Computing with Abstract Böhm Trees

Pierre-Louis Curien
CNRS-ENS, 45 rue d'Ulm, 75005 PARIS

Hugo Herbelin
INRIA, 78153 ROCQUENCOURT CEDEX

We focus on purely operational aspects of game semantics. We present abstract Böhm trees, which capture the computational core of various λ-calculus based proof systems and languages. We introduce a computing device over abstract Böhm trees, called Strategic Abstract Machine. This machine arises as an abstraction of (a stack-free version of) Krivine's abstract machine, and is step to step related to (abstractions of) known devices: the interaction mechanism of Coquand's, Hyland-Ong's or Nickau's game semantics, and Danos-Regnier's pointer machine. Pulling our abstract device back to concrete syntax yields simple abstract machines that can handle reduction of PCF trees or untyped λ-terms without stack nor dump. Our framework is thus unifying in two directions: on one hand it allows for precise comparisons of different abstract machines, and on the other hand any of these abstract machines can execute different sorts of code.

1 Introduction

In the recent years, game semantics has brought new insights on computation in typed programming languages [1,12] and proof theory [2,13,3], following older works by Lorenzen's school – see Felscher [8]. For these semantics, terms and proofs are interpreted by strategies in some game and reduction to normal form is interaction between the strategies.

The second author has observed [9] that interaction between strategies in Coquand's model of sequent calculus[3] follows step by step the process of weak-head cut-elimination. Then, observing the analogy between Hyland-Ong's style of games [12] and Danos-Regnier's variant [7] of Krivine's abstract machine [4], the step by step correspondence between call-by-name abstract machines and interaction in games has become more precise [6] [a]. (In our setting, Coquand's machine and the game-theoretic version of Danos-Regnier's machines are called View Abstract Machine (or *VAM*) and Pointer Abstract Machine (or *PAM*), respectively.)

In the present paper, we make the correspondence even more precise, between syntax and game semantics, and between call-by-name abstract machines and interaction. Our treatment is based on a notion of abstract Böhm

[a] In the cited paper [6], a relation between Abramsky-Jagadeesan's style of game and the Geometry of Interaction is also exhibited.

tree, which arises as a natural abstraction of λ-calculus' (η-long) Böhm trees. Abstract Böhm trees have to be compared with Coquand's strategies [3], with the functions underlying Hyland-Ong's innocent strategies [12], with Nickau's sequential strategies [14] and with the strategies for E-dialogues in the sense of Felscher [8]. Abstract Böhm trees capture the computational core of pure simply-typed λ-calculus but also of simply-typed λ-calculi with constants and case statements. They can be extended to capture untyped λ-calculus too.

In the first two sections, we exemplify the relation between concrete and abstract computing device. Two related machines are described. The first one, called *KAM*, is a simple variant of Krivine's abstract machine specially devised for computing over simply-typed η-long normal λ-terms: it has no stack and only one rule. The second machine, called Strategic Abstract Machine (*SAM*), is the abstract counterpart of the *KAM*. It works on abstract Böhm trees. In section 4, we present a handy geometric variation of the *SAM*.

In section 5, we show that the *SAM* can handle computation over PCF Böhm trees (which, roughly, are with respect to Scott's language PCF [15] what Böhm trees are with respect to λ-calculus), without any need to add instructions to handle the constants and the case statements. However, a slight extension is necessary to handle arbitrary untyped λ-terms (section 6). Pulling back in the original concrete syntax the work carried out in sections 5 and 6, we get simple stack-free and dump-free call-by-name environment machines for computing over PCF Böhm trees and over arbitrary untyped λ-terms, respectively (section 7). We let open the question whether the absence of dump and stack could be advantageously exploited in practice.

Finally, in section 8, we show how the *SAM* relates to the *VAM* and the *PAM* discussed above.

2 Stack-free machine (*KAM*)

We introduce an abstract machine with two components – the code and the environment – to implement the reduction of *η-long* λ-terms. In such terms, each occurrence of a variable must appear in a context where it 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 $N_1, \ldots, N_n$ which the term $\lambda\vec{x}.M$ can accept according to its type. Moreover, we suppose that $M, N_1, \ldots, N_n$ are *normal forms.* The two assumptions (η-long and normal) make the game-theoretic interpretation easier to grasp. The η-long restriction allows us to dispense of the stack component, which (in call-by-name) only serves to store arguments. Without the stack, we can discover a symmetry between the two remaining components. Indeed, the second restriction enables

$$(K) \quad (x\vec{M})[e] \to W[\langle \vec{z} \leftarrow \vec{M}\rangle[e] \bullet e_i]$$

$$\text{where } \begin{cases} e = B_0 \bullet \cdots \bullet B_n & B_i = \langle \vec{x_i} \leftarrow \vec{N_i}\rangle[e_i] \\ x = x_{ij} & N_{ij} = (\lambda \vec{z}.W) \end{cases}$$

Figure 1: The Stack-Free Krivine Abstract Machine

us to observe a strict alternation between the code and the environment. (The restrictions will be relaxed in section 6.)

To further smooth the way towards a reading of terms as strategies, we write a (head) normal form $M = \lambda\vec{x}.y\vec{N}$ as: $M = (\lambda\vec{x}.W)$, where $W = y\vec{N}$. We split M in this way even if $\vec{x}$ is empty: in this case we write $M = (W)$ and $W = y\vec{N}$. We call W a body, or *code*. The Stack-Free Krivine Abstract Machine (or *KAM*) has only one rule, given in figure 1. The states of the *KAM* are closures, i.e., pairs of the form $W[e]$, where W is a code, and where e is an environment, i.e. a list of environment items B each consisting of a list of closures:

$$\begin{array}{ll} M ::= (\lambda\vec{x}.W) & e ::= nil \mid B \bullet e \\ W ::= y\vec{M} & B ::= \langle \vec{z} \leftarrow \vec{M}\rangle[e]. \end{array}$$

Thus environments are lists of lists of closures, not just lists of closures as in usual environment machines. This two-level structure corresponds to the representation of normal forms: each environment item is to be matched with a *vector* of λ's, not just one. We usually do not write nil, e.g. $B \bullet nil$ is written B, or $(\vec{z} \leftarrow \vec{M})[nil]$ is written $(\vec{z} \leftarrow \vec{M})[\,]$. The notation $\langle \vec{z} \leftarrow \vec{M}\rangle[e]$ should be read as a vector $M_1[e], \ldots, M_n[e]$ of closures bound to $z_1, \ldots, z_n$.

Each machine step can be decomposed in two phases. Given $(x\vec{M})[e]$, the machine first looks for the value of x in e. We assume that all bound variables have distinct names. Thus, there exists at most one pair (i, j) such that $x = x_{ij}$: the index i is the environment item in which the binding of x lies, and j is the position of x in this environment item. If $(\lambda\vec{z}.W)$ is the value of x, then the second phase consists in extending the environment with an environment item that binds $\vec{M}$ to $\vec{z}$. Figure 2 shows the execution of

$$u(\lambda x.u(\lambda y.x))\,[u \leftarrow (\lambda r.r(r(z)))].$$

We take $u(\lambda x.u(\lambda y.x))\,[\langle u \leftarrow (\lambda r.r(r(z)))\rangle[\,]]$ as initial state. Notice the addition of $[\,]$ to turn $(\lambda r.r(r(z)))$ into a closure. The code part and the environment part of the state are displayed in the second and the third column, respectively. In this execution, we have numbered the machine steps $2', 3', 4', \ldots$. Notice the strict alternation between (subterms of) $(u(\lambda x.u(\lambda y.x)))$ and (subterms

