Math. Struct. in Comp. Science (1998), *vol.* 8, *pp.* 559–591. Printed in the United Kingdom
© 1998 Cambridge University Press

Abstract Böhm trees

PIERRE-LOUIS CURIEN

CNRS-ENS, LIENS, 45 Rue d'Ulm, 75230 Paris Cedex 05, France
Email: curien@dmi.ens.fr

Received 11 December 1996; revised 16 December 1997

We present a formalism of trees with pointers, called abstract Böhm trees, that provide a suitable abstract framework in which various cut-free proofs or normal terms of several λ-calculus based languages (including PCF and Parigot's $\lambda\mu$-calculus) can be faithfully encoded. A simple abstract machine called the View Abstract Machine (*VAM*) allows us to compute over abstract Böhm trees. The *VAM* is closely related to Coquand's interaction sequences and debates. The *VAM* execution over finite abstract Böhm trees always terminates. We next introduce an abstract notion of type that fits the purpose of guaranteeing that the *VAM* cannot go into deadlock, *i.e.*, that it always reaches a satisfactory final state. Typed abstract Böhm trees can be turned into a category – more naturally a 'multi-category' where the domains of arrows are sets of named objects or records. We then go from the abstract to the concrete by giving examples. Our sets of abstract (typed) Böhm trees are relative to an alphabet and a set of types. By instantiating these two parameter sets appropriately, we recover, successively: (η-long) typed Böhm trees; PCF trees as considered in the game models of Hyland–Ong or of Abramsky–Jagadeesan–Malacaria; a notion of classical Böhm tree due to Herbelin that provides a classical version of PCF trees in the style of $\lambda\mu$-calculus; and, finally, cut-free proofs in Novikov's infinitary propositional logic as investigated by Coquand. In a companion paper, we investigate the operational aspects of (untyped) Böhm trees in more depth.

1. Introduction

The origin of this work lies in a convergence between two lines of research:

— one originating from Lorenz-Lorenzen's game-theoretic interpretations of logical provability (Felscher 1986; Coquand 1995; Herbelin 1995),
— the other from the more recent game models of PCF (Hyland and Ong 1994; Abramsky *et al.* 1995; Nickau 1996), which provide syntax-free accounts of an interesting infinite term model for Scott's language PCF (Milner 1977; Plotkin 1977).

It turns out that cut-free proofs of various sequent calculi on the one hand, and Hyland and Ong's innocent strategies on the other, can all be expressed as trees with pointers, just as λ-terms (or more precisely Böhm trees). In Section 2, we develop a framework of abstract Böhm trees that can be instantiated to different calculi and languages, including those just mentioned. We introduce an abstract machine, called the View Abstract Machine, or *VAM*, that performs computations over abstract Böhm trees. It is an adaptation to

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

our general framework of Coquand's debates (Coquand 1995). Coquand has proved a 'no infinite chatting' property, which in our framework is a termination property: when executed on finite abstract Böhm trees, the *VAM* terminates. But the machine may stop without reaching a satisfactory final state. A typical example is provided by the execution of the (abstract Böhm tree version of) $\Delta\Delta$ (where $\Delta = \lambda x.xx$).

In Section 3, we define a notion of abstract type, and we show that correctly typed abstract Böhm trees are executed without deadlock, in other words that for such trees the *VAM* stops in a satisfactory final state that abstracts the λ-calculus notion of 'having reached a head normal form'. We define a strong reduction machine, by which we mean, in the terminology of the λ-calculus, reduction to full normal form, as opposed to reduction to head normal form. In the terminology of games, we mean computing the composition of two strategies, as opposed to computing the first player's move of the composition only.

In Section 4, we illustrate the wide coverage of abstract Böhm trees. We treat, successively, Böhm trees, PCF trees, classical Böhm trees and proofs in Novikov's infinitary propositional logic.

2. Untyped abstract Böhm trees

Throughout this paper, we assume a fixed set X whose elements are called abstract variable names. We want to define an abstract notion of tree with pointers, described as a set of branches, or positions.

Definition 2.1. (Position) We assume an alphabet A, and a special symbol $\bullet$. We assume that A, $\{\bullet\}$ and X are disjoint. We define opponent's moves, player's moves and positions as follows:

— An *initial opponent's move* is an element of $\{\bullet\} \cup X$ (see Definitions 2.2 and 2.3, respectively). An *opponent's move* is an element of $A \cup \{\bullet\} \cup X$. We let a range over $A \cup \{\bullet\} \cup X$; if $a \in A$, then a is called a non-initial opponent's move.

— A *player's move* is either a pair $[a, \overset{i}{\hookleftarrow}]$ where $a \in A$ and $i \in \omega$ (the set of natural numbers), or an element of X, written $[x, \overset{-}{\hookleftarrow}]$. We let α range over $A \cup X$ and κ range over $\omega \cup \{_\}$.

— An *even position* is a non-empty alternating sequence r of opponent's moves and player's moves, of the form:

$$a_1[\alpha_2, \overset{\kappa_1}{\hookleftarrow}] \cdots a_{2n-1}[\alpha_{2n}, \overset{\kappa_n}{\hookleftarrow}]$$

where $a_1 \in \{\bullet\} \cup X$ and $a_{2l-1} \in A$ for all $l > 1$ (whence the terminology of initial and non-inital opponent's moves). If $\kappa_l = _$, we say that $[\alpha_{2l}, \overset{\kappa_l}{\hookleftarrow}]$ occurs free. For every l such that $1 \leqslant l \leqslant n$ and $\kappa_l \neq _$, we make the following assumption:

$$\kappa_l < l. \tag{1}$$

We say that $[\alpha_{2l}, \overset{\kappa_l}{\hookleftarrow}]$ is bound to $a_{2(l-\kappa_l)-1}$. 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 an even position, an odd position or a suffix of a position.

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

Thus, as in de Bruijn's notation for λ-terms (Abadi *et al.* 1991), pointers are represented by numbers that count the number of opponent's moves between the pointing player's move and the pointed at opponent's move.

Definition 2.1 presents the kit out of which actual data can be constructed, much as cliques can be constructed out of coherence spaces, configurations out of event structures, and so on (Amadio and Curien 1997). Here the data are called strategies, or abstract Böhm trees.

Definition 2.2. (Strategy) A strategy, or *abstract Böhm tree*, is a tree consisting of a non-empty set ϕ of even positions of the form $\bullet p$ satisfying the following properties:

(a) ϕ is closed under even non-empty prefixes, and
(b) $q[\alpha_1, \overset{\kappa_1}{\hookleftarrow}], q[\alpha_2, \overset{\kappa_2}{\hookleftarrow}] \in \phi$ implies $[\alpha_1, \overset{\kappa_1}{\hookleftarrow}] = [\alpha_2, \overset{\kappa_2}{\hookleftarrow}]$.

Since ϕ is non-empty, it contains a unique position of the form $\bullet[\alpha, \overset{\kappa}{\hookleftarrow}]$, and $[\alpha, \overset{\kappa}{\hookleftarrow}]$ is called the *abstract head variable* of ϕ. If we do not insist on having Condition 2.1 (1), we say that ϕ is a *pseudo-strategy*. A (pseudo-) strategy is called name-free if it contains no occurrence of a player's move of the form $[x, \overset{-}{\hookleftarrow}]$.

Condition (b) says that, in a strategy, a 'query' (that is, a position of odd length), if answered, is uniquely so. Another way to put this is to say that ϕ is a tree of positions such that branching is only allowed under player's moves. By Condition (b), any strategy can be unambiguously presented as a partial function, also written ϕ, from odd positions to player's moves $[\alpha, \overset{\kappa}{\hookleftarrow}]$. We also need the related notion of multi-strategy.

Definition 2.3. (Multi-strategy) We define a renaming operation on strategies as follows:

$$[x \leftarrow \phi] = \{xr \mid \bullet r \in \phi\}$$

where $x \in X$. A multi-strategy ψ is a union $[x_1 \leftarrow \phi_1] \cup \cdots \cup [x_n \leftarrow \phi_n]$ of renamed strategies (with all the x_i distinct). We write ψ more suggestively as

$$\psi = [x_1 \leftarrow \phi_1, \ldots, x_n \leftarrow \phi_n].$$

We also write $\phi_i = \psi_{x_i}$, for all i.

Remark 1. In the presentation of a strategy ϕ as a partial function $q \mapsto \phi(q)$, the player's moves of q are actually irrelevant. The same remark applies to multi-strategies. We call *views* sequences of opponent's moves (starting with $\bullet$, or with x)[†]. Hence a strategy or a multi-strategy can be presented as a partial function from views to player's moves. We shall use this presentation below in the definition of the View Abstract Machine.

In Section 4 we shall see examples showing that abstract Böhm trees allow us to represent *terms* or *proofs* of various concrete calculi. This suggests that abstract Böhm trees in general are computational objects. We next show how to compute with abstract Böhm trees, or how to compose them. In Section 3, we shall present abstract typed Böhm trees as morphisms in a suitable (multi-) category. But just as the dynamics of λ-calculus is best understood in a type-free setting, our computation engine, called the View Abstract

[†] This terminology will become clear later in this section (see, in particular, Definition 2.5).

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

Machine, or *VAM* †, is best presented in an untyped setting. The *VAM* is the reformulation in our framework of Coquand's debate machinery (Coquand 1995).

The goal of a *VAM* execution is to compute the abstract head variable of a composition $\phi \circ \psi$ of

— a strategy ϕ whose free player's moves are $[x_1, \overleftarrow{\hookrightarrow}], \ldots, [x_n, \overleftarrow{\hookrightarrow}]$ with
— a multi-strategy $\psi = [x_1 \leftarrow \psi_{x_1}, \ldots, x_n \leftarrow \psi_{x_n}]$.

(Think of a substitution $M[\vec{x} \leftarrow \vec{N}]$ in the λ-calculus.) The *VAM* execution consists of playing opponent's moves in ϕ and ψ alternately, keeping pointers that serve to reconstruct the positions in ϕ and ψ under which these moves actually take place. Formally, each machine step consists of rewriting such a sequence Γ of (pointing) moves, called a history, to a history that extends Γ by one move.

Definition 2.4. (History) A *pointing opponent's move* is a pair $\langle a, m\rangle$, where $a \in A$ and $m \geqslant 1$ is a natural number. Initial opponent's moves do not carry pointers, but we call them pointing moves also for uniformity.
A *non-terminated history* is a sequence of pointing opponent's moves. More precisely, non-terminated histories are defined by the following syntax:

$$\Gamma ::= \bullet \mid \Gamma\, x \mid \Gamma\, \langle a, m\rangle\ .$$

Terminated histories are sequences of the form $\Gamma[x, \overleftarrow{\hookrightarrow}]$ or $\Gamma[a, \overset{0}{\overleftarrow{\hookrightarrow}}]$, where Γ is a non-terminated history. In the following, we let Γ range over histories, *i.e.*, over non-terminated and terminated histories. We denote the successive elements of a history Γ by $\Gamma{\cdot}1, \Gamma{\cdot}2, \ldots, \Gamma{\cdot}n$.

In a *VAM* execution, a terminated history corresponds to a final state of execution. The analysis of final states will be carried out in Section 3 (see Definition 3.5).

We next define two useful functions on histories.

Definition 2.5. (Jump,view) Let Γ be a non-terminated history.

1 We define a partial function $jump_\Gamma$ from numbers to numbers by setting:

$$jump_\Gamma(n) = m - 1 \text{ iff } \Gamma{\cdot}n = \langle a, m\rangle.$$

2 The related function $view_\Gamma$ takes a number n as argument and returns a sequence of opponent's moves:

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

Recall (*cf.* Remark 1) that we have called sequences of opponent's moves views, and that a strategy can be defined as a partial function from views to player's moves. Definition 2.5 offers a dynamic account of views: the function $view_\Gamma$ allows us to reconstruct views in ϕ and ψ from a history. Consider ϕ and ψ as strategies for two players and call them Player and Opponent, respectively. Opponent plays opponent's moves in ψ while Player

† In a companion paper, Curien and Herbelin (1997), we present different variations of the *VAM*, including the Strategic Abstract Machine, or *SAM*, which is a stack-free and abstract version of Krivine's abstract machine (Curien 1991).

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$$(1)^v \qquad \frac{}{\longrightarrow \bullet}$$

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

$$(2n+1)^v_b \qquad \frac{\psi(view_\Gamma(2n)) = [a, \overset{i}{\hookleftarrow}] \quad jump^i_\Gamma(2n) = 2m}{\Gamma \longrightarrow \Gamma\langle a, 2m\rangle}$$

$$(2n)^v_f \qquad \frac{\phi(view_\Gamma(2n-1)) = [x, \overset{-}{\hookleftarrow}]}{\Gamma \longrightarrow \Gamma\, x}$$

$$(2n+1)^v_! \qquad \frac{\psi(view_\Gamma(2n)) = [x, \overset{-}{\hookleftarrow}]}{\Gamma \longrightarrow \Gamma\, [x, \overset{-}{\hookleftarrow}]}$$

$$(2n)^v_! \qquad \frac{\phi(view_\Gamma(2n-1)) = [a, \overset{i}{\hookleftarrow}] \quad jump^i_\Gamma(2n-1) = 1}{\Gamma \longrightarrow \Gamma[a, \overset{0}{\hookleftarrow}]}$$

Fig. 1. The View Abstract Machine

plays opponent's moves in ϕ. Each player in turn uses his view $view_\Gamma(n)$ (where n is the length of the current history) to determine his next move, in a way that is explained below in detail. The notion of view was first defined in Hyland and Ong (1994). Hyland and Ong emphasise the idea of view by building it into the definition of strategy itself. In our terms, they define strategies as functions from histories to player's moves that are neither history-free nor completely history-sensitive, by which we mean that Player (or Opponent) is allowed to use only part of the history – his view, whence the terminology – to determine his next move. We refer the reader to Hyland and Ong (1994) for more intuition on the notion of view.

