Abstract Machines for Dialogue Games

P.-L. Curien (CNRS - Paris 7)
H. Herbelin (INRIA-Futurs)

July 16, 2005

Abstract

The notion of abstract Böhm tree has arisen as an operationally-oriented distillation of works on game semantics, and has been investigated in two papers [9, 11]. This paper revisits the notion, providing more syntactic support and more examples (like call-by-value evaluation) illustrating the generality of the underlying computing device. Precise correspondences between various formulations of the evaluation mechanism of abstract Böhm trees are established.

1 Introduction

This paper is a contribution to the paradigm of computation as interaction, by which we mean that a computation is described in terms of a game between two players, one representing the expression to be computed, the other its context, containing information such as the values of its free variables, or where the result should be returned to. This line of work has been pursued in different, but related perspectives, giving rise to rich theories and applications.

- The theory of sequential algorithms of Berry and Curien [4, 5] arose in the investigation of the full abstraction problem for PCF, a famous problem in the semantics of programming languages. PCF is a core pure functional programming language [31], and a fully abstract model is a model capturing the observable differences between programs exactly. Sequential algorithms are mathematical objects that in addition to input-output behaviour record some information about the order of computation. They turned out to provide a fully abstract model, not of PCF, but of a natural extension of PCF with a non-local control operator [8].
- Linear logic, and one of its models in particular – the geometry of interaction – originated in a fine-grained analysis of the cut-elimination process in proof theory, and has brought a wealth of new insights, such as formulas as resources, or proof nets [18, 19].

- Game semantics [10, 24, 2] was triggered by these previous works, and has allowed to give a neat account of a variety of programming features, such as control, non-determinism, references...

In this paper, we adopt a type-free, operationally-oriented view. Our work takes inspiration mostly from the works of Coquand [10] and of the second author [21, 22], and from those of Hyland, Ong, and Nickau [24, 28]. Our key object is the notion of abstract Böhm tree, which is a generalization of that of Böhm tree. Böhm trees are (potentially infinite) normal forms, and play an important role in the theory of the λ-calculus [3]. The main benefit of the generalization is that it offers the right level of generality for explaining the mechanism of computation at hand in the λ-calculus and similar sequential languages. Abstract Böhm trees have been defined and studied in the two articles [9, 11]. Here, we revisit the notion: we provide more syntactic support and more examples (like call-by-value evaluation) illustrating the generality of the underlying computing device. Precise statements on the correspondences between various formulations of the evaluation mechanism of abstract Böhm trees are established.

The paper is organized as follows. Abstract Böhm trees are defined in section 2, where we also introduce our computational engine, called the Geometric Abstract Machine (GAM). A concrete term notation with bound variables, in the style of the λ-calculus, is introduced at the end of this section. Section 3 is devoted to examples, that cover λ-calculus (both normal and non-normal forms), and extensions: PCF and classical PCF; call-by-value evaluation is also treated, and we show finally how Girard's ludics fits in our framework. A remarkable feature of our framework is that the computing device need not be extended or adjusted: only the compilation of the different source languages varies, and the machinery of abstract Böhm trees works as a "universal" device.

Sections 4 and 5 propose equivalent formulations of the GAM: the View Abstract Machine (VAM) highlights the important notion of view (basic to the works of Coquand [10], and of Hyland and Ong [24]), while the Environment Abstract Machine is a straightforward generalization of (a stack-free version) of Krivine Abstract Machine [26]. In the appendix, we establish precise correspondences between these machines.

In section 6, we show how to formalize a lazy, stream-like computational loop calling the GAM again and again in order to produce the full result of a composition; each call of the GAM gets us to the (abstract Böhm tree version of the) next head variable of the composition along a given exploration path. In section 7, we show how to extend the formalism of abstract Böhm trees and the GAM to "non-normal forms".

Finally, in section 8, we discuss η-expansion, which is needed to evaluate (the compilation of) untyped λ-terms. In this section, we also discuss the property of separation, which is the ability of observing differences through execution against a fixed counter-strategy.

2 The Geometric Abstract Machine

In this section, we present the ingredients of our theory, starting with moves, positions, strategies and counter-strategies (section 2.1), and continuing with our computing device governing the interaction strategy / counter-strategy, the Geometrical Abstract Machine (section 2.3). To this effect, we introduce the notions of multiplexed position, multiplexed strategy, multiplexed counter-strategy (section 2.2), which accommodate the process of duplication in the course of computation (when a function calls its argument several times). The termination cases of the machine are spelled out (section 2.4). A new contribution of this paper is section 2.5, where we provide a term notation for abstract Böhm trees.

2.1 Positions and strategies

We suppose given an alphabet A of move names, containing a special symbol $\bullet$, which is the initial move. Positions are sequences of moves with backward pointers for player's moves (that is, moves occurring at even places in the position). We choose to represent pointers by numbers which count the number of opponent's moves between the pointing player's move and the pointed opponent's move. These pointers may be used to relate the bound occurrences to their binders, or to relate values to their return address – i.e., to the root of the subexpression of which they are a (possible) value. Both of these kinds of pointing structure are present in the language PCF (see section 3.2). An even position, or *response*, i.e., a position of even length, is a sequence of the form:

$$a_1[a_2, \overset{i_1}{\hookleftarrow}] \dots a_{2n-1}[a_{2n}, \overset{i_n}{\hookleftarrow}]$$

where $a_j \in A$ for all $j \leq 2n$ and $i_l \in \omega \cup \{\_\}$ for all $l \leq n$. An odd position, or *query*, is defined in the same way, but ends with an opponent's move a_{2n-1}. In a position $p[a, \overset{i}{\hookleftarrow}]$, with $i \in \omega$, the intention is that the move $[a, \overset{i}{\hookleftarrow}]$ points to the move b of p which is at distance $2i+1$ from $[a, \overset{i}{\hookleftarrow}]$. For instance, if $i = 0$ and $p = p_1 b$, then $[a, \overset{i}{\hookleftarrow}]$ points to b. The number i has to be small enough to guarantee that the corresponding b exists. We *will always assume this*, and it will be an (easy) invariant of all the abstract machines presented in this paper that these pointers never become dangling while execution progresses. We use $\_$ to designate free occurrences of player's moves.

We shall let q and r range over queries and responses, respectively. We shall use $[a, \overset{\kappa}{\hookleftarrow}]$ to designate either $[a, \overset{i}{\hookleftarrow}]$ or $[a, \overset{-}{\hookleftarrow}]$.

A *strategy* is a set ϕ of positions sucht that:

- all positions of ϕ are of even length, and of the form $\bullet p$, where $\bullet$ does not occur in p,

- ϕ is closed under prefix,

- if $q[a_1, \overset{\kappa_1}{\hookleftarrow}], q[a_2, \overset{\kappa_2}{\hookleftarrow}] \in \phi$, then $[a_1, \overset{\kappa_1}{\hookleftarrow}] = [a_2, \overset{\kappa_2}{\hookleftarrow}]$.

The last property ensures that a "query", that is, a position of odd length, is uniquely answered in a strategy. Another presentation of a strategy is as a partial function, also written ϕ, from queries (whose player's moves are irrelevant) to player's moves . We write $dom(\phi)$ to denote the domain of definition of this partial function. We shall freely use either of the two presentations.

A counter-strategy is a forest of strategies, where the roots are renamed so as to hook-up with the free moves of the strategy against which they are placed to play with. The renaming is defined as follows:

$$[a \leftarrow \phi] = \{ar \mid \bullet r \in \phi\}$$

Hence $[a \leftarrow \phi]$'s root is labelled by a. A counter-strategy ψ is a union of renamed strategies $[a_1 \leftarrow \phi_1], \ldots, [a_n \leftarrow \phi_n]$ (with all the a_i's distinct and $\neq \bullet$). We write ψ as:

$$\psi = [a_1 \leftarrow \phi_1, \ldots, a_n \leftarrow \phi_n]$$

We assume that $\bullet$ is not only the initial move of all positions of ϕ, but does not occur either in ψ. (These conventions about $\bullet$ apply everywhere in the paper except in section 3.4, where $\bullet$ will be a "real" move expressing the convergence of a function in the weak sense, i.e. the presence of a head λ.)

Nothing prevents us from having infinite horizontal branching after player's moves, although in most examples branching will only be finite. Nothing prevents us either from having infinite positions and infinite depth strategies.

2.2 Multiplexing

Next we introduce multiplexed strategies, which will serve to trace dialogues between strategies and counter-strategies. The idea is that during the course of evaluation, nodes may be visited several times, whence the idea of "opening new copies" (see also section **??**). A *multiplexed* even position is a sequence $\mathbf{p}$ of the form:

$$\langle a_1, \mathbf{j_1} \rangle [a_2, \overset{i_1}{\hookleftarrow}], \ldots, \langle a_{2n-1}, \mathbf{j_n} \rangle [a_{2n}, \overset{i_n}{\hookleftarrow}]$$

where the a's and the i's are as for positions, and where $j_1, \ldots, j_n \in \omega$ encode the multiplexing of opponent's moves. In [11], we used the terminology "dynamic" for what we call "multiplexed" here. The new terminology reflects better the underlying idea of duplication. We could also use the word "thick", following [6].

A multiplexed strategy is defined as a tree Φ of multiplexed positions respecting the same conditions as a strategy, plus the following one: the collection of opponent's moves $\langle a, \mathbf{j}\rangle$ occurring in Φ is in one-to-one correspondence with the set of their second components $\mathbf{j}$. In other words, the multiplexing indices describe a traversal of the multiplexed tree. Moreover, if $\langle a, \mathbf{j}\rangle$ appears in a multiplexed position $\mathbf{p}$ of Φ, then all the $\langle a', \mathbf{j}'\rangle$'s occurring before $\langle a, \mathbf{j}\rangle$ in p must be such that $j' < j$. This is a common constraint of tree traversals: a node cannot be visited unless all its descendants have been visited before.

2.3 The machine

The Geometric Abstract Machine, or GAM, is a simple device describing the interaction between a strategy ϕ and a counter-strategy ψ. The machine duplicates progressively and in alternation (greater and greater portions of) ϕ and ψ. The state of the machine consists of a sequence Γ of multiplexed positions:

$$\text{GAM states:} \qquad \Gamma ::= \{1 \leftarrow \langle \bullet, \mathbf{1}\rangle\} \mid \Gamma\{\nu \leftarrow \mathbf{p}\}$$

The successive items of Γ are numbered $1, \overline{2}, 2, \overline{3}, 3, \ldots, n, \overline{n+1}, \ldots$ (and we use ν to range over these step numbers). The sequence Γ can be put apart, yielding two multiplexed strategies:

$$\begin{aligned} \Phi &= \{p \mid \exists\, n \;\; \Gamma\bullet(2n-1) = p \text{ or } \Gamma\bullet\overline{2n} = p\} \\ \Psi &= \{p \mid \exists\, n \;\; \Gamma\bullet(2n) = p \text{ or } \Gamma\bullet\overline{2n+1} = p\} \end{aligned}$$

Clearly, Φ and Ψ keep exactly the same information as Γ, thanks to the time stamps embodied in the opponent's moves. And while it is simpler to write Γ than to write the pair of Φ and Ψ, it is really Φ and Ψ that we have in mind, and that we shall draw in examples.

Here is some additional notation.

- Given an odd multiplexed position $\mathbf{q}$ ending with $\langle a, \mathbf{n}\rangle$, we set $n = \pi'(\mathbf{q})$.
- Given a multiplexed position $\mathbf{p}$, *erase*($\mathbf{p}$) is the position obtained by erasing the multiplexing information from $\mathbf{p}$.
- Given an odd multiplexed position $\mathbf{q} = \mathbf{q}_1[a_1, \overset{i}{\hookleftarrow}]\langle a_2, \mathbf{j}\rangle$, we write $\mathbf{q}_1 = \mathit{pop}(\mathbf{q})$.

The transition rule of the GAM are presented in figure 1. Notice that the transitions $(\overline{2n})$ and $(\overline{2n+1})$ are essentially the same, the only difference being the exchange of ϕ and ψ. Similarly, the only difference between the transitions $(2n)_b$ and $(2n+1)$ lies in parities. The transition $(2n)_f$ is a variation of $(2n)_b$ which is linked to our choice of encoding for free variables.

Less formally, the transitions $(\overline{2n})$ and $(2n)_b$ can be described as follows:

$$(1)\ \frac{}{\mapsto \{1 \leftarrow \langle \bullet, \mathbf{1} \rangle\}}$$

$$(\overline{2n})\ \frac{hd(\Gamma) = \{2n-1 \leftarrow \mathbf{q}\} \quad \phi(erase(\mathbf{q})) = [a, \overset{\kappa}{\hookleftarrow}]}{\Gamma \mapsto \Gamma\{\overline{2n} \leftarrow \mathbf{q}[a, \overset{\kappa}{\hookleftarrow}]\}}$$

$$(2n)_b\ \frac{hd(\Gamma) = \{\overline{2n} \leftarrow \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \quad \pi'(pop^i(\mathbf{q})) = 2m-1 \quad \Gamma \bullet \overline{2m-1} = \mathbf{r}'}{\Gamma \mapsto \Gamma\{2n \leftarrow \mathbf{r}'\langle a, \mathbf{2n} \rangle\}}$$

$$(2n)_f\ \frac{hd(\Gamma) = \{\overline{2n} \leftarrow \mathbf{q}[a, \overset{-}{\hookleftarrow}]\}}{\Gamma \mapsto \Gamma\{2n \leftarrow \langle a, \mathbf{2n} \rangle\}}$$

$$(\overline{2n+1})\ \frac{hd(\Gamma) = \{2n \leftarrow \mathbf{q}\} \quad \psi(erase(\mathbf{q})) = [a, \overset{\kappa}{\hookleftarrow}]}{\Gamma \mapsto \Gamma\{\overline{2n+1} \leftarrow \mathbf{q}[a, \overset{\kappa}{\hookleftarrow}]\}}$$

$$(2n+1)\ \frac{hd(\Gamma) = \{\overline{2n+1} \leftarrow \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \quad \pi'(pop^i(\mathbf{q})) = 2m \quad \Gamma \bullet \overline{2m} = \mathbf{r}'}{\Gamma \mapsto \Gamma\{2n+1 \leftarrow \mathbf{r}'\langle a, \mathbf{2n+1} \rangle\}}$$

Figure 1: The Geometric Abstract Machine (GAM)

- At stage $(2n-1)$, the machine has reached (a copy of) an opponent's position in ϕ. It looks up in ϕ a uniquely determined player's move $[a, \overset{\kappa}{\hookleftarrow}]$, and feeds it in Γ.

- The machine is now at stage $(\overline{2n})$, and points to a position $\mathbf{q}[a, \overset{i}{\hookleftarrow}]$. The machine uses the information i to retrieve the stage $(2m-1)$ at which the prefix of $\mathbf{q}$ to which $[a, \overset{i}{\hookleftarrow}]$ points has been built. At the stage $(\overline{2m-1})$ immediately preceding stage $(2m-1)$, the machine pointed to a position $\mathbf{r}'$ of ψ. Then the machine will place the move a right at the end of $\mathbf{r}'$, together with a multiplexing information, which is conveniently encoded as the current machine stage $(2n)$.

In summary, two mechanisms are mixed together:

1. The determinacy of ϕ and ψ are put to profit for resolving in turn conflicts about which among the possible opponent's moves pending from a given player's position should be played next.

2. The pointer structure together with the multiplexing structure is used to determine under which player's position the next opponent's move is to be placed.

We refer to these two ingredients of the machinery as to the tree interaction and pointer interaction, respectively.

2.4 Termination

There are exactly three situations in which the GAM is prevented to proceed further.

1. Suppose that when attempting to perform step $(2n)_b$ we find $m = 1$. Then since there is no step $(\overline{1})$ the machine stops. This is what happens in the example of section 3.2: the final move points to the root of ϕ.

2. Suppose that performing step $(\overline{2n+1})$ results in adding a move $[a, \overset{-}{\hookleftarrow}]$. Then the machine is unable to perform step $(2n+1)$, because there is no counterpart of rule $(2n)_f$ for the counter-strategy. This is what happens in the example of section 3.1: the computation terminates because a free variable of ψ has been met.

3. Let $\mathbf{p}$ be the opponent's multiplexed position reached at a stage $(2n-1)$, then step $\overline{2n}$ can be performed only if $\mathit{erase}(\mathbf{p}) \in \mathit{dom}(\phi)$ (and similarly for $(2n)$ and ψ).

We make some observations:

- In [9] we have defined a notion of (abstract) typing that guarantees that case (3) of termination never occurs. An important instance is that of typed, η-long Böhm trees, where each occurrence of a variable is applied to all its arguments, and in

each sequence of abstractions $\lambda\vec{x}.M$ the number of parameters $x_1, \ldots, x_n$ is exactly the number of arguments $N_1, \ldots, N_n$ that the term $\lambda\vec{x}.M$ accepts, according to the types. We consider that case (3) of termination is improper, and in section 8 we examine how to continue the execution when this case shows up.

- In [9] it is also proved, as a corollary of a result of Coquand [10], that the GAM stops after a finite number of steps, provided both ϕ and ψ have finite depth. This result holds without any typing assumptions, unlike usual termination results. This is because β-reduction is in fact more liberal than the GAM and embodies implicit η-expansions (see section 8), which are the only source of non-termination.