$2'$	$u(\lambda x.u(\lambda y.x))$	$\langle u \stackrel{2}{\leftarrow} (\lambda r.r(r(z)))\rangle[\,] = \rho_0$
$3'$	$r(r(z))$	$\langle r \stackrel{3}{\leftarrow} (\lambda x.u(\lambda y.x))\rangle[\rho_0] = \rho_1$
$4'$	$u(\lambda y.x)$	$\langle x \leftarrow (r(z))\rangle[\rho_1]\bullet(\rho_0 = \langle u \stackrel{4}{\leftarrow} (\lambda r.r(r(z)))\rangle[\,]) = \rho_2$
$5'$	$r(r(z))$	$\langle r \stackrel{5}{\leftarrow} (\lambda y.x)\rangle[\rho_2] = \rho_3$
$6'$	x	$\langle y \leftarrow (r(z))\rangle[\rho_3]\bullet(\rho_2 = \langle x \stackrel{6}{\leftarrow} (r(z))\rangle[\rho_1]\bullet\rho_0) = \rho_4$
$7'$	$r(z)$	$\langle\rangle[\rho_4]\bullet(\rho_1 = \langle r \stackrel{7}{\leftarrow} (\lambda x.u(\lambda y.x))\rangle[\rho_0]) = \rho_5$
$8'$	$u(\lambda y.x)$	$\langle x \leftarrow (z)\rangle[\rho_5]\bullet(\rho_0 = \langle u \stackrel{8}{\leftarrow} (\lambda r.r(r(z)))\rangle[\,]) = \rho_6$
$9'$	$r(r(z))$	$\langle r \stackrel{9}{\leftarrow} (\lambda y.x)\rangle[\rho_6] = \rho_7$
$10'$	x	$\langle y \leftarrow (r(z))\rangle[\rho_7]\bullet(\rho_6 = \langle x \stackrel{10}{\leftarrow} (z)\rangle[\rho_5]\bullet\rho_0) = \rho_8$
$11'$	z	$\langle\rangle[\rho_8]\bullet\rho_5$

Figure 2: The execution of $u(\lambda x.u(\lambda y.x))\,[u \leftarrow (\lambda r.r(r(z)))]$ (KAM)

of) $(\lambda r.r(r(z)))$. We have placed superscripts $2, 3, 4, \ldots$ on the bindings of the environments which are used "between" the steps. More precisely, we can identify successive "moves" $2', 2, 3', 3, 4', 4, \ldots n', n, (n+1)', \ldots$, which correspond to (copies of) subterms of $(u(\lambda x.u(\lambda y.x)))$ and $(\lambda r.r(r(z)))$:

$2'$	points to $u(\lambda x.u(\lambda y.x))$	(code)
2	points to $(\lambda r.r(r(z)))$	(term)
$3'$	points to $r(r(z))$	(code)
...		

The following invariants hold, for each n: (i) the subterm associated with n is the value (in the environment) of the head variable of the code associated with n'; (ii) if the subterm associated with n is $(\lambda\vec{y}.W)$, then the code associated with $(n+1)'$ is W.

3 Strategic machine (SAM)

In this section, we lift the machine presented in section 2 to a more abstract level, where terms are replaced by more general and more "semantic" objects, called abstract Böhm trees, or strategies. While a λ-term is defined algebraically, an abstract Böhm tree is characterized by the set of its (partial) branches (identified with positions in a game).

We suppose given an alphabet A of *moves*, containing a special symbol $\bullet$, which is to serve as initial move. *Positions* are sequences of moves with backward pointers from moves occurring at even places in the position to moves occurring at odd places. We choose to represent pointers by numbers that count the number of opponent's moves between the pointing player's move and the pointed opponent's move. Thus an even position, i.e. a position of even length, is a non-empty sequence r of the form:

$a_1[a_2, \overset{\kappa_1}{\hookleftarrow}] \cdots a_{2n-1}[a_{2n}, \overset{\kappa_n}{\hookleftarrow}]$

($a_j \in A$ for all $j \leq 2n$ and $\kappa_l \in \{0, 1, \ldots, l-1\} \cup \{\_\}$ for all $l \leq n$); $a_1, a_3, \ldots$ and $[a_2, \overset{\kappa_1}{\hookleftarrow}], [a_4, \overset{\kappa_2}{\hookleftarrow}], \ldots$ are called opponent's moves and player's moves, respectively. If $\kappa_l = \_$, we say that $[a_{2l}, \overset{\kappa_l}{\hookleftarrow}]$ occurs free. Otherwise, we say that $[a_{2l}, \overset{\kappa_l}{\hookleftarrow}]$ is bound to $a_{2(l-\kappa_l)-1}$. We use i to range over κ's different from $\_$. An odd position $q = a_1[a_2, \overset{\kappa_1}{\hookleftarrow}] \cdots a_{2n-1}$ is defined in the same way. We use p to denote a position which is either even or odd.

An abstract Böhm tree, or *strategy*, is a tree consisting of a set ϕ of even positions of the form $\bullet p$, where $\bullet$ does not occur in p, closed under even non-empty prefixes and such that $q[a_1, \overset{\kappa_1}{\hookleftarrow}], q[a_2, \overset{\kappa_2}{\hookleftarrow}] \in \phi$ implies $[a_1, \overset{\kappa_1}{\hookleftarrow}] = [a_2, \overset{\kappa_2}{\hookleftarrow}]$. The last condition says that, in a strategy, a "query", (that is, an odd position), if answered, is uniquely so. Therefore, any strategy can be unambiguously presented as a partial function, also written ϕ, from odd positions (in which the player's moves are actually irrelevant) to pointing player's moves $[a, \overset{\kappa}{\hookleftarrow}]$.

We also need the related notion of *multistrategy*. First we define a renaming operation: $[a \leftarrow \phi] = \{ar \mid \bullet r \in \phi\}$. Hence $[a \leftarrow \phi]$'s root is labelled by a. A multistrategy 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 more suggestively as:

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

As an illustration, we define a compilation of (η-long) λ normal forms. We take as alphabet $A = \{\bullet\} \cup \omega \cup \mathit{Var}$, where ω is the set of natural numbers, and where the variables $x \in \mathit{Var}$ are only used as player's moves of the form $[x, \overset{-}{\hookleftarrow}]$. We perform the following transformation on the tree representation

$$\overbrace{M_1 \quad \cdots \quad M_n}^{\lambda\vec{x}.y}$$

of a normal form M. The node "$\lambda\vec{x}.y$" is split into two moves: "$\lambda\vec{x}$" is associated to the opponent's move $\bullet$ and "y" to a player's move $\underline{y}$. One may think that two players are exploring M: the opponent asks for head variable and the player provides it. When y is free, $\underline{y}$ is $[y, \overset{-}{\hookleftarrow}]$. Otherwise, $\underline{y}$ is $[n, \overset{i}{\hookleftarrow}]$ such that the binder is found $2i + 1$ moves above and n denotes the position of y in the multiple abstraction $\lambda y_1 \cdots y_m$ which binds it, i.e. $y = y_n$. In the terminology of Danos and Regnier, n is called the *rank* and i is called the *lift*. The pairs (n, i), which we call *Böhm indices*, also appear in [11]. Finally, if ϕ_i is the translation of M_i, for all i, then the head opponent's move $\bullet$ of ϕ_i is replaced by i. Summarising, the translation of $M = (\lambda\vec{x}.yM_1 \cdots M_n)$ is the following tree:

$$\overbrace{[1 \leftarrow \phi_1] \quad \cdots \quad [n \leftarrow \phi_n]}^{\bullet \underline{y}}$$

Here is a formal translation involving a function which takes a list σ of vectors of variables and a term M, and returns a strategy M^σ:

$$(\lambda\vec{x}.y\vec{M})^\sigma = \{\bullet y^{\vec{x}\bullet\sigma}\} \bigcup_{i=1\cdots n} \{\bullet y^{\vec{x}\bullet\sigma} ir \mid \bullet r \in (M_i)^{\vec{x}\bullet\sigma}\}$$

where

$$\frac{}{x^{nil} = [x, \overset{-}{\hookleftarrow}]} \qquad \frac{x = x_i \quad \vec{x} = x_1 \cdots x_k}{x^{\vec{x}\bullet\sigma} = [i, \overset{0}{\hookleftarrow}]}$$

$$\frac{x^\sigma = [i, \overset{j}{\hookleftarrow}]}{x^{\vec{y}\bullet\sigma} = [i, \overset{j+1}{\hookleftarrow}]} \qquad \frac{x^\sigma = [x, \overset{-}{\hookleftarrow}]}{x^{\vec{y}\bullet\sigma} = [x, \overset{-}{\hookleftarrow}]}$$

The compilation of a term M is defined as M^{nil}. This is more than a compilation. It is injective (under our η-long assumption). We can even turn it into a bijection (whence the terminology of abstract Böhm trees), by imposing explicitly some additional conditions in the definition of a strategy ϕ.

Below, we display the compilation of the example of the previous section. The boxed comments give the correspondence with the source terms.

