies. We then aimed to study the categorical structure of these strategies, by adapting recent work of Harmer, Hyland and Mellies ([5]), in which innocence correponds to a functor "?" (which, along with dual "!", gives a model of linear logic on the category of games).

A second influence was our previous internship, which studied the decidability of an extension of modal logic on *Kripke structures*. We use thes stucture as basis to generalize the definition of [5], to fit both innocent and cellular strategies (the existing definitions, and thus proofs, were combinatiorial in nature, and very specific).

This generalization went beyond its original goal. It lead to the introduction of general notions of *embeddings* and *history operators* on games ; games themselves being defined as Kripke structures. Kripke structures are oriented graphs with different types of arrows ; thus our games can "inherit" the structure of another one. If this inheritance is faithful, we have an embedding. A history operator takes a game, and builds a new one in which the first is embedded.

As in [5], simple types correspond to very restrictive (linear) games ; and ? and ! are history operators. $\mathcal{C}$, which corresponds to cellular strategies, is also a history operator, as well as the maximal one, \$, in which all the others are embedded. Our main result is a generic syntax for all history operators. This gives terms for cellular strategies, which can interact with the terms for innocent strategies through a general β-reduction.

On the validity of this approach

The generic syntax exhibits an interesting a new kind of symmetry between functions and their environments, both of which can name the terms which are exchanged. This property comes from the symmetry present in the definitions of history operators and embeddings. Visibility conditions, when applied to the new argument names, translate to dependency conditions ; this is striking on cellular (generic) terms, for which recordings are not needed anymore.

Future work

The notion of history operator can be further generalized, by getting closer to the more local notion of embedding. This can be achieved by decomposing history operators in simpler "building blocks", defined as transformations on the modal logic formulas which describe games. This translation of all key concepts in terms of modal logic formulas was our initial approach ; although we did not use this tool to prove our main result, it seems like a very promising way of getting a more computational insight into the subject.

Introduction

Game semantics was first presented in [2] by Hyland and Ong (and independently in [3], with another presentation), to give a fully abstract model of PCF, the lambda calculus with fixpoint operator, booleans and integers.

$$t_1 = \lambda Pg.P(\lambda f.f)g$$

$$t_2 = \lambda xy.x(\lambda z.xy)$$

Their interaction (passing t_2 as the the argument P of t_1) can be seen as a game between the two terms : at the first move, t_1 is asked for its value, and in exchange it has the right to call an argument. This move corresponds to the atom λP in t_1. At the second move, t_1 calls P ; this asks t_2 for its value, in exchange giving it two arguments. This move corresponds to the atom P in t_1, and to λxy in t_2. The rest of the execution can be followed on the terms (the number of a move appearing below the corresponding atom) :

$t_1 =$	$\lambda Pg.$	P	$(\lambda f.$	$f)$	$($	$g)$
	1	2	3	4		
			5	6	7	8

$t_2 =$	$\lambda xy.$	x	$(\lambda z.$	x	$($	$y))$
	2	3	4	5	6	7

The move 6 corresponds in t_2 to a lambda absraction of zero arguments. At the move 5, t_2 calls x for the second time ; thus the move corresponds again to λf in t_1, creating a second copy of $\lambda f.f$ (in the same way that the β-reduction will duplicate the term $\lambda f.f$). The order in which we explored the terms can be seen as a reduction strategy ; it corresponds to linear head reduction.

The figure 1 shows the usual graphic representation of this play. It corresponds to the formal definition of games : each move is played at a certain position in the type of the interaction, and has a *justifying* move among the moves already played. The information of type distinguishes between arguments given at the same move (if t_2 had played y at the third move, it would correspond to a different position in the type). Justification arrows correspond to the link between variables and their abstraction (moves 2 and 5 in t_2 for example), or between a function and its argument (moves 2 and 5 in t_2).

In this definition of games, the current state is the full history of the interaction, and the next move can be played by chosing a position in the type as well as a justifying move (the justifying arrow must be in the tree representation of the type). But some of these moves will never be played

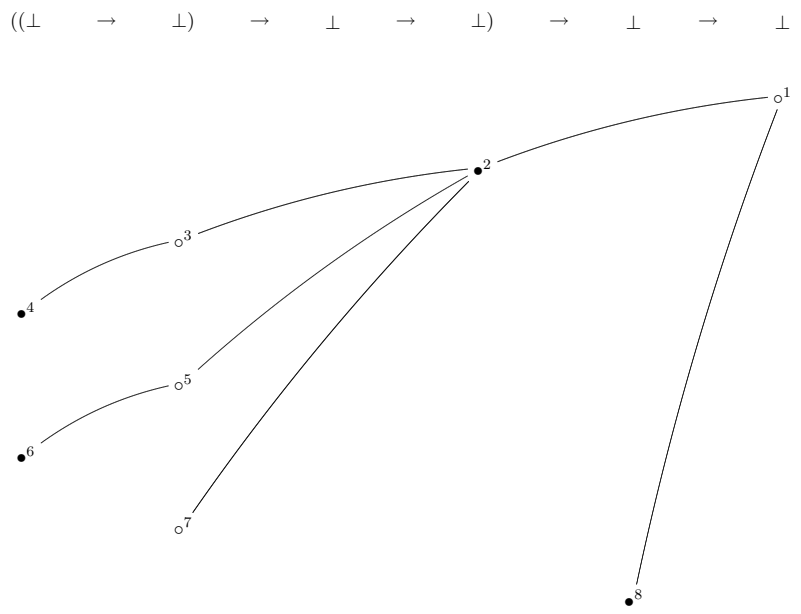

FIG. 1 – The game interaction of t_1 and t_2

by the strategy associated to a PCF term ; thus, to obtain a exactly these strategies, an additional condition is enforced. A subsequence of the history is computed, by iterating backward from the current move, following certain rules. This subsequence, called the P-view, corresponds exactly to the branch of the term already executed (for the active term). The actions of the strategies are restricted to the P-view in two ways : all the justifying moves must be contained in it (visibility), and the strategy may only use this information to decide its next move (dependency). Together these properties are called *innoncence* ; innocent strategies correspond exactly to PCF terms.