The rules of the *VAM* are given in Figure 1. We consider it implicit that

— the execution is relative to a fixed pair of a strategy ϕ and of a multi-strategy ψ, and
— there is a counter recording the current machine step number n. The succession of the steps is: $1, 2, 3, 4, \ldots$, corresponding to alternating opponent's moves in ϕ and ψ:

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

Here is an informal explanation of the rules of the *VAM*. Rule $(1)^v$ initiates the computation: the top move of ϕ may be considered as an initial query. Suppose that the machine has just performed $2n-1$ steps, for some n, the last step consisting of a move

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

in ϕ. Then the next move will be played in ψ. The machine computes the view at step $2n-1$, and then reads the value $[\alpha, \overset{\kappa}{\hookleftarrow}]$ of ϕ for this view. At this stage, we already know that α is the move played at step $2n$. There are two cases:

1 If $\alpha = x$ (and thus $\kappa = _$), the computation proceeds by playing the initial move x of ψ. This can be considered as the communication of ϕ and ψ along the channel x.
2 If $\alpha = a$ and $\kappa = i$, the iteration of the function $jump_\Gamma$ allows us to transform the *static* information i into the *dynamic* information $2m-1$. There are again two cases.
   (a) If $2m-1 = 1$, then a points to the initial move of the history, which constitutes an answer to the initial query, and terminates the execution.
   (b) If $2m-1 > 1$, the dialogue is not terminated: at step $2m-2$, a (pointing) move had been played in ψ, and the move a should be played in ψ right under that move: this is what the pointing information $\langle a, 2m-1\rangle$ in the conclusion of rule $(2n)^v_b$ is taking care of[†].

Now consider the situation after an even number of steps. One similarly computes a view, and the value $[\alpha, \overset{\kappa}{\hookleftarrow}]$ of ψ for this view. We revisit the above cases as follows:

1 If $\alpha = x$ (and thus $\kappa = _$), a 'free variable' of ψ has been reached, and this provides an answer to the initial query.
2 If $\alpha = a$ and $\kappa = i$, the computation proceeds as in subcase 2(b) above. There is no counterpart for the subcase 2(a), because if a is dynamically bound to the (copy of the) initial move x of ψ visited at step $2m$, the dialogue should be continued, because this step $2m$ had been performed in reaction to a step $2m-1$ played in ϕ.

Notice that the *VAM* stops after a step $(n)^v_!$: since we have not defined the view of a terminated history, there is no way to match the assumption of any rule labelled with $n+1$ (see Section 3 for a stronger framework where the machine can proceed further).

Remark 2. The following two kinds of interaction, corresponding to the two parts a and i in the player's move $[a, \overset{i}{\hookleftarrow}]$ present in the assumption of the rules $(n)^b_v$, respectively, are combined in the *VAM*.

— *Tree interaction* The determinism of the strategy and of the multi-strategy allows us to resolve the non-determinism of the choice of the next opponent's move in the multi-strategy.
— *Pointer interaction* The pointer structure allows us to determine the location of the next move to play.

Hence the two interactions correspond to the respective questions: 'which is the next move to play?', and 'where is this move to be played?'. They form the heart of the computational engine of the *VAM*.

Coquand has proved, in his terminology, that all debates between finite strategies terminate. In the present framework, his result implies that the *VAM* execution relative to *finite* ϕ and ψ terminates. For completeness, we include a proof of Coquand's result

[†] The move a in ψ can be understood as 'fetching the value of the bound variable' $[a, \overset{i}{\hookleftarrow}]$. We refer to Curien and Herbelin (1997) for precise connections between the *VAM* and environment abstract machines.

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

(Theorem 2.15), which is directly adapted from (a preliminary version of) Coquand (1995): the present proof is classical, while the proof published in Coquand (1995) is constructive. The proof is purely combinatorial, and only uses the pointer structure of the growing history (forgetting about the alphabet A) and the boundedness of views.

The following definition is taken from Coquand (1995), up to some change of notation.

Definition 2.6. (Interaction sequence) Let N be a natural number $\geqslant 2$, or infinite. An interaction sequence over the interval $[1, N]$ of ω is a pair (V, f) where

— f is a partial function from $[1, N]$ to itself, and where
— V is a function from $[1, N]$ to $P(\omega)$,

satisfying

$$V(n) = \begin{cases} \{n\} \cup V(f(n)-1) & \text{if } f(n) \text{ is defined} \\ \{n\} & \text{otherwise} \end{cases}$$

$$\begin{cases} f(1) \text{ is undefined} \\ (f(n+1) \text{ defined } \Rightarrow f(n+1) \in V(n)) \,. \end{cases}$$

Lemma 2.7. If (V, f) is an interaction sequence, then, for all n

1 $(f(n) \text{ defined} \Rightarrow f(n) < n)$,
2 $V(n) \subseteq \{1, \ldots, n\}$.

Proof. We write $1(n)$ to mean that Property 1 holds for n, and similarly for 2. We prove successively that

— $2(1)$: This is obvious since $V(1) = \{1\}$.
— $2(n) \Rightarrow 1(n+1)$: Since $f(n+1) \in V(n)$, we have $f(n+1) \in \{1, \ldots, n\}$, and hence $f(n+1) < n+1$.
— $1(n) \Rightarrow 2(n)$: If $V(n) = \{n\}$, then *a fortiori* $V(n) \subseteq \{1, \ldots, n\}$. If $V(n) = \{n\} \cup V(f(n)-1)$, we conclude by induction since $f(n) < n$ implies *a fortiori* $f(n) - 1 < n$. □

In an interaction sequence (V, f), V is entirely determined by f, as the following lemma shows.

Lemma 2.8. Let V and f be of the types indicated in Definition 2.6. Let g be defined by

$$g(n) = \begin{cases} f(n) - 1 & \text{if } f(n) \text{ is defined} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Then (V, f) forms an interaction sequence if and only if, for all $n \geqslant 1$

$$V(n) = \{g^i(n) \mid i \geqslant 0 \text{ and } g^i(n) \text{ defined}\}$$
$$(f(n+1) \text{ defined } \Rightarrow f(n+1) = g^i(n) \text{ for some } i \geqslant 0) \,.$$

Proof. Let (V, f) be an interaction sequence. By Lemma 2.7, we have $g(n) < n$ for any n. Hence, given n, there exists an i such that $g^i(n)$ is defined and $g^{(i+1)}$ is not defined. By unfolding we get

$$\begin{array}{rclcrcl} V(n) & = & \{n\} \cup V(g(n)) & = & \{n, g(n)\} \cup V(g^2(n)) & = & \cdots \\ & = & \{n, g(n), \ldots, g^{i-1}(n)\} \cup V(g^i(n)) & = & \{n, g(n), \ldots, g^i(n)\} & & . \end{array}$$

The converse direction is straightforward. □

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

Definition 2.9. (Combinatorial view) Let Γ be a non-terminated history. We define two functions f_Γ and V_Γ as follows:

$$f_\Gamma(n) = m \quad \Leftrightarrow \quad \Gamma{\bullet}n = \langle a, m\rangle$$

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

We call the $V_\Gamma(n)$'s the combinatorial views associated with Γ. (Clearly, for all n, the length of $view_\Gamma(n)$ is the cardinal of $V_\Gamma(n)$.)

Lemma 2.10. The pair (V_Γ, f_Γ) of Definition 2.9 forms an interaction sequence.

Proof. The two rules of the *VAM* adding a pointing move $\langle a, m\rangle$ to the history are $(2n)^b_v$ and $(2n+1)^b_v$. By considering the assumptions of these rules, we obtain, for all $k \geqslant 1$ (even or odd),

$$f_\Gamma(k) = jump^i_\Gamma(k-1) \quad \text{for some } i \geqslant 0.$$

Hence the condition of Lemma 2.8 is satisfied. □

Coquand's argument by contradiction consists in proving that if an execution does not terminate, there must exist an infinitely growing (combinatorial) view.

Definition 2.11. We write $m \prec n$ if $n < m$ and $n - 1 \in V(m-1)$.

By Lemmas 2.8 and 2.7, an alternative definition of $\prec$ is

$$m \prec n \;\Leftrightarrow\; n-1 \in g(V(m-1)) \;\Leftrightarrow\; n \in f(V(m-1))\,. \tag{2}$$

Lemma 2.12. If $f(f(n))$ is defined, $n \prec f(f(n))$.

Proof. We use the alternative definition (2) of $\prec$: we have $f(n) \in V(n-1)$, and hence $f(f(n)) \in f(V(n-1))$. □

For the next lemma, we adjust the definition of interaction sequence in order to be able to manipulate extracted interaction sequences. Replacing $[1, N]$ by a sequence $n_1, \ldots, n_k$ such that $n_1 < n_2 < \cdots < n_k$, we now define

$$g(n_p) = n_{q-1} \text{ if } f(n_p) = n_q$$

$$V(n_p) = \begin{cases} \{n_p\} \cup V(g(n_p)) & \text{if } f(n_p) \text{ is defined} \\ \{n_p\} & \text{otherwise} \end{cases}$$

and the condition to satisfy is now

$$(f(n_{p+1}) \text{ defined } \Rightarrow f(n_{p+1}) \in V(n_p)).$$

If this condition is satisfied, we say that (V, f) is an interaction sequence on $n_1, \ldots, n_k$. We likewise define an infinite interaction sequence on a sequence $(n_p)_{p\geqslant 0}$. A sensitive point here is that V now depends both on f and on $(n_p)_{p\geqslant 0}$. We call V the combinatorial view function associated with f and $(n_p)_{p\geqslant 0}$.

Definition 2.13. Suppose that (V, f) is an infinite interaction sequence on $(n_p)_{p\geqslant 0}$, and let $(n'_p)_{p\geqslant 0}$ be an infinite subsequence of $(n_p)_{p\geqslant 0}$. We say that (V, f) induces an interaction subsequence on $(n'_p)_{p\geqslant 0}$ if the following conditions hold:

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

1 $(n'_p)_{p\geqslant 0}$ is stable under f, that is,

$$\forall\, p\ (f(n'_p)\ \text{defined} \Rightarrow \exists\, q\ (f(n'_p) = n'_q)).$$

2 For all p, $V'(n_p) \subseteq V(n_p)$, where V' is the combinatorial view function associated with f and $(n'_p)_{p\geqslant 0}$.

Remark 3. Condition 2 of Definition 2.13 has the following consequence: if we have $m \prec n$ relatively to V', we also have $m \prec n$ relatively to V.

We can now state the key lemma.

Lemma 2.14. Let (V, f) be an infinite interaction sequence on $(n_p)_{p\geqslant 0}$. One of the following properties hold:

1 either for all q there exists $r > q$ such that $n_q = f(n_r)$,
2 or there exists a subsequence $(n'_p)_{p\geqslant 0}$ of the form

$$n_1, n_2, \ldots, n_{p-1}, m_p, m_{p+1}, \ldots$$

such that $m_p \prec n_p$ and on which (V, f) induces an interaction subsequence.

Proof. Suppose that Property 1 does not hold, and let q be such that $n_q \notin f((n_p)_{p\geqslant 0})$. Let $n_p = f(n_q)$ (the case where $f(n_q)$ is undefined is treated similarly), and consider the sequence

$$(n'_p)_{p\geqslant 0} = n_1, n_2, \ldots, n_{p-1}, n_{q+1}, n_{q+2}, \ldots\ .$$

We first show $n_{q+1} \prec n_p$, that is, $n_p \in f(V(n_q))$. This is obvious from $n_p = f(n_q)$ and $n_q \in V(n_q)$. We next show that the subsequence $(n'_p)_{p\geqslant 0}$ is stable under f, that is, that $f(n_r) \notin \{n_p, n_{p+1}, \ldots, n_q\}$, for all $r > q$. We call this property $P(r)$ and prove it together with the following related property, for any $r \geqslant q$, which we call $Q(r)$:

$$V(n_r) \cap \{n_p, n_{p+1}, \ldots, n_{q-1}\} = \varnothing\ .$$

We prove successively:

— $Q(q)$: This is obvious by Lemma 2.7, since $V(n_q) = \{n_q\} \cup V(n_{p-1})$.
— $Q(r-1) \Rightarrow P(r)$: Since $f(n_r) \in V(n_{r-1})$, we have $f(n_r) \notin \{n_p, n_{p+1}, \ldots, n_{q-1}\}$ by $Q(r-1)$. We also have $f(n_r) \neq n_q$ by assumption, so $P(r)$ holds.
— $(P(r), Q(q), \ldots Q(r-1)) \Rightarrow Q(r)$ $(r > q)$: We have $V(n_r) = \{n_r\} \cup V(n_{s-1})$, where $n_s = f(n_r)$. Clearly, $n_r \notin \{n_p, n_{p+1}, \ldots, n_{q-1}\}$. We know by $P(r)$ that $n_s \notin \{n_p, n_{p+1}, \ldots, n_q\}$. There are two cases: if $s < p$, then *a fortiori* $s - 1 < p$, which ensures $V(n_{s-1}) \cap \{n_p, n_{p+1}, \ldots, n_{q-1}\} = \varnothing$; if $s > q$, then $s - 1 \geqslant q$, so we can conclude using $Q(s-1)$.

Finally, we have to check Condition 2 of Definition 2.13. Let g' be the function defined by $g'(n'_p) = n'_{q-1}$ if $f(n'_p) = n'_q$. The only possibilty giving $g'(n_r) \neq g(n_r)$ is $f(n_r) = n_{q+1}$. Then $g(n_r) = n_q$, while $g'(n_r) = n_{p-1} = g(n_p) = g(g(n_r))$. Condition 2 then follows from Lemma 2.8. □

Theorem 2.15. Let (V, f) be an infinite interaction sequence. The following properties hold:

1 The relation $\prec$ is not well-founded.
2 The set $\{\sharp(V(n)) \mid n \geqslant 1\}$ (where $\sharp(V(n))$ denotes the cardinal of $V(n)$) has no upper bound.