Strategy	Multistrategy
$\overset{\boxed{(}}{\bullet}[u, \overset{-}{\hookleftarrow}] \left\{ \overset{\boxed{(\lambda x}}{1} [u, \overset{-}{\hookleftarrow}] \left\{ \overset{\boxed{(\lambda y}}{1} \overset{\boxed{x}}{[1, \overset{1}{\hookleftarrow}]} \right.\right.$	$\overset{\boxed{(\lambda r}}{u} \overset{\boxed{r}}{[1, \overset{0}{\hookleftarrow}]} \left\{ \overset{\boxed{(}}{1} \overset{\boxed{r}}{[1, \overset{1}{\hookleftarrow}]} \left\{ \overset{\boxed{(}}{1} [z, \overset{-}{\hookleftarrow}] \right.\right.$

We now express an abstract version of the *KAM*, called Strategic Abstract Machine (or *SAM*). The goal of a *SAM* execution is to compute (the "head normal form" of) a composition $\phi \circ \psi$ of a strategy ϕ whose free player's moves are $[a_1, \overset{-}{\hookleftarrow}], \ldots, [a_n, \overset{-}{\hookleftarrow}]$ with a multistrategy $\psi = [a_1 \leftarrow \phi_1, \ldots, a_n \leftarrow \phi_n]$. (Think of a substitution $M[\vec{x} \leftarrow \vec{N}]$.). The states of the *SAM* are abstract versions of closures: they are pairs $p[e]$ of a position and an environment. The syntax for environments is as follows (where r stands for an even position)::

$$e ::= nil \;\Big|\; B \bullet e \quad B ::= r[e]\,.$$

The rules of the *SAM* are given in figure 3. In rule $(n)_b$, $\pi_{i+1}(e)$ denotes the $(i+1)$-th element of e, i.e. $\pi_{i+1}(B_0 \bullet \cdots \bullet B_i \bullet e') = B_i$. For all the machines

$$
\begin{array}{ll}
(1) & \dfrac{}{\rightarrow \bullet[\,]} \\[2ex]
(2n') & \dfrac{\phi(q) = [a, \overset{\kappa}{\hookleftarrow}]}{q[e] \rightarrow (q[a, \overset{\kappa}{\hookleftarrow}])[e]} \\[2ex]
((2n+1)') & \dfrac{\psi(q) = [a, \overset{\kappa}{\hookleftarrow}]}{q[e] \rightarrow (q[a, \overset{\kappa}{\hookleftarrow}])[e]} \\[2ex]
(n)_b & \dfrac{r = q[a, \overset{i}{\hookleftarrow}] \text{ and } \pi_{i+1}(e) = r'[e']}{r[e] \rightarrow (r'a)[r[e]\bullet e']} \\[2ex]
(2n)_f & \dfrac{r = q[a, \overset{-}{\hookleftarrow}]}{r[e] \rightarrow a[r[e]]}
\end{array}
$$

Figure 3: The Strategic Abstract Machine

presented in the paper, we consider it implicit that there is a counter recording the current machine step number. The successive steps are numbered $1, 2', 2, 3', 3, 4', \ldots$, with the following player/opponent and ϕ/ψ alternation:

1	initial opponent's move in ϕ	$2'$	player's move in ϕ
2	opponent's move in ψ	$3'$	player's move in ψ
3	opponent's move in ϕ	$4'$	player's move in ϕ
...			

The machine starts with the initial position $\bullet$ and the empty environment. After a step $2n-1$ ($2n$), it looks up in ϕ (ψ) from which it gets a player's move $[a, \overset{\kappa}{\hookleftarrow}]$ (rules $(2n')$ and $((@n+1)')$. If $[a, \overset{\kappa}{\hookleftarrow}]$ is bound, then rule (n_b) is executed; if $[a, \overset{\kappa}{\hookleftarrow}]$ is free in ϕ, then the execution goes to the corresponding strategy of ψ (rule $(2n)_f$).

The following invariant holds: at each state $p[e]$, the length of e is the number of opponent's moves in p (without including $\bullet$).

The execution of the example of section 2 by the *SAM* is shown on figure 4. The empty environment in step $2'$ reflects a minor difference in the treatment of the initial substitution $[u \leftarrow (\lambda r.r(r(z)))]$. In section 2, we had turned this substitution into an initial environment $\langle u \leftarrow (\lambda r.r(r(z)))\rangle[\,]$. Here we treat it as a multistrategy stored outside, and the rule $(2n)_f$ is used for fetching the value bound to u.

Below, we formalize our claim that the *KAM* can be viewed as an instance of the *SAM*. Let $M = (\lambda\vec{x}.W)$ be a term and $\vec{N}$ a vector of η-long and normal terms. Let $\phi = M^{nil}$ and $\psi = [x_1 \leftarrow N_1^{nil}, \ldots, x_n \leftarrow N_n^{nil}]$ be their respective compilations. To each odd position $q = \bullet p$ in ϕ or $q = x_k p$ in ψ, we associate

$$
\begin{array}{lll}
2' & \bullet[u, \overset{-}{\hookleftarrow}] & [\,] \\
3' & u[1, \overset{0}{\hookleftarrow}] & (\bullet[u, \overset{-}{\hookleftarrow}])[\,] = e_1 \\
4' & \bullet[u, \overset{-}{\hookleftarrow}]1[u, \overset{-}{\hookleftarrow}] & (u[1, \overset{0}{\hookleftarrow}])[e_1] = e_2 \\
5' & u[1, \overset{0}{\hookleftarrow}] & (\bullet[u, \overset{-}{\hookleftarrow}]1[u, \overset{-}{\hookleftarrow}])[e_2] = e_3 \\
6' & \bullet[u, \overset{-}{\hookleftarrow}]1[u, \overset{-}{\hookleftarrow}]1[1, \overset{1}{\hookleftarrow}] & (u[1, \overset{0}{\hookleftarrow}])[e_3]\bullet e_2 = e_4 \\
7' & u[1, \overset{0}{\hookleftarrow}]1[1, \overset{1}{\hookleftarrow}] & (\bullet[u, \overset{-}{\hookleftarrow}]1[u, \overset{-}{\hookleftarrow}]1[1, \overset{1}{\hookleftarrow}])[e_4]\bullet e_1 = e_5 \\
8' & \bullet[u, \overset{-}{\hookleftarrow}]1[u, \overset{-}{\hookleftarrow}] & (u[1, \overset{0}{\hookleftarrow}]1[1, \overset{1}{\hookleftarrow}])[e_5] = e_6 \\
9' & u[1, \overset{0}{\hookleftarrow}] & (\bullet[u, \overset{-}{\hookleftarrow}]1[u, \overset{-}{\hookleftarrow}])[e_6] = e_7 \\
10' & \bullet[u, \overset{-}{\hookleftarrow}]1[u, \overset{-}{\hookleftarrow}]1[1, \overset{1}{\hookleftarrow}] & (u[1, \overset{0}{\hookleftarrow}])[e_7]\bullet e_6 = e_8 \\
11' & u[1, \overset{0}{\hookleftarrow}]1[1, \overset{1}{\hookleftarrow}]1[z, \overset{-}{\hookleftarrow}] & (\bullet[u, \overset{-}{\hookleftarrow}]1[u, \overset{-}{\hookleftarrow}]1[1, \overset{1}{\hookleftarrow}])[e_8]\bullet e_5 = e_9
\end{array}
$$

Figure 4: The execution of $(u(\lambda x.u(\lambda y.x)))^{nil} \circ [u \leftarrow (\lambda r.r(r(z)))^{nil}]$ (SAM)

a subterm $M_{|p}$ of M or $(N_k)_{|p}$ of N_k:

$$(\lambda\vec{y}.z\vec{P})_{|\epsilon} = (\lambda\vec{y}.z\vec{P}) \quad (\lambda\vec{y}.z\vec{P})_{|[a, \overset{\kappa}{\hookleftarrow}]kp} = (P_k)_{|p}$$

If $(\lambda\vec{y}.z\vec{P})$ is associated to q, we set $vars(q) = \vec{y}$, $code(q) = z\vec{P}$, and $args(q) = \vec{P}$. We can now define the translation from SAM states (at steps n') to KAM states: $\lceil(q[a, \overset{\kappa}{\hookleftarrow}][e])\rceil^{sk} = code(q)[\lceil e\rceil^{sk}_q]$, where environments are translated as follows:

$$\frac{}{\lceil nil\rceil^{sk}_{\bullet} = \langle\vec{x} \leftarrow \vec{N}\rangle[\,]} \qquad \frac{}{\lceil nil\rceil^{sk}_{a} = nil}$$

$$\frac{q = q_1[a, \overset{\kappa}{\hookleftarrow}]b \qquad r = q'[a', \overset{\kappa'}{\hookleftarrow}]}{\lceil(r[e]\bullet e')\rceil^{sk}_q = \langle vars(q) \leftarrow args(q')\rangle[\lceil e\rceil^{sk}_{q'}]\bullet\lceil e'\rceil^{sk}_{q_1}}$$

Proposition 3.1 *If $r[e]$ is the state at step n' of the SAM execution of (the compilation of) $(\lambda\vec{x}.W)\vec{N}$, then $\lceil r[e]\rceil^{sk}$ is the state at step n' of the KAM run started with $W[\langle\vec{x} \leftarrow \vec{N}\rangle[\,]]$.*

Thus each application of the unique rule of the KAM corresponds to a step n followed by a step n' of the SAM.