Cellular strategies are obtained by dissociating the visibility and the dependency condition. Visibility is taken on the more restrictive OP-view (the intersection of the P-view with the O-view, which is basically the opponent's P-view), but dependency relies on the more lenient "view" (the union of the P-view and the O-view). These strategies are defined by Russ Harmer in [4]. Their interest was prompted by the role of cellular strategies in the semantical world, and not, as is more usual, to give a model for an existing language. The properties they exhibit are also unusual : for example, if a cellular strategy σ is called as a function by τ, σ can count how many times it has been called before, but will only count those calls which were made by τ.

It is natural, at this point, to be curious about the language which would correspond to cellular strategies. We answer this by giving two different

languages.

The first one is specific to cellular strategies; it uses *recordings*, which represent parts of a function's P-view, and are automatically given to functions with which the term interacts. This shows that cellularity is a form of access control, with a neutral party (for example, an operating system) forcing functions to give information. But it does not show the general access control which would englobe both cellular and innocent strategies.

This is achieved by the second language, which is the result of a generalization of game semantics. In this language, arguments are named, not only by the function receiving them, but also by the term passing them; they can then be passed again, symmetrically to the way an argument can be called multiple times. This language is generic for a wide class of conditions, including innocence and cellularity. These conditions, like typing systems, can be checked statically on the terms.

The generalization of game semantics itself was prompted by the work of Harmer, Hyland and Mellies in [5], which shows a remarkable categorical structure of innocent strategies. The types of functions are here proper games, but very restricted (linear) ones. The games for innocent strategies are obtained with a functor "?" on games; it has a dual "!", with an associated model of multiplicative linear logic. Our generalization introduces *history operators* on graphs, which include ?,!, and a functor $\mathcal{C}$ corresponding to cellular strategies. A notion of embedding on games is also introduced, and is central in all our proofs. Finally the generic syntax is shown to correspond to the maximal history operator \$. Every other history operator is embedded in \$, and corresponds exactly to a certain subset of the terms.

In the first section we present the generic syntax along with generalized β-reduction, as well as the specific syntax. In the second section we give the definitions for generalized games, embeddings and history operators. In the third we describe the categorical structure of games, and rapidly sketch the approach of [5]. The fourth section gives the semantics of the generic semantics in terms of genralized games. A final section discusses future work, and explains the use of modal logic.

1 The generic syntax

We first give the generic syntax.

$$\begin{aligned} t ::= & \mid x \mid \Omega \\ & \mid \lambda x_1 \dots x_n.t \\ & \mid t^{u_1 \dots u_n}(v_1 : t) \dots (v_m : t) \end{aligned}$$

Where $x, x_1, \dots x_n$ are called *function* names, and $u_1, \dots u_n, v_1, \dots v_m$ are called *arguments* names. Function names are simply names in the usual sense; we call them this way because the variable they name can be called as

functions. Symmetrically, the objects named by argument names are passed as arguments. For the sake of simplicity, we did not mention the fixpoint operator ; but it is supposed to be present.

1.1 Reduction

We now give a first semantics for this syntax, as an abstract machine executing β-reduction on the terms. The machine maintains an environment Γ, which is an initially empty set of triples (x, u, t). The intention is that (x, u, t) is a reference with value t, and two names u and x. $\Gamma[(u, t)]$ will replace the only (x', u', t') in Γ such that $u' = u$ by (x', u', t) ; when doing this, α-renaming is applied to t. The machine is defined as follows :

$< x, \Gamma > \rightarrow < t, \Gamma >$, where$(x, u, t) \in \Gamma$ (there will be no ambiguity).

$< (\lambda x_1 \dots x_n.t)^{u_1 \dots u_n}(v_1 : t_1) \dots (v_m : t_m), \Gamma > \rightarrow < t, (\Gamma \bigcup_i (x_i, u_i, \Omega))[(v_1, t_m), \dots, (v_m, t_m)] >$

In the second case, the function names $x_1 \dots x_n$ are given corresponding argument names $u_1 \dots u_n$ (there must be an equal number of both, which will be ensured by types), with the default term Ω in the environment. By α-renaming, each name will only be entered once in the environment, preventing any ambiguity. A certain number of values in the environment are then modified.

Consider the two following terms, based on our previous example :

$$t'_1 = \lambda Pg.P^{u,v}(u : \lambda f.(f(u : \Omega))(v : g)$$

$$t'_2 = \lambda xy.x^w(w : \lambda z.x^{w'}(w' : y))$$

t'_1 and t'_2 are the translation in the generic syntax of t_1 and t_2, with the addition of $(u : \Omega)$ in t'_1. Until this subterm is reached, the reduction of the term $(t'_1)^s(s : t'_2)$ will correspond to the usual reduction of $t_1 t_2$ that we already described. After the first reduction step, the environment will contain the triple (P, s, t'_2) ; at the second one, $(x, u, \lambda f.(f(u : \Omega)))$ and (y, v, g) will be added to it ; at the third, $(f, w, \lambda z.x^{w'}(w' : y)))$ is added. Up to this point, the argument names are superfluous.

But, for the fith reduction step (corresponding to the move 6 in the game interaction of t_1 and t_2), we come accross the "application" $f(u : \Omega)$. Of course, f takes no argument ; instead, the value Ω replaces the term which was previously associated to u, and thus x, in the environment. At the next step, as the reduction replaces x by its value, the new term Ω is used, and the execution fails.

If we had not added $(u : \Omega)$ in t'_1, the existing value for x would have been used, in effect forcing the term t'_1 to backtrack to a previous position (the move 5 in the interaction of t_1 and t_2 backtracks to the position of the

move 3). This backtracking (corresponding to a duplication of the term $\lambda f.f$) would have forced t'_1 to "forget" the previous call of x. On the contrary, using the name u a second time (we say *calling* u a second time, as with function names) lets the term "remember" the previous call of x. As another example, the function $(u : \lambda f.f(u : \lambda f.f(u : \Omega)))$ would fail the third time it is called.

1.2 The specific syntax

This syntax was obtained as the first step in the study of cellular strategies, but the formalisation of its reduction rules was not completed. The reason is that the generic syntax, which we obtained later, gives a far better alternative as a formally defined syntax for cellular strategies.

However, an informal presentation of the specific syntax does present an interest. First as an illustration of an original language obtained purely from the study of a semantical object, and secondly as a point of comparison with the generic syntax. Indeed the symmetry which can be seen in the generic syntax is completly absent from the specific one.