- Even when the machine stops satisfactorily (cases (1) or (2)), we may not have the final word on the composition of ϕ and ψ. Consider for example $(\lambda y.y(x))[x \leftarrow tt]$ in PCF. Then the GAM does not produce $(\lambda y.y(tt))$, but $(\lambda y.y(x[x \leftarrow tt]))$. This is similar to the situation with environment machines, which do not compute under λ's. In section 6, we show how to compute (arbitrary long positions) of the composition $\phi \circ \psi$.

2.5 Syntax for abstract Böhm trees

Rather than defining strategies as sets of positions, one can define them (co-recursively) trough the following equations, which specify an abstract syntax for abstract Böhm trees.

Abstract syntax:

$$\begin{array}{l} P ::= [a, \overset{\kappa}{\hookleftarrow}]\ \{(b, M_b) \mid b \in B\} \quad (\kappa = i \text{ or } \kappa = \_) \\ M ::= (P) \end{array}$$

Note that the root of a term M is not named, unlike internal opponent's nodes (M_b is named by b). This is the syntactic counterpart of our convention to denote the initial move of a strategy by the special symbol $\bullet$.

For example, the abstract term denoting

$$\bullet[a_1, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} a_2[a_3, \overset{1}{\hookleftarrow}] \\ a_4[a_5, \overset{0}{\hookleftarrow}] \end{array} \right.$$

is $([a_1, \overset{-}{\hookleftarrow}]\ \{(a_2, ([a_3, \overset{1}{\hookleftarrow}]\{\ \})), (a_4, ([a_5, \overset{0}{\hookleftarrow}]\{\ \}))\})$.

This syntax is reminiscent of De Bruijn notation for the λ-calculus [7], and hence by "reverse engineering" suggests to replace pointers by bound variables. This leads us to the following concrete syntax:

Concrete syntax:

$$P ::= [a, \star]\ \{(b, M_b) \mid b \in B\} \quad (\star = \mathtt{x} \text{ or } \star = \_)$$
$$M ::= (\lambda \mathtt{x}.P)$$

There are free variables $[a, \_]$, and bound variables $[a, \mathtt{x}]$, where $\mathtt{x}$ stands for a set of variables bound at the same place. When $\mathtt{x}$ does not occur free in P, we shall freely write just (P) instead of $(\lambda \mathtt{x}.P)$ (note that the parenthesis is meaningful, as it signals the player's move from (P) to P).

The compilation from concrete to abstract syntax (see section 5.2 for a translation in the converse direction) takes as paramater a list of variable names (taken to be empty initially):

$$\lceil (\lambda \mathtt{x}.P) \rceil_L^{ca} = (\lceil P \rceil_{\mathtt{x} \bullet L}^{ca})$$
$$\lceil [a, \_]\ \{(b, M_b) \mid b \in B\} \rceil_L^{ca} = [a, \overset{-}{\hookleftarrow}]\ \{(b, \lceil M_b \rceil_L^{ca}) \mid b \in B\}$$
$$\lceil [a, \mathtt{x}]\ \{(b, M_b) \mid b \in B\} ca_L \rceil^{=} [a, \overset{L_{\mathtt{x}}}{\hookleftarrow}]\ \{(b, \lceil M_b \rceil_L^{ca}) \mid b \in B\}$$

where

$$\frac{}{(\mathtt{x} \bullet L)_{\mathtt{x}} = 0} \qquad \frac{L_{\mathtt{x}} = i \quad \mathtt{x} \neq \mathtt{y}}{(\mathtt{y} \bullet L)_{\mathtt{x}} = i + 1}$$

The Geometrical Abstract Machine can be formulated in terms of this syntax (see section 5.2).

3 Examples

In this section, we present a collection of examples of abstract Böhm trees. We start with the simplest and motivating example of Böhm trees (or λ-terms in normal form) (section 3.1). (We will show later how to fit λ-terms in general (section 7), up to the problem of η-conversion, which is discussed separately (section 8).) We next move on to a small variation, the PCF trees, which are λ-terms with case statements (section 3.2). A further variation is to add control, à la $\lambda\mu$-calculus [29] (section 3.3). The last two examples are new to this paper: in section 3.4, we present *call-by-value* trees and show how to compile them into abstract Böhm trees, and in section 3.5 we show how to fit Girard's designs (the notion central to ludics [20]) in our framework. These examples reinforce the generality and relevance of the notion of abstract Böhm tree. It is in particular remarkable that in order to execute call-by-value we do not need to change our engine: all happens during the compilation, much like in continuation-passing-style translations [30].

3.1 Böhm trees

We represent Böhm trees, or (potentially infinite) λ-terms in normal form, as trees with pointers. The traditional nodes $\lambda x_1 \dots x_m.y$ of Böhm trees are split in a pair of moves: an opponent's move $\lambda\vec{x}$ and a player's move y.

$$\begin{aligned} B &::= (\lambda x_1 \dots x_m.P) \\ P &::= yB_1 \dots B_n \end{aligned}$$

The alphabet is $A = \omega \cup \mathit{Var}$, where the variables $x \in \mathit{Var}$ are only used as player's moves of the form $[x, \overleftarrow{\hookrightarrow}]$.

We next show how to compile Böhm trees (through the concrete syntax of section 2.5). For defining the translation, it is convenient to prepare the source term in such a way that each bound variable has an indexed format $\mathtt{x}_i$, and that each free variable has a non-indexed format x. Then the translation into the concrete syntax of abstract Böhm trees is straightforward:

$$\begin{aligned} \lceil xB_1 \dots B_n \rceil &= [x, \_]\{(1, \lceil B_1 \rceil), \dots, (n, \lceil B_n \rceil)\} \\ \lceil \mathtt{x}_i B_1 \dots B_n \rceil &= [i, \mathtt{x}]\{(1, \lceil B_1 \rceil), \dots, (n, \lceil B_n \rceil)\} \end{aligned}$$

$$\lceil \lambda \mathtt{x}_1 \dots \mathtt{x}_m.P \rceil = (\lambda \mathtt{x}.\lceil P \rceil) \quad (m \geq 0)$$

We illustrate this with an example.

Strategy for $(u(\lambda x.u(\lambda y.x)))$:

$$\boxed{(}_{\bullet}\,\boxed{u}\,[u, \overleftarrow{\hookrightarrow}] \left\{ \boxed{(\lambda x}_{1}\,\boxed{u}\,[u, \overleftarrow{\hookrightarrow}] \left\{ \boxed{(\lambda y}_{1}\,\boxed{x}^{1}\,[1, \overleftarrow{\hookrightarrow}] \right.\right.$$

Strategy for $[u \leftarrow (\lambda r.r(r(z)))]$

$$\boxed{(\lambda r}_{u}\,\boxed{r}^{0}\,[1, \overleftarrow{\hookrightarrow}] \left\{ \boxed{(}_{1}\,\boxed{r}^{1}\,[1, \overleftarrow{\hookrightarrow}] \left\{ \boxed{(}_{1}\,[z, \overleftarrow{\hookrightarrow}] \right.\right.$$

The trace of the execution of $(u(\lambda x.u(\lambda y.x))[u \leftarrow (\lambda r.r(r(z)))]$, is displayed below. The boxed information (in this example, and in others to follow) maintains some reminders of the underlying concrete syntax that may help the reader to follow what is going on.

Function multiplexed tree:

$$\boxed{(}\ \langle \bullet, \mathbf{1}\rangle[u, \overset{-}{\hookleftarrow}]\ \begin{cases} \boxed{(\lambda x}\ \langle 1, \mathbf{3}\rangle[u, \overset{-}{\hookleftarrow}]\ \begin{cases} \boxed{(\lambda y}\ \langle 1, \mathbf{5}\rangle[1, \overset{1}{\hookleftarrow}] \end{cases} \\ \boxed{(\lambda x}\ \langle 1, \mathbf{7}\rangle[u, \overset{-}{\hookleftarrow}]\ \begin{cases} \boxed{(\lambda y}\ \langle 1, \mathbf{9}\rangle[1, \overset{1}{\hookleftarrow}] \end{cases} \end{cases}$$

Argument multiplexed tree:

$$\begin{cases} \boxed{(\lambda r}\ \langle u, \mathbf{2}\rangle[1, \overset{0}{\hookleftarrow}]\ \begin{cases} \boxed{(}\ \langle 1, \mathbf{6}\rangle[1, \overset{1}{\hookleftarrow}]\ \begin{cases} \boxed{(}\ \langle 1, \mathbf{10}\rangle\ [z, \overset{-}{\hookleftarrow}] \end{cases} \end{cases} \\ \boxed{(\lambda r}\ \langle u, \mathbf{4}\rangle[1, \overset{0}{\hookleftarrow}] \\ \boxed{(\lambda r}\ \langle u, \mathbf{8}\rangle[1, \overset{0}{\hookleftarrow}] \end{cases}$$

Here the strategy and counter-strategy are just paths, hence we could hardly illustrate the tree interaction. But we did illustrate pointer interaction. Let us describe the first steps of the execution. We start by applying rule (1), which lets us point to the root of the strategy:

$$\boxed{(}\ \langle \bullet, \mathbf{1}\rangle$$

Then we apply rule $(\overline{2})$, which leads us to a free move:

$$\boxed{(}\ \langle \bullet, \mathbf{1}\rangle[u, \overset{-}{\hookleftarrow}]$$

Therefore, we apply rule $(2)_f$, and we point now to the root of the (unique tree of the) counter-strategy:

$$\boxed{(\lambda r}\ \langle u, \mathbf{2}\rangle$$

By rule $(\overline{3})$ we reach a move 1 which points to the move reached at step (2), let us say for short that it points to 2:

$$\boxed{(\lambda r}\ \langle u, \mathbf{2}\rangle[1, \overset{0}{\hookleftarrow}]$$

Hence the move at step (3) is played under $\overline{2}$:

$$\boxed{(}\ \boxed{u}^{-}$$
$$\langle \bullet, \mathbf{1}\rangle[u, \hookleftarrow] \left\{ \begin{array}{l} \boxed{(\lambda x} \\ \langle 1, \mathbf{3}\rangle \end{array} \right.$$

Step ($\overline{4}$) leads us then to open a new copy of the counter-strategy:

$$\boxed{(}\ \boxed{u}^{-}$$
$$\langle \bullet, \mathbf{1}\rangle[u, \hookleftarrow] \left\{ \begin{array}{l} \boxed{(\lambda x}\ \boxed{u}^{-} \\ \langle 1, \mathbf{3}\rangle[u, \hookleftarrow] \end{array} \right.$$

This is where multiplexing begins:

$$\boxed{(\lambda r}$$
$$\langle u, \mathbf{4}\rangle$$

A little later, say, when we have performed step ($\overline{7}$), we arrive to a move 1 which points to 2:

$$\left\{ \begin{array}{l} \boxed{(\lambda r}\ \boxed{r}^{0} \\ \langle u, \mathbf{2}\rangle[1, \hookleftarrow] \left\{ \begin{array}{l} \boxed{(}\ \boxed{r}^{1} \\ \langle 1, \mathbf{6}\rangle[1, \hookleftarrow] \end{array} \right. \\ \\ \boxed{(\lambda r}\ \boxed{r}^{0} \\ \langle u, \mathbf{4}\rangle[1, \hookleftarrow] \end{array} \right.$$

Then (7) tells us to play 7 under $\overline{2}$, opening a new copy of (a subtree of the) strategy:

$$\boxed{(}\ \boxed{u}^{-}$$
$$\langle \bullet, \mathbf{1}\rangle[u, \hookleftarrow] \left\{ \begin{array}{l} \boxed{(\lambda x}\ \boxed{u}^{-} \\ \langle 1, \mathbf{3}\rangle[u, \hookleftarrow] \left\{ \begin{array}{l} \boxed{(\lambda y}\ \boxed{x}^{1} \\ \langle 1, \mathbf{5}\rangle[1, \hookleftarrow] \end{array} \right. \\ \\ \boxed{(\lambda x} \\ \langle 1, \mathbf{7}\rangle \end{array} \right.$$

Etc... (in general, when $\overline{n}$ points to m, then n is to be placed under $\overline{m}$). Note the importance of keeping the multiplexing information precise: for example, the move $\overline{10}$ points to 7, not to 3, and this is precisely what gets us out of a loop.

3.2 PCF trees

Our next example comes from the language PCF. We refer to [1] for background. Let us just mention here that the trees presented here, which we call PCF trees, provide a term model for the language PCF, a core functional programming language which has been a subject of focus of many works in denotational semantics, much in the same way as Böhm trees provide a term model for λ-calculus. PCF trees have the form

$$B ::= (\lambda \vec{x}.P)$$
$$P ::= case\ yB_1 \cdots B_n\ [\underline{n_1} \Rightarrow B'_1, \ldots, \underline{n_i} \Rightarrow B'_i, \ldots]$$

where the $\underline{n_i}$'s are distinct natural numbers (or boolean values). (We use underlining for integer constants to distinguish them from the moves i used in the compilation of variables and arguments.) We compile PCF Böhm trees much like Böhm trees, adopting the same convention about variables (cf. section 3.1):

$$\lceil case\ xB_1 \cdots B_n\ [\ldots, \underline{n_i} \Rightarrow B'_i, \ldots]\rceil = [x, \_]\{(1, \lceil B_1\rceil), \ldots, (n, \lceil B_n\rceil), \ldots, (\underline{n_i}, \lceil B'_i\rceil), \ldots\}$$
$$\lceil case\ \mathtt{x}_j B_1 \cdots B_n\ [\ldots, \underline{n_i} \Rightarrow B'_i, \ldots]\rceil = [j, \mathtt{x}]\{(1, \lceil B_1\rceil), \ldots, (n, \lceil B_n\rceil), \ldots, (\underline{n_i}, \lceil B'_i\rceil), \ldots\}$$

$$\lceil \lambda \mathtt{x}_1 \ldots \mathtt{x}_m.P\rceil = (\lambda \mathtt{x}.\lceil P\rceil) \quad (m \geq 0)$$

With respect to λ-calculus, we add the constants to the alphabet (in this section as well as in section 3.3). So, if we limit ourselves to the type hierarchy built over *Bool*, the alphabet is $A = \omega \cup Var \cup \{tt, ff\}$. The example below illustrates tree interaction.

Strategy for $(case\ f(tt)\ [ff \Rightarrow tt])$:

$$\begin{array}{c} \boxed{(}\ \boxed{case\ f} \\ \bullet\ [f, \overset{-}{\hookleftarrow}] \end{array} \left\{ \begin{array}{l} \boxed{(} \\ 1\ [tt, \overset{0}{\hookleftarrow}] \\ ff\,[tt, \overset{1}{\hookleftarrow}] \end{array} \right.$$

Strategy for $[f \leftarrow (\lambda x.case\ x\ [tt \Rightarrow ff, ff \Rightarrow tt])]$:

$$\begin{array}{c} \boxed{(\lambda x}\ \boxed{case\ x} \\ f\ \ [1, \overset{0}{\hookleftarrow}] \end{array} \left\{ \begin{array}{l} tt[ff, \overset{1}{\hookleftarrow}] \\ ff\,[tt, \overset{1}{\hookleftarrow}] \end{array} \right.$$

Function multiplexed tree:

$$\begin{array}{c} \boxed{(}\ \ \boxed{case\ f} \\ \langle \bullet, \mathbf{1}\rangle\ [f, \overset{-}{\hookleftarrow}] \end{array} \left\{ \begin{array}{l} \boxed{(} \\ \langle 1, \mathbf{3}\rangle\ [tt, \overset{0}{\hookleftarrow}] \\ \langle ff, \mathbf{5}\rangle [tt, \overset{1}{\hookleftarrow}] \end{array} \right.$$

Argument multiplexed tree:

$$\begin{array}{c} \boxed{(\lambda x}\ \boxed{case\ x} \\ \langle f, \mathbf{2}\rangle\ [1, \overset{0}{\hookleftarrow}] \end{array} \left\{ \begin{array}{l} \langle tt, \mathbf{4}\rangle [ff, \overset{1}{\hookleftarrow}] \end{array} \right.$$

3.3 Classical PCF

The following variant of PCF allows us to introduce explicit control on where the values are to be sent to.

$$\begin{array}{l} M ::= (\lambda \vec{x}.\mu\beta.c) \\ c ::= \mathit{case}\ x\vec{M}\ [\underline{\vec{n}} \to \vec{c}] \mid [\alpha]\underline{n} \end{array}$$

Terms of this syntax are called classical PCF trees. Their execution is driven by the following abstract machine:

$$\frac{\rho(x) = (\lambda \vec{z}\mu\beta.c')[\rho']}{\langle \mathit{case}\ x\vec{M}\ [\underline{\vec{n}} \to \vec{c}] \mid \rho\rangle \quad\longrightarrow\quad \langle c' \mid \rho'[\vec{z} \leftarrow \vec{M}[\rho], \beta \leftarrow [\underline{\vec{n}} \to \vec{c}][\rho]]\rangle}$$

$$\frac{\rho(\alpha) = [\underline{\vec{n}} \to \vec{c}[\rho'] \text{ and } n = n_i}{\langle [\alpha]\underline{n} \mid \rho\rangle \quad\longrightarrow\quad \langle c_i \mid \rho'\rangle}$$