4 Geometric machine (GAM)

In this section, we introduce a variation of the SAM aimed to represent graphically the execution in the SAM. We call it Geometric Abstract Machine, or GAM. In the GAM, execution consists in a travel in the strategy ϕ and the multistrategy ψ. Any step n or n' of the SAM corresponds to an odd or even

$$(1)^g \qquad \frac{}{\to (\{\langle \bullet, \mathbf{1}\rangle\}, \emptyset)}$$

$$(2n')^g \qquad \frac{\Phi(\mathbf{2n-1}) = Q \quad \phi(\mathit{erase}(Q)) = [a, \overset{\kappa}{\hookleftarrow}]}{(\Phi, \Psi) \to (\Phi \bigcup \{Q[a, \overset{\kappa}{\hookleftarrow}]\}, \Psi)}$$

$$((2n+1)')^g \qquad \frac{\Psi(\mathbf{2n}) = Q \quad \psi(\mathit{erase}(Q)) = [a, \overset{\kappa}{\hookleftarrow}]}{(\Phi, \Psi) \to (\Phi, \Psi \bigcup \{Q[a, \overset{\kappa}{\hookleftarrow}]\})}$$

$$(2n)^g_b \qquad \frac{\Phi(\mathbf{2n'}) = Q[a, \overset{i}{\hookleftarrow}] \quad \pi_{i+1}(Q) = 2m-1 \quad \Psi((\mathbf{2m-1})') = R'}{(\Phi, \Psi) \to (\Phi, \Psi \bigcup \{R'\langle a, \mathbf{2n}\rangle\})}$$

$$(2n)^g_f \qquad \frac{\Phi(\mathbf{2n'}) = Q[a, \overset{-}{\hookleftarrow}]}{(\Phi, \Psi) \to (\Phi, \Psi \bigcup \{\langle a, \mathbf{2n}\rangle\})}$$

$$(2n+1)^g_b \qquad \frac{\Psi((\mathbf{2n+1})') = Q[a, \overset{i}{\hookleftarrow}] \quad \pi_{i+1}(Q) = 2m \quad \Phi(\mathbf{2m'}) = R'}{(\Phi, \Psi) \to (\Phi \bigcup \{R'\langle a, \mathbf{2n+1}\rangle\}, \Psi)}$$

Figure 5: The Geometric Abstract Machine

position of ϕ or ψ, respectively. The *GAM* execution tags opponent's moves with step numbers, opening new copies when reaching a position that has already been visited. The *GAM* proceeds by constructing such tagged positions, which we call dynamic positions.

Formally, given the same alphabet of moves as in section 3, an even dynamic position is a sequence R of the form

$$\langle a_1, \mathbf{j_1}\rangle[a_2, \overset{\kappa_1}{\hookleftarrow}] \cdots \langle a_{2n-1}, \mathbf{j_n}\rangle[a_{2n}, \overset{\kappa_n}{\hookleftarrow}],$$

where the a's and the κ's are as for positions, and where the tags $j_1, \ldots, j_n \in \omega$ are distinct natural numbers. A move $\langle a, \mathbf{j}\rangle$ is called a dynamic opponent's move, or a copy of opponent's move. An odd dynamic position

$$Q = \langle a_1, \mathbf{j_1}\rangle[a_2, \overset{\kappa_1}{\hookleftarrow}] \cdots \langle a_{2n-1}, \mathbf{j_n}\rangle$$

is defined in the same way. Any dynamic position P can be mapped to a position $\mathit{erase}(P)$ obtained by erasing the tags. We write (for $l \leq n$):

$$\pi_{l+1}(\langle a_1, \mathbf{j_1}\rangle \cdots \langle a_{2n-1}, \mathbf{j_n}\rangle) = j_{n-l}.$$

Thus, $\pi_{l+1}(P)$ is the $(l+1)$-th tag of P (starting from the right).

A state of the *GAM* is a pair (Φ, Ψ), where Φ and Ψ are sets of dynamic positions. More precisely, each of the even positions of Φ has the form $R = R_1\langle a, \mathbf{2j-1}\rangle[b, \overset{\kappa}{\hookleftarrow}]$, and the function which maps R to $2j-1$ is injective. We write $R = \Phi(\mathbf{2j'})$ and $R_1\langle a, \mathbf{2j-1}\rangle = \Phi(\mathbf{2j-1})$. The same property holds for Ψ (with even numbers), and we define $\Psi(\mathbf{2j'+1})$ and $\Psi(\mathbf{2j})$ similarly.

Figure 5 presents the rules of the *GAM*. At each step ν ($\nu = n$ or n'), the

GAM extends a dynamic position of $\Phi \cup \Psi$ with a player's move or a dynamic opponent's move, and adds the extended position P to $\Phi \cup \Psi$; P and ν are called the *current position* and the *current step number*, respectively.

The computation engine is concentrated in the rules $(n)_b^g$. The index i is used to retrieve the tag $2m-1$ or $2m$ of the copy of the opponent's move to which $[a, \overset{i}{\hookleftarrow}]$ is bound. This tag is in turn paired with (or points to) the current position R' of the machine at step $(2m-1)'$ or $2m'$, and R', extended with a copy of a, becomes the current position.

The two informations a and i in the move $[a, \overset{i}{\hookleftarrow}]$ give rise to two combined sorts of interaction, which we call *tree interaction* and *pointer interaction*, respectively.

– a allows us to choose opponent's move a: the determinism of ϕ at odd positions resolves the non-determinism of ψ at even positions, and conversely;

– i allows us to determine which odd dynamic position the next opponent's move should extend (in ψ or ϕ).

The example given in section 3 illustrates pointer interaction only (the strategy and the multistrategy are each reduced to a single position). An example of tree interaction is given in section 5.

The following invariant holds: if $\langle a, \mathbf{j} \rangle$ appears in a dynamic position r of Φ or Ψ, then all the opponent's moves $\langle a', \mathbf{j}' \rangle$ occurring before $\langle a, \mathbf{j} \rangle$ in r are such that $j' < j$. Hence the successive tags determine a traversal of $\Phi \cup \Psi$, and a node cannot be visited before his descendants.

Here is the final state of the execution of $(u(\lambda x.u(\lambda y.x))[u \leftarrow (\lambda r.r(r(z)))]$ by the *GAM*.

$$\Phi \quad \begin{array}{c}\boxed{(}\\ \langle \bullet, \mathbf{1} \rangle\end{array} [u, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} \begin{array}{c}\boxed{(\lambda x}\\ \langle 1, \mathbf{3} \rangle\end{array} [u, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{c}\boxed{(\lambda y} \ \boxed{x}\\ \langle 1, \mathbf{5} \rangle [1, \overset{1}{\hookleftarrow}]\end{array} \right. \\ \begin{array}{c}\boxed{(\lambda x}\\ \langle 1, \mathbf{7} \rangle\end{array} [u, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{c}\boxed{(\lambda y} \ \boxed{x}\\ \langle 1, \mathbf{9} \rangle [1, \overset{1}{\hookleftarrow}]\end{array} \right. \end{array} \right.$$

$$\Psi \quad \left\{ \begin{array}{l} \begin{array}{c}\boxed{(\lambda r}\\ \langle u, \mathbf{2} \rangle\end{array} [1, \overset{0}{\hookleftarrow}] \left\{ \begin{array}{c}\boxed{(}\\ \langle 1, \mathbf{6} \rangle\end{array} [1, \overset{1}{\hookleftarrow}] \left\{ \begin{array}{c}\boxed{(}\\ \langle 1, \mathbf{10} \rangle\end{array} [z, \overset{-}{\hookleftarrow}] \right. \right. \\ \begin{array}{c}\boxed{(\lambda r}\\ \langle u, \mathbf{4} \rangle\end{array} [1, \overset{0}{\hookleftarrow}] \\ \begin{array}{c}\boxed{(\lambda r}\\ \langle u, \mathbf{8} \rangle\end{array} [1, \overset{0}{\hookleftarrow}] \end{array} \right.$$

Note the importance of the copying information contained in the tags: for example, the move $10'$ (right after $\langle 1, \mathbf{9} \rangle$) points to 7, not to 3, and this is precisely what gets us out of a loop.