The syntax by itself extends λ-calculus with operations to start, and test *recordings*. Recordings give terms informations on the term they are interacting with, which corresponds to a weakening of the dependency condition. To obtain cellular strategies, the visibility condition must at the same time be strengthened ; this is achieved independantly by a typing of the terms. The syntax is as follows :

$$\begin{aligned} t ::= & \mid x \mid \Omega \\ & \mid \mathrm{record}_{r_1 \dots r_m}(\lambda x_1 \dots x_n.t) \\ & \mid \mathrm{record}_{r_1 \dots r_m}((t)t_1 \dots t_n) \\ & \mid \varphi(r)t_1 \dots t_n \end{aligned}$$

Here the names $x, x_1 \dots x_n$ are used for (usual) variables, $r, r_1 \dots r_n$ for recordings and φ for *tests*.

The idea is that each step of reduction will increment a data structure, a recording, which is associated to the term. This assumes a certain representation of the execution of β-reduction (or the associated play in game semantics, once it is defined). If all steps of reduction for a term are recorded in this way, the recording will contain information about corresponding exactly to the P-view of its term (as seen for PCF terms).

The recording of a term F is then automatically passed to the term a in the interaction Fa (function and argument) : the name of the recording of F, say r, should then be bound in a. a can then make a choice based on the value of r, with the construction $\varphi(r)t_1 \dots t_n$: this term will reduce to

one of the $t_1 \dots t_n$, depending the test φ, and the value of r. At this level of description, we do not give any explicit way to build φ. We simply assume a set of tests to exist, and that their behavior can be decided from the value of r by some computable function.

This gives a access to the P-view of F ; but we want a more precise access control, granting only the O-view. This corresponds to moves of the P-view of F which were played while interacting with a and its copies (this is a consequence of the definition of the O-view, as a subsequence of the play in the game). To enforce this, we add the record_r construction, which a must use to explicitly request that the following action be recorded in r. Thus, when a later tests the value of r, it will only provide informations on actions that occured while F interacted with copies of a : only copies of a will have the name r bound (which is necessary to ask that the action be recorded in r) ; and r is kept, not as a variable of a, but of F. This means that r is kept in the environment specific to F ; a can only read its value through tests, and can only modify its value through "record" ; but multiple copies of (subterms of) a can access the same recording r (meaning that recording names should not be subjected to α-renaming).

As an example, here is the translation in this syntax of the terms t'_1 and t'_2 :

$$t''_1 = \lambda P g.P(\text{record}_r(\lambda f.(\varphi(r)\ f\ \Omega))\ g$$

$$t''_2 = \lambda xy.x(\lambda z.xy)$$

Here the only action recorded is the call of the function $\lambda f.f$; t''_1 asks that it be recorded in r (the syntax allows to not record an action, by specifying an empty list of recordings ; in this case the action is writen as in PCF).

When x is called for the first time in t''_2 (move 3), a new recording r is created in the environment of t''_2, containing only the information corresponding to this first call. We suppose that, with this value of r, $\text{test}_\varphi(r) f \Omega$ then reduces to f (move 4), calling f. The variable x is then called a second time (move 5), recording a second action in r. At the next step (corresponding to the move 6 in the game interaction of t_1 and t_2), $\text{test}_\varphi(r) f \Omega$ reduces to Ω (such a test can be chosen), and the execution fails.

Thus t''_1 and t''_2 interact in the same way as t'_1 and t'_2.

1.3 Typing method for the specific syntax

We now briefly describe the typing system, which selects terms of the specific syntax which are OP-visible (and thus cellular).

We base our typing method on the usual method for λ-calculus : an environment Γ is maintained, which is a set of names ; when coming across a lambda abstraction $\lambda x_1 \dots x_n$, $x_1 \dots x_n$ are added in Γ ; when coming accross a variable x, it must be bound, i.e., present in Γ.

We make the following modifications to this method : Γ is a stack of sets of names ("baskets"), and we add all $x_1 \dots x_n$ as a single basket on top of the stack. When we come across x, there are two accepted cases : either x is in the top basket, in which case we continue, or x is in a lower basket, in which case we pop baskets off the stack until x is in the top basket ; after which we continue. All the names in the popped baskets are lost.

The following (Kierstead) term does not satisfy OP-visibility (the possibility to call the name x is "lost" when calling y) :

$$\lambda F.F(\lambda x(F)\lambda y.x)$$

2 Generalized games

2.1 Kripke structures

Kripke structures are oriented graphs with different types of edges, called *modalities*. These modalities correspond to symbols, which exist independently to the structures. Kripke structures are used as models of modal logic. For this purpose, they associate to each of their nodes (called *states*) a set of atomic formulas (which are said to be true at this node). Although it was our initial intention to make extensive use of these formulas, they are not necessary for the proofs in this work, and thus we do not detail their definition. The only formulas which will appear will be self explanatory. The formal definition of Kripke structures follows.

From this point, we assume two sets of symbols, MS and PS (modalities and atomic propositions).

A *Kripke structure* is a triple $\mathcal{K} = (S, R, \lambda)$, where S is a set and its elements are called *states*.

R is a function whose domain contains all $m \in$ MS and $R(m)$ is a relation called a *transition relation* on S, namely $R(m) \in \mathscr{P}(S \times S)$. We call $(a, b) \in R(m)$ an arrow of m, and note it $a \xrightarrow{m} b$. If m is a modality, we note m^{-1} the modality such that $a \xrightarrow{m^{-1}} b$ when $b \xrightarrow{m} a$, and m^* the modality obtained by following any number of arrows of m. A modality is said to be functional if, for every $a \in S$, there is at most one $b \in S$ such that $a \xrightarrow{m} b$.

λ : PS $\to \mathscr{P}$(S) ; we say that the proposition p holds at a if $a \in \lambda(p)$.

2.2 Games

A game $A = (K_A, m_A, I_A, P_A)$ is a Kripke structure K, whoses states are the *moves* of A, along with a distinguished modality m_A, and two propositions I_A (the initial moves of A), and P_A, the moves belonging to player (P) ($O_A = \neg P_A$ corresponds to the opponent's moves). For a modality m, we

define m_P (resp. m_O) as all the arrows of m starting from a player's (resp. opponent's) move.