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

Proof. Suppose that $\prec$ is well-founded. We can construct a sequence $(m_p)_{p\geqslant 0}$ as follows. For each p, we choose m_{p+1} $\prec$-minimal among the set of all m's for which f induces an interaction on an infinite subsequence starting with $m_1, m_2, \ldots, m_p, m$. (This set is maintained non-empty by the induction.) By construction and Lemma 2.14, we can extract from $(m_p)_{p\geqslant 0}$ an infinite sequence $(n_p)_{p\geqslant 0}$ such that $n_p = f(n_{p+1})$ for all p. This implies $n_{2p} = f(f(n_{2(p+1)}))$, and hence $n_{2(p+1)} \prec n_{2p}$ by Lemma 2.12, and contradicts the well-foundedness assumption. Notice the implicit use of Remark 3 in the above argument: as the construction proceeds, successive subsequences are extracted, and the $\prec$-minimality is meant relative to each of the successive subsequences; Condition 2 of Definition 2.13 allows us to inherit the well-foundedness of $\prec$ along the construction.

We next prove that views are unbounded. Since $\prec$ is not well-founded, there exists a subsequence $(l_n)_{n\geqslant 0}$ such that $l_{n+1} \prec l_n$, for all n. If we set $k_n = l_n - 1$, this can be rephrased as ($k_n < k_{n+1}$ and $k_n \in V(k_{n+1})$). In particular, all the k_n's are distinct. We observe, as an immediate consequence of the characterisation of V given in Lemma 2.8, that, say,

$$(k_{n-1} \in V(k_n) \text{ and } k_n \in V(k_{n+1})) \Rightarrow k_{n-1} \in V(k_{n+1}).$$

It follows that $\{k_1, \ldots, k_n\} \subseteq V(k_{n+1})$, and hence the conclusion follows. □

Proposition 2.16. If ϕ and ψ are a finite strategy and a finite multi-strategy, respectively, then the *VAM* execution relative to ϕ and ψ terminates.

Proof. Suppose that the *VAM* does not terminate, and let Δ be the infinite history whose finite prefixes are the histories Γ such that $\longrightarrow^\star \Gamma$. Let (V_Δ, f_Δ) be the associated infinite interaction sequence (*cf.* Lemma 2.10). We observe that

— the combinatorial views have an unbounded length, by Theorem 2.15, while
— the views have a bounded length, being in the domain of definition of ϕ or ψ.

This yields a contradiction, since the lengths of the views are the cardinals of their underlying combinatorial views. □

In order to illustrate the *VAM*, we anticipate Subsection 4.1 and describe a compilation of (untyped) λ normal forms, or finite Böhm trees, as defined by the following syntax:

$$M ::= \lambda\vec{x}.W \qquad W ::= yM_1 \cdots M_n \quad (n \geqslant 0)\,.$$

We take as alphabet $A = \omega$. We perform the following transformation on the tree representation of a normal form M:

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

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 $\underline{y}$. When y is free, $\underline{y}$ is $[y, \overleftarrow{\hookrightarrow}]$. Otherwise, $\underline{y}$ is $[n, \overset{i}{\overleftarrow{\hookrightarrow}}]$ such that the binder is found $2i+1$ moves above, and n denotes the position of y in the multiple abstraction $\lambda y_1 \cdots y_m$ that binds it, that is, $y = y_n$. 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 Huet (1993). Finally, if ϕ_i is the translation of M_i for all i, the head opponent's

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

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

where

$$\frac{}{x^{nil} = [x, \overleftarrow{\hookrightarrow}]} \qquad \frac{x = x_i \quad \vec{x} = x_1 \cdots x_k}{x^{L,\vec{x}} = [i, \overset{0}{\overleftarrow{\hookrightarrow}}]} \qquad \frac{x^L = [i, \overset{j}{\overleftarrow{\hookrightarrow}}]}{x^{L,\vec{y}} = [i, \overset{j+1}{\overleftarrow{\hookrightarrow}}]} \qquad \frac{x^L = [x, \overleftarrow{\hookrightarrow}]}{x^{L,\vec{y}} = [x, \overleftarrow{\hookrightarrow}]}$$

Fig. 2. Compilation of untyped λ normal forms

move $\bullet$ of ϕ_i is replaced by i. In this way, we record that M_i is the i-th argument of y. To summarise, the translation of $M = (\lambda\vec{x}.yM_1\cdots M_n)$ is the following tree:

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

Formally, we define in Figure 2 a translation function involving a function that takes a list L of vectors of variables and a term M, and returns a strategy M^L. The compilation of a term M is defined as M^{nil}. We adapt this compilation to a typed setting in Section 3, and show that it defines a bijective transformation to a suitable set of typed strategies.

Example 2.17. We display the computation of $(u(\lambda x.u(\lambda y.x)))^{nil} \circ [u \leftarrow (\lambda r.r(r(z)))^{nil}]$. The boxed comments give the correspondence with the source terms.

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

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

— Multi-strategy for $[u \leftarrow (\lambda r.r(r(z)))]$:

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

The execution of this example by the *VAM* is as follows (the step numbers are placed below the successive moves, as a help to the reader):

$\bullet$	u	$\langle 1,2\rangle$	u	$\langle 1,4\rangle$	$\langle 1,3\rangle$	$\langle 1,2\rangle$	u	$\langle 1,8\rangle$	$\langle 1,7\rangle$	$[z, \overleftarrow{\hookrightarrow}]$
1	2	3	4	5	6	7	8	9	10	11

The successive views are as follows:

$$\begin{array}{lll} view_\Gamma(1) = \bullet & view_\Gamma(2) = u & view_\Gamma(3) = \bullet 1 \\ view_\Gamma(4) = u & view_\Gamma(5) = \bullet 11 & view_\Gamma(6) = u1 \\ view_\Gamma(7) = \bullet 1 & view_\Gamma(8) = u & view_\Gamma(9) = \bullet 11 \\ view_\Gamma(10) = u11 & & \end{array}$$

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

3. Typed abstract Böhm trees

We introduce a set of abstract types and revisit the definitions of Section 2, imposing type constraints.

Definition 3.1. (Typed position) In addition to the alphabet A, we assume a set T of *abstract types*, or types for short, together with a partial function $(__)$ from $T \times A$ to T, written $(\sigma, a) \mapsto \sigma_a$. A typed position is a position all of whose moves are decorated with a type superscript as follows:

$$a_1^{\sigma_1}[\alpha_2, \overset{\kappa_1}{\hookleftarrow}]^{\sigma_2} \cdots a_{2n-1}^{\sigma_{2n-1}}[\alpha_{2n}, \overset{\kappa_n}{\hookleftarrow}]^{\sigma_{2n}}.$$

We say that $a_1[\alpha_2, \overset{\kappa_1}{\hookleftarrow}] \cdots a_{2n-1}[\alpha_{2n}, \overset{\kappa_n}{\hookleftarrow}]$ is the underlying (untyped) position.

Definition 3.2. (Typed strategy) A typed strategy, or *typed abstract Böhm tree*, is a set ϕ of typed positions whose underlying set of positions forms a strategy and that satisfies the following two properties:

(1) If $q[\alpha, \overset{\kappa}{\hookleftarrow}]^{\sigma} \in \phi$, then, for any $\tau \in T$ and $b \in A$

$$(q[\alpha, \overset{\kappa}{\hookleftarrow}]^{\sigma} b^{\tau} \text{ is answered in } \phi) \Leftrightarrow (\sigma_b \text{ is defined and } \sigma_b = \tau).$$

(2) If $q[a, \overset{i}{\hookleftarrow}]^{\sigma} \in \phi$ and if $[a, \overset{i}{\hookleftarrow}]$ is bound to b^{τ} in q, then $\sigma = \tau_a$.

Moreover, we require that all occurrences of a free player's move $[x, \overset{-}{\hookleftarrow}]$ bear the same type. The type decorating $\bullet$ is called the type of ϕ.

For economy of notation, we shall freely omit the type decorations on moves when they are not relevant to the discussion. But the reader should keep in mind that they are implicitly carried through the constructions in the obvious way.

We now give a typed version of pseudo-strategies, relative to a list Σ of types.

Definition 3.3. Let ϕ be a set of typed positions whose underlying set of positions forms a pseudo-strategy, and let Σ be a list of types. Let $r = q[a, \overset{i}{\hookleftarrow}]^{\sigma} \in \phi$ of length $2l$ such that $[a, \overset{i}{\hookleftarrow}]$ is not bound in ϕ. We write r^{Σ} for the $(i - l + 1)$-th type in the sequence Σ. We assume therefore that Σ has length at least $i - l + 1$ for all $r = q[a, \overset{i}{\hookleftarrow}]^{\sigma} \in \phi$ such that $[a, \overset{i}{\hookleftarrow}]$ violates Condition 2.1 (1). We call ϕ a *typed pseudo-strategy* relative to Σ if it satisfies the conditions listed in Definition 3.2 plus the following one:

(3) If $r = q[a, \overset{i}{\hookleftarrow}]^{\sigma} \in \phi$, and if $[a, \overset{i}{\hookleftarrow}]$ is not bound in ϕ, then $\sigma = (r^{\Sigma})_a$.

As for strategies, the type decorating $\bullet$ is called the type of ϕ.

In order to establish precise correspondences between concrete syntaxes of terms or proofs and abstract Böhm trees relative to appropriate A, T, and $(__)$, we shall rely on the following simple decomposition property for typed pseudo-strategies.

Lemma 3.4. Let ϕ and Σ be such that ϕ is a typed pseudo-strategy relative to Σ. We can write uniquely

$$\phi = \{\bullet^{\tau}[\alpha, \overset{\kappa}{\hookleftarrow}]^{\sigma}\} \bigcup \Big(\bigcup_{\{a \mid \sigma_a \text{ is defined}\}} \{\bullet[\alpha, \overset{\kappa}{\hookleftarrow}]ap \mid \bullet p \in \phi_a\} \Big).$$

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

Moreover, all ϕ_a's are pseudo-strategies relative to Σ, τ. We use the following synthetic notation for the decomposition:

$$\phi \rightsquigarrow \bullet^{\tau}[\alpha, \overset{\kappa}{\hookleftarrow}]^{\sigma}, \{\phi_a \mid \sigma_a \text{ is defined}\}.$$

or, if $\{a \mid \sigma_a \text{ is defined}\} = \{a_1, \ldots, a_n\}$:

$$\phi \rightsquigarrow \bullet^{\tau}[\alpha, \overset{\kappa}{\hookleftarrow}]^{\sigma}, \phi_{a_1}, \ldots, \phi_{a_n}.$$

Proof. The underlying untyped positions of ϕ_a inherit the closure properties of the corresponding underlying untyped positions of ϕ, and hence form a pseudo-strategy. Condition (1), which is entirely local, is also clearly inherited from Condition (1) for ϕ. For Conditions (2) and (3), there are three cases. Let $r = \bullet q[b, \overset{i}{\hookleftarrow}]^{\sigma} \in \phi_a$.

— If $[b, \overset{i}{\hookleftarrow}]$ is bound in r, then it is *a fortiori* bound in $\bullet[\alpha, \overset{\kappa}{\hookleftarrow}]aq[b, \overset{i}{\hookleftarrow}]$, and Condition (2) follows from Condition (2) for ϕ.

— By contraposing this implication, if $[b, \overset{i}{\hookleftarrow}]$ is not bound in $\bullet[\alpha, \overset{\kappa}{\hookleftarrow}]aq[b, \overset{i}{\hookleftarrow}]$, then Condition (3) follows from Condition (3) for ϕ.

— If $[b, \overset{i}{\hookleftarrow}]$ is bound in $\bullet^{\tau}[\alpha, \overset{\kappa}{\hookleftarrow}]aq[b, \overset{i}{\hookleftarrow}]$ but not in r, it is bound to $\bullet^{\tau}$ in ϕ, so Condition (3) relative to Σ, τ follows from Condition (2) for ϕ. □

We now show that typed strategies and pseudo-strategies are deadlock-free. We first define the legal final states of the *VAM*.

Definition 3.5. (Legal final state) We say that the *VAM* is in a final (F) state when it has reached a step $(2n+1)^{v}_{!}$, and that it is in a final (B) state when it has reached a step $(2n)^{v}_{!}$. (F and B stand for 'free' and 'bound', respectively.) A state is called legal if it is either (F) final or (B) final.

At an (F) final state, the computation terminates because a free variable of ψ has been met. This is what happens in Example 2.17. We can interpret this as a demand to the input multi-strategy. A simple example of (B) final state is obtained by composing $\bullet[1, \overset{0}{\hookleftarrow}]$ (corresponding to $(\lambda x.x)$) with an arbitrary multi-strategy: the execution results in the terminated history $\bullet[1, \overset{0}{\hookleftarrow}]$. See Section 4.2 for another flavour of (B) final state.

In both cases (F) and (B), the machine stops because a 'head normal form' has been reached. More precisely, for λ normal forms, the two cases of termination correspond to the following partition of head normal forms:

$$\begin{array}{lll} (\lambda\vec{x}.y\vec{M}) & (y \text{ free}) & (F) \\ (\lambda\vec{x}.x\vec{M}) & (x \text{ bound}) & (B)\,. \end{array}$$

In Figure 3, we present a typed version of the *VAM*. The rules are those of the *VAM*, with the addition that the type decorations indicated by ϕ and ψ are faithfully recorded. The typed *VAM* executes the composition of

— a strategy ϕ whose free player's moves are $[x_1, \overset{-}{\hookrightarrow}]^{\sigma_1}, \ldots, [x_n, \overset{-}{\hookrightarrow}]^{\sigma_n}$ with

— a multi-strategy $\psi = [x_1 \leftarrow \psi_{x_1}, \ldots, x_n \leftarrow \psi_{x_n}]$ whose initial moves x_i are decorated with the respective types σ_i. In other words, x_i and ψ_{x_i} (bound to x_i) must have the same type.