The information contained in the tags allows us to reconstruct the *SAM* environments. As an illustration, we show how to obtain the environment e_8 of figure 4 starting from $\Phi(\mathbf{9})$ in the above example. The environment items of e_8 are reconstructed using the pointer information contained in the non-initial dynamic opponent's moves of $\Phi(\mathbf{9}) = \langle \bullet, \mathbf{1} \rangle [u, \overset{-}{\hookleftarrow}] \langle 1, \mathbf{7} \rangle [u, \overset{-}{\hookleftarrow}]$. The first element is a closure $r[e]$ where $r = \mathit{erase}(\Psi(\mathbf{9}')) = u[1, \overset{0}{\hookleftarrow}]$ and where $e = e_7$ is the environment reconstructed from $\Psi(\mathbf{8})$. The second element is constructed similarly from $\Phi(\mathbf{7})$.

5 PCF trees

In this section, we consider a variation over the syntax of section 2 with constants of base type and case statements, and we show how to encode it into abstract Böhm trees, without any need to extend the target language (except for the alphabet of moves). The syntax is given by:

$$\begin{array}{lcl} M & ::= & (\lambda \vec{x}.W) \\ W & ::= & v \;\Big|\; \mathit{case}\ y\vec{M}\ [v_1 \rightarrow (W_1), \ldots, v_k \rightarrow (W_k)] \\ v & ::= & tt \;\Big|\; \mathit{ff} \;\Big|\; \underline{n} \quad (n \in \omega) \end{array}$$

The terms M are called finite PCF Böhm trees, or simply PCF trees, because they arise as the compact elements in a term model of the language PCF [12,1]. We compile PCF trees much like λ normal forms. The alphabet is $A = \{\bullet\} \cup \omega \cup \mathit{Var} \cup \{\underline{n} \mid n \in \omega\} \cup \{tt, \mathit{ff}\}$. We adopt the following tree representation for a PCF term M:

$$\overbrace{M_1 \quad \cdots \quad M_n \quad v_1 \rightarrow (W_1) \quad \cdots \quad v_k \rightarrow (W_k)}^{\lambda \vec{x}.\ \mathit{case}\ y}$$

We call $M_1, \ldots, M_n$ the argument branches, and $v_1 \rightarrow (W_1), \ldots, v_k \rightarrow (W_k)$ the case branches. The node "$\lambda \vec{x}.\ \mathit{case}\ y$" is split into the opponent's move $\bullet$ followed by a player's move $\underline{y}$ as in section 3. An occurrence of a basic constant v is compiled as $[v, \overset{i}{\hookleftarrow}]$, where i counts the number of case branches above it before the first argument branch is met. Finally, we replace the head opponent's moves $\bullet$ in the translations ϕ_i,ϕ'_j of M_i,(W_j) by i,v_j, respectively:

$$\overbrace{[1 \leftarrow \phi_1] \ \cdots \ [n \leftarrow \phi_n] \ [v_1 \leftarrow \phi'_1] \ \cdots \ [v_k \leftarrow \phi'_k]}^{\bullet \underline{y}}$$

The formal translation is as follows:

$$
\begin{aligned}
(\lambda \vec{x}.W)^{m;\sigma} &= \{\bullet r \mid r \in W^{m;\vec{x}\bullet\sigma}\} \\
v^{m;\sigma} &= [v, \overset{m}{\hookleftarrow}] \\
(case\ y\vec{M}\ [v_1 \rightarrow (W_1) \cdots v_k \rightarrow (W_k)])^{m;\sigma} &= \begin{cases} \{y^\sigma\} \\ \bigcup_{i=1\cdots n}\{y^\sigma ir \mid \bullet r \in M_i^{0;\sigma}\} \\ \bigcup_{j=1\cdots k}\{y^\sigma v_j r \mid r \in (W_j)^{(m+1);\sigma}\} \end{cases}
\end{aligned}
$$

where y^σ is as in section 3. The compilation of a term M is defined as $M^{0;nil}$. As an illustration, we compile $(case\ f(tt)\ [ff \rightarrow tt])[f \leftarrow (\lambda x.case\ x\ [tt \rightarrow ff, ff \rightarrow tt])]$ and we exhibit the final state of its GAM execution:

Strategy

$$
\begin{array}{c}\boxed{(}\\ \bullet\end{array}\begin{array}{c}\boxed{case}\\ {[f, \bar{\hookleftarrow}]}\end{array}\begin{cases} \begin{array}{c}\boxed{(}\\ 1\end{array}\ [tt, \overset{0}{\hookleftarrow}] \\ ff[tt, \overset{1}{\hookleftarrow}] \end{cases}
$$

Dynamic strategy

$$
\begin{array}{c}\boxed{(}\\ \langle \bullet, \mathbf{1}\rangle\end{array}\begin{array}{c}\boxed{case}\\ {[f, \bar{\hookleftarrow}]}\end{array}\begin{cases} \begin{array}{c}\boxed{(}\\ \langle 1, \mathbf{3}\rangle\end{array}\ [tt, \overset{0}{\hookleftarrow}] \\ \langle ff, \mathbf{5}\rangle[tt, \overset{1}{\hookleftarrow}] \end{cases}
$$

Multistrategy

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

Dynamic multistrategy

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

6 Executing arbitrary λ-terms

In this section, we successively remove the normal form restriction and the η-long restriction. We first describe how to compile an arbitrary η-long λ-term, and how to compute it. The basic idea, which is folklore, is to replace any redex $(\lambda x.M)N$ by a term $(*\ N)[* \leftarrow (\lambda x.M)]$, where $*$ is a special variable name. The alphabet is now $\{\bullet, *\} \cup \omega \cup Var$. To compile non-normal forms, we proceed in two phases, as in section 2. The first phase is unchanged. In the second phase, there are two cases. If $M = (\lambda \vec{x}.yM_1 \cdots M_n)$, then we proceed as in section 2. If $M = (\lambda \vec{x}.(\lambda y.M_0)M_1 \cdots M_n)$, let $\phi_0, \phi_1, \ldots \phi_n$ be the translations of $(\lambda y.M_0), M_1, \ldots M_n$, respectively. Then M is translated to:

$$
\overbrace{[1 \leftarrow \phi_1] \quad \cdots \quad [n \leftarrow \phi_n] \quad [* \leftarrow \phi_0]}^{\bullet[*, \bar{\hookleftarrow}]}
$$

How should we adjust the GAM or the SAM in order to execute a tree χ obtained as the target of this compilation? Here, there is no alternation between

$$\begin{array}{c}\boxed{(}\\ \langle\bullet,\mathbf{1}\rangle\end{array}\ [*,\overset{-}{\hookleftarrow}]\begin{cases} \begin{array}{l}\boxed{(\lambda z}\ \ \boxed{z}\\ \langle 1,\mathbf{6}\rangle[1,\overset{0}{\hookleftarrow}]\end{array} \\ \begin{array}{c}\boxed{(\lambda x}\\ \langle *,\mathbf{2}\rangle\end{array}\ [*,\overset{-}{\hookleftarrow}]\begin{cases} \begin{array}{l}\boxed{(\lambda y}\ \ \boxed{x}\\ \langle 1,\mathbf{5}\rangle[1,\overset{1}{\hookleftarrow}]\end{array}\begin{cases}\begin{array}{l}\boxed{(}\ \ \boxed{y}\\ \langle 1,\mathbf{7}\rangle[1,\overset{1}{\hookleftarrow}]\end{array}\end{cases} \\ \begin{array}{c}\boxed{(\lambda f}\\ \langle *,\mathbf{3}\rangle\end{array}\ [*,\overset{-}{\hookleftarrow}]\begin{cases}\begin{array}{c}\boxed{(\lambda z}\\ 1\end{array}\ [u,\overset{-}{\hookleftarrow}] \\ \begin{array}{l}\boxed{(\lambda x}\ \ \boxed{f}\\ \langle *,\mathbf{4}\rangle[1,\overset{1}{\hookleftarrow}]\end{array}\begin{cases}\begin{array}{c}\boxed{(}\\ \langle 1,\mathbf{8}\rangle\end{array}\ [t,\hookleftarrow]\end{cases}\end{cases}\end{cases}\end{cases}$$

Figure 6: Final state of the execution of $((\lambda x.(\lambda f.(\lambda x.f(t))(\lambda z.u))(\lambda y.x(y)))(\lambda z.z))$

a strategy and a multistrategy. Instead, χ is a "strategy" in which the $*$'s signal (possibly nested) compositions of strategies. We therefore need to replace the rule $(2n)_f$ by a new rule, which we shall call the $*$ rule. In the framework of the *GAM*, the $*$ rule can be described as follows: when the current position ends with $[*,\overset{-}{\hookleftarrow}]$, the machine moves to (a copy) of the opponent's move $*$ occurring immediately below. We illustrate this in figure 6.