A P (resp. O) strategy s on A is a modality of A such that $R(s) \subseteq R(m_A)$, s_O is functional and $s_P = (m_A)_P$. Given a O-move a, the move b such that $a \xrightarrow{s} b$, if any, is the answer of s to a. Thus the interaction between a P-strategy s and an O-strategy s' on A is the unique branch of A obtained by following $s \cap s'$ ($s \cap s'$ being short for $R_A(s) \cap R_A(s')$).

2.3 Embeddings

An *embedding* G of A in B is a partial function sending moves of B to moves of A. When $G(x) = a$, we say that a is the label of x in A, noted it x^a. G must verify these properties :

- if $a \xrightarrow{m_A} b$ and $x^a \in A$, then there is exactly one $y \in S_B$ such that y is labelled by b, $x^a \xrightarrow{m_B} y^b$ and $x^a \xrightarrow{m_A} y^b$ ($x^a \xrightarrow{m_A} y^b$ is an arrow in B, the symbol m_A corresponding to a relation on B by R_B)
- if $a \xrightarrow{{m_A}^{-1}} b$ and x^a then there is exactly one y^b such that $x^a \xrightarrow{{m_A}^{-1}} y^b$
- in the previous condition, we also ask that $x^a \xrightarrow{{m_B}^{-1^*}} y^b$.

When there is an embedding of A in B, we note it $A \sqsubseteq B$. Figure 2 gives an example of embedding ; the numbers correspond to moves of A (they are labels for the moves of B). The solid arrows are in m_a, the dotted arrows in m_B.

This allows us to derive from G a function sending strategies of A to strategies of B (this function is still called G) : if s is a strategy of A, $G(s)$'s answer to a move x^a is the unique y^b such that $x^a \xrightarrow{m_B \cap m_A} y^b$, where s's answer to a is b ($m_B \cap m_A$ is short for $R_B(m_B) \cap R_B(m_A)$, which is a modality of B).

Thus embeddings are related to the concept of sub-types : a strategy s on a game A can represent a term of the "type" A (as we saw for PCF). If B is a game in which A is embedded, the behavior of s is well defined as a strategy on B ; this means that every behavior allowed on A will also be allowed on B (in A, every possible action corresponds to an arrow of m_A ; in B, an arrow of $m_B \cap m_A$ corresponds to an action of B and at the same time to an action inherited from A). Thus B is a subtype of the more restrictive type A.

Another way to see this is that B contains a (faithful) copy of A, which can be explored by following arrows of $m_A \cap m_B$ in B. But strategies of B are also allowed to follow arrows of m_B ; by doing this, they switch to a new state, which is also in a copy of A. But it might be a different copy. On the example of figure 2, the only arrow of m_B which is not also an arrow of m_A switchs between two copies of A (which share the initial move 1). A strategy of A, when playing on B through the embedding, can only stay in

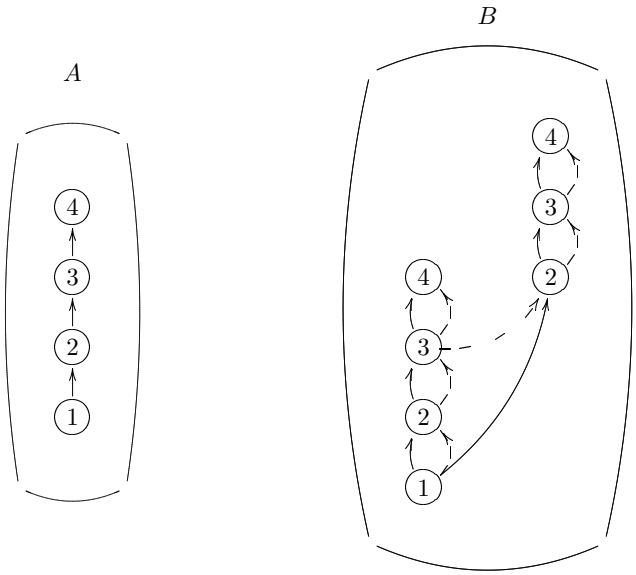

FIG. 2 – an embedding of A in B

the same copy of A. Furthermore, it must answer based only on the label of the move ; it cannot, for example, distinguish between the two moves of B labeled by 2.

In this sense, embeddings enforce access control on B. Not only do they restrict the actions available (in the same way that a sub-type specifies a restriction on the actions of a term), they also restrict the information available to a strategy. These two types of restrictions will correspond to visibility and dependency, respectively.

2.4 Operations on games

We define $A \times B$ as $(K_A + K_B, m_A \cup m_B, I_A \cup I_B, P_A \vee P_B)$, where $K_A + K_B$ is the obvious disjoint union on Kripke structures.

We define $A \Rightarrow B$ as $(K_A + K_B, m, I_B, \neg P_A \vee P_B)$, where m is $m_A \cup m_B$, with additional arrows from every initial move of B to every initial move of A. Using $\Rightarrow$ and $\bot$ (the game with only one initial O-move, and all modalities the empty relation), we build the simple types as in the usual presentation of game semantics (using $\Rightarrow$ for the arrow type). The difference is that here, types are actual games ; although very limited ones. Indeed a player can never backtrack to a previous position in the type ; this will be achieved by the following class of operators.

H is a *history* operator if, for any game A, HA is defined in the following way :

– As the first step, for every initial move a of A, build a move in HA labelled by a (we are at the same time defining a cannonical embedding of A in HA). To each of these moves, associate a set of "bound" moves. The exact computation of this set is what differs from one history operator to another.
– Following steps : to every move x^a built at the previous step, add $x^a \xrightarrow{m_A \cup m_{HA}} y^b$ to HA for each $a \xrightarrow{m}_A b$ (thus one new move and two new arrows were constructed in HA).

  Then, for each move $x'^{a'}$ bound at x^a, and for each $a' \xrightarrow{m}_A b'$, add $x^a \xrightarrow{m}_{HA} y'^{b'}$ and $x'^{a'} \xrightarrow{A} y'^{b'}$ to HA. Here y' is played by "calling" the bound move $x'^{a'}$.

  Compute then the sets of bound moves for the new moves y and y'. They must be in the m_{HA} branch of y (or y') (the branch of a move being all moves in the unique path from an initial move to itself). Finally, for each new move y (or y'), and for each $x^a \xrightarrow{A^*} y^b$ such that $a \xrightarrow{m} b$, where m is a modality of A, add $x^a \xrightarrow{m} y^b$ in HA. The computation of the bound moves may not depend on these modalities, only on m_A and m_{HA}.