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$$(1)^v \qquad \frac{\phi \text{ has type } \sigma}{\longrightarrow \bullet^\sigma}$$

$$(2n)^v_b \qquad \frac{\phi(view_\Gamma(2n-1)) = [a, \overset{i}{\hookleftarrow}]^\sigma \quad jump^i_\Gamma(2n-1) = 2m-1 > 1}{\Gamma \longrightarrow \Gamma\langle a, 2m-1\rangle^\sigma}$$

$$(2n+1)^v_b \qquad \frac{\psi(view_\Gamma(2n)) = [a, \overset{i}{\hookleftarrow}]^\sigma \quad jump^i_\Gamma(2n) = 2m}{\Gamma \longrightarrow \Gamma\langle a, 2m\rangle^\sigma}$$

$$(2n)^v_f \qquad \frac{\phi(view_\Gamma(2n-1)) = [x, \overset{-}{\hookleftarrow}]^\sigma}{\Gamma \longrightarrow \Gamma\, x^\sigma}$$

$$(2n+1)^v_! \qquad \frac{\psi(view_\Gamma(2n)) = [x, \overset{-}{\hookleftarrow}]^\sigma}{\Gamma \longrightarrow \Gamma\, [x, \overset{-}{\hookleftarrow}]^\sigma}$$

$$(2n)^v_! \qquad \frac{\phi(view_\Gamma(2n-1)) = [a, \overset{i}{\hookleftarrow}]^\sigma \quad jump^i_\Gamma(2n-1) = 1}{\Gamma \longrightarrow \Gamma[a, \overset{0}{\hookleftarrow}]^\sigma}$$

Fig. 3. The typed *VAM*

Proposition 3.6. The execution of a typed strategy composed with a typed multi-strategy by the typed *VAM* can only terminate in a legal final state.

Proof. The proposition is proved by induction on the length of the execution. Let Γ be a non-terminated history resulting from the n first steps of execution. We have to show that nothing prevents the next step being executed. We claim that $\chi(view_\Gamma(n))$ is defined, where χ is ϕ or ψ depending on the parity of n. We first show how to conclude from the claim. The statement follows immediately from the claim for rules $(2n)^v_f$ and $(2n+1)^v_!$. For all the other rules, we have to show that if $\chi(view_\Gamma(n)) = [a, \overset{i}{\hookleftarrow}]$, then $jump^i_\Gamma(n)$ is defined. By Condition 2.1 (1), $view_\Gamma$ is long enough to have a suffix $a_i \cdots a_0$. Unfolding the recursive definition of $view_\Gamma$, we have

$$view_\Gamma(n) = view_\Gamma(jump_\Gamma(n))a_0 = view_\Gamma(jump^i_\Gamma(n))a_{i-1} \cdots a_0\ . \tag{3}$$

Hence $jump^i_\Gamma(n)$ is defined, and the next step can be performed.

We are left with the proof of the claim. If $\Gamma{\cdot}n = \bullet$, the claim follows from the non-emptyness of ϕ. If $\Gamma{\cdot}n = x$, the claim follows similarly from the non-emptyness of ψ. If $\Gamma{\cdot}n = \langle a, m\rangle$, the last rule applied is $(n)^v_b$. Suppose, say, that n is even. Reading the assumptions, and using Equation (3) above as well as Condition (2) of Definition 3.2, we get that $m = jump^i_\Gamma(n)$ for some i and that $\sigma = \tau_a$, where τ is the type decoration of $\Gamma{\cdot}m$. By the definition of the typed *VAM*, τ is also the type decoration of $\psi(view_\Gamma(m-1))$. Since $view_\Gamma(n) = view_\Gamma(m-1)\, a$, we conclude using Condition (1) of Definition 3.2. □

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

In order to appreciate the previous proposition, we should see how it fails for non-typable strategies. We show how the *VAM* executes $\Delta\Delta = (\lambda x.xx)(\lambda x.xx)$, or in compiled form

$$\{\bullet[x, \overset{-}{\hookleftarrow}], \bullet[x, \overset{-}{\hookleftarrow}]1[x, \overset{-}{\hookleftarrow}]\}[x \leftarrow \{x[1, \overset{0}{\hookleftarrow}], x[1, \overset{0}{\hookleftarrow}]1[1, \overset{1}{\hookleftarrow}]\}].$$

The execution by the *VAM* results in the following history:

$$\begin{array}{cccccc} \bullet & x & \langle 1,2\rangle & x & \langle 1,4\rangle \\ 1 & 2 & 3 & 4 & 5 \end{array}$$

with the following associated views:

$$view_\Gamma(1) = \bullet \quad view_\Gamma(2) = x \quad view_\Gamma(3) = \bullet 1 \quad view_\Gamma(4) = x \quad view_\Gamma(5) = \bullet 11\ .$$

The *VAM* stops after step $(5)^v_b$ without having reached a legal final state.

Remark 4. The *VAM* could have computed a bit further if the strategy had been 'η-expanded': the compilation of $\lambda x.x(\lambda y.xy)$ answers the view $\bullet 11$ with the compilation of y. In the companion paper, Curien and Herbelin (1997), we show how the *VAM* can be adapted to mimic η-expansions dynamically, yielding a non-terminating computation (involving unbounded views, *cf.* Proposition 2.16).

We now formulate a strong version of the *VAM*, called Strong *VAM*, which can compute full normal forms. The machine works on demand. Given a strategy ϕ and a multi-strategy ψ, it builds any position of the composition $\phi \circ \psi$, provided it is asked by an external agent, called the *user*, to progressively do so. We signal the user's demands and the answers to the user by underlining. The user raises an initial query, call it $\underline{\bullet}$ in first approximation. The *VAM* is started, and runs until it reaches a legal final state that produces a player's move $\underline{[\alpha, \overset{\kappa}{\hookleftarrow}]}$ answering the initial query. We then know that $\bullet[\alpha, \overset{\kappa}{\hookleftarrow}] \in \phi \circ \psi$. The user may raise a new query $\bullet[\alpha, \overset{\kappa}{\hookleftarrow}]b$. A new run of the *VAM* is started, until a new 'final' state is reached. If the final player's move is bound, should we bind it to $\bullet$ or to b? To keep a proper scoping, we use a mechanism of names. This leads to the following notion of strong history and the following extension of the function *view*. We do not include type decorations to avoid overloading the notation.

Definition 3.7. (Strong history) The syntax of strong histories is as follows:

$$\Gamma ::= \underline{\bullet_v} \mid \Gamma \underline{a_v} \mid \Gamma x \mid \Gamma\langle a, m\rangle \mid \Gamma\underline{[a, \overset{v}{\hookleftarrow}]} \mid \Gamma\underline{[x, \overset{-}{\hookleftarrow}]}$$

where v ranges over an infinite set N of names. The underlined moves correspond to the progressive formation of the positions of the composition (see Definition 3.9 and Proposition 3.11).

Definition 3.8. (Strong view) The strong views associated with a strong history are defined by the rules of Definition 2.5 plus the following one:

$$\frac{\Gamma{\cdot}n = \underline{a_v}}{view_\Gamma(n) = view_\Gamma(n-2)\ a}$$

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$$(1)^s_? \qquad \frac{v \text{ new}}{\longrightarrow (\underline{\bullet_v}\ ?)}$$

$$(2n)^s_b \qquad \frac{\phi(view_\Gamma(2n-1)) = [a, \overset{i}{\hookleftarrow}] \quad jump^i_\Gamma(2n-1) = 2m-1 \quad \Gamma{\bullet}(2m-1) \neq \underline{b_v}}{(\Gamma\ ?) \longrightarrow (\Gamma\langle a, 2m-1\rangle\ ?)}$$

$$(2n+1)^s_b \qquad \frac{\psi(view_\Gamma(2n)) = [a, \overset{i}{\hookleftarrow}] \quad jump^i_\Gamma(2n) = 2m \quad \Gamma{\bullet}2m \neq \underline{b_v}}{(\Gamma\ ?) \longrightarrow (\Gamma\langle a, 2m\rangle\ ?)}$$

$$(2n)^s_f \qquad \frac{\phi(view_\Gamma(2n-1)) = [x, \overline{\hookleftarrow}]}{(\Gamma\ ?) \longrightarrow (\Gamma x\ ?)}$$

$$(2n)^s_! \qquad \frac{\phi(view_\Gamma(2n-1)) = [a, \overset{i}{\hookleftarrow}] \quad jump^i_\Gamma(2n-1) = 2m-1 \quad \Gamma{\bullet}(2m-1) = \underline{b_v}}{(\Gamma\ ?) \longrightarrow (\Gamma\underline{[a, \overset{v}{\hookleftarrow}]}\ !)}$$

$$(2n+1)^s_! \qquad \frac{\psi(view_\Gamma(2n)) = [a, \overset{i}{\hookleftarrow}] \quad jump^i_\Gamma(2n) = 2m \quad \Gamma{\bullet}2m = \underline{b_v}}{(\Gamma\ ?) \longrightarrow (\Gamma\underline{[a, \overset{v}{\hookleftarrow}]}\ !)}$$

$$(2n)^s_? \qquad \frac{\psi(view_\Gamma(2n-1)a) \text{ defined} \quad v \text{ new}}{(\Gamma\ !) \longrightarrow (\Gamma\underline{a_v}\ ?)}$$

$$(2n+1)^s_? \qquad \frac{\phi(view_\Gamma(2n)a) \text{ defined} \quad v \text{ new}}{(\Gamma\ !) \longrightarrow (\Gamma\underline{a_v}\ ?)}$$

Fig. 4. The Strong *VAM*

The rules of the Strong *VAM* are displayed in Figure 4. The rules are

— those of the *VAM* (up to some change of notation), plus
— two new rules that formalise the user's new demands.

The execution proceeds by progressively extending a strong history, which is flagged with a sign ? or !, where

? indicates that the machine is engaged in an internal dialogue whose goal is to answer a user's query,
! indicates that the last user's query has just been answered.

Hence the flag ! stays for one step exactly.

Remark 5. While the *VAM* is deterministic, the transition relation of the Strong *VAM* is non-deterministic: at steps $n^s_?$, the choice of the move a is free, as long as the extended view is in the domain of ϕ or ψ. If the user knows which view he wants to explore, the

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

evaluation is deterministic with respect to that goal. We say that the execution is guided by the user.

Finally, we need to read back the positions of the composition, *i.e.*, to strip off the moves that correspond to the internal dialogue of the machine. This is formalised in the following definition, which also takes care of installing the nameless notation for bindings.

Definition 3.9. (Readback) The following function *readback* takes as argument a suffix of a strong history and an auxiliary list L of names, and returns a position.

$$\begin{array}{lcl}
readback^{nil}(\underline{\bullet_v}\Gamma) & = & \bullet\, readback^{v}(\Gamma) \\
readback^{L}(\underline{a_v}\Gamma) & = & a\, readback^{v\bullet L}(\Gamma) \\
readback^{L}(x\Gamma) & = & readback^{L}(\Gamma) \\
readback^{L}(\langle a, m\rangle\Gamma)) & = & readback^{L}(\Gamma) \\
readback^{L}(\underline{[a, \overset{v}{\hookleftarrow}]}\Gamma) & = & [a, \overset{v_L}{\hookleftarrow}]\, readback^{L}(\Gamma) \\
readback^{L}(\underline{[x, \overset{-}{\hookleftarrow}]}\Gamma) & = & [x, \overset{-}{\hookleftarrow}]\, readback^{L}(\Gamma)
\end{array}$$

where

$$v_{\mu\bullet L} = \begin{cases} 0 & \text{if } \mu = v \\ v_L + 1 & \text{if } \mu \neq v. \end{cases}$$

For a strong history we set $readback(\Gamma) = readback^{nil}(\Gamma)$.

We now have all the material to define a 'category' of typed abstract Böhm trees. We first define a notion of named multi-category that is well adapted to our purposes.

Definition 3.10. (Named multi-category) A named multi-category consists of a collection of objects and a collection of morphisms; each morphism f has a target object B, and has a source consisting of a context of objects, that is, a list $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ of named objects, with $x_1, \ldots, x_n$ all distinct. One writes $f : \Gamma \to B$. For each context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ and each $i \leqslant n$, there exists a projection arrow π_i from Γ to A_i. Moreover, there exists a composition operation: if $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ and Δ are contexts, and if $g_1 : \Delta \to A_1, \ldots, g_n : \Delta \to A_n, f : \Gamma \to B$ are morphisms, then there is a composition written $f \circ [x_1 \leftarrow g_1, \ldots, x_n \leftarrow g_n]$ from Δ to B. The usual monoid laws are required:

$$\begin{array}{l}
(f \circ [x_1 \leftarrow g_1, \ldots, x_n \leftarrow g_n]) \circ [\vec{y} \leftarrow \vec{h}] = f \circ [x_1 \leftarrow g_1 \circ [\vec{y} \leftarrow \vec{h}], \ldots, x_n \leftarrow g_n \circ [\vec{y} \leftarrow \vec{h}]] \\
f \circ [x_1 \leftarrow \pi_1, \ldots, x_n \leftarrow \pi_n] = f \\
\pi_i \circ [x_1 \leftarrow g_1, \ldots, x_n \leftarrow g_n] = g_i.
\end{array}$$

Proposition 3.11. Let A, T, and $_$ be fixed. Typed abstract Böhm trees form a named multi-category. The objects are the elements of T, and a morphism from Γ to σ is a typed abstract Böhm tree ϕ such that all the occurrences of moves of the form $[x, \overset{-}{\hookleftarrow}]^{\sigma}$ in ϕ are such that $x : \sigma$ is in Γ. The projection morphisms are $\pi_i = \bullet[x_i, \overset{-}{\hookleftarrow}]$, and the composition is given by

$$\phi \circ \psi = \{readback(\Gamma) \mid \longrightarrow^{\star} (\Gamma\ !)\}.$$

The right-hand side of this definition does not mention ϕ, ψ explicitly, but one should keep in mind that the transition relation $\longrightarrow$ is parametrised by ϕ and ψ.