We now address the second restriction. The basic idea is that the machine can cope with situations where a head variable has not enough arguments by making (or rather mimicking) an η-expansion dynamically, as the need arises. We illustrate this with $(z(\lambda xy.y)vw)[z \leftarrow (\lambda u.u)]$ (we should actually write $(z(\lambda xy.y)vw)[z \leftarrow (\lambda u.u), v \leftarrow v, w \leftarrow w]$, according to our conventions):

Strategy	Multistrategy
$\begin{array}{c}\boxed{(}\\ \bullet\end{array}\ [z,\overset{-}{\hookleftarrow}]\begin{cases}\begin{array}{l}\boxed{(\lambda xy}\ \ \boxed{y}\\ \quad 1\ \ [2,\overset{0}{\hookleftarrow}]\end{array}\\ \begin{array}{c}\boxed{(}\\ 2\end{array}\ [v,\overset{-}{\hookleftarrow}]\\ \begin{array}{c}\boxed{(}\\ 2\end{array}\ [w,\overset{-}{\hookleftarrow}]\end{cases}$	$\begin{array}{l}\boxed{(\lambda u}\ \ \boxed{u}\\ \quad\bullet\ [1,\overset{0}{\hookleftarrow}]\end{array}$

The *SAM* is blocked after step 4, because u has no argument, and a fortiori no second argument. If we had performed the η-expansion transforming $(\lambda u.u)$ into $(\lambda u u_2 u_3.u(u_2)(u_3))$, then we would have had no difficulty. The variables u_2 and u_3, which we call expansion variables, are compiled to moves of the form $[i,\overset{1}{\hookleftarrow}]$. We have $i = 2$ for u_2 and $i = 3$ for u_3. We can let the machine

$$
\begin{array}{lll}
(2') & \bullet[z,\overset{-}{\hookleftarrow}] & [\;] \\
(2) & z & (\bullet[z,\overset{-}{\hookleftarrow}])[\;] = e_1 \\
(3') & z[1,\overset{0}{\hookleftarrow}] & e_1 \\
(3) & \bullet[z,\overset{-}{\hookleftarrow}]1 & (z[1,\overset{0}{\hookleftarrow}])[e_1] = e_2 \\
(4') & \bullet[z,\overset{-}{\hookleftarrow}]1[2,\overset{0}{\hookleftarrow}] & e_2 \\
(4) & z[1,\overset{0}{\hookleftarrow}]2 & (\bullet[z,\overset{-}{\hookleftarrow}]1[2,\overset{0}{\hookleftarrow}])[e_2]\bullet e_1 = e_3 \\
(5') & z[1,\overset{0}{\hookleftarrow}]2[3,\overset{1}{\hookleftarrow}] & e_3 \\
(5) & \bullet[z,\overset{-}{\hookleftarrow}]3 & (z[1,\overset{0}{\hookleftarrow}]2[3,\overset{1}{\hookleftarrow}])[e_3] = e_4 \\
(6') & \bullet[z,\overset{-}{\hookleftarrow}]3[w,\overset{-}{\hookleftarrow}] & e_4
\end{array}
$$

Figure 7: The execution of $(z(\lambda xy.y)vw)[z \leftarrow (\lambda u.u)]$

think that u has a second argument by stating that if $\psi(ra)$ is undefined (with $r \in \psi$), then the run continues, and a move $[i,\overset{1}{\hookleftarrow}]$ (for an appropriate i) is played. The full execution is shown in figure 7.

In order to compute the appropriate i in general, we reformulate the compilation of λ-terms. We tag every player's move corresponding to the head variable of a subterm $(\lambda\vec{x}.y\vec{M})$ with a pair (m,n) of two natural numbers, which count the length of the two vectors $\vec{x}$ and $\vec{M}$. The modified compilation is as follows:

$$(\lambda x_1 \cdots x_m.yM_1 \cdots M_n)^\sigma = \{\bullet(y^{\vec{x}\bullet\sigma})^{m,n}\} \bigcup_{i=1\cdots n}\{\bullet(y^{\vec{x}\bullet\sigma})^{m,n}ir \mid \bullet r \in (M_i)^{\vec{x}\bullet\sigma}\}$$

$$(\lambda x_1 \cdots x_m.(\lambda y.P)M_1 \cdots M_n)^\sigma = \begin{cases} \{\bullet[*,\overset{-}{\hookleftarrow}]^{m,n}\} \\ \bigcup\{\bullet[*,\overset{-}{\hookleftarrow}]^{m,n} * r \mid \bullet r \in (\lambda y.P)^{\vec{x}\bullet\sigma}\} \\ \bigcup_{i=1\cdots n}\{\bullet[*,\overset{-}{\hookleftarrow}]^{m,n}ir \mid \bullet r \in (M_i)^{\vec{x}\bullet\sigma}\} \end{cases}$$

Cumulating the two adjustments for non-normal forms and for non η-long terms (and hence for arbitrary untyped λ-terms), we arrive at an abstract machine which we call the SAM^* (figure 8). The rule $(n')^*$ is just an obvious reformulation of the rules $((2n+2)')$ and $((2n+1)')$ of the SAM. The rule $(n')^*_\eta$ implements the dynamic η-expansions, and the last rule is the $*$ rule dealing with non-normal forms.

7 Back to the KAM

We can walk the reverse way from the game-theoretic machines to the KAM and express some of the implementation ideas developed in the earlier sections

$$(1)^* \quad \frac{}{\rightarrow \bullet[\,]}$$

$$(n')^* \quad \frac{\chi(q) = [a, \overset{\kappa}{\hookleftarrow}]^{m,n}}{q[e] \rightarrow (q[a, \overset{\kappa}{\hookleftarrow}]^{m,n})[e]}$$

$$(n')^*_\eta \quad \frac{q = q_1[a_1, \overset{\kappa_1}{\hookleftarrow}]^{m_1,n_1} a \quad n_1 < a}{q[e] \rightarrow (q[a - n_1 + m_1, \overset{1}{\hookleftarrow}]^{0,0})[e]}$$

$$(n)^*_b \quad \frac{r = q[a, \overset{i}{\hookleftarrow}]^{m,n} \text{ and } \pi_{i+1}(e) = r'[e']}{r[e] \rightarrow (r'a)[(r[e])\bullet e']}$$

$$(n)^*_* \quad \frac{r = q[*, \overset{-}{\hookleftarrow}]^{m,n}}{r[e] \rightarrow (r*)[r[e]\bullet e]}$$

Figure 8: The SAM^*

in the language of the λ-calculus or of PCF trees. In this section, we present a Stack-Free KAM (or KAM^{PCF}) for PCF trees, and a full Stack-Free KAM (or KAM^*) for the untyped λ-calculus.

We address PCF trees first. We adopt a mild change of syntax with respect to section 5 (reflecting a small optimisation described below), and write $case\ y\vec{M}\ [\vec{v} \rightarrow \vec{W}]$ instead of $case\ y\vec{M}\ [\vec{v} \rightarrow (\vec{W})]$. We define a syntax of PCF environments as follows:

$$\begin{array}{l} e ::= nil \;\big|\; B\bullet e \\ B ::= \langle \vec{x} \leftarrow \vec{M}; \vec{v} \rightarrow \vec{V} \rangle[e]. \end{array}$$

The rules of the KAM^{PCF} are given in figure 9. There is no counterpart for the encoding of v as v^i given in section 5. This corresponds to an optimisation consisting in changing $r \in (W_j)^{(m+1);\sigma}$ into $r \in W_j^\sigma$ in the compilation of section 5 and in removing the m's altogether — this allows not to "cons" the empty closure in rule $(K)_v^{\text{PCF}}$.