It is immediate that this defines an embedding of A in HA, with a total labelling function ; we still note it H. Another important property is that each move of HA is uniquely determined by its m_{HA} branch, enriched by the modality m_A.

2.5 Examples of history operators

The most restrictive history operator is I, in which no move is ever bound ; the game can only proceed by following arrows in A. Thus $IA = A$.

On the contrary, the maximal history operator is \$, in which the whole branch of a move is bound. For any history operator H, the moves in HA have a corresponding move in $\$A$ (they are uniquely determined by their branch). H simply cuts some branches of \$, according to its restrictions on bound moves. This gives a cannonical embedding of every HA in $\$A$, for every game A.

?, resp. !, are the history operators where all O-moves, resp. P-moves, are bound. The operator ? gives the player the right to backtrack, while forbidding the opponent to call a bound move. Thus, if a strategy on $?A$, is asked to play in a larger game (say, $\$A$) through an embedding, it will not "see" moves where the opponent called a bound move (in these cases, opponent switched to another copy of $?A$).

We formally define the *innocent* strategies on a game A as the P-strategies

on $?A$. Indeed, when A is a simple type, this corresponds to the definition of innocent strategies of type A, in the usual presentation of game semantics.

In the figure 2, B is equal to (part of) the game $?A$: the arrow of m_B from 3 to 2 corresponds to calling the bound O-move 1 to play a new move labelled by 2.

$\mathcal{C}$ is the history operator such that, at a move x in $\mathcal{C}A$, the m_A branch of x (in $\mathcal{C}A$) is bound. We formally define *cellular* strategies on a game A to be P-strategies on $\mathcal{C}A$. Again this corresponds exactly to the original definition. The simplicity of this definition, and its symmetry (moves of player and opponent are treated in the same way) pave the way for the symmetry on the syntax that we mentioned in the introduction.

To give a categorical structure on games, we will need an application operator which is less restricted than $\Rightarrow$. This will be achieved by "$\multimap$". $A \multimap B$ is built from $A \Rightarrow B$ in a way similar to history operators, with one extra limitation.

The initial moves of $A \multimap B$ have only themselves as bound move. When appending a move y to x, if we are calling a bound move z, x becomes the (only) bound move at y. If not, y keeps the same bound moves that x had, with no addition (thus there will always be exactly one bound move). As an extra limitation, when calling a (the) bound move for the first time, we may only proceed to an (initial) move of A.

In words, to a move in $A \multimap B$ correspond two (distinguished) moves of $A \Rightarrow B$, one in B, and one in A, one of them being stored as the bound move. This gives the right to switch from one side to the other (the move of the now inactive side becoming the bound one). But we do not allow both moves to be in B, which is why we need the extra condition. Also note that only player has the right to switch (by induction, bound moves are always O-moves)[4]. We have obvious embeddings of A and B in $A \multimap B$, with partial labelling functions.

Figure 3 shows a move in the game $A \multimap B$, where $A = B = \bot \Rightarrow \bot \Rightarrow \bot$. The solid arrows correspond to the structure of $A \Rightarrow B$; the dashed arrows correspond to a certain sequence (or history) of moves of $A \Rightarrow B$; this history constitutes a move of $A \multimap B$. The last move of each side in the sequence are the distinguished moves (noted by a double circle) ; here the move in B is the bound move. This information is sufficient to deduce the whole history (this is because $A \multimap B$, as an history operator, is simple ; in particular it does not allow a state of $A \Rightarrow B$ to be visited twice).

[4]this assuming that O-moves and P-moves alternate in A and B, which is the case for all the games we consider, in particuliar simple types

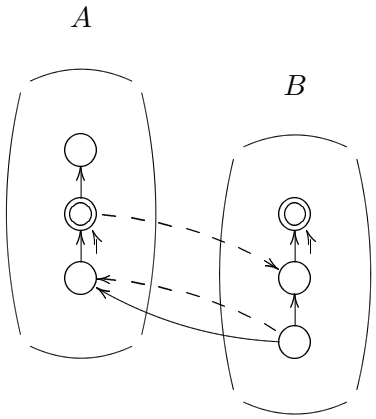

FIG. 3 – a move in the game $(\bot \Rightarrow \bot \Rightarrow \bot) \multimap (\bot \Rightarrow \bot \Rightarrow \bot)$

3 Categorical structures

3.1 The category of games

Games form a category $\mathcal{G}$, with maps from A to B being P-strategies of $A \multimap B$.

Let σ and τ be maps from A to B and B to C, respectively. To define their composition, consider the game $(A \multimap B) \multimap C$; to every move of this game corresponds a triple of moves in A, B and C. We have obvious embeddings of $A \multimap B$, $B \multimap C$ and $A \multimap C$ in it, by taking as label the right pair of moves in this triple; call them $G_{A,B}, G_{B,C}$ and $G_{A,C}$. Now $G_{A,B}(\sigma) \cup G_{B,C}(\tau)$ is a P-strategy of $(A \multimap)B \multimap C)$.

We first define a strategy $\sigma|\tau$ on $(A \multimap B) \multimap C$. To answer to an O-move x in C, $G_{A,B}(\sigma) \cup G_{B,C}(\tau)$ will follow an arrow coming from τ to either y' in C or switch to B. In the second case, it will alternate arrows from σ and τ until it (possibly) emerges to a move y in A or C. If it does, we take y as the answer of $\sigma|\tau$ to x (in the other case, its answer is y'). We define the answer to an O-move of A in the same way.

Here, all the moves played in B were deterministic choices made by σ or τ; the opponent on $A \multimap B$, $B \multimap C$ only had a chance to play on A or C, corresponding to O-moves in $A \multimap C$. Thus $\sigma|\tau$, which was defined on $(A \multimap)B \multimap C$, indeed projects to a strategy on $A \multimap C$, which we take as the composition of σ and τ, and note by $\sigma;\tau$.

One can make the familiar observation here that σ and τ, which are both P-strategies on the respective smaller games, play respectively as player and opponent on B in the larger game.

The identity for this composition is the strategy of $A \multimap A$ which, to a move x^a, responds by switching side and playing the (unique) move labelled by a : the *copycat* strategy.