The compilation in concrete syntax is as follows (again, we assume that bound variables take an indexed format $\mathtt{x}_i$ and that free variables take a non-indexed format x):

$$\lceil(\lambda\vec{\mathtt{x}}\mu\beta.c)\rceil_\rho = (\lambda\mathtt{x}.\lceil c\rceil_{\rho[\beta\leftarrow x]})$$

$$\lceil[\alpha]\underline{n}\rceil_\rho = (!\underline{n}, \rho(\alpha))$$

$$\begin{array}{l} \lceil \mathit{case}\ xM_1 \ldots M_m[\ldots, \underline{n_j} \to c_j, \ldots]\rceil_\rho \\ \quad = [x, \_]\ \{\ldots, (?i, \lceil M_i\rceil_\rho), \ldots, (!\underline{n_j}, (\lceil c_j\rceil_\rho), \ldots\} \end{array}$$

$$\begin{array}{l} \lceil \mathit{case}\ \mathtt{x}_i M_1 \ldots M_m[\ldots, \underline{n_j} \to c_j, \ldots]\rceil_\rho \\ \quad = [i, \mathtt{x}]\ \{\ldots, (?i, \lceil M_i\rceil_\rho), \ldots, (!\underline{n_j}, (\lceil c_j\rceil_\rho), \ldots\} \end{array}$$

3.4 Classical call-by-value PCF

In call-by-value λ-calculus [30], the β-rule $(\lambda x.M)N \to [M \leftarrow x]N$ is allowed only when N is a value, where a value V is an abstraction $\lambda x.N'$ or a variable. Hence a non-value is a term of the form MN. Then, if V, M_1 are normal forms, $(\lambda y.M_1)(xV)$ is also a normal form. It is more readable and suggestive to write the latter *let* $y = xV$ *in* M_1. Adding case statements and control as in section 3.3, we arrive at the following syntax of *classical call-by-value Böhm trees*:

$$\begin{array}{ll} \text{Values} & V ::= \lambda(z, \beta).c \mid \underline{n} \mid x \\ \text{Commands} & c ::= \mathit{let}\ x = yV\ \mathit{in}\ c \mid \mathit{case}\ x\ [\underline{\vec{n}} \to \vec{c}] \mid [\alpha]V \end{array}$$

(An alternative syntax for $(\lambda(z,\beta)$ is $\lambda z\mu\beta.)$ The expressions of the second category are called commands, and in a typed setting have the special type $\perp$ (see below). In this syntax, a value x is meant to be of basic type (and is indeed of basic type in the typed version given at the end of the section). This is not restrictive in terms of expressive power, as we can express higher-type variables y as values in η-expanded form. For example, at first-order:

$$\lambda(z,\alpha).(let\ u = yz\ in\ [\alpha]u)\ .$$

It is not restrictive either to limit application of a variable to one argument, as we can encode $(let\ xV_1V_2)$ as $(let\ z = xV_1\ in\ let\ y = g)$. Moreover, a key difference between call-by-name and call-by-value is that in call-by-value an abstraction is a value, or a result of computation. Hence the evaluation stops at the first abstraction met, while in call by–name, the machine treats abstractions by blocks and proceeds until it finds the head variable. Compare for example

$\lambda xy.xM$	call-by-name
$\lambda(x,\alpha).[\alpha](\lambda(y,\beta).(let\ z = xV\ in\ [\alpha]z)$	call-by-value

In the call-by-name case, the dialogue behind $\lambda xy.xM$ can be paraphrased as:

- Question: What is the value of the head variable?
- Answer: x.

In the call-by-value setting, we must also know y, even it if does not occur in V. So the dialogue that we want to formalize is different:

- Answer: I am an abstraction $(\lambda(x,\alpha))$.
- Question: If I give you an abstraction as value for x, what more can you tell me about you?
- Answer: I shall become an abstraction $(\lambda(y,\beta))$ and my result should be returned to where α was declared bound.
- If I give you a value for y, what will you do next?
- I shall apply x to V.

We hope that these intuitions will be helpful for the rest of this section, which is a bit technical. The two syntactic categories are called values and commands, respectively.

The evaluation of our call-by-value trees is performed by the following abstract machine (we refer to [12]) for background on the notation $\tilde{\mu}$):

$$\frac{\rho(y) = (\lambda(z,\beta).c')[\rho']}{\langle \mathit{let}\ x = yV\ \mathit{in}\ c \mid \rho\rangle \quad\longrightarrow\quad \langle c' \mid \rho'\,[z \leftarrow V[\rho], \beta \leftarrow (\tilde{\mu}x.c)[\rho]]\rangle}$$

$$\frac{\rho(x) = \underline{n_i}[\rho']}{\langle \mathit{case}\ x\ [\vec{\underline{n}} \to \vec{c}] \mid \rho\rangle \quad\longrightarrow\quad \langle c_i \mid \rho\rangle}$$

$$\frac{\rho(\alpha) = (\tilde{\mu}x.c')[\rho']}{\langle [\alpha]V \mid \rho\rangle \quad\longrightarrow\quad \langle c' \mid \rho'\rangle x \leftarrow V[\rho]]}$$

Here is the compilation into our concrete syntax (we have been guided by the game semantics of call-by-value proposed by Honda and Yoshida [23]):

$\lceil \lambda(z^{\iota}, \beta).c \rceil_\rho = \bullet\,\{\ldots, (?\underline{n}, (\lambda \mathtt{x}.\lceil c \rceil_{\rho[z \leftarrow \underline{n}, \beta \leftarrow \mathtt{x}]})), \ldots\}$ ($\mathtt{x}$ fresh)
$\lceil \lambda(z^{\sigma_1 \to \sigma}, \beta).c \rceil_\rho = \bullet\,\{(?\bullet, (\lambda \mathtt{x}.\lceil c \rceil_{\rho[z \leftarrow \mathtt{x}, \beta \leftarrow \mathtt{x}]}))\}$ ($\mathtt{x}$ fresh)
$\lceil \underline{n} \rceil_\rho = \underline{n}$
$\lceil x^{\iota} \rceil_\rho = \rho(x)$

$\lceil \mathit{case}\ x\ [\vec{\underline{n}} \to \vec{c}] \rceil_\rho = \lceil c_i \rceil_\rho \quad (\rho(x) = \underline{n_i})$
$\lceil [\alpha]V \rceil_\rho = [!*, \rho(\alpha)]\ \{\vec{T}\} \quad (\rho(\alpha)\downarrow, \lceil V \rceil_\rho = *\,\{\vec{T}\})$
$\lceil [\alpha]V \rceil_\rho = [(!*, \alpha), \_]\ \{\vec{T}\} \quad (\rho(\alpha)\uparrow, \lceil V \rceil_\rho = *\,\{\vec{T}\})$
$\lceil \mathit{let}\ x^{\iota} = yV\ \mathit{in}\ c \rceil_\rho = [?*, \rho(\mathtt{y})]\ \{\vec{T}, \ldots, (!\underline{n}, (\lceil c \rceil_{\rho[x \leftarrow \underline{n}]})), \ldots\} \quad (\rho(y)\downarrow, \lceil V \rceil_\rho = *\,\{\vec{T}\})$
$\lceil \mathit{let}\ x^{\iota} = yV\ \mathit{in}\ c \rceil_\rho = [(?*, y), \_]\ \{\vec{T}, \ldots, (!\underline{n}, (\lceil c \rceil_{\rho[x \leftarrow \underline{n}]})), \ldots\} \quad (\rho(y)\uparrow, \lceil V \rceil_\rho = *\,\{\vec{T}\})$
$\lceil \mathit{let}\ x^{\sigma_1 \to \sigma} = yV\ \mathit{in}\ c \rceil_\rho = [?*, \rho(\mathtt{y})]\ \{\vec{T}, (!\bullet, (\lambda \mathtt{z}.\lceil c \rceil_\rho[x \leftarrow \mathtt{z}]))\} \quad (\rho(y)\downarrow, \lceil V \rceil_\rho = *\,\{\vec{T}\}, \mathtt{z}$ fresh)
$\lceil \mathit{let}\ x^{\sigma_1 \to \sigma} = yV\ \mathit{in}\ c \rceil_\rho = [(?*, y), \_]\ \{\vec{T}, (!\bullet, (\lambda \mathtt{z}.\lceil c \rceil_\rho[x \leftarrow \mathtt{z}]))\} \quad (\rho(y)\uparrow, \lceil V \rceil_\rho = *\,\{\vec{T}\}, \mathtt{z}$ fresh)

$\lceil c\,[\ldots, x_i \leftarrow \underline{n_i}, \ldots, x_j \leftarrow V_j^{\iota \to \sigma}, \ldots, x_k \leftarrow V_k^{(\sigma_3 \to \sigma_2) \to \sigma_1}, \ldots] \rceil_{[]} =$
$\quad \lceil c \rceil_{[\ldots x_i \leftarrow \underline{n_i}, \ldots]}\,[\ldots, (?\underline{n}, x_j) \leftarrow (?\underline{n}, T_n), \ldots, (?\bullet, x_k) \leftarrow (?\bullet, T)]$
$\qquad (\lceil V_j \rceil_{[]} = \bullet\,\{\ldots, (?\underline{n}, T_n), \ldots\}, \lceil V_k \rceil_{[]} = \bullet\,\{(?\bullet, T)\})$

The following explanations should help parsing the definition. The compilation is relative to an environment, which records the values of variables of basic types, and which also rearranges the names of bound variables so as to pass correctly from the syntax of call-by-value Böhm trees to the syntax of abstract Böhm trees. The compilation of $\lambda(z,\beta).c$ is conform to the informal dialogue suggested above. The compilations of $\underline{n}$ and x are self-explanatory (remember that x must have a basic type in our syntax). As

for commands, the compilation of *case* $x\ [\vec{\underline{n}} \to \vec{c}]$ is easy, since the environment ρ has "precomputed" the branch to be chosen. The compilation of $[\alpha]V$ splits into two cases according to whether α is bound or free. The compilation of *let* $x = yV$ *in* c splits into four cases, taking additionally into account the type (basic or not) of x. The common idea between the four cases for the *let* construct is that the compilation of *let* $x = yV$ *in* c with respect to ρ is obtained by adding sons to the root of the compilation of V (with respect to ρ) corresponding to the different possible values of yV, under which the compilation of c (with respect to ρ extended with a suitable value for x) is placed.

The notation $(?\underline{n}, x_j) \leftarrow (?\underline{n}, T_n)$ is an obvious variation on the notation $[a \leftarrow \phi]$ of section 2.1. (Note also that in this section $\bullet$ is a player's move from the point of view of the strategy ϕ. In call-by-value we use the symbol $\bullet$ to denote the information "I am a function".) Finally, we notice that the translation fixes some details about the names of moves. Our use of ! and ? here is somewhat reminiscent of the use of n and $\underline{n}$ in the two previous sections to distinguish between arguments and continuations.

The alphabet in this section is thus the following:

$$A ::= (?\bullet, \xi) \mid (?n, \xi) \mid ?\bullet \mid ?n \mid !\bullet \mid !n$$

where ξ ranges over ordinary variables x and continuation variables α.

As an illustration, take:

$$\begin{array}{l} \mathit{let}\ y = x(\lambda(z,\beta).\mathit{case}\ z\ [3 \to [\beta]5, 4 \to [\beta]9])\ \mathit{in}\ \mathit{let}\ u = vy\ \mathit{in}\ [\alpha]u \\ \quad [x \leftarrow \lambda(t,\gamma).\mathit{let}\ r = t3\ \mathit{in}\ \mathit{case}\ r\ [5 \to [\gamma]7]\ ,\ v \leftarrow \lambda(w,\delta).\mathit{case}\ w\ [7 \to [\delta]8]] \end{array}$$

The compilation gives:

$$\begin{array}{l} [(?\bullet, x), \overleftarrow{\hookrightarrow}]\{(?3, (\lambda \mathtt{x}_1.[!5, \mathtt{x}_1])), (?4, (\lambda \mathtt{x}_1.[!9, \mathtt{x}_1])), .., \\ \qquad\qquad (!a, ([(?a, v), \hookrightarrow]\{.., (!b, ([(!b, \alpha), \overleftarrow{\hookrightarrow}])), ..\})), ..\} \end{array}$$

$$\begin{cases} ((?\bullet, x), (\lambda \mathtt{x}_2.[?3, \overset{\mathtt{x}_2}{\hookleftarrow}]\{(!5, ([!7, \overset{\mathtt{x}_2}{\hookleftarrow}]\})) \\ \\ ((?7, v), (\lambda \mathtt{x}_3.[!8, \overset{\mathtt{x}_3}{\hookleftarrow}])) \end{cases}$$

Or, as trees with pointers:

$$\phi \quad [(?\bullet, x), \overleftarrow{\hookrightarrow}] \begin{cases} ?3[!5, \overset{0}{\hookleftarrow}] \\ ?4[!9, \overset{0}{\hookleftarrow}] \\ \vdots \\ !a[(?a, v), \hookrightarrow] \begin{cases} \vdots \\ !b[(!b, \alpha), \overleftarrow{\hookrightarrow}] \\ \vdots \end{cases} \\ \vdots \end{cases}$$

$$\psi \quad \begin{cases} (?\bullet, x)[?3, \overset{0}{\hookleftarrow}]!5[!7, \overset{1}{\hookleftarrow}] \\ \\ (?7, v)[!8, \overset{0}{\hookleftarrow}] \end{cases}$$

Here is the execution:

$$\phi \quad [(?\bullet, x), \overleftarrow{\hookrightarrow}] \begin{cases} \langle ?3, \mathbf{3}\rangle[!5, \overset{0}{\hookleftarrow}] \\ \langle !7, \mathbf{5}\rangle[(?7, v), \overleftarrow{\hookrightarrow}] \{ \ \langle !8, \mathbf{7}\rangle[(!8, \alpha), \overleftarrow{\hookrightarrow}] \end{cases}$$

$$\psi \quad \begin{cases} \langle (?\bullet, x), \mathbf{2}\rangle[?3, \overset{0}{\hookleftarrow}]\langle !5, \mathbf{4}\rangle[!7, \overset{1}{\hookleftarrow}] \\ \\ \langle (?7, v), \mathbf{6}\rangle[!8, \overset{0}{\hookleftarrow}] \end{cases}$$

Notice that the first move is $\overline{2}$: in call-by-value, it is Player who starts! The final result is the value 8, which is sent to the continuation α. The reader may check that the machine given above indeed yields this result when it is run on the source code. (In later work, we intend to prove formally that this source machine is correctly simulated by the GAM.)

We give two further examples, in less detail. The compilation of

$$\mathit{let}\ x = y3\ \mathit{in}\ \mathit{let}\ v = x4\ \mathit{in}\ [\gamma]v \quad [y \leftarrow (\lambda(z, \alpha).[\alpha](\lambda(u, \beta).[\beta](u + z)]$$

(assuming a binary adddition operation) is:

$$[(?3, y), \overleftarrow{\hookrightarrow}] \begin{cases} !\bullet\ [?4, \overset{0}{\hookleftarrow}] \begin{cases} \vdots \\ !m[(!m, \gamma), \overleftarrow{\hookrightarrow}] \\ \vdots \end{cases} \end{cases}$$

$$\left\{\begin{array}{l}\vdots\\ (?n,y)[!\bullet,\overset{0}{\hookleftarrow}]\left\{\begin{array}{l}\vdots\\ ?p\,[!(n+p),\overset{0}{\hookleftarrow}]\\ \vdots\end{array}\right.\\ \vdots\end{array}\right.$$

with the following execution:

$$[(?3,y),\overset{-}{\hookleftarrow}]\left\{\ \langle!\bullet,\mathbf{3}\rangle[?4,\overset{0}{\hookleftarrow}]\left\{\ \langle!7,\mathbf{5}\rangle[(!7,\gamma),\overset{-}{\hookleftarrow}]\right.\right.$$
$$\left\{\ \langle(?3,y),\mathbf{2}\rangle[!\bullet,\overset{0}{\hookleftarrow}]\left\{\ \langle?4,\mathbf{4}\rangle\ [!7,\overset{0}{\hookleftarrow}]\right.\right.$$

The compilation of

$$let\ x = y(\lambda(z,\beta).[\beta]z)\ in\ [\alpha]x \quad [y \leftarrow (\lambda(f,\gamma).let\ u = f3\ in\ [\gamma]u)]$$

is:

$$[(?\bullet,y),\overset{-}{\hookleftarrow}]\left\{\begin{array}{l}\vdots\\ ?n[!n,\overset{0}{\hookleftarrow}]\\ \vdots\\ !p[(!p,\alpha),\overset{-}{\hookleftarrow}]\\ \vdots\end{array}\right.$$

$$\left\{\begin{array}{l}(?\bullet,y)[?3,\overset{0}{\hookleftarrow}]\left\{\begin{array}{l}\vdots\\ !n[!n,\overset{1}{\hookleftarrow}]\\ \vdots\end{array}\right.\end{array}\right.$$

with the following execution:

$$[(?\bullet,y),\overset{-}{\hookleftarrow}]\left\{\begin{array}{l}\langle?3,\mathbf{3}\rangle[!3,\overset{0}{\hookleftarrow}]\\ \langle!3,\mathbf{5}\rangle[(!3,\alpha,\overset{-}{\hookleftarrow}]\end{array}\right.$$
$$\left\{\ \langle(?\bullet,y),\mathbf{2}\rangle[?3,\overset{0}{\hookleftarrow}]\left\{\ \langle!3,\mathbf{4}\rangle[!3,\overset{1}{\hookleftarrow}]\right.\right.$$

We end the section by a short incursion into types. In figure 2, we present typing rules for classical PCF call-by-value trees, in a simple type system built over the base type ι of naturals, and over the function type:

$$\sigma :: \iota \mid (\sigma \to \sigma)$$

There is also a special type $\perp$, which is not used in compound types and serves only to type commands. A typing context consists of a pair $\Gamma;\Delta$ of two multisets containing

$$\frac{}{\Gamma;\Delta \vdash \underline{n} : \iota} \qquad \frac{}{\Gamma, x:\iota;\Delta \vdash x:\iota} \qquad \frac{\Gamma, z:\sigma;\Delta,\beta:\tau \vdash c:\bot}{\Gamma:\Delta \vdash \lambda(z,\beta).c:\sigma\to\tau}$$

$$\frac{\Gamma;\Delta,\alpha:\sigma \vdash V:\sigma}{\Gamma;\Delta,\alpha:\sigma \vdash [\alpha]V:\bot} \qquad \frac{\forall i\ \ \Gamma, x:\iota;\Delta \vdash c_i:\bot}{\Gamma, x:\iota;\Delta \vdash \mathit{case}\ x\ [\vec{\underline{n}} \to \vec{c}] : \bot}$$

$$\frac{\Gamma, y:\sigma\to\tau, x:\tau;\Delta \vdash c:\bot \qquad \Gamma, y:\sigma\to\tau;\Delta \vdash V:\sigma}{\Gamma, y:\sigma\to\tau;\Delta \vdash \mathit{let}\ x = yV\ \mathit{in}\ c:\bot}$$

Figure 2: Typed classical PCF call-by-value trees

declarations $x:\sigma$ (ordinary variables) and $\alpha:\sigma$ (continuation variables), respectively. We can type the successive states as closed judgements $\vdash \langle c \,|\, \rho\rangle : \bot$, with $\Gamma;\Delta \vdash c:\bot$ for some Γ and Δ such that for all $x:\sigma$ (resp. $\alpha:\tau$) appearing in Γ (resp. Δ), we have $\vdash \rho(x):\sigma$ (resp. $\vdash \rho(\alpha):\tau$). The construct $\tilde{\mu}$ is typed as follows:

$$\frac{\Gamma, x:\tau;\Delta \vdash c:\bot}{\Gamma;\Delta \vdash \tilde{\mu}x.c:\tau}$$

Moreover, well-typed states are guaranteed to evolve without encountering the "bad" termination case of section 2.4. We omit the details, since we do not want to put emphasis on types here.