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$$M ::= \lambda\vec{x} : \vec{\sigma}.W \qquad W ::= yM_1 \cdots M_n \quad (n \geqslant 0)\,.$$

$$\frac{\Gamma{\bullet}y = \sigma_1 \to \cdots \to \sigma_n \to C \quad \Gamma \vdash M_1 : \sigma_1 \ \cdots \ \Gamma \vdash M_n : \sigma_n}{\Gamma \vdash yM_1 \cdots M_n : C}$$

$$\frac{\Gamma, \vec{x} : \vec{\sigma} \vdash W : C}{\Gamma \vdash \lambda\vec{x} : \vec{\sigma}.W : \vec{\sigma} \to C}$$

Fig. 5. Typing rules for finite Böhm trees

Proof. We first check that $\phi \circ \psi$ is a strategy. It is non-empty by Propositions 2.16 and 3.6. If $\mathit{readback}(\Gamma) \in \phi \circ \psi$, then for each even prefix r of $\mathit{readback}(\Gamma)$, there exists a prefix Δ of Γ such that

$$\longrightarrow^\star (\Delta\ !) \longrightarrow^\star (\Gamma\ !).$$

Hence $\phi \circ \psi$ is closed under prefixes. The last condition (no branching at an opponent's move) follows from the observations made in Remark 5. The proof of associativity is tedious, but conceptually easy. Both sides $(\phi \circ [x_1 \leftarrow \psi_{x_1}, \ldots, x_n \leftarrow \psi_{x_n}]) \circ \chi$ and $\phi \circ [x_1 \leftarrow \psi_{x_1} \circ \chi, \ldots, x_n \leftarrow \psi_{x_n} \circ \chi]$ of the associativity equation are equal to the result of the three-ary composition $\phi \circ [x_1 \leftarrow \psi_{x_1}, \ldots, x_n \leftarrow \psi_{x_n}] \circ \chi$, which can be computed by a strong *VAM* parametrised by ϕ, ψ, and χ. □

4. Some examples of abstract Böhm trees

4.1. *η-long Böhm trees*

We revisit the translation of finite Böhm trees given in Section 2 in a typed setting. We restrict our attention to η-long normal forms, as defined by the following typing system. We assume a collection of basic types $C, C_1, C', \ldots$. The simple types are then generated by the following syntax:

$$\sigma ::= C \mid \sigma \to \tau.$$

The *η-long* normal forms, or finite *η-long* typed Böhm trees, or *finite typed Böhm trees* for short, are defined via typing judgements $\Gamma \vdash M : \sigma$, where Γ is a context consisting of a list of non-empty vectors $\vec{x} : \vec{\sigma}$ of typed variables. In each of these vectors $\vec{x} = x_1 \cdots x_n$, the variables $x_1, \ldots, x_n$ are supposed distinct. The following auxiliary function allows us to retrieve the type $\Gamma{\bullet}y$ of a free variable y from the context Γ:

$$(\Gamma, \vec{x} : \vec{\sigma}){\bullet}y = \begin{cases} \Gamma{\bullet}y & \text{if } y \text{ does not appear in } \vec{x} \\ \sigma_i & \text{if } y = x_i. \end{cases}$$

If $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$, we adopt the abbreviation $\vec{\sigma} \to \kappa$ for $\sigma_1 \to \cdots \to \sigma_n \to \kappa$. If $n = 0$, we then just mean κ. The typing rules are given in Figure 5. Notice that the bodies W are forced to have a basic type, that is, variables must be fully applied and terms of function types must be fully abstracted according to their type. The η-long format is essential to guarantee the completeness of the *VAM* execution (*cf.* Remark 4).

For the other end, as anticipated in Section 2, we take $A = \omega$ as alphabet, and the above set of simple types as set T of abstract types. We shall define the translation of *closed* finite typed Böhm trees. As with the untyped case, the translation function involves a function

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$$\frac{y^{\Gamma,\vec{x}:\vec{\sigma}} = [i, \overset{j}{\hookleftarrow}]^{\tau_1 \to \cdots \tau_n \to C}}{(\lambda\vec{x} : \vec{\sigma}.y\vec{M})^{\Gamma} = \{\bullet^{\vec{\sigma} \to C} y^{\Gamma,\vec{x}:\vec{\sigma}}\} \bigcup_{k=1\cdots n} \{\bullet^{\vec{\sigma} \to C} y^{\Gamma,\vec{x}:\vec{\sigma}} k^{\tau_k} r \mid \bullet^{\tau_k} r \in (M_k)^{\Gamma,\vec{x}:\vec{\sigma}}\}}$$

where

$$\frac{x = x_i \quad \vec{x} = x_1 \cdots x_k}{x^{\Gamma,\vec{x}:\vec{\sigma}} = [i, \overset{0}{\hookleftarrow}]^{\sigma_i}} \qquad \frac{x^{\Gamma} = [i, \overset{j}{\hookleftarrow}]^{\tau} \text{ and } x \text{ does not appear in } \vec{y}}{x^{\Gamma,\vec{y}:\vec{\sigma}} = [i, \overset{j+1}{\hookleftarrow}]^{\tau}}$$

Fig. 6. Compilation of typed Böhm trees

that takes a context Γ and a term M, and returns a strategy M^{Γ}. This function itself uses an auxiliary function that takes a context Γ and a variable x and returns a player's move decorated with a type. These functions are defined in Figure 6. The compilation of a closed finite typed Böhm tree M is defined as M^{nil}. A type decoration σ in M^{nil} should be read as follows:

(player's move)	σ = type of the corresponding variable
(opponent's move)	σ = type of the corresponding subterm.

Remark 6. We indicate briefly how the compilation can be extended to open typed Böhm trees. Suppose we want to compile M such that $\Gamma \vdash M : \sigma$. As a first try, we might define the compilation by M^{Γ}, but then the free variable occurrences x are not transformed into $[x, \overset{-}{\hookleftarrow}]$, but in those 'hybrid' player's moves $[i, \overset{j}{\hookleftarrow}]$ that are bound in a companion list (*cf.* Definition 3.3). In order to produce moves of the form $[x, \overset{-}{\hookleftarrow}]$, which serve to hook up strategies for composition, one may adapt the compilation of Figure 6 as follows: we should compile M as M^{nil}_{Γ}, where Γ is carried out everywhere in a dummy way except when x^{nil}_{Γ} is met, which has to be compiled to $[x, \overset{-}{\hookleftarrow}]^{\Gamma \bullet x}$.

The set of finite typed Böhm trees is in bijection with a subset of finite typed strategies for A, T, and a suitable function $(__)$. Furthermore, in order to characterise this subset, we have to impose a discipline for basic types.

Definition 4.1. Let $A = \omega$ and let T be the set of types built as terms over the signature consisting of basic type symbols $C, C_1, \ldots$ and of $\to$ of arity 2. If $\sigma = \tau_1 \to \cdots \to \tau_n \to C$, we set

$$\sigma_i = \begin{cases} \tau_i \text{ if } 1 \leqslant i \leqslant n \\ \text{undefined otherwise.} \end{cases}$$

We define the following additional operation $(_)^{\star}$ from types to basic types: $(\vec{\sigma} \to \kappa)^{\star} = \kappa$. A *typed λ strategy* is a typed strategy ϕ relative to A and T that, moreover, satisfies the following condition:

(0) If $ra^{\sigma}[\alpha, \overset{\kappa}{\hookleftarrow}]^{\tau} \in \phi$, then $\sigma^{\star} = \tau^{\star}$.

A typed λ pseudo-strategy is defined likewise as a typed pseudo-strategy satisfying Condition (0).

We next define a compilation from finite *name-free* typed λ strategies (*cf.* Definition 2.2 and Remark 6) to finite closed typed Böhm trees. The translation involves a function that

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$$\frac{\phi \rightsquigarrow \bullet^{\vec{\sigma}\to C}[i, \overset{j}{\hookleftarrow}], \phi_1, \ldots, \phi_n \quad \Gamma^{\vec{\sigma}\to C} = \Gamma, \vec{x} : \vec{\sigma}}{\phi^{\Gamma} = \lambda\vec{x} : \vec{\sigma}.[i, \overset{j}{\hookleftarrow}]^{\Gamma^{\vec{\sigma}\to C}} \phi_1^{\Gamma^{\vec{\sigma}\to C}} \cdots \phi_n^{\Gamma^{\vec{\sigma}\to C}} \cdots}$$

where

$\Gamma^{\vec{\sigma}\to C} = \Gamma, \vec{x} : \vec{\sigma}$, with $x_1, \ldots, x_n$ distinct and new

$[i, \overset{0}{\hookleftarrow}]^{\Gamma,\vec{x}:\vec{\sigma}} = x_i \quad [i, \overset{j+1}{\hookleftarrow}]^{\Gamma,\vec{x}:\vec{\sigma}} = [i, \overset{j}{\hookleftarrow}]^{\Gamma}$

Fig. 7. Compilation of typed λ strategies

takes a name-free pseudo-strategy of type σ and a context Γ and yields an (open) finite typed Böhm tree. The translation is given in Figure 7.

Definition 4.2. We say that a context Γ *agrees* with a list of types Σ if Γ and Σ have the same length, and if for each vector $\vec{x} : \vec{\tau}$ of Γ the corresponding type in the list Σ has the form $\vec{\tau} \to C$ for some C.

Proposition 4.3. The following properties hold:

1 If $\vdash M : \sigma$, then M^{nil} is a finite name-free typed λ strategy.
2 If ϕ is a finite name-free typed λ strategy of type σ, then ϕ^{nil} is a finite typed Böhm tree and $\vdash \phi^{nil} : \sigma$.
3 Moreover, these two transformations are inverse to each other.

Proof. (*hint*)
(1) We show a stronger result: if P is a subterm of M and $\Gamma \vdash P : \sigma$, then P^{Γ} is a pseudo-strategy relative to a list Σ that agrees with Γ. We prove this by induction on the size of P, and the list Σ is constructed along the induction. Let

$$(\lambda\vec{x} : \vec{\sigma}.y\vec{M})^{\Gamma} = \{\bullet^{\vec{\sigma}\to C} y^{\Gamma,\vec{x}:\vec{\sigma}}\} \bigcup_{k=1\cdots n} \{\bullet^{\vec{\sigma}\to C} y^{\Gamma,\vec{x}:\vec{\sigma}} k^{\tau_k} r \mid \bullet^{\tau_k} r \in (M_k)^{\Gamma,\vec{x}:\vec{\sigma}}\}.$$

Then, if Σ is associated with Γ, we associate $\Sigma, \vec{\sigma} \to C$ with $\Gamma, \vec{x} : \vec{\sigma}$, and the agreement is obvious by induction and by definition. Conditions (0) and (1) are immediate by construction and by the typing rules. We sketch the argument for Conditions (2) and (3). If $[i, \overset{j}{\hookleftarrow}]$ occurs in $(\lambda\vec{x} : \vec{\sigma}.y\vec{M})^{\Gamma}$, then it occurs either as the abstract head variable or in $(M_i)^{\Gamma,\vec{x}:\vec{\sigma}}$ for some i. Suppose the latter holds (the head variable case is similar). There are three cases (*cf.* the proof of Lemma 3.4):

(a) $[i, \overset{j}{\hookleftarrow}]$ occurs bound in $(M_i)^{\Gamma,\vec{x}:\vec{\sigma}}$. Then it occurs *a fortiori* bound in $(\lambda\vec{x} : \vec{\sigma}.y\vec{M})^{\Gamma}$, and Condition (2) follows from Condition (2) at M_i.
(b) $[i, \overset{j}{\hookleftarrow}]$ does not occur bound $(\lambda\vec{x} : \vec{\sigma}.y\vec{M})^{\Gamma}$. Then, contraposing from the previous case, Condition (3) follows from Condition (3) at M_i.
(c) $[i, \overset{j}{\hookleftarrow}]$ occurs bound in $(\lambda\vec{x} : \vec{\sigma}.y\vec{M})^{\Gamma}$ and does not occur bound in $(M_i)^{\Gamma,\vec{x}:\vec{\sigma}}$. Then $[i, \overset{j}{\hookleftarrow}]$ is bound to the head type $\Sigma, \vec{\sigma} \to C$ in $(M_i)^{\Gamma,\vec{x}:\vec{\sigma}}$ and is bound to $\bullet^{\vec{\sigma}\to C}$ in $(\lambda\vec{x} : \vec{\sigma}.y\vec{M})^{\Gamma}$, so Condition (2) follows from Condition (3) at M_i.

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

(2) One shows that if ϕ is a typed λ pseudo-strategy of type σ relative to Σ and if Γ agrees with Σ, then $\Gamma \vdash \phi^\Gamma : \sigma$. This is proved by induction on the size of ϕ. Conditions (0)–(3) ensure the well-typing of ϕ^Γ.

(3) Let us consider ϕ^Γ arising in the recursive computation of some ψ^{nil}. We shall prove $(\phi^\Gamma)^\Gamma = \phi$ for all such ϕ's, and hence *a fortiori* $(\psi^{nil})^{nil} = \psi$. Suppose that

$$\phi \rightsquigarrow \bullet^{\vec{\sigma}\to C}[i, \overset{j}{\hookleftarrow}], \phi_1, \ldots, \phi_n$$

and that $\Gamma^{\vec{\sigma}\to C} = \Gamma, \vec{x} : \vec{\sigma}$. Then by construction,

$$\phi^\Gamma = \lambda\vec{x} : \vec{\sigma}.[i, \overset{j}{\hookleftarrow}]^{\Gamma,\vec{x}:\vec{\sigma}} \phi_1^{\Gamma^{\vec{\sigma}\to C}} \cdots \phi_n^{\Gamma^{\vec{\sigma}\to C}} \cdots.$$

If we now analyse the construction of $(\phi^\Gamma)^\Gamma$, we recover:

— $\bullet^{\vec{\sigma}\to C}$ thanks to Condition (0),

— the correct type decoration for $[i, \overset{j}{\hookleftarrow}]$ thanks to Conditions (2) and (3), and

— $\phi_1, \ldots, \phi_n$ by induction.