3.2 Functors

Only the results are presented for this section ; the proofs are given in the appendix.

The operator \$ can be extended to an endofunctor of this category, using an embedding of $A \multimap B$ in $\$A \multimap \$B$ which can be inductively built. The contruction makes use of certain bound moves in $\$A$ and $\$B$; if these moves are also bound in the history operator H, then H is also an endofunctor. $\mathcal{C}$, ? and ! all satisfy this condition.

3.3 The categorical structure of innocent strategies

We now state the general idea of the approach of [5], using these new definitions.

The starting point is the observation that, for any games A and B, $?(A \Rightarrow B) = !A \multimap ?B$; this is true in particular when A and B are simple types. Additionally, ! and ? are a monad and a comonad (respectively), and along with other operators, they give $\mathcal{G}$ the structure of a model for full Intuitionistic Linear Logic.

The question is then to give a categorical meaning to the interaction of two innocent strategies σ and τ, which are now strategies on $!A \multimap ?B$ and $!B \multimap ?C$. This is achieved by taking the image of these categorical maps by the functors ! and ?, respectively, and by invoking a law λ to build the following composition :

$$!A \xrightarrow{\delta} !!A \xrightarrow{!\sigma} !?B \xrightarrow{\lambda} ?!B \xrightarrow{?\tau} ??C \xrightarrow{\mu} ?C$$

which gives the composition of σ and τ as a map of $!A \multimap ?C$ (λ needs to verify some coherence properties, which correspond to being a *distributive law* with respect to the monad and comonad).

3.4 The categorical structure of cellular strategies

Unfortunately, such a rich categorical structure cannot be given to cellular strategies with this approach. Indeed, we would need the equality $\mathcal{C}(A \twoheadrightarrow B) = \mathcal{C}A \twoheadrightarrow \mathcal{C}B$, for some operator $\twoheadrightarrow$. We cannot take $\multimap$ for $\twoheadrightarrow$, because any simple type A with at least three consecutive moves (for example, $\bot \multimap \bot \multimap \bot$) will allow strategies in $A \multimap A$ which are not cellular (if two moves are played on the left side and then one on the right side, $A \multimap A$ allows us to switch back to the left side and continue to the third move ; but doing so violates OP-visibility : we should only be allowed to play an initial move when switching side).

Moreover to define $\twoheadrightarrow$, we need to distinguish moves of $\mathcal{C}A$ according to m_A, not just $m_{\mathcal{C}A}$ (when on the right side, a move on the left side may only be bound if its label in A is initial (in A)). Thus $\twoheadrightarrow$ could only be defined on the image of $\mathcal{C}$, and not as a general operator on games.

Still, our generalization has brought us some isight into this matter. Indeed, for every history operator H, we can show an embedding of $H(A \multimap B)$ into $\$A \multimap \$B$ (call this embedding G_H) . this allows us to define, in categorical terms, the composition of strategies corresponding to any history operators : if σ and τ are respectively strategies on $H(A \multimap B)$ and $H'(B \multimap C)$, we can build the following diagram :

$$\$A \xrightarrow{G_H\sigma} \$B \xrightarrow{G_{H'}\tau} \$C$$

Which gives us the composition of σ and τ as a strategy of $\$A \multimap \$B$. Compared to the diagram for innocent strategies, the result is weaker since the composition is in a bigger game ; but this results applies to more than just innocent strategies. In particular, we can describe the composition of innocent strategies with cellular strategies.

The next section will, formally, give a syntactical equivalent to this result.

4 Game semantics for the generic syntax

4.1 Preliminary

To give the semantics of terms of the generic syntax, we need to extend to these terms some standard notions. For concision we only sketch this section.

For a term t, if we remove all $(u : t')$ except when u has just been defined (with a subterm $t^{u_1 \cdots u_n}$ with $u = u_i$), we obtain a basic lambda-calculus term, which can be well typed in the usual sense. If this is the case, for every $(u : t')$ we previously removed, we can check if it is "compatible" with this type, by putting it back into t, just after u is defined, in place of any existing $(u : t'')$. If all such $(u : t')$ are compatible, we say that t is well typed. This gives us a simple type for t.

For example t'_1 and t'_2 are well typed, with the same types as t_1 and t_2.

We also assume a notion of normal form for terms of the generic syntax.

4.2 Game semantics

We now give a game semantics for the terms in the generic syntax.

Let t be a term which is closed, in normal form and well typed. Let A be the game corresponding to its simple type. We inductively define a P-

strategy $[\![t]\!]$ on $\$A$ for t. We do this while associating subterms of t to the moves reachable by this strategy.

- First step : t is associated to the initial (opponent) move of $\$A$.
- Next steps, first case : if a subterm of the form $\lambda x_1 \dots x_n.t'$ was associated to a move z at the previous step, then it was an O-move (by induction), and t' is of the form $x^{u_1 \dots u_n}(v_1 : t_1) \dots (v_m : t_m)$ (this is because t is in normal form ; the only thing we needed here was that t' starts with a variable x).
  Now, if x is one of the x_i, the strategy answers by following the arrow of $m_A \cap m_{\$A}$ corresponding to x_i (this is a choice in A, given by the position of the type of x_i in the type A of t), and we associate $t' = x^{u_1 \dots u_n}(v_1 : t_1) \dots (v_m : t_m)$ to that move.
  If x is not one of the x_i, there is a move in the m_A branch of z to which we associated a subterm of the form $\lambda y_1 \dots y_r.t''$, with $x = y_j$. This move is bound at z ; the strategy's answer is to call this move. Again we associate t' to the reached move.
- Next steps, second case : if a subterm of the form

$$(t')^{u_1 \dots u_n}(v_1 : t_1) \dots (v_m : t_m)$$

  was associated to a move z at the previous step, then it was a P-move (by induction), and t' is a variable x (normal form). Let z' be a move such that $z \xrightarrow{m_{\$A}} z'$. The move z' corresponds to one of the possible behaviors of x, which is a function played by the opponent (by definition, the strategy $[\![t]\!]$ must contain all arrows like $z \xrightarrow{m_{\$A}} z'$, to distinguish between these possible behaviors).
  The move which justifies z' (the unique move obtained by following m_A^{-1}) indicates when in the interaction this argument was named, and the label of z' distinguishes between the different arguments named at that move. This uniquely determines an argument name u''. If there is a matching $(u'' : t'')$[5] we associate t'' to z'. (If there is no matching $(u'' : t'')$, the strategy will have no answer if opponent plays z' : this corresponds to the default value of Ω in the environment).