We now adapt the encodings of section 6, using a syntax of terms rather than a syntax of positions. We recall that in the notation of section 6, when we write ^{m,n} to decorate a head variable, m records the number of simultaneously abstracted λ's, while n records the number of arguments. In a term-based syntax, the second information is visible, therefore we care only for the first information, which we now record on λ abstractions and then on environment components. This leads to the following syntax:

$$\begin{array}{ll} M ::= (\lambda^n.W) \qquad & e ::= nil \;\big|\; B\bullet e \\ W ::= [j, \overset{i}{\hookleftarrow}]\vec{M} \qquad & B ::= \langle \vec{M} \rangle^n[e] \end{array}$$

$$(K)^{\text{PCF}}\ (case\ x\vec{M}\ [\vec{v} \rightarrow \vec{V}])[e] \rightarrow W[\langle \vec{z} \leftarrow \vec{M}; \vec{v} \rightarrow \vec{V}\rangle[e]{\bullet}e_i]$$

$$\text{where} \begin{cases} e = B_0{\bullet}\cdots{\bullet}B_n & B_i = \langle \vec{x_i} \leftarrow \vec{N_i}; \vec{v_i} \rightarrow \vec{V_i}\rangle[e_i] \\ x = x_{ij} & N_{ij} = (\lambda\vec{z}.W) \end{cases}$$

$$(K)_v^{\text{PCF}}\ v[e] \rightarrow V_{0j}[e_0]$$

$$\text{where} \begin{cases} e = B_0{\bullet}e' & B_0 = \langle \vec{x_0} \leftarrow \vec{N_0}; \vec{v_0} \rightarrow \vec{V_0}\rangle[e_0] \\ v = v_{0j} & \end{cases}$$

Figure 9: The Stack-Free *KAM* for PCF

Transposing the rules of the SAM^*, we arrive at the KAM^* (figure 10). In a binding $\langle\vec{N}\rangle^n$, n stands for the number of variables actually bound and is not necessarily the size of $\vec{N}$. When $n < |\vec{N}|$, the remaining N's may be considered as an implicit stack. The converse situation happens in rule $(K)^*_\eta$, where necessarily $n_i > |\vec{N_i}|$. In order to determine where a variable of rank $j > |\vec{N_i}|$ is bound, we investigate $\langle\vec{M_{i0}}\rangle^{n_{i0}}[e_{i0}]$. If $j - |\vec{N_i}| + n_{i0} > |\vec{M_{i_0}}|$, then the exploration will continue in e_{i0}, and so on.

8 Computation as interaction

In this section, we relate the *SAM* to (an abstract versions of) Danos-Regnier's Pointer Abstract Machine (*PAM*) where the environment is implicit [7], and to a machine called View Abstract Machine which formalises directly the notion of interaction between two persons playing moves (not positions) in turn (as in [3]). The two machines can be recovered from an analysis of the information redundancy in the *SAM*.

We observe that the environment component of the *SAM* states can be retrieved from the previous states of the execution: in rule $(n)'$, the environment e is just a copy of the environment component of the state at step $n-1$; in rules $(n)_b$ or $(2n)_f$, the environment becomes $r[e]{\bullet}e'$ or $r[e]$. Its head component $r[e]$ is just a copy of the state of the machine at step n'; the tail component e' is not directly accessible from the *SAM* state at step n', since it comes from $\pi_{i+1}(e) = r'[e']$, but by construction of the environments along the execution, e' was the environment component of the *SAM* state at some previous step. Thus, we can turn the *SAM* – a "history-free" machine, i.e. where any computation step depends on the previous state only – into a machine dealing only with positions, but history dependent.

$$(K)^* \;\; ([j, \stackrel{i}{\hookleftarrow}]\vec{M})[e] \rightarrow W[\langle \vec{M}\rangle^n[e]{\bullet}e_i]$$

$$\text{where} \begin{cases} e = B_0{\bullet}\cdots{\bullet}B_n \;\; B_i = \langle \vec{N_i}\rangle^{n_i}[e_i] \\ j \leq |\vec{N_i}| \qquad\quad N_{ij} = (\lambda^n.W) \end{cases}$$

$$(K)^*_\eta \;\; ([j, \stackrel{i}{\hookleftarrow}]\vec{M})[e] \rightarrow [j - |\vec{N_i}| + n_{i0}, \stackrel{1}{\hookleftarrow}][\langle \vec{M}\rangle^0[e]{\bullet}e_i]$$

$$\text{where} \begin{cases} e = B_0{\bullet}\cdots{\bullet}B_n \;\; B_i = \langle \vec{N_i}\rangle^{n_i}[e_i] \\ j > |\vec{N_i}| \qquad\quad e_i = B_{i0}{\bullet}\cdots{\bullet}B_{in_i} \;\; B_{i0} = \langle \vec{M_{i0}}\rangle^{n_{i0}}[e_{i0}] \end{cases}$$

$$(K)^*_* \;\; ((\lambda^n.W)\vec{M})[e] \rightarrow W[\langle \vec{M}\rangle^n[e]{\bullet}e]$$

Figure 10: The full Stack-Free *KAM*

In the *PAM*, the environments are implicit. The states of the *PAM* are $\bullet$ (at step 1), even positions (at steps n'), or tagged odd positions of the form $\langle q, m'\rangle$, called *pointing positions* (at steps n, $n \geq 2$). There are no tags on positions of length 1. A *history* Γ is a sequence of states:

$$\Gamma ::= \bullet \;\Big|\; \Gamma a \;\Big|\; \Gamma(q[a, \stackrel{\kappa}{\hookleftarrow}]) \;\Big|\; \Gamma\langle ra, m'\rangle$$

We denote the successive elements of Γ by $\Gamma{\bullet}1, \Gamma{\bullet}2', \Gamma{\bullet}2, \ldots, \Gamma{\bullet}\nu$ (where $\nu = n$ or n'). If $\Gamma{\bullet}n = \langle q, m'\rangle$, we set $jump_\Gamma(n) = m - 1$. The rules of the *PAM* (figure 11) follow those of the *SAM*, forgetting the environment component of each *SAM* state, but recording the history. The main difference is in rule $(n)^p_b$: the pointer m', which is precisely $(jump_\Gamma(n)+1)'$, is recursively computed by using *jump* on smaller values.

We next define a compilation that maps a *PAM* history Γ to a *SAM* state $\lceil\Gamma\rceil^{ps} = p_{\Gamma\bullet\nu}[e_\Gamma(\nu)]$, where:

$$\begin{array}{lll} p_\bullet = \bullet & p_a = a & e_\Gamma(n') = e_\Gamma(n-1) \\ p_{q[a,\stackrel{\kappa}{\hookleftarrow}]} = q[a, \stackrel{\kappa}{\hookleftarrow}] & p_{\langle ra, m'\rangle} = ra & e_\Gamma(n) = (\Gamma{\bullet}n')[e_\Gamma(n')]{\bullet}e_\Gamma(jump_\Gamma(n)) \end{array}$$

Proposition 8.1 *The SAM simulates the PAM: if* $\Gamma \rightarrow \Gamma_1$, *then* $\lceil\Gamma\rceil^{ps}\!\rightarrow \lceil\Gamma_1\rceil^{ps}$.

PROOF HINT. We consider only rule $(n)^p_b$. We have $\Gamma{\bullet}n' = q[a, \stackrel{i}{\hookleftarrow}]$, $jump^i_\Gamma(n-1) = m$, $\Gamma{\bullet}m' = r'$, and $\Gamma_1 = \Gamma\langle r'a, m'\rangle$. The compilations of Γ and Γ_1 are as follows: $\lceil\Gamma\rceil^{ps} = (q[a, \stackrel{i}{\hookleftarrow}])[e_\Gamma(n')]$, and

$$
\begin{array}{ll}
(1)^p & \dfrac{}{\to \bullet} \\[2ex]
(2n')^p & \dfrac{\pi(\Gamma\bullet(2n-1)) = q \quad \phi(q) = [a, \stackrel{\kappa}{\hookleftarrow}]}{\Gamma \to \Gamma(q[a, \stackrel{\kappa}{\hookleftarrow}])} \\[2ex]
((2n+1)')^p & \dfrac{\pi(\Gamma\bullet 2n) = q \quad \psi(q) = [a, \stackrel{\kappa}{\hookleftarrow}]}{\Gamma \to \Gamma(q[a, \stackrel{\kappa}{\hookleftarrow}])} \\[2ex]
(n)^p_b & \dfrac{\Gamma\bullet n' = q[a, \stackrel{i}{\hookleftarrow}] \quad jump^i_\Gamma(n-1) = m \quad \Gamma\bullet m' = r'}{\Gamma \to \Gamma\langle r'a, m'\rangle} \\[2ex]
(2n)^p_f & \dfrac{\Gamma\bullet 2n' = q[a, \stackrel{-}{\hookleftarrow}]}{\Gamma \to \Gamma\ a}
\end{array}
$$