The other direction $(M^\Gamma)^\Gamma$ can be handled similarly, choosing names according to the concrete syntax of M in the context extension operations. □

4.2. PCF *trees*

In this section, we adapt the syntax of Section 4.1 in order to cope with constants of basic type and case statements, and we show how to encode this extension into abstract Böhm trees, without any need to extend the target language (except for the alphabet of moves). The types are the same as in Section 4.1, and the typing judgements have the same form. The terms 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 (Hyland and Ong 1994; Abramsky *et al.* 1995). The syntax and the typing rules of PCF trees are given in Figure 8, which uses the notation $\Gamma{\bullet}y$ defined in Section 4.1. In the first rule, $v_1, \ldots, v_k : C$ should be read as: '$v_1, \ldots, v_k$ are all the constants of type C'.

Remark 7. We make the simplifying assumption that there are only finitely many constants at any given basic type. This allows us to keep our representations finite. There is no difficulty in extending the results of this section to the case of infinite basic types, like the natural numbers: one then needs to consider terms that may be horizontally infinite, that is, that may have infinite branching (at player's moves). The induction arguments on the size of terms must then be replaced by arguments on the length of a maximal branch.

We compile PCF trees much like finite typed Böhm trees.

Definition 4.4. We set

$$A = \{v \mid v : C\} \cup \{v_1 \mid v_1 : C_1\} \cup \cdots .$$

Hence the moves are either natural numbers or constants, called *questions* and *answers*, respectively. We suppose that $\{v \mid v : C\} \cap \{v_1 \mid v_1 : C_1\} = \varnothing$, for any distinct basic types

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$$M ::= \lambda \vec{x} : \vec{\sigma}.W \qquad W ::= v \mid case\ yM_1 \cdots M_n\ [v_1 \to W_1, \ldots, v_k \to W_k]\ .$$

$$\frac{v : C}{\Gamma \vdash v : C}$$

$$\frac{\Gamma \bullet y = \sigma_1 \to \cdots \to \sigma_n \to C \quad v_1, \ldots, v_k : C \quad \Gamma \vdash M_1 : \sigma_1 \ \cdots\ \Gamma \vdash M_n : \sigma_n \quad \Gamma \vdash W_1 : C_1 \ \cdots\ \Gamma \vdash W_k : C_1}{\Gamma \vdash case\ yM_1 \cdots M_n\ [v_1 \to W_1, \ldots, v_k \to W_k] : C_1}$$

$$\frac{\Gamma, \vec{x} : \vec{\sigma} \vdash W : C}{\Gamma \vdash \lambda \vec{x} : \vec{\sigma}.W : \vec{\sigma} \to C}$$

Fig. 8. Typing rules for finite PCF trees

C and C_1. Let T be as in Definition 4.1. If $\sigma = \tau_1 \to \cdots \to \tau_n \to C$, we set

$$\sigma_a = \begin{cases} \tau_i & \text{if } a = i \text{ and } 1 \leqslant i \leqslant n \\ C & \text{if } a = v \text{ and } v : C \\ \text{undefined} & \text{otherwise.} \end{cases}$$

A PCF *strategy* is a typed strategy ϕ relative to A, T, and $(_)$ that, moreover, satisfies the following conditions:

(Q) If $q[v, \overset{i}{\hookleftarrow}]^{\sigma} \in \phi$, where v is an answer, and if $[v, \overset{i}{\hookleftarrow}]$ is bound to b^{τ} in q, then b is a question.

(S) Moreover, all the opponent's moves between the binding move b and the answer must be answers.

Conditions (Q) and (S) are called the question–answer discipline and the stack discipline, respectively. A PCF pseudo-strategy is defined likewise as a typed pseudo-strategy satisfying Conditions (Q) and (S).

Remark 8. Conditions (1), (2) and (Q), together with the definition of σ_v, ensure that opponent's moves of the form v, that is, opponent's questions, are never pointed to. Suppose this is not the case, and consider a player's move that points to v. It cannot be an answer by condition (Q). Hence it is a question $[i, \overset{j}{\hookleftarrow}]$. By Condition (2), v should then be decorated with a type σ such that σ_i is defined. But by Condition (1), $\sigma = \tau_v$ for some τ, which by definition of τ_v implies $\sigma = C$. Hence σ_i is undefined, which is a contradiction. Since questions are not pointed to, it is safe to omit them in the count of opponent's moves from a player's move to its binder. Therefore, in the setting of PCF strategies (and of classical strategies), we shall *modify* the encoding of the pointers accordingly. When writing $[a, \overset{i}{\hookleftarrow}]$, we mean that there are i opponent's *questions* between $[a, \overset{i}{\hookleftarrow}]$ and its binder. As a consequence, due to condition (S), all occurrences of player's answers have the form $[v, \overset{0}{\hookleftarrow}]$.

The compilation of closed PCF trees is given via a function M^{Γ} whose definition is given in Figure 9. The following are more informal explanations. We adopt the following

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$$\frac{W \text{ has type } C}{(\lambda\vec{x} : \vec{\sigma}.W)^\Gamma = \{\bullet^{\vec{\sigma}\to C} r \mid r \in W^{\Gamma,\vec{x}:\vec{\sigma}}\}}$$

$$\frac{v : C}{v^\Gamma = \{[v, \overset{0}{\hookleftarrow}]^C\}}$$

$$\frac{v_1, \ldots, v_k : C}{(case\ y M_1 \cdots M_n\ [v_1 \to W_1 \cdots v_k \to W_k])^\Gamma \quad = \quad \{y^\Gamma\} \quad \bigcup_{i=1\cdots n} \{y^\Gamma i^{\tau_i} r \mid \bullet^{\tau_i} r \in M_i^\Gamma\} \quad \bigcup_{j=1\cdots k} \{y^\Gamma v_j^C r \mid r \in W_j^\Gamma\}}$$

where y^Γ is as in Figure 6.

Fig. 9. Compilation of PCF trees

tree representation for a PCF term M:

$$\overbrace{M_1 \quad \cdots \quad M_n \quad v_1 \to W_1 \quad \cdots \quad v_k \to W_k}^{\lambda\vec{x}.\, case\ y}$$

We call $M_1, \ldots, M_n$ the argument branches, and $v_1 \to W_1, \ldots, v_k \to W_k$ the case branches. The node '$\lambda\vec{x}.\, case\ y$' is split into the opponent's move $\bullet$ followed by a player's move $\underline{y}$ as in Section 2. An occurrence of a basic constant v is compiled as $[v, \overset{0}{\hookleftarrow}]$ (*cf.* Remark 8). Finally, we replace the head opponent's moves $\bullet$ in the translations ϕ_i and ϕ'_j of M_i and W_j by i and v_j, respectively:

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

In Figure 9, the assumption 'W has type C' can be read in either of the following two ways. We may understand the compilation as translating typing judgements, in which case the information C is immediately available, or we may retrieve C through a simple recursive exploration: we have $C = W^\star$, where

$$(case\ y\vec{M}\ [v_1 \to W_1 \cdots v_k \to W_k])^\star = W_j^\star$$

for an arbitrary choice of j, and where $v^\star = C$ if $v : C$. The compilation of a term M is defined as M^{nil}.

Example 4.5. We compile the following term:

$$(case\ f(tt)\ [ff \to tt])\ [f \leftarrow (\lambda x.case\ x\ [tt \to ff, ff \to tt])]$$

where we take a basic type of Booleans and the two constant tt and ff of this type.

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

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

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

— Multi-strategy for $[f \leftarrow (\lambda x.case\ x\ [tt \rightarrow ff, ff \rightarrow tt])]$:

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

The execution of this example by the *VAM* is as follows:

$$\begin{array}{cccccc} \bullet & f & \langle 1,2\rangle & \langle tt,3\rangle & \langle ff,2\rangle & [tt, \overset{0}{\hookleftarrow}] \\ 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

The successive views are as follows:

$$\begin{array}{lll} view_\Gamma(1) = \bullet & view_\Gamma(2) = f & view_\Gamma(3) = \bullet 1 \\ view_\Gamma(4) = f\ tt & view_\Gamma(5) = \bullet ff & \end{array}$$

The displayed terminated history corresponds to a (B) final state, which can be seen as providing an answer to the following query: what is the value returned by the application of the functional $\lambda f.case\ f(tt)\ [ff \rightarrow tt]$ to the negation function?

For PCF trees, the correspondence between final states and 'head normal forms' is as follows:

$$\begin{array}{lll} (\lambda\vec{x}.\ case\ y\vec{M}\ [\vec{v} \rightarrow \vec{W}]) & (y \text{ free}) & (F) \\ (\lambda\vec{x}.\ case\ x\vec{M}\ [\vec{v} \rightarrow \vec{W}]) & (x \text{ bound}) & (B) \\ (\lambda\vec{x}.v) & & (B)\ . \end{array}$$

Proposition 4.6. The compilation function of Figure 9 maps finite closed PCF trees to name-free PCF strategies bijectively.

Proof. (*hint*)
The proof is similar to the proof of Proposition 4.3. It goes via the definition of an inverse compilation. We do not give the details, which are rather straightforward. We just remark that the counterpart of the stack condition (S) in the type system is to be found in the typing rule for *case*, where the common type C_1 of the W_j's is 'passed' to the conclusion. □

4.3. *Classical Böhm trees*

Classical Böhm trees have been introduced in Herbelin (1997). They provide a classical version of PCF trees, using the ideas of the $\lambda\mu$-calculus. We refer to Parigot (1992) and Ong (1996) for an introduction to the $\lambda\mu$-calculus (see also the comments at the end of the section). In comparison with Böhm trees and PCF trees, classical Böhm trees incorporate two new constructs: the μ abstraction, and the naming operation $[\alpha]$. They correspond to a logical treatment of the classical absurdity, and involve a discipline of continuation variables separate from the discipline of ordinary variables. The syntax and the typing rules for classical Böhm trees are given in Figure 10. The typing judgements are of the form $\Gamma; \Delta \vdash M : \sigma$, where Γ is a context of typed variables as usual, and Δ is a context of typed continuation variables. In classical Böhm trees, all the continuation variables have

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$$M ::= \lambda\vec{x} : \vec{\sigma}.\mu\alpha.W \qquad W ::= [\alpha]v \mid case\ yM_1 \cdots M_n\ [v_1 \to W_1, \ldots, v_k \to W_k]\ .$$

$$\frac{\Delta{\bullet}\alpha = C \quad v : C}{\Gamma;\Delta \vdash [\alpha]v : \bot}$$

$$\frac{\Gamma{\bullet}y = \sigma_1 \to \cdots \to \sigma_n \to C \quad v_1, \ldots, v_k : C \quad \Gamma;\Delta \vdash M_1 : \sigma_1 \ \cdots\ \Gamma \vdash M_n : \sigma_n \quad \Gamma;\Delta \vdash W_1 : \bot \ \cdots\ \Gamma;\Delta \vdash W_k : \bot}{\Gamma;\Delta \vdash case\ yM_1 \cdots M_n\ [v_1 \to W_1, \ldots, v_k \to W_k] : \bot}$$

$$\frac{\Gamma, \vec{x} : \vec{\sigma};\Delta, \alpha : C \vdash W : \bot}{\Gamma \vdash \lambda\vec{x} : \vec{\sigma}.\mu\alpha.W : \vec{\sigma} \to C}$$

Fig. 10. Typing rules for classical Böhm trees

$$\overline{(\lambda\vec{x} : \vec{\sigma}.\mu\alpha : C.W)^{\Gamma;\Delta} = \{\bullet^{\vec{\sigma}\to C} r \mid r \in W^{\Gamma,\vec{x}:\vec{\sigma};\Delta,\alpha:C}\}}$$

$$\frac{v : C}{([\alpha]v)^{\Gamma;\Delta} = \{[v, \overset{\alpha^\Delta}{\hookleftarrow}]^C\}}$$

where $\alpha^{\Delta,\beta:C} = \begin{cases} \alpha^\Delta + 1 \text{ if } \alpha \neq \beta \\ 0 \text{ if } \alpha = \beta \end{cases}$

$$\frac{y^{\Gamma,\vec{x}:\vec{\sigma}} = [i, \overset{j}{\hookleftarrow}]^{\tau_1\to\cdots\tau_n\to C} \quad v_1, \ldots, v_k : C}{(case\ y\vec{M}\ [v_1 \to W_1 \cdots v_k \to W_k])^\Gamma \quad = \quad \{y^{\Gamma,\vec{x}:\vec{\sigma}}\} \quad \bigcup_{i=1\cdots n} \{y^{\Gamma,\vec{x}:\vec{\sigma}} i^{\tau_i} r \mid \bullet^{\tau_i} r \in M_i^{\Gamma,\vec{x}:\vec{\sigma}}\} \quad \bigcup_{j=1\cdots k} \{y^{\Gamma,\vec{x}:\vec{\sigma}} v_j^C r \mid r \in W_j^{(\Gamma,\vec{x}:\vec{\sigma})}\}}$$

where y^Γ is as in Figure 6.

Fig. 11. Compilation of classical Böhm trees

a basic type. In the figure, the notation $\Delta{\bullet}\alpha$ is similar to the notation $\Gamma{\bullet}y$, and is defined by

$$(\Delta, \beta : C){\bullet}\alpha = \begin{cases} \Delta{\bullet}\alpha & \text{if } \alpha \neq \beta \\ C & \text{if } \alpha = \beta\ . \end{cases}$$

One should read $v_1, \ldots, v_k : C$ as '$v_1, \ldots, v_k$ are all the constants of type C' (*cf.* Figure 8). All the bodies W receive a special type $\bot$, which stands for absurdity.

Definition 4.7. Let A, T, and $(__)$, be as in Definition 4.4. A *classical strategy* is a typed strategy ϕ relative to A and T that, moreover, satisfies the question–answer discipline (Condition (Q) of Definition 4.4).

In other words, classical strategies are the same as PCF strategies, except that the stack condition (S) has been removed.

Proposition 4.8. The compilation function of Figure 11 maps bijectively finite closed classical Böhm trees to finite name-free classical strategies.