This shows the symmetry between argument and function names, since $\$A$ is absolutely symmetric between player and opponent.

It is clear that, for every simple type A, and every strategy s on $\$A$, we can also gives a term corresponding to s by this method.

4.3 History operators as typing methods

This approach seems to single out the history operator \$. But, since we have embeddings of every HA into $\$A$, this also give a syntax to strategies

[5] We take the latest occurence of $(u'' : t'')$ in the $m_{\$A}$ branch, which is what the computation of the environment does in the β-reduction.

of HA. Moreover, the different history operators are obtained from \$ simply by restricting the sets of bound moves. By definition of history operators, these restrictions may only depend on the $m_{\$A}$ branch (along with the m_A structure on this branch), which corresponds exactly to a prefix of the term. This means that the restrictions enforced by H can be checked statically on the terms : it can be seen as a typing method. This easily gives an exact correspondence between terms "well typed according to H" (with the previous meaning), and strategies of HA (with A any simple type).

This does not necessarly mean that these typing methods can be efficiently computed ; we would have to impose further restrictions on history operators for this to be true (with our current definition, the selection of the bound moves could be the result of a non-computable function...). The reason for this (possible) inefficiency is that terms can match a branch of the execution of arbitrary length.

But for the operators we considered, the typing methods are indeed simple. For example, a term is well typed according to ? if "$(u : t)$" only appears just after u is defined, as in the standard way of passing arguments (this because defining u is done at a player move, which is never bound in $?A$).

The case of $\mathcal{C}$ is more interesting : its typing method is obtained simply by taking the method described in section 1 (for the specific syntax), and treating the argument names in exactly the same way as function names (they are put on the stack structure of the environment). The information granted to a term by these bound argument names is exactly the same that would have been granted by the recordings. This result is quite striking when one considers how unnatural the restrictions for storing and passing recordings are.

5 Future work

Some further generalizations of embeddings and history operators can be investigated.

In embeddings, the third condition, which imposes in particular that justifying moves be in the m_A branch, could be removed. This would allow a player to place the opponent in an arbitrary state (a generalization of the backtracking which a term would be subjected to when β-reduction duplicates one of its subterms). This would correspond to the player choosing the opponent's move until a certain point - the equivalent of a debugging software modifying the values of variables in another function. The resulting notion of embedding would then be entierly local.

History operators include all the transformations on games that we were interested in, while being sufficient for the result we wanted to prove. But

they are not the weakest conditions needed to obtain those results. A better method (which was our initial approach) is to define more basic operators on games, as building blocks for the others. For example a "travel" operator would build moves for each branches of the existing game (the resulting game having the structure of a tree).

These basic operators can be seen as transformations on modal logic formulas ; an operator on games would then build the image of a game A by building a new structure B which inherits the modality m_A, and then define m_B by applying a transformation on the formula of m_A (some modalities can be described by a formula).

The motivation for this approach is that, as long as the operations used are simple enough, the decidability results on modal logic ([6]) could be used. It is possible to decide if a formula is true on a given structure (which could allow to check that a strategy verifies properties like innocence). But, given a formula, it is also possible to decide if there exists a structure on which the formula is verified (at some state), through the *tableau* method. This could allow to check a general property of HA, for all A. This method relies on the fact that two states are indistinguishable if the same paths can be followed from them (with respect to modalities and atomic propositions encountered). This is compatible with our definitions for games, in particular the property that moves in HA are uniquely determined by their m_A branch.

Finally, to make full use of the generality of history operators, we intend to study their effects on arbitrary games. The generic syntax was obtained by studying only HA, where A is a simple type ; in [5], $!?A$ was considered, not its strategies (only those corresponding to $?A$). But, in the same way that types can be seen as (linear) games, any game can be seen as the extended "type" of its strategies ; we showed for example that the game HA is a type in the usual sense for strategies of $\$A$ (whereas the usual presentation only made explicit the simple type A for the same strategies). This hints at a way to build more precise types, expressing, statically, a more precise access control (which would benefit from our further generalizations of embeddings and history operators). It would also be interesting to apply a Curry-Howard isomorphism to these terms. This will be the subject of our next investigations.

Conclusion

As a study of semantics, our generalization of games has allowed us to describe general concepts of embeddings and history operators, which are interesting in their own right. They make apparent some important notions which are implicit in the usual presentation, such as the fine grained access

control imposed on interacting strategies, and the symmetry of visibility and dependency.

But these results have counterparts in languages. Indeed, the generic syntax, by its tight correspondence to the semantical object $, often makes regularities of this object apparent - statically - on the terms.

Thus the study of game semantics can indeed help discover new languages.

A Functors on the category of games

Let H be a history operator ; we want to give a sufficient condition to be able to expand H to an endofunctor in the category $\mathcal{G}$.

We first need to define a strategy of $HA \multimap HB$ for every strategy in $A \multimap B$. To do this we attempt to define inductively an embedding of $A \multimap B$ in $HA \multimap HB$.