3.5 Ludics

Ludics [20] is a recent theory whose aim is to reconstruct logic on interactive principles. While denotational semantics has been category-theory-oriented and type-oriented in the last 25 years, ludics deliberately adopts the view that computation is untyped, and that types may be built out of untyped objects as collections of such objects "behaving the same way". The objects of ludics, called *designs*, are derived from the skeletons of proofs in linear logic obtained after removing the type information (just retaining the subformula relations).

In [15], we have shown that the designs can be presented by a syntax which is accessible

without prior knowledge of linear logic and of the genesis of ludics:

$$M ::= \{J = \lambda\{x_j \mid j \in J\}.P_J \mid J \in \mathcal{P}_f(\omega)\}$$
$$P ::= (x \cdot I)\{M_i \mid i \in I\} \mid \Omega \mid \maltese$$

Girard's original ludics enforces *affinity* assumptions (see [15] for details). Without these assumptions, this syntax reflects Maurel's exponential ludics [27].
The execution is driven by the following abstract machine.

$$\langle (x \cdot I)\{M_i \mid i \in I\} \mid \rho \rangle \quad \longrightarrow \quad \langle P_I \mid \rho_I\{x_i \leftarrow M_i[\rho] \mid i \in I\} \rangle$$

where $\rho(x) = \{\ldots, I = (\lambda\{x_i \mid i \in I\}.P_I)[\rho_I], \ldots\}$. The reduction combines substitution of actual parameters (the M_i's) for formal ones (the x_i's), typical of β-reduction, with a prior selection of a "field" I. In this framework, abstractions are replaced by a collection of abstractions, one for each possible indexing set I. Whence our choice of a notation reminiscent of that used for records in some object-oriented programming languages.

We say that the evaluation *converges* when it reaches a stage $\langle \maltese | \rho \rangle$, and that it diverges if either it reaches a stage $\langle \Omega \mid \rho \rangle$, or the computation never ends (this cannot happen if both the design and the counter-design have finite depth, cf. section 2.4).

It is easy to make this syntax fit into the format of abstract Böhm trees. One first groups the two layers of indexing (by $i \in I$ and by $J \in \mathcal{P}_f(\omega)$). This gives (by a textual transformation):

$$(x \cdot I)\{((i, J), (\lambda\{x_j \mid j \in J\}.P_{i,J})) \mid i \in I, J \in \mathcal{P}_f(\omega)\}$$

One then replaces $\{x_j \mid j \in J\}$ by $\mathtt{x}$ and every (bound) $(x_j.K)$ by $((j, K), \mathtt{x})$. The two constants can be treated as special (terminal) free player's moves.

4 The View Abstract Machine

In this section, we give a "lighter" version of the GAM, where the state is a sequence of *moves*, called a *play* rather than a sequence of positions. The price to pay is that relevant information must be reconstructed at each step on the fly. This version of the GAM is called the View Abstract Machine (VAM).

Before we define it formally, let us examine the play underlying the execution of the example in section 3.1. The following moves are played successively at steps (1), $(\overline{2})$ and (2), $(\overline{3})$ and (3), etc...: $\bullet$, $[u, \overset{-}{\hookleftarrow}]$ and u, $[1, \overset{0}{\hookleftarrow}]$ and 1, etc... . But the resulting sequence is not a position (but rather an interleaving of positions) of ϕ or ψ. At each step $(2n-1)$ (resp. $(2n)$), we have to reconstruct the appropriate position of ϕ (resp. ψ) – the *view* at this step –, in order to apply ϕ (resp. ψ), as before. For example, the view at step (2) consists of u only.

The syntax of VAM states is as follows:

$$\text{VAM states:} \qquad \Gamma ::= \{1 \leftarrow \bullet\} \mid \Gamma\{\overline{n} \leftarrow [a, \overset{\kappa}{\hookleftarrow}]\} \mid \Gamma\{2n \leftarrow a\} \mid \Gamma\{n \leftarrow \langle a, \overline{m}\rangle\}$$

We next define *jumps* and *views*, which are our tools for reconstructing information:

$$\frac{\Gamma{\bullet}n = \langle a, \overline{m}\rangle}{jump_\Gamma(n) = m - 1}$$

$$\frac{}{view_\Gamma(1) = \bullet} \qquad \frac{\Gamma{\bullet}(2n) = a}{view_\Gamma(2n) = a} \qquad \frac{\Gamma{\bullet}n = \langle a, \overline{m}\rangle}{view_\Gamma(n) = view_\Gamma(m-1)\ (\Gamma{\bullet}\overline{m})\ a}$$

The rules of the VAM are given in figure 3.

We collect here a few observations.

- Most works on game semantics do not define strategies as sets of views like we do. For example, Hyland and Ong's interpretation of $(u(\lambda x.u(\lambda y.x))$ (cf. section 3.1) contains the position

  $$\bullet\ [u, \overset{-}{\hookleftarrow}]\ \underline{1\ [u, \overset{-}{\hookleftarrow}]\ 1\ [1, \overset{1}{\hookleftarrow}]}\ 1[u, \overset{-}{\hookleftarrow}]$$

  whose view $\bullet\ [u, \overset{-}{\hookleftarrow}]\ 1\ [u, \overset{-}{\hookleftarrow}]$ is obtained by removing the underlined portion. There are some advantages to this "plethorous" definition of strategies:

  - In such positions, there is no dissymetry between Player and Opponent, since one has broken the requirement that Opponent always plays just below Player (this is why there are no pointers originating from opponent's moves in abstract Böhm trees).
  - The composition of strategies can be defined algebraically in one line as "composition + hiding" [24, 2].

  The drawback is that it gives an infinite representation of finite objects, since, say, also

  $$\bullet\ [u, \overset{-}{\hookleftarrow}]\ 1\ [u, \overset{-}{\hookleftarrow}]\ 1\ [1, \overset{1}{\hookleftarrow}]\ 1[u, \overset{-}{\hookleftarrow}]\ 1[u, \overset{-}{\hookleftarrow}]$$

  etc... belong to the strategy. The existence of these two kinds of representation of a strategy – the sober one, isomorphic to the underlying term, and the plethoric one – was first recognized by Felscher, who calls them E-dialogue and D-dialogue, respectively [17]. (Felscher's work continues a school of thought initiated by Lorenz and Lorenzen, where proofs are viewed as a dialogue between a defendant and an opponent, a bit like in a PhD defence. Unfortunately, these works emphasized provability rather than proofs.)

$$(1)\ \frac{}{\mapsto \{1 \leftarrow \bullet\}}$$

$$(\overline{2n})\ \frac{\phi(view_\Gamma(2n-1)) = [a, \overset{\kappa}{\hookleftarrow}]}{\Gamma \mapsto \Gamma\{\overline{2n} \leftarrow [a, \overset{\kappa}{\hookleftarrow}]\}}$$

$$(2n)_b\ \frac{hd(\Gamma) = \{\overline{2n} \leftarrow [a, \overset{i}{\hookleftarrow}]\} \quad jump^i_\Gamma(2n-1) = 2m-1}{\Gamma \mapsto \Gamma\{2n \leftarrow \langle a, \overline{2m-1}\rangle\}}$$

$$(2n)_f\ \frac{hd(\Gamma) = \{\overline{2n} \leftarrow p[a, \overset{-}{\hookleftarrow}]\}}{\Gamma \mapsto \Gamma\{2n \leftarrow a\}}$$

$$(\overline{2n+1})\ \frac{\psi(view_\Gamma(2n)) = [a, \overset{\kappa}{\hookleftarrow}]}{\Gamma \mapsto \Gamma\{\overline{2n+1} \leftarrow [a, \overset{\kappa}{\hookleftarrow}]\}}$$

$$(2n+1)\ \frac{hd(\Gamma) = \{\overline{2n+1} \leftarrow [a, \overset{i}{\hookleftarrow}]\} \quad jump^i_\Gamma(2n) = 2m}{\Gamma \mapsto \Gamma\{2n+1 \leftarrow \langle a, \overline{2m}\rangle\}}$$

Figure 3: The View Abstract Machine (VAM)

- The translation from a VAM state to a GAM state (see section A.1) can be understood as a desenquentialization process: the play Γ gets translated to a collection of views (with repetitions) – a thick tree of views. The equivalence between the GAM and the VAM (again, see section A.1) guarantees that the information collected in the multiplexed opponent's moves of the translation contains all the original sequential information, implicitly. However, by moving to the tree isomorphism class of $\lceil \Gamma \rceil^{VG}$, this information is really lost, and then it makes sense to talk about desequentialization. This forgetful desequentialization is close in spirit to the work of Boudes [6] on the analysis of the relation between game semantics and coherence semantics (a more traditional kind of model).

- The concept of multiplexing is not only relevant for dynamic issues, but also for static ones. In a typed setting, each position in a strategy encodes a multiplexed traversal of its type. The two kinds of multiplexing obey a dual discipline: the dynamic one multiplexes the opponent's moves, whereas the static one multiplexes the player's moves. We illustrate this with a simple example: the most general type of $M = (\lambda u.u(\lambda x.u(\lambda y.x)))$, in Hindley's sense, is

$$((X \to Y) \to Y) \to Y$$

which we represent as

$$Y \left\{ \; Y \left\{ \; Y \left\{ \; X \right.\right.\right.$$

We show how M, which consists of one path only, encodes a multiplexing of its type:

$$\boxed{\begin{matrix}(\lambda u)\\ Y\end{matrix}} \left\{ \begin{array}{l} \boxed{u_1} \\ \langle Y, \mathbf{2} \rangle \end{array} \left\{ \begin{array}{l} \boxed{\begin{matrix}(\lambda x)\\ Y\end{matrix}} \\ \end{array} \left\{ \begin{array}{l} \boxed{x} \\ \langle X, \mathbf{6} \rangle \end{array} \right. \right. \right. \\ \left. \begin{array}{l} \boxed{u_2} \\ \langle Y, \mathbf{4} \rangle \end{array} \left\{ \begin{array}{l} \boxed{(\lambda y} \\ \end{array} \right. \right.$$

We shall make a timid incursion in linear logic, by suggesting that this multiplexing of types can be understood as rewriting of trees by means of the rule $!A \to (!A \otimes A)$. For example, the type $((X \to Y_3) \to Y_2) \to Y_1$, expressed in linear logic ($\to$ being now the linear arrow), is:

$$!(!(!X \to Y_3) \to Y_2) \to Y_1$$

and can be rewritten to:

$$(!(!(!X \to Y_3) \to Y_2) \otimes !(!X \to Y_3) \to Y_2) \to Y_1$$

which corresponds to the multiplexing above.

5 A concrete version of the GAM

In this section, we complete the operational picture by introducing two other formulations of the GAM: the Strategic Abstract Machine (SAM), and its concrete version, the Environment Abstract Machine (EAM), formulated in terms of the concrete syntax of section 2.5.

5.1 Strategic abstract machine

Instead of marking the multiplexed nodes with numbers n, which in their form $\overline{n}$ point us to player's positions in the other strategy, we could directly mark them with these positions. This yields the following version of the GAM:

$$\frac{}{\mapsto \{1 \leftarrow \bullet\}}$$

$$\frac{hd(\Gamma) = \{2n-1 \leftarrow p\} \quad \phi(erase(p)) = [a, \overset{i}{\hookleftarrow}]}{\Gamma \mapsto \Gamma\{\overline{2n} \leftarrow p[a, \overset{i}{\hookleftarrow}]\}}$$

$$\frac{hd(\Gamma) = \{\overline{2n} \leftarrow p[a, \overset{i}{\hookleftarrow}]\} \quad \pi'(pop^i(p)) = q}{\Gamma \mapsto \Gamma\{2n \leftarrow q\langle a, p[a, \overset{i}{\hookleftarrow}]\rangle\}}$$

$$\frac{hd(\Gamma) = \{\overline{2n} \leftarrow p[a, \overset{-}{\hookleftarrow}]\}}{\Gamma \mapsto \Gamma\{2n \leftarrow \langle a, p[a, \overset{-}{\hookleftarrow}]\rangle\}}$$

$$\frac{hd(\Gamma) = \{2n \leftarrow p\} \quad \psi(erase(p)) = [a, \overset{i}{\hookleftarrow}]}{\Gamma \mapsto \Gamma\{\overline{2n+1} \leftarrow p[a, \overset{i}{\hookleftarrow}]\}}$$

$$\frac{hd(\Gamma) = \{\overline{2n+1} \leftarrow p[a, \overset{i}{\hookleftarrow}]\} \quad \pi'(pop^i(p)) = q}{\Gamma \mapsto \Gamma\{2n+1 \leftarrow q\langle a, p[a, \overset{i}{\hookleftarrow}]\rangle\}}$$

The distinction between n and $\overline{n}$ becomes then useless, but we keep it in order to stay with the same number of rules.

But now we observe that the machine, expressed in this format, is "history-free". We can thus remove Γ altogether, and describe the machine as a rewriting system on (nested) positions:

$$n \qquad\qquad \overline{n}$$

$$\text{SAM states} \qquad \mathtt{q} ::= \bullet \mid \langle a, \mathtt{r}\rangle \mid \mathtt{r}'\langle a, \mathtt{r}\rangle \qquad \mathtt{r} ::= \mathtt{q}[a, \overset{\kappa}{\hookleftarrow}]$$

We arrive at the machine described in figure 4, called here Strategic Abstract Machine (SAM).

5.2 The Environment Abstract Machine

The SAM can be formulated in terms of the concrete syntax of section 2.5, and in this form, it appears as the natural generalization of (a stack-free version of) Krivine abstract machine. We call it the Environment Abstract Machine. The states of the EAM are pairs (code, environment), recursively defined as follows:

Closures:	$M[\rho]$
Environments:	maps ρ from variables to closures
EAM states:	$\langle P \mid \rho\rangle$

The rules of the EAM are given in figure 5.

6 Strong reduction

In this section, we show how to extend the GAM to a *strong* machine, which computes a complete strategy. The strong machine which we present can build on demand, in a *stream-like* fashion, any position of the composition of ϕ and ψ. The initial step of the machine builds the initial position $\bullet$ of the composition. Either of the (two first) terminating states discussed in section 2.4, when first encountered, produces a player's move in the composition that answers the initial query $\bullet$: in the first case this player's move is written $[a, 1]$, in the second case, it is written as usual $[a, \overset{-}{\hookleftarrow}]$. Then the pilot of the strong machine may decide to raise a new query, expressed as a position $\bullet[a, 1]\langle b, \mathbf{2n}\rangle$ (in the first case) or $\bullet[a, \overset{-}{\hookleftarrow}]\langle b, \mathbf{2n+1}\rangle$ (in the second case). Note that the alternation is broken: this new step, which is of the form $(2n)$ in the first case, and $(2n+1)$ in the second case, is performed in the same strategy as the previous step.