We can summarise the transition from Böhm trees to classical Böhm trees through PCF trees as follows:

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

— In order to characterise typed Böhm trees, we have to impose an explicit basic type discipline (Condition (0) of Definition 4.1). This condition becomes implied as soon as values, or answers are introduced (both in PCF trees and classical Böhm trees). Answers allow us to encode the continuations of the computations using pointers: they are bound, just as variables are.

— In order to characterise PCF Böhm trees, we have to impose a discipline for answers:

1 Answers are bound to questions.
2 Moreover, this binding forces us to answer the last pending question (stack discipline).

— If the stack discipline is removed, we have the classical Böhm trees.

From this analysis we conclude that classical Böhm trees provide the best fit for our abstract computational setting of abstract Böhm trees. Condition (0) is an *ad hoc* condition. The stack discipline is certainly not *ad hoc*, since it sets the boundary between intuitionistic and classical. But from the computational point of view, it makes no difference whether we have it or not.

In the rest of this section, we substantiate rather informally the claim that classical Böhm trees are to '$\lambda\mu$+PCF' what PCF trees are to PCF. We first briefly describe the version of Parigot's $\lambda\mu$-calculus (Parigot 1992) given by Hofmann and Streicher (Hofmann and Streicher 1997) (which is itself closely inspired by Ong's version (Ong 1996)). Let T be the collection of types considered so far, the basic types being now looked at as propositional atoms. There is also a special type $\perp$, which Hofmann and Streicher allow in T (that is, for example, $\sigma \to \perp \in T$), but we shall leave it apart. The terms of the $\lambda\mu$-calculus are given by the following syntax:

$$M ::= x \mid MN \mid \lambda x : \sigma.M \mid \mu\alpha : A.M \mid [\alpha]M.$$

where x ranges over term variables as usual, and where α ranges over a disjoint set of continuation variables. Typing judgements have the form $\Gamma;\Delta \vdash M : \sigma$, where Δ is a list of typed continuation variables $\alpha : \sigma$, with $\sigma \neq \perp$. The judgements are derived by the following rules:

$$\frac{x : \sigma \text{ in } \Gamma}{\Gamma;\Delta \vdash x : \sigma} \qquad \frac{\Gamma, x : \sigma;\Delta \vdash M : \tau}{\Gamma;\Delta \vdash \lambda x : \sigma.M : \sigma \to \tau} \qquad \frac{\Gamma;\Delta \vdash M : \sigma \to \tau \qquad \Gamma;\Delta \vdash N : \sigma}{\Gamma;\Delta \vdash MN : \tau}$$

$$\frac{\Gamma;\Delta, \alpha : \sigma \vdash M : \perp}{\Gamma;\Delta \vdash \mu\alpha : \sigma.M : \sigma} \qquad \frac{\Gamma;\Delta \vdash M : \sigma \qquad \alpha : \sigma \text{ in } \Delta}{\Gamma;\Delta \vdash [\alpha]M : \perp}$$

In order to understand these rules, one should read $\alpha : \sigma$ as an assumption that the negation $\sigma^{\perp}$ of σ holds. Then the typing rules for μ-abstractions $\mu\alpha : \sigma.M$ and for named terms $[\alpha]M$ read as absurdity introduction and elimination rules. The rewriting rules of

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

the $\lambda\mu$-calculus are the following [†]:

$$\begin{array}{ll}(\beta) & (\lambda x : \sigma.M)N = M[N/x] \\ (\mu-\zeta) & \mu\alpha : \sigma \to \tau.\, M = \lambda x : \sigma.\, \mu\beta : \tau.\, M[x :: \beta/\alpha] \\ (\mu-\beta) & [\alpha](\mu\beta : \sigma.M) = M[\alpha/\beta].\end{array}$$

In these rules, three substitution operations are involved. The first is the ordinary substitution in the λ-calculus. The third is the substitution of (all free occurrences of) the continuation variable β by α. The second consists of spotting all subterms of the form $[\alpha]N$ of M and replacing them by $[\beta](Nx)$.

In addition to these rules, we consider an expansion rule for a term M of type $\sigma \neq \bot$:

$$(\mu-\eta) \quad M \to \mu\alpha : \sigma.\, [\alpha]M$$

where α is new. It is easy to see that this rule, together with the other rules, implies the η-rule $N = \lambda x : \sigma.\, Nx$ for any term N of a non-basic type $\sigma \to \tau$. The following properties are easy to check:

1 There cannot be a well-typed term of the form $\mu\alpha : \sigma.\mu\beta : \tau.M$ nor of the form $[\alpha][\beta]M$. For example, $\mu\beta : \tau.\, M$ has type $\tau \neq \bot$, but should have type $\bot$ if $\mu\alpha : \sigma.\, \mu\beta : \tau.\, M$ were it to be well typed.
2 Under the assumption $\bot \notin T$, a term of type $\bot$ can only be of the form $[\beta]M$.
3 In a normal form, all the continuation variables have a basic type, thanks to the rule $(\mu-\zeta)$.

Let us analyse informally an expanded $\lambda\mu$ normal form M, by which we mean the $\lambda\mu$ version of η-long finite Böhm trees. Making use, if needed, of the expansion rule $(\mu-\eta)$ followed by the rule $(\mu-\zeta)$, we see that M has the form $\lambda\vec{x} : \vec{\sigma}.M_1$, where M_1 is of a basic type, which is $\neq \bot$, since we assume $\bot \notin T$ here. Possibly using the expansion rule, we have $M_1 = \mu\alpha : C.M_2$ for some α and C. By Property (2) above, M_2 must be of the form $[\beta]M_3$, where by Property (3), M_3 is of basic type. The first symbol of M_3 cannot be a $[\gamma]$ by Property (1), nor a λ because of the type, nor a $\mu\gamma$, as otherwise there would be a $(\mu-\beta)$ redex. It is thus either a variable x or an aplication M_4M_5; since M_4 has a non-basic type, by Property (3) it cannot start with a $\mu\gamma$, nor with a λ, as otherwise there would be a β-redex, nor with a $[\gamma]$, because of the type. Continuing in this way, we see that M_3 has the form $xM_n \cdots M_5$. Summarising, the expanded normal forms have (recursively) the form

$$\lambda\vec{x} : \vec{\sigma}.\, \mu\alpha : C.\, [\beta](x\vec{M})$$

where β has a basic type.

If we now incorporate the basic constants and the case construct, we arrive at the classical Böhm trees of Figure 10, thanks to the following additional rule that allows to distribute the naming operator $[\alpha]$ all the way down to the constants:

$$[\alpha](case\ y\vec{M}\ [v_1 \to W_1, \ldots, v_k \to W_k]) \to (case\ y\vec{M}\ [v_1 \to [\alpha]W_1, \ldots, v_k \to [\alpha]W_k]).$$

[†] The rule $(\mu-\zeta)$ in the case $B = \bot$ is slightly different, since continuation variables of type $\bot$ are not allowed, see Hofmann and Streicher (1997).

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

Remark 9. The bijective correspondences stated in Propositions 4.3, 4.6 and 4.8 can be extended without difficulty to order-isomorphisms by allowing a special player's move $[\Omega, \overleftarrow{\hookrightarrow}]$. We indicate briefly how this works. When we compile a partial term, we should include explicit player's moves $[\Omega, \overleftarrow{\hookrightarrow}]$ corresponding to each of the occurrences of Ω. For example, $(\lambda x.x\Omega(y\Omega))^{nil}$ contains $\bullet[1, \overset{0}{\overleftarrow{\hookrightarrow}}]1[\Omega, \overleftarrow{\hookrightarrow}]$ and $\bullet[1, \overset{0}{\overleftarrow{\hookrightarrow}}]2[y, \overleftarrow{\hookrightarrow}]1[\Omega, \overleftarrow{\hookrightarrow}]$. Here is how we justify that the composition operation is monotonic (and hence Scott-continuous, being defined by means of finitary rules). First, we need a minor adjustment in the setting of the *VAM*. The special 'free player's move' $[\Omega, \overleftarrow{\hookrightarrow}]$ is allowed to occur in ϕ, without being bound in ψ. Hence we allow for (F) final states in ϕ that correspond to having reached an Ω of ϕ. Next, we define the order relation between strategies (and similarly between multi-strategies) as follows: $\phi \leqslant \phi'$ if each position of ϕ' is either in ϕ, or has a query prefix q such that $q[\Omega, \overleftarrow{\hookrightarrow}] \in \phi$. Suppose now that $\phi \leqslant \phi'$ and $\psi \leqslant \psi'$, and that $r \in \phi' \circ \psi'$. The corresponding computation of the strong *VAM* has explored parts of ϕ' and ψ'. If those parts are also present in ϕ and ψ, then r is also in $\phi \circ \psi$. Otherwise, we run two executions of the strong *VAM* in parallel, the first for ϕ and ψ, the second for ϕ' and ψ', both executions being guided by r (*cf.* Remark 5). The machines have the same history up to some step at which the view q in ϕ' and ψ' is answered by Ω in ϕ or ψ, and by something else in ϕ' or ψ'. The first execution then returns an answer Ω to the current query, which is a prefix of r. Hence $\phi \circ \psi \leqslant \phi' \circ \psi'$.

4.4. *Novikov's logic*

In this section, we present examples taken from Novikov's classical infinitary propositional logic. The basic formulas of this logic are of the form $\exists\, x\ P$ (or $\exists\, \vec{x}\ P$), where P is a predicate over natural numbers, like $x \leqslant y$ or $f(x) \leqslant f(y)$ (where f is a function from natural numbers to natural numbers whose definition is given as an oracle outside the logic). The (existential) formulas are then built by the following rule:

$$\text{If } A \text{ is a formula, then } \exists\, x\ \forall\, y\ A \text{ is a formula.}$$

In this logic, one proves disjunctions of formulas, represented as sequences $\Gamma = A_1, \ldots, A_{n+1}$. Proofs are formulated as strategies. At the beginning, the opponent objects the whole disjunction. Then the player chooses to justify one of the disjuncts, say $A_{n+1} = \exists\, x\ \forall\, y\ A(x, y)$. To materialise his justification, the player also chooses a particular x_0. Then the proof of the formula amounts to proving each of the following disjunctions:

$$A_1, \ldots, A_n, A_{n+1}, A(x_0, i)$$

for $i = 0, 1, 2, \ldots$. Notice that we have kept A_{n+1}, because, in classical reasoning, it might be the case that the player discovers that he was wrong in his choice of x_0, and that another choice would have been wiser.

A proof of a universal quantification $\forall\, z\ A(z)$ is represented as an infinite multi-strategy $[\ldots, m \leftarrow A(m), \ldots]$. All such proofs can be formulated as trees with pointers, and the (Strong) *VAM* can be used for cut-elimination. More precisely, in the examples presented below,

— the strategy proves a disjunction $(\exists\, x\ \forall\, y\ A(x, y)), \Gamma$,

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

— the multi-strategy provides proofs of $\exists\, y\ \neg A(m, y)$ for all m, and
— the cut-elimination, or composition, yields a (cut-free) proof of Γ.

The alphabet for Novikov's calculus is $A = \omega$. The set T of abstract types is defined as follows (in first approximation):

$$T = \{A \mid A \text{ is a formula}\} \cup \{\forall\, y\ A \mid A \text{ is a formula}\}.$$

We set

$$(\forall\, y\ A)_i = A(i) \qquad (\exists\, x\ B)_j = B(j)\ .$$

We shall not give a formal statement that characterises the proofs of Novikov's logic as typed strategies relative to A, T and $(_)$, but it should be clear that such a bijection exists (see also remark 10). We shall just present two examples, taken from Coquand (1995) and from Berardi *et al.* (1994), respectively. In Example 4.9, given a function f from natural numbers to natural numbers, we show how to construct three numbers u_1, u_2, u_3 such that $f(u_1) \leqslant f(u_2) \leqslant f(u_3)$, relying on a lemma that asserts the existence of relative minima (the formula $K(x)$ below). In Example 4.10, we show a dialogue interpretation of the Axiom of Choice: given a predicate $P(x, y)$ and a multi-strategy witnessing $(\exists\, x\ P(m, x))$ for all m, we show how the interaction with the axiom of choice results in the progressive construction of f such that $P(n, f(n))$ (for all n in the domain of definition of f).