- An initial move Y of $HA \multimap HB$ corresponds to an initial move y of HB, labelled by b in B (with itself as bound move) ; we label it by the initial move b in $A \multimap B$, with itself as bound move.
- Let X be a move in $HA \multimap HB$, labelled by x^a in HA (a is x's label in A ; the case where $x \in HB$ will be defined symmetrically), with bound move Y labled by y^b in HB.[6]
  Suppose that we have already labelled X by a move α of $A \multimap B$, with bound move β labelled by c in B (α is necessarily labelled by a). We note that α is uniquely determined by its $m_{A \multimap B}$ branch, and, by conditions (2) and (3) of the embedding we are building, the moves of this branch correspond exactly to the $m_{A \multimap B}$ branch of X in $HA \multimap HB$. Thus β is the label in $A \multimap B$ of a move Z in the $m_{A \multimap B}$ branch of X, with Z labelled by z^c in HB.
  We can now label (some) moves reachable from X.
  First, for all arrows $\alpha \xrightarrow{m_{A \multimap B}} \alpha'$ (in $A \multimap B$) with α' labelled by a' in A, there is exactly one X' in $HA \multimap HB$ such that $X \xrightarrow{m_A \cap m_{HA \multimap HB}} X'$ and such that X' is labelled by a' (by the embedding of A in HA, since $\alpha \xrightarrow{m_A} \alpha'$). We label X' by α', and $X \xrightarrow{m_{A \multimap B}} X'$.
  Secondly, for every $\alpha \xrightarrow{m_{A \multimap B}} \beta'$ with β' labelled by b' in B, we have $\beta \xrightarrow{m_B} \beta'$ (with β the bound move of α and the label of Z, as before). If Z is bound at Y (in HB), or if $Z = Y$, there is exactly one Y' in $HA \multimap HB$ such that $Z \xrightarrow{m_B} Y'$ and $Y \xrightarrow{mHB} Y'$ (which means that $X \xrightarrow{m_{HA \multimap HB}} Y'$). We label Y' by β', and $X \xrightarrow{m_{A \multimap B}} Y'$.
  The situation is summed up in the figure 4 : the dashed arrow can be built if the two solid arrows exist. X (with bound move Y), is the current position in $HA \multimap HB$, while α (with bound move β), is the current position in $A \multimap B$.

The construction will succeed if, in the last case, Z is always bound at Y when $Z \neq Y$.

Let H be any history operator such that the previous construction succeeds. We will now prove that H is a functor.

Let σ and τ be P-strategies on $A \multimap B$ and $B \multimap C$; we note $H\sigma$ and $H\tau$ their images in $HA \multimap HB$ and $HB \multimap HC$ (even though $H\sigma$ was already

[6] We do not give the details for the case where X is initial with bound move itself, but it can easily be deduced from the general case.

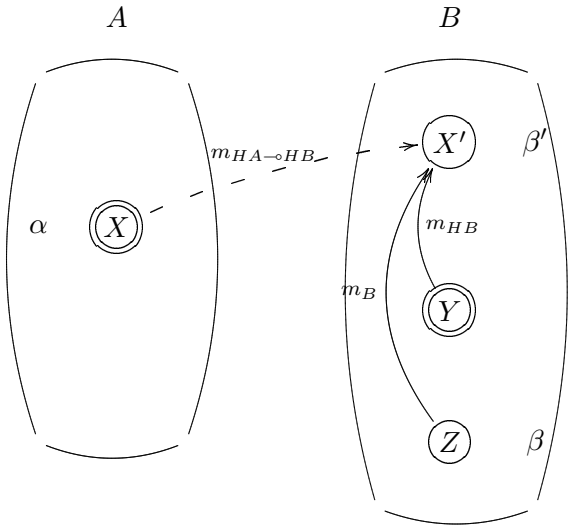

FIG. 4 – Building a new move X' in $HA \multimap HB$

defined as the image of σ in $H(A \multimap B)$; no confusion will arise).

Consider now the composition of $H\sigma$ and $H\tau$ in $(HA \multimap HB) \multimap HC$. Suppose the external opponent plays in HC ; then the composition will follow an arrow of $H\tau$, which will stay in the same copy of $B \multimap C$, then possibly alternate arrows of $H\sigma$ and $H\tau$, staying in the intersection of their copies of $B \multimap C$ and $A \multimap C$, which we can see as a single copy of $(A \multimap B) \multimap C$. Thus their answer will be the same as $H(\sigma;\tau)$, by the embedding of $A \multimap C$ in $HA \multimap HC$ (which is compatible with the two other embeddings, meaning that two moves in $HA \multimap C$ and $HA \multimap HB$ with the same label in HA, for example, will have the same label in A by the corresponding embeddings given by H).

This shows that H, as an operation on maps in the category of games, preserves composition. Since the image by H of the identity is clearly the identity, H defines a functor.

Proposition A.1. *The history operator* \$ *defines a functor on* $\mathcal{G}$

Démonstration. The construction shown previously succeeds ; indeed, using the same notation as before, Z is always in the m_{HB} branch of Y, and thus bound at Y. □

Proposition A.2. *The history operator* $\mathcal{C}$ *defines a functor on* $\mathcal{G}$

Démonstration. Because Z corresponds to β in $A \multimap B$, it is always in the m_B branch of Y, and thus it is bound at Y. □

Again a result on cellular strategy is made almost trivial once we have the general case.

Proposition A.3. *The history operators* ? *and* ! *define functors on* $\mathcal{G}$

Démonstration. If X is on the left side of $?A \multimap ?B$ (if it corresponds to a move in $?A$), then Z is on the right side, it which case it is bound in $?B$ (as an O-move of $?B$). If X is on the right side, this does not work, because Z now corresponds to a P-move of $?A$, which is not bound (recall that the polarity of moves is reversed on the left side of $\multimap$). But if we follow $m^{-1}_{?A \multimap ?B}$ from X, as long as we stay on the right side, every arrow we follow is also in m_B : player will never call a bound move because it plays according to a strategy defined on $A \multimap B$, and player is the only one able to call bound moves in $?B$. Thus we stay in the same copy of $A \multimap B$ until we switch side, which happens at Z (opponent cannot switch side, and β was the last move on the left side of $A \multimap B$ in the branch of α). Thus $Z = Y$.

For !, the argument is symmetric : Z is bound if it is on the left side, and only player can call bound move in the copy of $!A$ (with the polarity inversion), yielding $Z = Y$ if Z was on the right side.

□

Références

[1] P.-L. Curien and H. Herbelin. *Abstract machines for dialogue games. Panoramas et synthèses, 2009.* To appear.

[2] J. M. E. Hyland and C.-H. L. Ong. *On full abstraction for PCF : I, II and III.* Information and Computation, 163 :285-408. 2000.

[3] S. Abramsky, R. Jagadeesan and P. Malacaria. *Full Abstraction for PCF.* information and Computation Vol. 163, pages 409-470. 2000.

[4] R. Harmer. *Cellular strategies and innocent interaction.* 2008. To appear

[5] Russ Harmer, Martin Hyland and Paul-André Melliès. *Categorical combinatorics for innocent strategies.* LICS'07. 2007.

[6] P A. Bonatti, C. Lutz, A. Murano and M Y. Vardi. *The Complexity of Enriched μCalculi.* Logical Methods in Computer Science Vol. 4 (3 :11), pages 1-27. 2008