The machine states consist now of triplets of the form:

- $(\Gamma\ ?\ q)$, where Γ is as before, and where ? witnesses that we are working on answering a query q,

$$(1)\ \frac{}{\xrightarrow{1} \bullet}$$

$$(\overline{2n})\ \frac{\phi(erase(\mathtt{q})) = [a, \overset{\kappa}{\hookleftarrow}]}{\mathtt{q} \xrightarrow{\overline{2n}} \mathtt{q}[a, \overset{\kappa}{\hookleftarrow}]}$$

$$(2n)_b\ \frac{\pi'(pop^i(\mathtt{q})) = \mathtt{r}'}{\mathtt{q}[a, \overset{i}{\hookleftarrow}] \xrightarrow{2n} \mathtt{r}'\langle a, \mathtt{q}[a, \overset{i}{\hookleftarrow}]\rangle}$$

$$(2n)_f\ \frac{}{\mathtt{q}[a, \overset{-}{\hookleftarrow}] \xrightarrow{2n} \langle a, \mathtt{q}[a, \overset{-}{\hookleftarrow}]\rangle}$$

$$(\overline{2n+1})\ \frac{\psi(erase(\mathtt{q})) = [a, \overset{\kappa}{\hookleftarrow}]}{\mathtt{q} \xrightarrow{\overline{2n+1}} \mathtt{q}[a, \overset{\kappa}{\hookleftarrow}]}$$

$$(2n+1)\ \frac{\pi'(pop^i(\mathtt{q})) = \mathtt{r}'}{\mathtt{q}[a, \overset{i}{\hookleftarrow}] \xrightarrow{2n+1} \mathtt{r}'\langle a, \mathtt{q}[a, \overset{i}{\hookleftarrow}]\rangle}$$

Figure 4: The Strategic Abstract Machine (SAM)

$$\frac{\rho([a,\star]) = (\lambda \mathtt{y}.Q)[\rho']}{\langle [a,\star]\,\{(b,M_b) \mid b \in B\} \mid \rho\rangle \quad \longrightarrow \quad \langle Q \mid \rho'[[b,\mathtt{y}] \leftarrow M_b[\rho] \mid b \in B]\rangle}$$

$$\frac{\rho([a,\star]) = (Q)[\rho']}{\langle [a,\star]\,\{\,\} \mid \rho\rangle \quad \longrightarrow \quad \langle Q \mid \rho'\rangle}$$

Figure 5: The Environment Abstract Machine (EAM)

- $(\Gamma\ !\ r)$, where Γ is as before, and where ! witnesses that the query at the previous stage is just answered.

The complete set of rules of the strong GAM is given in figure 6. The rules $(1)^s$, $(\overline{n})^s_\phi$, $(n)^s_{b,\phi}$, $(n)^s_{f,\phi}$, $(\overline{n})^s_\psi$, and $(n)^s_\psi$, are just rephrasings of the rules of the (weak) GAM.

We have been a bit vague about the syntax of the positions appearing as the third component of the states of the machine. The point is that the dynamics of the computation does not allow us to immediately extract pointers encoded as offsets like those we had for the positions of ϕ, ψ we started with. Instead, we have pointers in the form of addresses: when a position $q[a,m]$ is reached (at a step $(!)^s_\phi$ or $(!)^s_{b,\psi}$), we mean that the move $[a,m]$ points to the unique opponent's move of q of the form $\langle a,m\rangle$.

We have already remarked that when going from "weak" to "strong", the nice alternation between ϕ and ψ is broken (rules $(?)^s_\phi$ and $(?)^s_\psi$). This could be one abstract definition of weak evaluation: evaluation is weak as long as it strictly alternates between the strategy and the counter-strategy.

The composition of ϕ and ψ is defined as the collection of the positions obtained through the various non-deterministic executions of the GAM (recompiled in order to obtain standard abstract Böhm tree positions). We obtain in this way a category (actually, a multicategory, where morphisms have a sequence or set of objects as domain and an object as codomain) of strategies (see [9]). The identities, which in the multicategorical setting are in fact projections (from the domain to one of the objects of the domain), are so-called copy-cat or fax strategies (see section 8). The associativity of the composition amounts to define a three-way machine that lets all the strategies involved interact.

We give two examples which illustrate the use of the new rules:

Example 1

Function: the strategy for $(\lambda u.x(\lambda v.u(v)))$ is:

$$(1)^s \ \frac{}{\mapsto (\{1 \overset{\phi}{\leftarrow} \langle \bullet, \mathbf{1} \rangle\} \ ? \ \langle \bullet, 1 \rangle)} \qquad (n)^s_{f,\phi} \ \frac{hd(\Gamma) = \{\overline{n} \overset{\phi}{\leftarrow} \mathbf{q}[a, \overset{-}{\hookleftarrow}]\}}{(\Gamma \ ? \ r) \mapsto (\Gamma\{n \overset{\psi}{\leftarrow} \langle a, \mathbf{n} \rangle\} \ ? \ r)}$$

$$(\overline{n})^s_{\phi} \ \frac{hd(\Gamma) = \{n-1 \overset{\phi}{\leftarrow} \mathbf{q}\} \quad \phi(erase(\mathbf{q})) = [a, \overset{i}{\hookleftarrow}]}{(\Gamma \ ? \ r) \mapsto (\Gamma\{\overline{n} \overset{\phi}{\leftarrow} \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \ ? \ r)}$$

$$(n)^s_{b,\phi} \ \frac{hd(\Gamma) = \{\overline{n} \overset{\phi}{\leftarrow} \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \quad \pi'(pop^i(\mathbf{q})) = m \quad \Gamma \bullet \overline{m} = \mathbf{r}' \in \psi}{(\Gamma \ ? \ r) \mapsto (\Gamma\{n \overset{\psi}{\leftarrow} \mathbf{r}'\langle a, \mathbf{n} \rangle\} \ ? \ r)}$$

$$(\overline{n})^s_{\psi} \ \frac{hd(\Gamma) = \{n-1 \overset{\psi}{\leftarrow} \mathbf{q}\} \quad \psi(erase(\mathbf{q})) = [a, \overset{i}{\hookleftarrow}]}{(\Gamma \ ? \ r) \mapsto (\Gamma\{\overline{n} \overset{\psi}{\leftarrow} \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \ ? \ r)}$$

$$(n)^s_{\psi} \ \frac{hd(\Gamma) = \{\overline{n} \overset{\psi}{\leftarrow} \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \quad \pi'(pop^i(\mathbf{q})) = m \quad \Gamma \bullet \overline{m} = \mathbf{r}' \in \phi}{(\Gamma \ ? \ r) \mapsto (\Gamma\{n \overset{\phi}{\leftarrow} \mathbf{r}'\langle a, \mathbf{n} \rangle\} \ ? \ r)}$$

$$(!)^s_{\phi} \ \frac{hd(\Gamma) = \{\overline{n} \overset{\phi}{\leftarrow} \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \quad \pi'(pop^i(\mathbf{q})) = m \quad (m = 1 \text{ or } \Gamma \bullet \overline{m} \in \phi)}{(\Gamma \ ? \ r) \mapsto (\Gamma \ ! \ r[a, m])}$$

$$(!)^s_{b,\psi} \ \frac{hd(\Gamma) = \{\overline{n} \overset{\psi}{\leftarrow} \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \ \ \pi'(pop^i(\mathbf{q})) = m \ \ \Gamma \bullet \overline{m} \in \psi}{(\Gamma \ ? \ r) \mapsto (\Gamma \ ! \ r[a, m])} \qquad (!)^s_{f,\psi} \ \frac{hd(\Gamma) = \{\overline{n} \overset{\psi}{\leftarrow} \mathbf{q}[a, \overset{-}{\hookleftarrow}]\}}{(\Gamma \ ? \ r) \mapsto (\Gamma \ ! \ r[a, \_])}$$

$$(?)^s_{\chi} \ \frac{hd(\Gamma) = \{\overline{n} \overset{\phi}{\leftarrow} \mathbf{r}'\} \ \ erase(\mathbf{r}')a \in dom(\chi) \ \ (\chi = \phi \text{ or } \psi)}{(\Gamma \ ! \ r) \mapsto (\Gamma\{n \overset{\chi}{\leftarrow} p\langle a, \mathbf{n} \rangle\} \ ? \ \mathbf{r}'\langle a, n \rangle)}$$

Figure 6: The Strong GAM

$$\boxed{(\lambda u}_{\bullet}\,[x,\overset{-}{\hookleftarrow}]\left\{\begin{array}{l}\boxed{(\lambda v}_{1}\,[1,\overset{1}{\hookleftarrow}]\left\{\begin{array}{l}\boxed{(}_{1}\,[1,\overset{1}{\hookleftarrow}]\end{array}\right.\\ \boxed{u}\end{array}\right.$$

Argument: the strategy for $[x \leftarrow (\lambda r.r(z))]$ is:

$$\boxed{(\lambda r}_{x}\,[1,\overset{0}{\hookleftarrow}]\left\{\begin{array}{l}\boxed{(}_{1}\,[z,\overset{-}{\hookleftarrow}]\\ \boxed{r}\end{array}\right.$$

Function multiplexed tree:

$$\boxed{(\lambda u\ ?}_{\langle\bullet,\mathbf{1}\rangle}\,[x,\overset{-}{\hookleftarrow}]\left\{\begin{array}{l}\langle 1,\mathbf{3}\rangle\,[1,\overset{1}{\hookleftarrow}]\left\{\begin{array}{l}\boxed{(\ ?_{\phi}}_{\langle 1,\mathbf{4}\rangle}\,[1,\overset{1}{\hookleftarrow}]\\ \boxed{u\ !_{\phi}}\end{array}\right.\\ \langle x,\mathbf{2}\rangle[1,\overset{0}{\hookleftarrow}]\end{array}\right.$$

Argument multiplexed tree:

$$\langle x,\mathbf{2}\rangle[1,\overset{0}{\hookleftarrow}]\left\{\begin{array}{l}\langle 1,\mathbf{5}\rangle\,[z,\overset{-}{\hookleftarrow}]\\ \boxed{!_{f,\psi}}\end{array}\right.$$

Reading back of the composition: $(\lambda u.u(z))$.

Example 2

Function: the strategy for $(x(\lambda u.u))$ is:

$$\boxed{(}_{\bullet}\,[x,\overset{-}{\hookleftarrow}]\left\{\begin{array}{l}\boxed{(\lambda u}_{1}\,[1,\overset{0}{\hookleftarrow}]\\ \boxed{u}\end{array}\right.$$

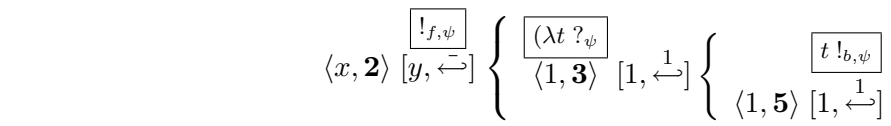
Argument: the strategy for $[x \leftarrow (\lambda z.y(\lambda t.z(t)))]$ is:

$$\boxed{(\lambda z}_{x}\,[y,\overset{-}{\hookleftarrow}]\left\{\begin{array}{l}\boxed{(\lambda t}_{1}\,[1,\overset{1}{\hookleftarrow}]\left\{\begin{array}{l}\boxed{(}_{1}\,[1,\overset{1}{\hookleftarrow}]\\ \boxed{t}\end{array}\right.\\ \boxed{z}\end{array}\right.$$

Function multiplexed tree:

$$\boxed{(\ ?}_{\langle\bullet,\mathbf{1}\rangle}\,[x,\overset{-}{\hookleftarrow}]\left\{\begin{array}{l}\langle 1,\mathbf{4}\rangle[1,\overset{0}{\hookleftarrow}]\\ \boxed{(\ ?}\end{array}\right.$$

Argument multiplexed tree:



- Reading back of the composition: $(y(\lambda t.t))$.

7 Evaluating non-normal forms

The syntactic formulation of abstract Böhm trees (cf. section 2.5) suggests us to extend the syntax to "non-normal forms", as follows:

$$P ::= M\,\{(b, M_b) \mid b \in B\} \quad | \quad [a, \star]\,\{(b, M_b) \mid b \in B\} \quad (\star = \mathtt{x} \text{ or } \star = \_)$$
$$M ::= (\lambda \mathtt{x}.P)$$

and to equip the language with the following notion of reduction (a generalized version of the β-rule of the λ-calculus):

$$(\lambda \mathtt{x}.P)\,\{(b, M_b) \mid b \in B\} \quad \longrightarrow \quad P\,[[b, \overset{\mathtt{x}}{\hookleftarrow}] \leftarrow M_b \mid b \in B]$$

The machinery of the GAM can be extended to such terms. Expressed in terms of λ-calculus, the basic idea, which is somewhat folklore, is to replace any redex $(\lambda x.M)N$ by an indirection of the form $(\star(N))[\star \leftarrow (\lambda x.M)]$, where $\star$ is a special variable name. We make the following ajustments:

1. We compile the terms in two steps. First we obtain a term of the core syntax of normal forms, where we have reserved a second special move $\star$:

   $$\lceil M\,\{\ldots, (b, M_b), \ldots\}\rceil^\star = ([\star, \overset{-}{\hookleftarrow}]\,\{(\star, M), \ldots, (b, M_b), \ldots\}$$

   And then we compile the core term.

2. We add the following rule to the GAM:

   $$(n)_\star \quad \frac{hd(\Gamma) = \{\overline{n} \leftarrow \mathbf{q}[\star, \overset{-}{\hookleftarrow}]\}}{\Gamma \mapsto \Gamma\{n \leftarrow \mathbf{q}[\star, \overset{-}{\hookleftarrow}]\langle \star, \mathbf{n}\rangle\}}$$

   In addition, the rules $(\overline{2n})$ and $(\overline{2n+1})$ collapse in a single rule $(\overline{n})$, since there is now a single strategy χ interacting internally with itself. Similarly, the rules $(2n)_b$ and $(2n+1)$ become just one rule (n), and the rule $(2n)_f$ disappears. The changes are summarized in figure 7, and the resulting machine is called the GAM*.

$$(1)\ \frac{}{\mapsto \{1 \leftarrow \langle \bullet, \mathbf{1} \rangle\}}$$

$$(\overline{n})\ \frac{hd(\Gamma) = \{n-1 \leftarrow \mathbf{q}\} \quad \chi(erase(\mathbf{q})) = [a, \overset{i}{\hookleftarrow}]}{\Gamma \mapsto \Gamma\{\overline{n} \leftarrow \mathbf{q}[a, \overset{i}{\hookleftarrow}]\}}$$

$$(n)\ \frac{hd(\Gamma) = \{\overline{n} \leftarrow \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \quad \pi'(pop^i(\mathbf{q})) = m \quad \Gamma \bullet \overline{m} = \mathbf{r}'}{\Gamma \mapsto \Gamma\{n \leftarrow \mathbf{r}'\langle a, \mathbf{n} \rangle\}}$$

$$(n)_\star\ \frac{hd(\Gamma) = \{\overline{n} \leftarrow \mathbf{q}[\star, \overset{-}{\hookleftarrow}]\}}{\Gamma \mapsto \Gamma\{n \leftarrow \mathbf{q}[\star, \overset{-}{\hookleftarrow}]\langle \star, \mathbf{n} \rangle\}}$$

Figure 7: The GAM$_\star$

With this machine we can accommodate λ-terms. We illustrate the GAM* with a simple example, which is a variation on the theme of static binding. Let

$$M = ((\lambda x.(\lambda f.(\lambda x.f(t))(\lambda z.u))(\lambda y.x(y)))(\lambda z.z))$$

which is easier to read with syntactic sugar:

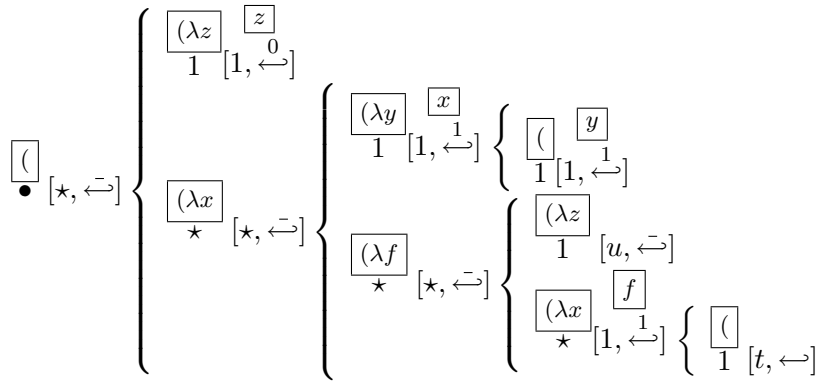
$$M = (\text{let } x = (\lambda z.z) \text{ in } (\text{let } f = (\lambda y.x(y)) \text{ in } (\text{let } x = (\lambda z.u) \text{ in } (f(t)))))$$

It illustrates static binding: the final result is t, not u, i.e., the relevant value for x is $(\lambda z.z)$, which was the value of x at the time of declaration of f, and not $(\lambda z.u)$ which is the execution time value of x when f is called.

We can define the compilation of λ-terms by factoring through the term notation of section 2.5. One adds the following obvious clause in the translation of section 3.1:

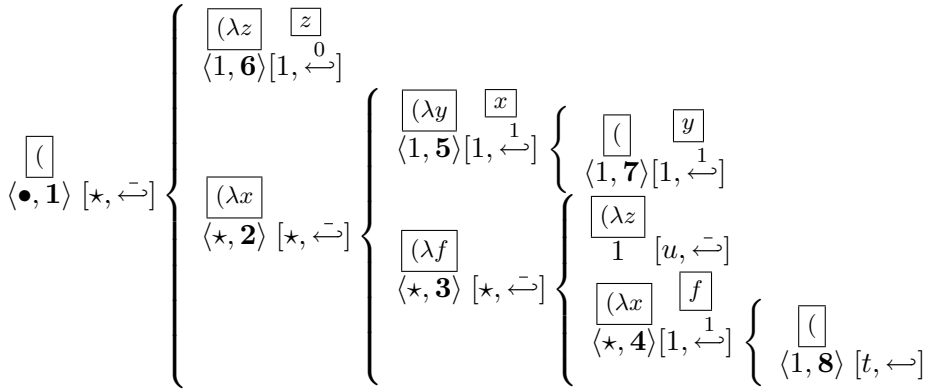
$$\lceil MB_1 \dots B_n \rceil = M\{(1, \lceil B_1 \rceil), \dots, (n, \lceil B_n \rceil)\}$$

Here is the full compilation of M:



We now execute this term, using the acquired skills.