Figure 11: The Pointer Abstract Machine

$$
\begin{aligned}
\lceil\Gamma_1\rceil^{ps} &= (r'a)[e_{\Gamma_1}(n)] \\
&= (r'a)[(q[a, \stackrel{i}{\hookleftarrow}])[e_{\Gamma_1}(n')]\bullet e_{\Gamma_1}(jump_{\Gamma_1}(n))] \\
&= (r'a)[\lceil\Gamma\rceil^{ps}\bullet e_{\Gamma_1}(jump_{\Gamma_1}(n))] \\
&= (r'a)[\lceil\Gamma\rceil^{ps}\bullet e_\Gamma(m')]
\end{aligned}
$$

since, by definition of $jump_\Gamma$, we have

$$e_{\Gamma_1}(jump_{\Gamma_1}(n)) = e_{\Gamma_1}(m-1) = e_{\Gamma_1}(m') = e_\Gamma(m').$$

To perform the *SAM* step n on $\lceil\Gamma\rceil^{ps}$, we unfold the definition of $e_\Gamma(n-1)$:

$$
\begin{aligned}
\pi_{i+1}(e_\Gamma(n')) &= \pi_{i+1}(e_\Gamma(n-1)) \\
&= (\Gamma\bullet jump^i_\Gamma(n-1)')[e_\Gamma(jump^i_\Gamma(n-1)')] \\
&= (\Gamma\bullet m')[e_\Gamma(m')]
\end{aligned}
$$

hence $\lceil\Gamma\rceil^{ps} \to \lceil\Gamma_1\rceil^{ps}$. □

Note that the complexity of the machine is the same for the *SAM* and for the *PAM* ($jump^i$ and π_{i+1} are both in time proportional to i). However, the *PAM* handles a linear sequence of "positions". This suggests to implement weak head reduction for Böhm trees by pushing pointing positions (in practice: physical pointers on terms) on a stack rather than by allocating environments in the heap as usual.

Next, we recover the *VAM* from an analysis of information redundancy in the *PAM*. Observing the rules of the *PAM*, we remark that the prefixes q at steps n' and r' at steps n can be retrieved from the positions in previous steps: in rule $(n)^p_b$, the information $\langle a, m'\rangle$ is enough, because r' is known from the history at step m'; similarly, in rule $(n')^p$, q is known from the history at step

$$(1)^v \quad \frac{}{\rightarrow \bullet}$$

$$(n)^v_b \quad \frac{\Gamma{\bullet}n' = [a, \overset{i}{\hookleftarrow}] \quad jump^i_\Gamma(n-1) = m}{\Gamma \rightarrow \Gamma\langle a, m'\rangle} \qquad (2n')^v \quad \frac{\phi(view_\Gamma(2n-1)) = [a, \overset{\kappa}{\hookleftarrow}]}{\Gamma \rightarrow \Gamma[a, \overset{\kappa}{\hookleftarrow}]}$$

$$(2n)^v_f \quad \frac{\Gamma{\bullet}2n' = [a, \overset{-}{\hookleftarrow}]}{\Gamma \rightarrow \Gamma\, a} \qquad ((2n+1)')^v \quad \frac{\psi(view_\Gamma(2n)) = [a, \overset{\kappa}{\hookleftarrow}]}{\Gamma \rightarrow \Gamma[a, \overset{\kappa}{\hookleftarrow}]}$$

Figure 12: The View Abstract Machine

$n-1$ and the information $[a, \overset{\kappa}{\hookleftarrow}]$ is enough. This leads to a simplified syntax of histories:

$$\Gamma ::= \bullet \,\Big|\, \Gamma a \,\Big|\, \Gamma[a, \overset{\kappa}{\hookleftarrow}] \,\Big|\, \Gamma\langle a, m'\rangle$$

But now, the information q in rule $(n')^p$ is no longer computable in one step. The move $\Gamma{\bullet}(n-1)$ only carries the last move of $q = r'a$. The rest r' is the position implicit at step $(n-1)'$, which can be computed recursively via a function *view* defined as follows:

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

This leads to the machine described in figure 12. Finally, we define a compilation from *VAM* histories to *PAM* histories:

$$\lceil\bullet\rceil^{vp} = \bullet \qquad \lceil\Gamma[a, \overset{\kappa}{\hookleftarrow}]\rceil^{vp} = \lceil\Gamma\rceil^{vp}((view_\Gamma)[a, \overset{\kappa}{\hookleftarrow}])$$
$$\lceil\Gamma a\rceil^{vp} = \lceil\Gamma\rceil^{vp} a \qquad \lceil\Gamma\langle a, m'\rangle\rceil^{vp} = \lceil\Gamma\rceil^{vp}\langle(view_{\Gamma\langle a,m'\rangle}), m'\rangle$$

where $view_{\Gamma\bullet 1\cdots\Gamma\bullet n} = view_\Gamma(n)$.

Proposition 8.2 *The PAM simulates the VAM: if* $\Gamma \rightarrow \Gamma_1$*, then* $\lceil\Gamma\rceil^{vp} \rightarrow \lceil\Gamma_1\rceil^{vp}$*.*

At each step, the *VAM* has an additional computational cost proportional to the view's length, hence the *VAM* is slower than the *PAM* and the *SAM*.

9 Conclusion

The *SAM* and the *PAM* are the analogues of machines that have been formulated in the framework of the λ-calculus, while the *GAM* and the *VAM* seem more specific of the language of games. Our framework allows us to express all these machines in a uniform notation and to relate them by mutual encodings, and hence enables us to transfer properties established in game semantics. For example, the *VAM* terminates[3,5], hence the *PAM*, the *GAM*, and the stack-free machines *SAM*, *KAM*, and KAM^{PCF} terminate. Types are not involved in the proof of termination. It can be shown [5] they only play a role in guaranteeing that the final states correspond to (an abstract form of) head normal forms.

The *SAM* or the *PAM* are not merely paraphrasing abstract machines for the λ-calculus. They can serve to execute other kinds of code. Here we have dealt with PCF trees. Other examples are $\lambda\mu$-calculus [10,5] and cut elimination in Novikoff's logic [3,5].

References

1. S. Abramsky, R. Jagadeesan, and P. Malacaria, Full abstraction for PCF, available by ftp at theory.doc.ic.ac.uk/papers (1995).
2. A. Blass, Degrees of indeterminacy for games, Fundamenta Mathematicae **77**, 151–166 (1972).
3. T. Coquand, A semantics of evidence for classical arithmetic, J. of Symb. Logic **60**, 325–337 (1995).
4. P.-L. Curien, An abstract framework for environment machines, TCS **82**, 389–402 (1991).
5. P.-L. Curien, Abstract Böhm trees, to appear in Math. Structures in Computer Science (1998).
6. V. Danos, H. Herbelin, L. Regnier, Game semantics and abstract machines, in proceedings of LICS 96, 394–405 (1996).
7. V. Danos, L. Regnier, Machina ex deo, ou encore quelque chose à dire sur la machine de Krivine, unpublished (1990).
8. W. Felscher, Dialogues as a foundation of intuitionistic logic, Handbook of Phil. Logic **3**, 341–372 (1986).
9. H. Herbelin, Séquents qu'on calcule, Thèse de Doctorat, Université Paris VII (1995).
10. H. Herbelin, Games and Weak-Head Reduction for Classical PCF, Proceedings of TLCA 97, LNCS **1210**, 214–230 (1997).
11. G. Huet, An analysis of Böhm's theorem, TCS **121**, 145–167 (1993).
12. M. Hyland, L. Ong, On full abstraction for PCF, available by ftp at ftp.comlab.ox.ac.uk/pub/ Documents/techpapers/Luke.Ong (1994).
13. F. Lamarche, Games Semantics for Full Propositional Linear Logic, in Proceedings of LICS 95, 464–473 (1995).
14. H. Nickau, Hereditarily Sequential Functionals, Proceedings of LFCS, LNCS **813**, 253–254 (1994).
15. D. Scott, A type-theoretical alternative to ISWIM, CUCH, OWHY, TCS **121**, 411–440 (1993).