Example 4.9.
— Strategy for

$$L = (\exists\, x\ \forall\, y \geqslant x\ \exists\, z \geqslant x\ f(y) > f(z))\ ,\ (\exists\, u_1 < u_2 < u_3\ f(u_1) \leqslant f(u_2) \leqslant f(u_3))\ :$$

$$\bullet\ \underset{[0, \overset{-}{\hookleftarrow}]}{\boxed{\exists x}} \left\{ y_1\ \underset{[y_1 + 1, \overset{-}{\hookleftarrow}]}{\boxed{\exists x}} \left\{ \begin{array}{l} \dfrac{f(y_1) > f(y_2)}{y_2}\ \underset{[y_2, \overset{1}{\hookleftarrow}]}{\boxed{\exists z}} \\[2ex] \dfrac{f(y_1) \leqslant f(y_2)}{y_2}\ \underset{[y_2 + 1, \overset{-}{\hookleftarrow}]}{\boxed{\exists x}} \left\{ \begin{array}{l} \dfrac{f(y_2) > f(y_3)}{y_3}\ \underset{[y_3, \overset{1}{\hookleftarrow}]}{\boxed{\exists z}} \\[2ex] \dfrac{f(y_2) \leqslant f(y_3)}{y_3}\ \underset{[y_1, y_2, y_3, \overset{3}{\hookleftarrow}]}{\boxed{\exists u_1 < u_2 < u_3}} \end{array} \right. \end{array} \right. \right.$$

— Multi-strategy for $K(x) = \exists\, y \geqslant x\ \forall\, z \geqslant x\ f(y) \leqslant f(z)$:

$$x\ \underset{[x, \overset{0}{\hookleftarrow}]}{\boxed{\exists y}}\ \frac{f(x) > f(z_1)}{z_1}\ \underset{[z_1, \overset{1}{\hookleftarrow}]}{\boxed{\exists y}}\ \frac{f(z_1) > f(z_2)}{z_2}\ \underset{[z_2, \overset{2}{\hookleftarrow}]}{\boxed{\exists y}}\ \cdots$$

We explain briefly how to read the strategy for L. The player chooses to justify the first disjunct of L, with $x = 0$. This is just to let the opponent pick a value $y_1 \geqslant 0$ for y. Next the player retries the first disjunct, but now with $x = y_1 + 1$. The opponent must then choose $y_2 > y_1$. The player has access to the oracle defining f. If $f(y_1) > f(y_2)$, the player plays $z = y_2$, and he has justified the first disjunct. Otherwise, he retries the first disjunct from the very beginning, with $x = y_2 + 1$. The opponent chooses $y_3 > y_2$. If $f(y_2) > f(y_3)$, the player can justify the first disjunct by playing $z = y_3$. Otherwise, we have $f(y_1) \leqslant f(y_2) \leqslant f(y_3)$, and the player justifies the second disjunct by playing

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

$u_1, u_2, u_3 = y_1, y_2, y_3$. We have omitted the type decorations in the above strategy and multi-strategy. Here are the typed positions corresponding to the first branch and to the third branch of the proof of L:

$$\begin{array}{ll} \bullet^{\exists\, u_1<u_2<u_3\ f(u_1)\leqslant f(u_2)\leqslant f(u_3)} & [0, \overset{-}{\hookleftarrow}]^{\forall\, y\geqslant 0\ \exists\, z\geqslant 0\ f(y)>f(z)} \\ y_1^{\exists\, z\geqslant 0\ f(y_1)>f(z)} & [y_1+1, \overset{-}{\hookleftarrow}]^{\forall\, y\geqslant y_1+1\ \exists\, z\geqslant y_1+1\ f(y)>f(z)} \\ y_2^{\exists\, z\geqslant y_1+1\ f(y_2)>f(z)} & [y_2, \overset{1}{\hookleftarrow}]^{f(y_1)>f(y_2)} \end{array}$$

$$\begin{array}{ll} \bullet^{\exists\, u_1<u_2<u_3\ f(u_1)\leqslant f(u_2)\leqslant f(u_3)} & [0, \overset{-}{\hookleftarrow}]^{\forall\, y\geqslant 0\ \exists\, z\geqslant 0\ f(y)>f(z)} \\ y_1^{\exists\, z\geqslant 0\ f(y_1)>f(z)} & [y_1+1, \overset{-}{\hookleftarrow}]^{\forall\, y\geqslant y_1+1\ \exists\, z\geqslant y_1+1\ f(y)>f(z)} \\ y_2^{\exists\, z\geqslant y_1+1\ f(y_2)>f(z)} & [y_2+1, \overset{-}{\hookleftarrow}]^{\forall\, y\geqslant y_2+1\ \exists\, z\geqslant y_2+1\ f(y)>f(z)} \\ y_3^{\exists\, z\geqslant y_2+1\ f(y_3)>f(z)} & [y_1, y_2, y_3, \overset{3}{\hookleftarrow}]^{f(y_1)\leqslant f(y_2)\leqslant f(y_3)} \end{array}$$

For K, we give only a typed position of length 4 (with $x = 0$, $z_1 = 1$):

$$\begin{array}{ll} 0^{\exists\, y\geqslant 0\ \forall\, z\geqslant 0\ f(y)\leqslant f(z)} & [0, \overset{0}{\hookleftarrow}]^{\forall\, z\geqslant 0\ f(0)\leqslant f(z)} \\ 1^{f(0)\leqslant f(1)} & [1, \overset{1}{\hookleftarrow}]^{\forall\, z\geqslant 0\ f(1)\leqslant f(z)} \end{array}$$

Given f such that

$$f(0) = 10, f(1) = 5, f(2) = 3, f(3) = 7, f(4) = 4, f(5) = 11, f(6) = 29,$$

the history of the proof of $\exists\, u_1 < u_2 < u_3\ f(u_1) \leqslant f(u_2) \leqslant f(u_3)$ by cut-elimination performed by the *VAM* is as follows.

$\bullet$	0	$\langle 0,2\rangle$	1	$\langle 1,4\rangle$	$\langle 1,3\rangle$	$\langle 1,2\rangle$	2	$\langle 2,8\rangle$	$\langle 2,7\rangle$
1	2	3	4	5	6	7	8	9	10

$\langle 2,2\rangle$	3	$\langle 3,12\rangle$	4	$\langle 4,14\rangle$	$\langle 4,13\rangle$	$\langle 4,12\rangle$	5	$\langle 5,18\rangle$	$[2,4,5,\overset{0}{\hookleftarrow}]$
11	12	13	14	15	16	17	18	19	20

The successive views are as follows:

$$\begin{array}{lll} view_\Gamma(1) = \bullet & view_\Gamma(2) = 0 & view_\Gamma(3) = \bullet 0 \\ view_\Gamma(4) = 1 & view_\Gamma(5) = \bullet 01 & view_\Gamma(6) = 01 \\ view_\Gamma(7) = \bullet 1 & view_\Gamma(8) = 2 & view_\Gamma(9) = \bullet 12 \\ view_\Gamma(10) = 012 & view_\Gamma(11) = \bullet 2 & view_\Gamma(12) = 3 \\ view_\Gamma(13) = \bullet 23 & view_\Gamma(14) = 4 & view_\Gamma(15) = \bullet 234 \\ view_\Gamma(16) = 34 & view_\Gamma(17) = \bullet 24 & view_\Gamma(18) = 5 \\ view_\Gamma(19) = \bullet 245 & & \end{array}$$

Remark 10. The above type decorations do not fit exactly with the framework of the typed *VAM*: if ϕ is the strategy associated to L, then $\phi(\bullet) = [0, \overset{-}{\hookleftarrow}]^{\forall\, y\geqslant 0\ \exists\, z\geqslant 0\ f(y)>f(z)}$, while the first move played in the multi-strategy is $0^{\exists\, y\geqslant 0\ \forall\, z\geqslant 0\ f(y)\leqslant f(z)}$. What we need to do to repair this mismatch is to take as the set of abstract types a quotient of the set of formulas by de Morgan duality.

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

Example 4.10.

— Strategy for $AC = (\exists m \, \forall x \, \overline{P}(m,x)) \, , \, (\exists f \, \forall n \, P(n, f(n)))$:

$$\bullet \, [\bot, \overset{0}{\hookleftarrow}] \left\{ \begin{array}{l} n^0 \, [n^0, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} x^0 \, [\begin{pmatrix} x^0 \\ n^0 \end{pmatrix}, \overset{2}{\hookleftarrow}] \left\{ \begin{array}{l} n^1 \, [n^1, \overset{-}{\hookleftarrow}] \left\{ \begin{array}{l} x^1 \, [\begin{pmatrix} x^0 & x^1 \\ n^0 & n^1 \end{pmatrix}, \overset{4}{\hookleftarrow}] \ldots \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \right.$$

where $\begin{pmatrix} x^0 \\ n^0 \end{pmatrix}, \begin{pmatrix} x^0 & x^1 \\ n^0 & n^1 \end{pmatrix}, \ldots$ are the (partial) functions defined by:

$$\begin{pmatrix} x^0 \\ n^0 \end{pmatrix}(n^0) = x^0$$
$$\begin{pmatrix} x^0 & x^1 \\ n^0 & n^1 \end{pmatrix}(n^0) = x^0 \quad \begin{pmatrix} x^0 & x^1 \\ n^0 & n^1 \end{pmatrix}(n^1) = x^1$$
$$\vdots$$

— Multi-strategy for $(\exists x \, P(m,x))$:

$$\left\{ \begin{array}{l} \vdots \\ m[x_m, \overset{0}{\hookleftarrow}] \\ \vdots \end{array} \right.$$

This example requires strong reduction: the demand for more values of f must come from the user. Thus we describe the execution in the framework of the Strong *VAM*. The first 10 steps of (generic) execution are as follows:

$$\underset{1}{\underline{\bullet_v}} \qquad \underset{2}{\underline{[\bot, \overset{v}{\hookleftarrow}]}} \qquad \underset{3}{\underline{n^0_{v^0}}} \qquad \underset{4}{n^0} \qquad \underset{5}{\langle x_{n^0}, 4 \rangle}$$

$$\underset{6}{\underline{[\begin{pmatrix} x_{n^0} \\ n^0 \end{pmatrix}, \overset{v}{\hookleftarrow}]}} \qquad \underset{7}{\underline{n^1_{v^1}}} \qquad \underset{8}{n^1} \qquad \underset{9}{\langle x_{n^1}, 8 \rangle} \qquad \underset{10}{\underline{[\begin{pmatrix} x_{n^0} & x_{n^1} \\ n^0 & n^1 \end{pmatrix}, \overset{v}{\hookleftarrow}]}}$$

with the following successive views:

$$\begin{array}{lll} view_\Gamma(1) = \bullet & & view_\Gamma(3) = \bullet n^0 \\ view_\Gamma(4) = n^0 & view_\Gamma(5) = \bullet n^0 x_{n^0} & \\ view_\Gamma(7) = \bullet n^0 x_{n^0} n^1 & view_\Gamma(8) = n^1 & view_\Gamma(9) = \bullet n^0 x_{n^0} n^1 x_{n^1} \end{array}$$

5. Conclusion

Abstract Böhm trees seem to provide an adequate level of abstraction for the description of abstract machines for functional languages (and some of their extensions, such as those with control operators) and for cut-elimination in a range of logical proof systems. They embody just as much structure as is needed to describe two sorts of interaction: tree

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

interaction and pointer interaction (*cf.* Remark 2). These two forms of interaction are not only present in calculi such as those presented as examples in Section 4, they also suffice to account for the whole evaluation mechanism (restricted to η-long terms) of these calculi.

Developing a theory at this level of abstraction allows us to direct more efforts towards compilation than towards execution. For example, by compiling PCF trees or classical Böhm trees in the language of abstract Böhm trees, we have freed ourselves from the need to design extensions of the execution model: the same abstract machine is prepared to handle any kind of abstract Böhm trees.

Here we have only presented one abstract machine (and its strong extension). It turns out that several abstract machines in the literature (including a call-by-name environment machine in the style of Landin's SECD machine) can be expressed at this level of abstraction (Curien and Herbelin 1997), and can be inter-translated, yielding step-to-step correspondences. Hence the framework of abstract Böhm trees can contribute to the expression of what these abstract machines have in common, and how they differ (in data representation, or in memory management).

We would like to stress that the role played by types in this study is more typical of concurrency theory than of type theory. We have seen in Section 2 that the abstract machine always terminates. We do not use types to prevent non-termination, but to ensure that the termination is satisfactory (no dead-lock). The non-termination, say, of the untyped λ-calculus, is then explained by the fact that untyped reduction infinitely expands the original term behind the scenes (*cf.* Remark 4). We formulate the following question:

Is there a characterisation (for example, by some type system in the style of Coppo–Dezani's intersection types) of the untyped λ-terms for which the VAM execution succeeds, that is, ends in a legal final state? More informally, among the terms possessing a head normal form, which ones can reach it without implicit η-expansions?

It would also be interesting to extend our approach to cover more sophisticated type systems such as second-order types. This is certainly possible, at the price of a sophistication of the basic abstract machine presented here, that seems inherently linked with simple types.

References

Abadi, M., Cardelli, L., Curien, P.-L. and Lévy, J. J. (1991) Explicit substitutions. *Journal of Functional Programming* **1** (4) 375–416.

Abramsky, S., Jagadeesan, R. and Malacaria, P. (1995) Full abstraction for PCF (available by ftp at `theory.doc.ic.ac.uk/papers`).

Amadio, R. and Curien, P.-L. (1997) Domains and Lambda-calculi (to appear).

Berardi, S., Bezem, M. A. and Coquand, T. (1994) On the computational content of the axiom of choice, Technical Report of Utrecht University.

Coquand, T. (1995) A semantics of evidence for classical arithmetic, revised version. *Journal of Symbolic Logic* **60** 325–337.

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press

Curien, P.-L. (1991) An abstract framework for environment machines. *Theoretical Computer Science* **82** 389–402.

Curien, P.-L. and Herbelin, H. (1997) Computing with Abstract Böhm trees (submitted).

Danos, V., Herbelin, H. and Regnier, L. (1996) Game semantics and abstract machines. *Proc. LICS 96*.

Danos, V. and Regnier, L. (1990) Machina ex deo, ou encore quelque chose à dire sur la machine de Krivine (unpublished).

Felscher, W. (1986) Dialogues as a foundation of intuitionistic logic. *Handbook of Philosophical Logic* **3** 341–372.

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

Herbelin, H. (1997) Games and weak head reduction for classical PCF. *Proc. TLCA 96*.

Huet, G. (1993) An analysis of Böhm's theorem. *Theoretical Computer Science* **121** 145–167.

Hofmann, M. and Streicher, T. (1997) Continuation models are universal for $\lambda\mu$-calculus (submitted).

Hyland, M., Ong, L. (1994) On full abstraction for PCF (available by ftp at `ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Luke.Ong`).

Milner, R. (1977) Fully abstract models of typed lambda-calculi. *Theoretical Computer Science* **4**.

Nickau, H. (1996) *Hereditary sequential functionals: a game-theoretic approach to sequentiality*, Dissertation, Universität GH Siegen, Shaker-Verlag.

Ong, L. (1996) A semantic view of classical proofs: type-theoretic, categorical, and denotational characterisations. *Proc. LICS 96*.

Parigot, M. (1992) $\lambda\mu$-calculus, an algorithmic interpretation of classical natural deduction. In Proc. LPAR 92. *Springer-Verlag Lecture Notes in Computer Science* **624**.

Plotkin, G. (1977) LCF as a programming Language. *Theoretical Computer Science* **5**.

https://doi.org/10.1017/S0960129598002631 Published online by Cambridge University Press