Multiplexed (compilation of) M:



8 Evaluating and separating untyped λ-terms

Consider the following very simple example, expressed in concrete syntax:

$$\langle [x, \_]\{\ \} \mid [x \leftarrow (\lambda \mathtt{y}.[a, \mathtt{y}]] \rangle$$

whose compilation in terms of trees of positions is:

$\bullet[x, \overset{-}{\hookleftarrow}]$

$x[a, \overset{0}{\hookleftarrow}]$

The computation is blocked at step $(\overline{3})$ (third case of termination, cf. section 2.4):

$\langle \bullet, \mathbf{1} \rangle [x, \overset{-}{\hookleftarrow}]$

$\langle x, \mathbf{2} \rangle [a, \overset{0}{\hookleftarrow}]$

The problem is that x should have an argument. Evaluation can proceed if we perform an η-expansion of x. Recall that in the λ-calculus, the η-rule asserts

$$\lambda x.Mx = x$$

for all M and x such that x does not occur freely in M. In the present setting, we need to replace $([x, \_])$ by $(\lambda \mathtt{z}.[x, \_]\{(a, ([1, \mathtt{z}]))\}$. This looks simple enough, but, in fact, we cannot really do this for arbitrary abstract Böhm trees. Consider $\lambda x_1 \ldots x_p.yM_1 \ldots M_q$ which η-expands to

$$\lambda x_1 \ldots x_p z.yM_1 \ldots M_q z \ .$$

In compiled form, we have that $\lambda \mathtt{x}.[y, \_]\{(1, B_1), \ldots, (q, B_q)\}$ expands to

$$\lambda \mathtt{x}.[y, \_]\{(1, B_1), \ldots, (q, B_q), (q+1, ([p+1, \mathtt{x}]))\}$$

The point is that we need to name (or give an address to) the two moves which have been added: the new opponent's move has address $q+1$ since z is to be the $(q+1)$-th argument of y, and the new player's move has address $p+1$ since it is to be the $(p+1)$–th abstracted variable. This supposes a notion of sequencing among arguments and abstracted variables, which is present in the examples treated in sections 3.1, 3.2, 3.3, and 3.4, but was "lost in translation". In particular, in order to evaluate untyped λ-terms, we need to record more information during the compilation (section 8.1).

The η-expansion plays an essential role in the proof of an important theorem of the λ-calculus due to Corrado Böhm. We briefly recall what this theorem is about, and illustrate it through an example (section 8.2).

There is however one instance where η-expansion makes sense for arbitrary Böhm trees: when the sets of arguments and abstracted variables are empty. This special case is very important, since it provides us with the missing bit of section 6: the projections, or copy-cat (or *fax*, in the nice terminology of [20]) strategies (section 8.3). We conclude with a discussion contrasting Böhm's theorem with the separation theorem of ludics (section 8.4).

8.1 Incorporating η into the GAM

We redefine the compilation of Böhm trees, keeping now the number of abstracted variables (resp. the number of arguments) as a superscript on opponent's moves (resp. on player's moves):

Abstract syntax:

$$P ::= [a, \overset{\kappa}{\hookleftarrow}]^m \{(1^{n_1}, M_1), \ldots, (m^{n_m}, M_m)\}$$
$$M ::= (P)$$

Concrete syntax:

$$P ::= [a, \star]^m \{(1, M_1), \ldots, (m, M_m)\}$$
$$M ::= (\lambda \mathtt{x}^n.P)$$

Compiling from untyped λ-calculus to concrete syntax:

$$\lceil x B_1 \ldots B_m \rceil = [x, \_]^m \{(1, \lceil B_1 \rceil, \ldots, (m, \lceil B_m \rceil^{)}\}$$
$$\lceil \mathtt{x}_i B_1 \ldots B_m \rceil = [i, \mathtt{x}]^m \{(1, \lceil B_1 \rceil, \ldots, (m, \lceil B_m \rceil^{)}\}$$

$$\lceil \lambda \mathtt{x}_1, \ldots, \mathtt{x}_n.P \rceil = (\lambda \mathtt{x}^n.\lceil P \rceil)$$

Compiling from concrete syntax to abstract syntax:

$$\lceil (\lambda \mathtt{x}^n.P) \rceil^{ca}_L = (\lceil P \rceil^{ca}_{\mathtt{x}\bullet L})^n$$

$$\begin{aligned}
&\lceil [a, _]^m \{(1, M_1), \ldots, (m, M_m) \mid\} \rceil^{ca}_L \\
&\quad = [a, \overset{-}{\hookleftarrow}]^m \{\mathtt{c}((1, \lceil M_1 \rceil^{ca}_L)), \ldots, \mathtt{c}((m, \lceil M_m \rceil^{ca}_L))\} \\
&\lceil [a, \mathtt{x}]^m \{(1, M_1), \ldots, (m, M_m) \mid\} \rceil^{ca}_L \\
&\quad = [a, \overset{L_{\mathtt{x}}}{\hookleftarrow}] \{\mathtt{c}((1, \lceil M_1 \rceil^{ca}_L)), \ldots, \mathtt{c}((m, \lceil M_m \rceil^{ca}_L))\}
\end{aligned}$$

where $\mathtt{c}((i, M^n) = (i^n, M)$.

The execution is driven by a variant of the GAM, which we call GAM$_\eta$. The GAM$_\eta$ continues to manipulate positions which do not have superscripts. But the function which maps the positions of ϕ to the same positions where the superscripts have been erased is injective. We use the notation $\mathit{filter}(\mathbf{r}, \phi)$ to denote the inverse to this injection (the execution keeps within the range of the injection).

The η-expansion process is taken care of by the following additional rules:

$$(\overline{2n})_\eta \quad \frac{\begin{array}{l} \mathit{hd}(\Gamma) = \{2n - 1 \leftarrow \mathbf{q}\} \quad \mathbf{q} = \mathbf{r}\langle m, \mathbf{2n-1}\rangle \\ \mathit{filter}(\mathbf{r}, \phi) = r_1 a^{n_1}[b, \overset{\kappa}{\hookleftarrow}]^{n_2} \quad m > n_2 \end{array}}{\begin{array}{l} \Gamma \mapsto \Gamma\{\overline{2n} \leftarrow \mathbf{q}[m - n_2 + n_1, \overset{1}{\hookleftarrow}]\} \\ (\Phi, \Psi) \mapsto (\Phi \cup \{\mathit{filter}(\mathbf{r}, \phi) m^0 [m - n_2 + n_1, \overset{1}{\hookleftarrow}]^0\}, \Psi) \end{array}}$$

$$(\overline{2n+1})_\eta \quad \frac{\begin{array}{l} \mathit{hd}(\Gamma) = \{2n \leftarrow \mathbf{q}\} \quad \mathbf{q} = \mathbf{r}\langle m, \mathbf{2n-1}\rangle \\ \mathit{filter}(\mathbf{r}, \psi) = r_1 a^{n_1}[b, \overset{\kappa}{\hookleftarrow}]^{n_2} \quad m > n_2 \end{array}}{\begin{array}{l} \Gamma \mapsto \Gamma\{\overline{2n+1} \leftarrow \mathbf{q}[m - n_2 + n_1, \overset{1}{\hookleftarrow}]\} \\ (\Phi, \Psi) \mapsto (\Phi, \Psi \cup \{\mathit{filter}(\mathbf{r}, \phi) m^0 [m - n_2 + n_1, \overset{1}{\hookleftarrow}]^0\}) \end{array}}$$

Notice that Φ and Ψ are progressively updated: they are η-expanded, as the need arises.

We can now complete the execution of the example of the preamble of this section:

$$\langle \bullet, \mathbf{1}\rangle[x, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle a, \mathbf{3}\rangle[a, \overset{1}{\hookleftarrow}] \\ \\ \langle x, \mathbf{2}\rangle[a, \overset{0}{\hookleftarrow}] \end{array} \right.$$

An example of endless η-expansion is provided by the well-known term $\Delta\Delta$ (with $\Delta = \lambda x.xx$):

$$(x(x))[x \leftarrow (\lambda y.y(y))]$$

The terms $(x(x))$ and $(\lambda y.y(y))$ are not η-long – a concept which is meaningless without types. So the machine has to make η-expansions dynamically.

Strategy for $(x(x))$:

$$\underset{\bullet}{\boxed{(}}\overset{\boxed{x_1}}{}[x, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \underset{1}{\boxed{(}}\overset{\boxed{x_2}}{}[x, \overset{-}{\hookleftarrow}] \end{array} \right.$$

Strategy for $[x \leftarrow (\lambda y.y(y))]$:

$$\underset{x}{\boxed{(\lambda y}}\overset{\boxed{y_1}}{}[1, \overset{0}{\hookleftarrow}] \left\{ \begin{array}{l} \underset{1}{\boxed{(}}\overset{\boxed{y_2}}{}[1, \overset{1}{\hookleftarrow}] \end{array} \right.$$

We display the steps of the GAM$_\eta$ up to step $(\overline{21})$. Note the (exponential) explosion of steps related with the traversal of the expansion variables that are progressively introduced. Notice also that the GAM is stuck as early as step $(\overline{5})$.

Multiplexed, expanded strategy for $(x(x))$:

$$\langle \bullet, \mathbf{1}\rangle[x, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{3}\rangle[x, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{5}\rangle \overset{\boxed{x_2^\eta}}{}[\overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{11}\rangle \overset{\boxed{x_2^{2\eta}}}{}[\overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 1, \mathbf{13}\rangle \overset{\boxed{x_2^\eta}}{}[\overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{19}\rangle \overset{\boxed{x_2^{2\eta}}}{}[\overset{1}{\hookleftarrow}] \end{array} \right. \end{array} \right. \\ \langle 1, \mathbf{7}\rangle[x, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{9}\rangle \overset{\boxed{x_2^\eta}}{}[\overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 1, \mathbf{15}\rangle[x, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{17}\rangle \overset{\boxed{x_2^\eta}}{}[\overset{1}{\hookleftarrow}] \end{array} \right. \end{array} \right.$$

Multiplexed, expanded strategy for $[x \leftarrow (\lambda y.y(y))]$:

$$\left\{\begin{array}{l} \langle x, \mathbf{2}\rangle[\overset{0}{\hookleftarrow}] \left\{\begin{array}{l} \langle 1, \mathbf{6}\rangle[\overset{1}{\hookleftarrow}] \left\{\begin{array}{r} \boxed{y_2^\eta} \\ \langle 1, \mathbf{10}\rangle\ [\overset{1}{\hookleftarrow}] \end{array}\right. \\ \langle 1, \mathbf{14}\rangle[\overset{1}{\hookleftarrow}] \left\{\begin{array}{r} \boxed{y_2^\eta} \\ \langle 1, \mathbf{18}\rangle\ [\overset{1}{\hookleftarrow}] \end{array}\right. \end{array}\right. \\ \langle x, \mathbf{4}\rangle[\overset{0}{\hookleftarrow}] \left\{\begin{array}{l} \langle 1, \mathbf{12}\rangle[\overset{1}{\hookleftarrow}] \left\{\begin{array}{r} \boxed{y_2^\eta} \\ \langle 1, \mathbf{20}\rangle\ [\overset{1}{\hookleftarrow}] \end{array}\right. \end{array}\right. \\ \langle x, \mathbf{8}\rangle[\overset{0}{\hookleftarrow}] \\ \langle x, \mathbf{16}\rangle[\overset{0}{\hookleftarrow}] \end{array}\right.$$

8.2 Böhm's theorem

Böhm's theorem asserts that in the λ-calculus one can separate two distinct normal forms M_1 and M_2 that are not η-convertible [3]. This means that we can find a context (that is, a term with a hole) C such that $C[M_1]$ (C whose hole has been filled with M_1) and $C[M_2]$ β-reduce to distinct (fresh) variables. This result in turn entails that $\beta\eta$ is a maximal consistent theory, since adding any new equation (between normalizable terms) would result in equating all pairs of terms.

As an illustration, we show how to separate xy et $x(xy)$. We follow Joly's proof of Böhm's theorem [25]. Instead of distinct fresh variables, we use distinct constants Ω and ✠, as in ludics (cf. section 3.5).

Joly's proof makes use of auxiliary terms:

$$\begin{array}{l} 0 = \lambda xy.y \\ 1 = \lambda xy.x \\ (M, N)_n = \lambda y_1 \dots y_n z.z(My_1 \dots y_n)(Ny_1 \dots y_n) \\ R = \lambda z.1 \\ S = 0 \end{array}$$

We observe:

$$\begin{array}{l} (M, N)_0 0 \to N \\ (M, N)_0 1 \to M \\ (M, N)_{k+1} P \to (MP, NP)_k \end{array}$$

The separating context is the following:

$$[\,][x \leftarrow (\lambda x.0, \lambda x.x)_3, y \leftarrow 1]RS01\Omega✠$$

We set $P \equiv y$ or $P \equiv xy$, and $P' \equiv P[x \leftarrow (\lambda x.0, \lambda x.x)_3, y \leftarrow 1]$. We have:

$$\begin{array}{l}(xP)[x \leftarrow (\lambda x.0, \lambda x.x)_3, y \leftarrow 1]RS01\Omega\Lsh \\ \quad \equiv (\lambda x.0, (\lambda x.x)_3 P'RS01\Omega\Lsh \\ \quad \to^* (\_, \lambda x.x)P'RS)_0 01\Omega\Lsh \\ \quad \to (\lambda x.x)P'RS1\Omega\Lsh \\ \quad \to P'RS1\Omega\Lsh\end{array}$$

- $P = y$. Then $P' \equiv 1$ and:

$$\begin{array}{l}P'RS1\Omega\Lsh \equiv 1RS1\Omega\Lsh \\ \quad \to R1\Omega\Lsh \equiv (\lambda z.1)1\Omega\Lsh \\ \quad \to 1\Omega\Lsh \\ \quad \to \Omega\end{array}$$

- $P = xy$. Then $P' \equiv (\lambda x.0, \lambda x.x)_3 1$ and:

$$\begin{array}{l}P'RS1\Omega\Lsh \equiv (\lambda x.0, \lambda x.x)_3 1RS1\Omega\Lsh \\ \quad \to^* ((\lambda x.0)1RS, \_)_0 1\Omega\Lsh \\ \quad \to (\lambda x.0)1RS\Omega\Lsh \\ \quad \to 0RS\Omega\Lsh \quad \to S\Omega\Lsh \\ \quad \to \Lsh\end{array}$$

Here is the execution (case $P \equiv y$). The compilation of

$$(zRS01\Omega\Lsh)[z \leftarrow (xy)[x \leftarrow Q, y \leftarrow 1]]$$

where

$$Q = (\lambda x_1 x_2 x_3 z.z(ax_1x_2x_3)(bx_1x_2x_3))[a \leftarrow \lambda x.0, b \leftarrow \lambda x.x]$$

is as follows:

$$\bullet^0[z, \overleftarrow{\rightarrow}]^6 \begin{cases} 1^3[2, \overset{0}{\hookleftarrow}]^0 \\ 2^2[2, \overset{0}{\hookleftarrow}]^0 \\ 3^2[2, \overset{0}{\hookleftarrow}]^0 \\ 4^2[1, \overset{0}{\hookleftarrow}]^0 \\ 5^0[\Omega, \overleftarrow{\rightarrow}] \\ 6^0[\Lsh, \overleftarrow{\rightarrow}] \end{cases} \qquad z^0[x, \overleftarrow{\rightarrow}]^1 \left\{ \; 1^0[y, \overleftarrow{\rightarrow}]^0 \right.$$

$$x^4[4,\overset{0}{\hookleftarrow}]^2 \begin{cases} 1^0[a,\overset{-}{\hookleftarrow}]^3 \begin{cases} 1^0[1,\overset{2}{\hookleftarrow}]^0 \\ 2^0[2,\overset{2}{\hookleftarrow}]^0 \\ 3^0[3,\overset{2}{\hookleftarrow}]^0 \end{cases} \\ 2^0[b,\overset{-}{\hookleftarrow}]^3 \begin{cases} 1^0[1,\overset{2}{\hookleftarrow}]^0 \\ 2^0[2,\overset{2}{\hookleftarrow}]^0 \\ 3^0[3,\overset{2}{\hookleftarrow}]^0 \end{cases} \end{cases}$$

$$a^3[3,\overset{0}{\hookleftarrow}]^0$$

$$b^1[1,\overset{0}{\hookleftarrow}]^0$$

$$y^2[1,\overset{0}{\hookleftarrow}]^0$$

And here is the dynamics:

$$\langle \bullet, \mathbf{1}\rangle[z,\overset{-}{\hookleftarrow}] \begin{cases} \langle 1, \mathbf{17}\rangle[2,\overset{0}{\hookleftarrow}] \\ \langle 3, \mathbf{5}\rangle[2,\overset{0}{\hookleftarrow}] \begin{cases} \langle 2, \mathbf{29}\rangle[4,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 5, \mathbf{33}\rangle[\Omega,\overset{-}{\hookleftarrow}] \end{cases}$$

$$\langle z, \mathbf{2}\rangle[x,\overset{-}{\hookleftarrow}] \begin{cases} \langle 1, \mathbf{10}\rangle[y,\overset{-}{\hookleftarrow}] \begin{cases} \langle 1, \mathbf{12}\rangle[1,\overset{1}{\hookleftarrow}] \begin{cases} \langle 2, \mathbf{22}\rangle[2,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 4, \mathbf{24}\rangle[4,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 2, \mathbf{16}\rangle[1,\overset{1}{\hookleftarrow}] \begin{cases} \langle 2, \mathbf{18}\rangle[2,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 4, \mathbf{4}\rangle[3,\overset{1}{\hookleftarrow}] \begin{cases} \langle 2, \mathbf{6}\rangle[2,\overset{1}{\hookleftarrow}] \begin{cases} \langle 2, \mathbf{28}\rangle[2,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 4, \mathbf{30}\rangle[4,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 6, \mathbf{32}\rangle[5,\overset{1}{\hookleftarrow}] \end{cases}$$

$$\langle x, \mathbf{3}\rangle[4,\overset{0}{\hookleftarrow}] \begin{cases} \langle 2, \mathbf{7}\rangle[b,\overset{-}{\hookleftarrow}] \begin{cases} \langle 1, \mathbf{9}\rangle[1,\overset{2}{\hookleftarrow}] \begin{cases} \langle 1, \mathbf{13}\rangle[1,\overset{1}{\hookleftarrow}] \begin{cases} \langle 2, \mathbf{21}\rangle[2,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 4, \mathbf{25}\rangle[4,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 2, \mathbf{15}\rangle[2,\overset{2}{\hookleftarrow}] \begin{cases} \langle 2, \mathbf{19}\rangle[2,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 5, \mathbf{27}\rangle[2,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 4, \mathbf{31}\rangle[6,\overset{1}{\hookleftarrow}] \end{cases}$$

$$\langle b, \mathbf{8}\rangle[1,\overset{0}{\hookleftarrow}] \begin{cases} \langle 1, \mathbf{14}\rangle[2,\overset{1}{\hookleftarrow}] \begin{cases} \langle 2, \mathbf{20}\rangle[2,\overset{1}{\hookleftarrow}] \end{cases} \\ \langle 4, \mathbf{26}\rangle[5,\overset{1}{\hookleftarrow}] \end{cases}$$

$\langle y, \mathbf{11}\rangle[1, \overset{0}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{23}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right.$

Taking now $P = xy$, we replace the tree starting with z by the following one:

$z^0[x, \overset{-}{\hookleftarrow}]^1 \left\{ \begin{array}{l} 1^0[x, \overset{-}{\hookleftarrow}]^1 \left\{ \begin{array}{l} 1^0[y, \overset{-}{\hookleftarrow}]^0 \end{array} \right. \end{array} \right.$

The execution is the same until step 10, where the visit of P starts. We give the full execution below:

$$\langle \bullet, \mathbf{1}\rangle[z, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{39}\rangle[2, \overset{0}{\hookleftarrow}] \\ \langle 3, \mathbf{5}\rangle[2, \overset{0}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{17}\rangle[3, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{25}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{53}\rangle[2, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 4, \mathbf{61}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 3, \mathbf{73}\rangle[5, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 4, \mathbf{21}\rangle[1, \overset{0}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{57}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 6, \mathbf{77}\rangle[\text{✠}, \overset{-}{\hookleftarrow}] \end{array} \right.$$

$$\langle z, \mathbf{2}\rangle[x, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{10}\rangle[x, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 3, \mathbf{34}\rangle[2, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{44}\rangle[2, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 4, \mathbf{12}\rangle[3, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{30}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{48}\rangle[2, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 4, \mathbf{66}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 6, \mathbf{68}\rangle[5, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 3, \mathbf{38}\rangle[2, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{40}\rangle[2, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 4, \mathbf{4}\rangle[3, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{6}\rangle[2, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{16}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{26}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{52}\rangle[2, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 4, \mathbf{62}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 3, \mathbf{72}\rangle[3, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 3, \mathbf{18}\rangle[3, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{24}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{54}\rangle[2, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 4, \mathbf{60}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 5, \mathbf{74}\rangle[5, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 5, \mathbf{20}\rangle[4, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{22}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{56}\rangle[2, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 4, \mathbf{58}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 7, \mathbf{76}\rangle[6, \overset{1}{\hookleftarrow}] \end{array} \right.$$

$$\langle x, \mathbf{3}\rangle[4, \overset{0}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{7}\rangle[b, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{9}\rangle[1, \overset{2}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{35}\rangle[2, \overset{1}{\hookleftarrow}] \left\{ \langle 2, \mathbf{43}\rangle[2, \overset{1}{\hookleftarrow}] \right. \\ \langle 3, \mathbf{13}\rangle[3, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{29}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \langle 2, \mathbf{49}\rangle[2, \overset{1}{\hookleftarrow}] \right. \\ \langle 4, \mathbf{65}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 5, \mathbf{69}\rangle[5, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 3, \mathbf{37}\rangle[3, \overset{2}{\hookleftarrow}] \left\{ \langle 2, \mathbf{41}\rangle[2, \overset{1}{\hookleftarrow}] \right. \\ \langle 4, \mathbf{15}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{27}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \langle 2, \mathbf{51}\rangle[2, \overset{1}{\hookleftarrow}] \right. \\ \langle 4, \mathbf{63}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 6, \mathbf{71}\rangle[3, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 3, \mathbf{19}\rangle[5, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{23}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \langle 2, \mathbf{55}\rangle[2, \overset{1}{\hookleftarrow}] \right. \\ \langle 4, \mathbf{59}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 5, \mathbf{75}\rangle[7, \overset{1}{\hookleftarrow}] \end{array} \right.$$

$$\langle x, \mathbf{11}\rangle[4, \overset{0}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{31}\rangle[a, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 3, \mathbf{33}\rangle[3, \overset{2}{\hookleftarrow}] \left\{ \langle 2, \mathbf{45}\rangle[2, \overset{1}{\hookleftarrow}] \right. \\ \langle 5, \mathbf{47}\rangle[2, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 4, \mathbf{67}\rangle[6, \overset{1}{\hookleftarrow}] \end{array} \right.$$

$$\langle a, \mathbf{32}\rangle[3, \overset{0}{\hookleftarrow}] \left\{ \langle 2, \mathbf{46}\rangle[5, \overset{1}{\hookleftarrow}] \right.$$

$$\langle b, \mathbf{8}\rangle[1, \overset{0}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 2, \mathbf{36}\rangle[3, \overset{1}{\hookleftarrow}] \left\{ \langle 2, \mathbf{42}\rangle[2, \overset{1}{\hookleftarrow}] \right. \\ \langle 3, \mathbf{14}\rangle[4, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{l} \langle 1, \mathbf{28}\rangle[1, \overset{1}{\hookleftarrow}] \left\{ \langle 2, \mathbf{50}\rangle[2, \overset{1}{\hookleftarrow}] \right. \\ \langle 4, \mathbf{64}\rangle[4, \overset{1}{\hookleftarrow}] \end{array} \right. \\ \langle 5, \mathbf{70}\rangle[6, \overset{1}{\hookleftarrow}] \end{array} \right.$$

8.3 A degenerated general instance

When there are neither arguments nor abstracted variables, then η-expansion makes sense even for arbitrary Böhm trees: $([a, \star]\{\ \})$ expands to

$$(\lambda \mathbf{z}.[a, \star]\{(a, [a, 1]) \mid a \in A'\})$$

where A' is an arbitrary subset of A. Taking $[a, \star] = [x_i, \_]$ and $A' = A$, and co-inductively iterating the process, we obtain the identity morphisms of our category, which completes the categorical picture of section 6. We check on one example that this works, by playing $([x_2, \overset{-}{\hookleftarrow}]\{\ \})$ against

$$\left\{\begin{array}{l} x_1\,\{\ \ldots \\ \\ x_2[a,\overset{0}{\hookleftarrow}]\,\{\ b[y,\overset{-}{\hookleftarrow}] \\ \\ x_3\,\{\ \ldots \end{array}\right.$$

We η-expand $([x_2,\overset{-}{\hookleftarrow}]\{\ \})$ dynamically, using the (strong version of the) machinery as in section 8.1, replacing in the rules m, n_1, and n_2 by a (arbitrary move), 0 (empty set of abstracted variables), and 0 (empty set of arguments), respectively:

$$\langle \bullet, \mathbf{1}\rangle[x_2,\overset{-}{\hookleftarrow}]\left\{\ \langle a, \mathbf{3}\rangle[a,\overset{1}{\hookleftarrow}]\left\{\ \langle b, \mathbf{4}\rangle[b,\overset{1}{\hookleftarrow}]\right.\right.$$

$$\left\{\begin{array}{l} x_1\,\{\ \ldots \\ \\ \langle x_2, \mathbf{2}\rangle[a,\overset{0}{\hookleftarrow}]\,\{\ b[y,\overset{-}{\hookleftarrow}]\,\{\ \langle b, \mathbf{5}\rangle[y,\overset{-}{\hookleftarrow}] \\ \\ x_3\,\{\ \ldots \end{array}\right.$$

8.4 Discussion

Böhm's theorem works only modulo η-expansion (we refer to [13] for a detailed analysis). But, as we have seen, η-conversion does not make sense in the general setting of abstract Böhm trees. As a matter of fact, Böhm's theorem does not hold [16] for the $\lambda\mu$-calculus in its original version, while only a small increase of flexibility in the syntax makes it hold again [32]. In other words, Böhm's theorem is rather fragile and does not extend easily to other syntaxes.

What about separation in the strict sense (not modulo η)? Maurel has exhibited the following simple counter-example to separation in ludics with pointers:

$$M_1 = (x \cdot \{0\})\{\lambda\{x_0\}.\maltese\}$$
$$M_2 = (x \cdot \{0\})\{\lambda\{x_0\}.(x \cdot \{0\})\{\lambda\{x_0\}.\maltese\}\}$$

Here, say, $\lambda\{x_0\}.\maltese$ is a notation for

$$\{\lambda\{x_0\}.\maltese\} \cup \{J = \lambda\{x_j \mid j \in J\}.\Omega \mid J \neq \{0\}\}$$

(This example is a variant of the terms xy and $x(xy)$ considered in section 8.2.)

The only (closed) opponents N able to interact with M_1 in such a way that $\langle M_1 \,|\, [x \leftarrow N]\rangle$ converges have the form

$$\{\ldots, (\{0\} = \lambda\{y_0\}.\maltese), \ldots\} \text{ or}$$
$$\{\ldots, (\{0\} = \lambda\{y_0\}.(y_0 \cdot \{0\})\{M\}), \ldots\}$$

The opponents to M_2 must also have the same form. But then it is easily checked that for any N in this class, the evaluations of *both* $\langle M_1 \mid [x \leftarrow N]\rangle$ and $\langle M_2 \mid [x \leftarrow N]\rangle$ converge. For example, with N of the second form, we have

$$\begin{array}{rcl}\langle M_2 \mid [x \leftarrow N]\rangle & \longrightarrow & \langle y_0 \cdot \{0\})\{M\} \mid [y_0 \leftarrow M_1[x \leftarrow N]]\rangle \\ & \longrightarrow & \langle M_1 \mid [x \leftarrow N]\rangle \\ & \longrightarrow & \langle y_0 \cdot \{0\})\{M\} \mid [y_0 \leftarrow (\lambda\{x_0\}.✠)[x \leftarrow N]]\rangle \\ & \longrightarrow & \langle ✠ \mid \ldots\rangle\end{array}$$

But why have we been able to separate xy and $x(xy)$? We can reformulate the separating context as follows. We have let

$$M_1 = \lambda xy.xy \quad \text{and} \quad M_2 = \lambda xy.x(xy)$$

interact with $zQ1RS01\Omega✠$, through the substitution of M_1 or M_2 for z. But in terms of ludics, this means that we have accepted an interaction between $(z \cdot \{1,2,3,4,5,6,7,8\}\ldots$ and $\lambda\{1,2\}\ldots$, which violates the machinery of ludics (and of abstract Böhm trees in general). The point of η-expansion is to allow to fill the gap, and to pretend that both M_1 and M_2 have arity 8.

Separation is recovered under Girard's affinity conditions, and the proof is then simple (see [20, 15]). In this respect, the situation is the same as for Böhm's theorem, which becomes tricky only when a head variable occurs in one of its arguments (like in $y(yx)$).

Maurel has shown how to recover separation without sacrificing nested occurrences of variables by extending the framework of designs to probabilistic designs [27]. The idea is to assign probabilities to actions, i.e., to head variables. When assigning probability $\frac{1}{2}$ to $y_0 \cdot \{0\}$ in $\lambda\{y_0\}.(y_0 \cdot \{0\})\{M\}), \ldots$, we get that $\langle M_1 \mid [x \leftarrow N]\rangle$ and $\langle M_2 \mid [x \leftarrow N]\rangle$ converge with probabilities $\frac{1}{2}$ and $\frac{1}{4}$, respectively, and hence M_1 and M_2 can be separated.

We summarize the discussion in figure 8.

We believe that Maurel's probabilistic ludics can be lifted to the general setting of abstract Böhm trees, and leave this as further work.

A Machine equivalences

We define precise translations between the machines. The translation functions are written $\lceil\_\rceil^{XY}$ (from machine X to machine Y).

A.1 VAM-GAM equivalence

We define two-way translations between the GAM and the VAM. The GAM state associated to a VAM state is (essentially) its associated set of multiplexed views, which are defined as follows:

exponential ludics : NO

(affine) ludics: YES exponential probabilistic ludics : YES

λ-calculus: yes (modulo η)

Figure 8: Three separation theorems

$$\frac{}{dview_\Gamma(1) = \bullet} \qquad \frac{\Gamma\bullet(2n) = a}{dview_\Gamma(2n) = \langle a, \mathbf{2n}\rangle}$$

$$\frac{\Gamma\bullet n = \langle a, \overline{m}\rangle}{dview_\Gamma(n) = dview_\Gamma(m-1)\ (\Gamma\bullet\overline{m})\ \langle a, \mathbf{n}\rangle}$$

VAM to GAM:

$$\frac{}{\lceil\{1 \leftarrow \bullet\}\rceil^{VG} = \{1 \leftarrow \langle\bullet, \mathbf{1}\rangle\}}$$

$$\frac{}{\lceil\Gamma\{n \leftarrow \_\}\rceil^{VG} = \lceil\Gamma\rceil^{VG}\{n \leftarrow dview_\Gamma(n)\}}$$

$$\frac{\lceil\Gamma\rceil^{VG}\bullet(n-1) = \mathbf{q}}{\lceil\Gamma\{\overline{n} \leftarrow [a, \overset{\kappa}{\hookleftarrow}]\}\rceil^{VG} = \lceil\Gamma\rceil^{VG}\{\overline{n} \leftarrow \mathbf{q}[a, \overset{\kappa}{\hookleftarrow}]\}}$$

where $\_$ stands for either a or $\langle a, \overline{m}\rangle$.

The GAM to VAM direction is essentially forgetful:

GAM to VAM:

$$\frac{}{\lceil\{1 \leftarrow \langle \bullet, \mathbf{1}\rangle\}\rceil^{GV} = \{1 \leftarrow \bullet\}}$$

$$\frac{}{\lceil\Gamma\{\overline{n} \leftarrow \mathbf{q}[a, \overset{\kappa}{\hookleftarrow}]\}\rceil^{GV} = \lceil\Gamma\rceil^{GV}\{\overline{n} \leftarrow [a, \overset{\kappa}{\hookleftarrow}]\}}$$

$$\frac{\Gamma\bullet\overline{2n} = \mathbf{q}[a, \overset{-}{\hookleftarrow}]}{\lceil\Gamma\{2n \leftarrow \langle a, \mathbf{2n}\rangle\}\rceil^{GV} = \lceil\Gamma\rceil^{GV}\{2n \leftarrow a\}}$$

$$\frac{\pi'(pop^i(\Gamma\bullet(n-1))) = m}{\lceil\Gamma\{n \leftarrow \mathbf{r}\langle a, \mathbf{n}\rangle\}\rceil^{GV} = \lceil\Gamma\rceil^{GV}\{n \leftarrow \langle a, \overline{m}\rangle\}}$$

We shall show that these transformations are inverse (on reachable states, i.e., on states that arise at some stage in the execution of the machine), and that the two machines simulate each other in lock step. We shall use the following invariants:

(1) $\pi'(pop^i(dview_\Gamma(n))) = jump^i_\Gamma(n) \quad (n > 1)$
(2) $dview_{\lceil\Gamma\rceil^{GV}}(n) = \Gamma\bullet n$ (for Γ reachable)

We prove claim (1) by induction on i:

1. $i = 0$. Then the claim reduces to $\pi'(dview_\Gamma(n)) = n$, which holds by definition of the multiplexed view function.

2. $i > 0$. Then $\Gamma\bullet n = \langle a, \overline{m}\rangle$ for some $\overline{m}$. We have, by definition of *jump* and *dvview*:

$$m - 1 = jump_\Gamma(n) \quad \text{and} \quad dview_\Gamma(n) = dview_\Gamma(m-1)\,(\Gamma\bullet\overline{m})\,\langle a, \mathbf{n}\rangle$$

   It follows that the claim reduces to its $(i{-}1, m{-}1)$ instance, which holds by induction.

The base cases of claim (2) are obvious. For the induction case, we have

$$\begin{array}{rcl} dview_{\lceil\Gamma\rceil^{GV}}(n) & = & dview_{\lceil\Gamma\rceil^{GV}}(m-1)\,(\lceil\Gamma\rceil^{GV}\bullet\overline{m})\,\langle a, \mathbf{n}\rangle \\ & = & (\Gamma\bullet(m-1))\,(\lceil\Gamma\rceil^{GV}\bullet\overline{m})\,\langle a, \mathbf{n}\rangle \\ & = & (\Gamma\bullet\overline{m})\,\langle a, \mathbf{n}\rangle \\ & = & \Gamma\bullet n \end{array}$$

using successively the induction hypothesis, the definition of the $\overline{m}$-rule of the GAM, the definition of the translation, and the definition of the n-rule of the GAM. Appeals to the rules of the GAM are legitimate because the claim is restricted to reachables states.

We now show that the machines simulate each other in lock-step. Let $\{1 \leftarrow \langle \bullet, \mathbf{1} \rangle\} = \Gamma_1, \ldots, \Gamma_\nu, \ldots$ be the successive states of the GAM. Then the VAM runs successively through the states $\{1 \leftarrow \bullet\} = \lceil \Gamma_1 \rceil^{GV}, \ldots \lceil \Gamma_\nu \rceil^{GV}, \ldots$, and terminates only when the GAM terminates; and the same holds conversely in the VAM to GAM direction. These properties are an easy consequence of the two claims. We check one case in detail: let $\Gamma \mapsto \Gamma\{2n \leftarrow \mathbf{r}'\langle a, \mathbf{2n} \rangle\}$, with $\pi'(pop^i(\Gamma\bullet(2n-1))) = 2m-1$. Using the two claims, we have:

$$\pi'(pop^i(\Gamma\bullet(2n-1))) = \pi'(pop^i(dview_{\lceil\Gamma\rceil^{GV}}(2n-1))) = jump^i_{\lceil\Gamma\rceil^{GV}}(2n-1)$$

from which it follows that

$$\lceil\Gamma\rceil^{GV} \mapsto \lceil\Gamma\rceil^{GV}\{2n \leftarrow \langle a, \overline{2m-1}\rangle\} = \lceil\Gamma\{2n \leftarrow \mathbf{r}'\langle a, \mathbf{2n}\rangle\}\rceil^{GV}$$

The fact that the two translations are inverse on reachable states is an immediate consequence of the mutual lock-step simulation.

A.2 SAM-GAM equivalence

We follow the same scheme as in section A.1. Since the proofs are fairly similar, we shall limit ourselves to stating the relevant invariants.

If Γ is a state of the GAM, we use Γ_ν to denote the prefix of Γ whose last item is the ν-th item of Γ. The inverse translations are defined as follows (the second translation necessitates to define at the same time a step number associated to a SAM state):

GAM to SAM:

$$\frac{}{\lceil\{1 \leftarrow \langle \bullet, \mathbf{1}\rangle\}\rceil^{GS} = \bullet}$$

$$\frac{hd(\Gamma) = \{\overline{n} \leftarrow \mathbf{q}[a, \overset{i}{\hookleftarrow}]\} \quad \pi'(pop^i(\mathbf{q})) = m}{\lceil\Gamma\{n \leftarrow \mathbf{r}'\langle a, \mathbf{n}\rangle\}\rceil^{GS} = \lceil\Gamma_{\overline{m}}\rceil^{GS}\langle a, \lceil\Gamma\rceil^{GS}\rangle}$$

$$\frac{hd(\Gamma) = \{\overline{2n} \leftarrow \mathbf{q}[a, \overset{-}{\hookleftarrow}]\}}{\lceil\Gamma\{2n \leftarrow \langle a, \mathbf{2n}\rangle\}\rceil^{GS} = \langle a, \lceil\Gamma\rceil^{GS}\rangle}$$

$$\frac{}{\lceil\Gamma\{\overline{n+1} \leftarrow \mathbf{q}[a, \overset{\kappa}{\hookleftarrow}]\}\rceil^{GS} = \lceil\Gamma\rceil^{GS}[a, \overset{\kappa}{\hookleftarrow}]}$$

SAM to GAM:

$$\frac{}{\lceil \bullet \rceil^{SG} = \{1 \leftarrow \langle \bullet, \mathbf{1} \rangle\}} \qquad \frac{}{\bullet^{\#} = 1}$$

$$\frac{\mathtt{q}^{\#} = n}{\lceil \mathtt{q}[a, \overset{\kappa}{\hookleftarrow}] \rceil^{SG} = \lceil \mathtt{q} \rceil^{SG} \{\overline{n+1} \leftarrow (\lceil \mathtt{q} \rceil^{SG} \bullet n)[a, \overset{\kappa}{\hookleftarrow}]\}} \qquad \frac{\mathtt{q}^{\#} = n}{(\mathtt{q}[a, \overset{\kappa}{\hookleftarrow}])^{\#} = \overline{n+1}}$$

$$\frac{\mathtt{r}^{\#} = \overline{n} \quad \mathtt{r'}^{\#} = \overline{m}}{\lceil \mathtt{r'}\langle a, \mathtt{r} \rangle \rceil^{SG} = \lceil \mathtt{r} \rceil^{SG} \{n \leftarrow (\lceil \mathtt{r'} \rceil^{SG} \bullet \overline{m}) \langle a, \mathbf{n} \rangle\}} \qquad \frac{\mathtt{r}^{\#} = \overline{n}}{(\mathtt{r'}\langle a, \mathtt{r} \rangle)^{\#} = n}$$

$$\frac{\mathtt{r}^{\#} = \overline{2n}}{\lceil \langle a, \mathtt{r} \rangle \rceil^{SG} = \lceil \mathtt{r} \rceil^{SG} \{2n \leftarrow \langle a, \mathbf{2n} \rangle\}} \qquad \frac{\mathtt{r}^{\#} = \overline{2n}}{\langle a, \mathtt{r} \rangle^{\#} = 2n}$$

In order to express one of the invariants of these translations, we introduce the following substate relation among SAM states. It is the transitive closure of the relation defined by the following rules:

$$\frac{}{\mathtt{q} \prec \mathtt{q}[a, \overset{\kappa}{\hookleftarrow}]} \qquad \frac{}{\mathtt{r'} \prec \mathtt{r'}\langle a, \mathtt{r} \rangle} \qquad \frac{}{\mathtt{r} \prec \mathtt{r'}\langle a, \mathtt{r} \rangle}$$

We also need the following auxiliary definition. Let σ be the function assigning to ν the state reached by the SAM at stage (ν), i.e., $\xrightarrow{1} \bullet = \sigma(1) \xrightarrow{\overline{2}} \sigma(\overline{2}) \ldots \xrightarrow{\nu} \sigma(\nu) \ldots$ We are now ready to express the invariants needed to prove the equivalence between the GAM and the SAM:

$(\sigma(\nu))^{\#} = \nu$
$\sigma(\mathtt{p}^{\#}) = \mathtt{p}$
$\mathtt{p}_1 \prec \mathtt{p}_2$, then $(\mathtt{p}_1)^{\#} < (\mathtt{p}_2)^{\#}$
the head item of $\lceil \mathtt{p} \rceil^{SG}$ is numbered $\mathtt{q}^{\#}$
$(\lceil \Gamma_\nu \rceil^{GS})^{\#} = \nu$
$(\lceil \mathtt{p}_1 \rceil^{SG})_{(\mathtt{p}_2)^{\#}} = \lceil \mathtt{p}_2 \rceil^{SG} \quad (\mathtt{p}_2 \prec \mathtt{p}_1)$
$(\pi'(pop^i(\lceil \mathtt{q} \rceil^{SG} \bullet \mathtt{q}^{\#})))' = (\pi'(pop^i(\mathtt{q})))^{\#}$
$\lceil \Gamma_{(\pi'(pop^i(\Gamma \bullet n)))'} \rceil^{GS} = \pi'(pop^i(\lceil \Gamma_n \rceil^{GS}))$
$erase(\lceil \Gamma_n \rceil^{GS}) = erase(\Gamma \bullet n)$

A.3 SAM-EAM equivalence

We relate the SAM to the EAM. Let

$$\langle P \mid [a_1, \_] \leftarrow M_1[\,], \ldots [a_n, \_] \leftarrow M_n[\,] \rangle$$

be an initial state of the EAM. We shall show that the EAM execution from there is lock-step simulated by the SAM execution of

$$\langle (\lceil P \rceil^{ca}_{[\,]}) \mid a_1 \leftarrow \lceil M_1 \rceil^{ca}_{[\,]}, \ldots, a_1 \leftarrow \lceil M_n \rceil^{ca}_{[\,]} \rangle$$

We translate a SAM state back to an EAM state as follows:

$$\frac{\begin{array}{l}\lceil \mathtt{r}' \rceil^{SE} = \langle [a_1, \overset{\star}{\hookleftarrow}] \{\ldots, (a, (\lambda \mathtt{y}.P)), \ldots\} \mid \rho' \rangle \\ \lceil \mathtt{r} \rceil^{SE} = \langle [b_1, \overset{\star}{\hookleftarrow}] \{(b, M_b) \mid b \in B\} \mid \rho \rangle\end{array}}{\lceil \mathtt{r}' \langle a, \mathtt{r} \rangle \rceil^{SE} = \langle P \mid \rho' [[b, \mathtt{y}] \leftarrow M_b \mid b \in B] \rangle}$$

$$\frac{\begin{array}{l}\lceil \mathtt{r}' \rceil^{SE} = \langle [a_1, \overset{\star}{\hookleftarrow}] \{\ldots, (a, (P)), \ldots\} \mid \rho' \rangle \\ \lceil \mathtt{r} \rceil^{SE} = \langle [b_1, \overset{\star}{\hookleftarrow}] \{\,\} \mid \rho \rangle\end{array}}{\lceil \mathtt{r}' \langle a, \mathtt{r} \rangle \rceil^{SE} = \langle P \mid \rho' \rangle}$$

In order to formulate the invariants of the simulation, we need to define the following translation function:

$$\frac{}{\lceil \bullet \rceil^{ac} = ((P), [\,])} \qquad \frac{}{\lceil a_i \rceil^{ac} = (M_i, [\,])}$$

$$\frac{\lceil r \rceil^{ac} = ([a, \star] \{\ldots, (b, M_b), \ldots\}, L)}{\lceil rb \rceil^{ac} = (M_b, L)}$$

$$\frac{\lceil q \rceil^{ac} = ((\lambda \mathtt{x}.P), L)}{\lceil q[a, \overset{\kappa}{\hookleftarrow}] \rceil^{ac} = (P, \mathtt{x} \bullet L)}$$

The function $\lceil \_ \rceil^{ac}$ reconstructs the matching subterm of (P) or of the M_i, together with the concrete binding information above it. The translation $\lceil \_ \rceil^{SE}$ satisfies the following invariants:

(1) $$\frac{\lceil \mathtt{r} \rceil^{SE} = \langle P \mid \rho \rangle}{P = \lceil erase(r) \rceil^{ac}}$$

(2) $$\frac{\lceil \mathtt{q} \rceil^{SE} = \langle P \mid \rho \rangle \qquad \lceil \pi'(pop^i(\mathtt{q})) \rceil^{SE} = [a_1, \star] \{\ldots, (a, M), \ldots\}}{\rho([a, pop^{\mathtt{i}}(\lceil erase(\mathtt{q}) \rceil^{ac}]) = M}$$

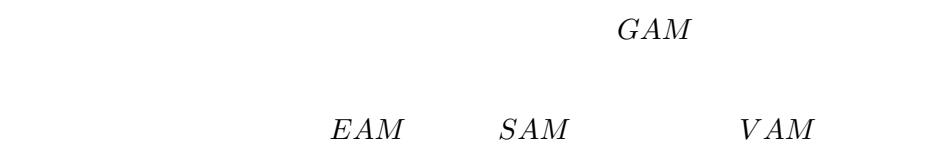


Figure 9: A unique computing device in four disguises

which in turn entail (easy check) that the SAM can proceed from $\mathtt{r}$ (to $\mathtt{q}$) if and only if the EAM can proceed from $\lceil \mathtt{r} \rceil^{SE}$ (to $\lceil \mathtt{q} \rceil^{SE}$).

Our picture is now complete. We have four equivalent presentations of the same computing device, as summarized in figure 9.

References

[1] R. Amadio and P.-L. Curien, Domains and lambda-calculi, Cambridge Univ. Press (1998).

[2] S. Abramsky, R. Jagadeesan, and P. Malacaria, Full abstraction for PCF, Information and Computation 163, 409-470 (2000).

[3] H. Barendregt, The lambda calculus; its syntax and semantics, North-Holland (1984).

[4] G. Berry and P.-L. Curien, Sequential algorithms on concrete data structures, Theoretical Computer Science 20, 265-321 (1982).

[5] G. Berry, P.-L. Curien, and J.-J. Lévy, Full abstraction for sequential languages: state of the art, in 'Algebraic methods in semantics', M. Nivat, J. Reynolds eds, Cambridge Univ. Press, 35-87 (1985).

[6] P. Boudes, Desequentialization of games and experiments on proof-nets, submitted (2005).

[7] N. de Bruijn, Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, Indag. Math. 34 ,381-392 (1972).

[8] R. Cartwright, P.-L. Curien, and M. Felleisen, Fully abstract semantics for observably sequential languages, Information and Computation 111 (2), 297-401 (1994).

[9] P.-L. Curien, Abstract Böhm trees, Mathematical Structures in Computer Science 8(6), 559-591(1998).

[10] T. Coquand, A semantics of evidence for classical arithmetic, Journal of Symb. Logic 60, 325–337 (1995).

[11] P.-L. Curien and H. Herbelin, Computing with Abstract Böhm Trees, in *Proceedings of the 3rd Fuji International Symposium on Functional and Logic Programming*, Eds M. Sato & Y. Toyama, World Scientific, 20-39 (1998).

[12] P.-L. Curien and H. Herbelin, The duality of computation, in Proc. ICFP 2000 (International Conference on Functional Programming), Montréal, sept. 2000, ACM Press.

[13] P.-L. Curien, Sur l'η-expansion infinie, Comptes-Rendus de l'Académie des Sciences 334, Sec. I, 77-82 (2002).

[14] P.-L; Curien, Playful, streamlike computation, invited paper, in Domain theory, logic and computation, Proceedings of the International Symposium on Domain Theory (ISDT 2001), Chengdu, China, October 2001, Series Semantic structures in computation, 1-24, Kluwer Academic Publishers (2003).

[15] P.-L. Curien, Introduction to linear logic and ludics, part II, to appear in Advances in Mathematics, China.

[16] R. David and W. Py, $\lambda\mu$-calculus and Böhm's theorem, Journal of Symbolic Logic 66(1) (2001).

[17] W. Felscher, Dialogues, strategies, and intuitionistic provability, Annals of Pure and Applied Logic 28, 217-254 (1985)

[18] J.-Y. Girard, Linear logic, Theoretical Computer Science 50, 1-102 (1987).

[19] J.-Y. Girard, Geometry of interaction I: interpretation of system F, in Proc. Logic Colloquium '88, 221-260, North Holland (1989).

[20] J.-Y. Girard, Locus solum: from the rules of logic to the logic of rules, Mathematical Structures in Computer Science 11(3), 301-506 (2001).

[21] H. Herbelin, Séquents qu'on calcule, Thèse de Doctorat, Université Paris VII (1995).

[22] H. Herbelin, Games and Weak-Head Reduction for Classical PCF, Proceedings of TLCA 97, LNCS 1210, 214–230.

[23] K. Honda and N.Yoshida, Game-theoretic analysis of call-by-value computation, Proc. ICALP 97, Lecture Notes in Computer Science 1256, Springer (1997).

[24] M. Hyland and L. Ong, On full abstraction for PCF, Information and Computation 163(2), 285-408 (2000).

[25] Th. Joly, Codages, séparabilité et représentation de fonctions en λ-calcul simplement typé et dans d'autres systèmes de types, Thèse de doctorat, Université Paris 7, 2000.

[26] J.-L. Krivine, A call-by-name lambda-calculus machine, Higher-Order and Symbolic Computation, to appear.

[27] F. Maurel,Un cadre quantitatif pour la ludique, Thèse de Doctorat, Université Paris 7 (2004).

[28] H. Nickau, Hereditarily Sequential Funtionals, In: Proc. Symp. Logical Foundations of Computer Science: Logic at St. Petersburg, Eds. A. Nerode and Yu. V. Matiyasevich, Lecture Notes in Computer Science, volume 813, pages 253-264, Springer-Verlag, (1994).

[29] M. Parigot, $\lambda\mu$-calculus, an algorithmic interpretation of classical natural deduction, in Proc. LPAR 92, LNCS 624 (1992).

[30] G. Plotkin, Call-by-name, call-by-value and the lambda-calculus, Theoretical Computer Science 1, 125-159 (1975).

[31] G. Plotkin, LCF as a programming language, Theoretical Computer Science 5, 223-257 (1977).

[32] A. Saurin, Separation and the $\lambda\mu$-calculus, in Proceedings of Logic in Computer Sciencer 2005.