

## Regular Böhm trees

GÉRARD HUET

INRIA-Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France  
Email: Gerard.Huet@inria.fr

Received 16 December 1996; revised 12 March 1998

We give a decision procedure for the extensional equality of total Böhm trees presented by regular systems of recursion equations.

### 1. Böhm trees presentations

Böhm trees are the natural infinite generalisations of normal forms in pure  $\lambda$ -calculus. They arose from the work of Böhm on separability (Böhm 1968), and were first identified by Barendregt, who devotes Chapter 10 of his book Barendregt (1980) to their study, and relates denotational models such as  $D_\infty$  to appropriate quotients over Böhm trees.

There is, however, no generally agreed presentation of Böhm trees, and the various partial orderings considered on them make this topic a difficult one (Lévy 1993). Here we will adopt the point of view that Böhm trees are potentially infinite computational objects similar to the streams studied in the theory of communicating processes, and treat them accordingly as maximal solutions to systems of recursive definitions.

#### 1.1. Systems of guarded combinators

**Definitions.** We assume that we are given two disjoint denumerable alphabets of symbols:  $\mathcal{X} = \{X_1, X_2, \dots\}$  is the set of *combinator* symbols;  $\mathcal{U} = \{u_1, u_2, \dots\}$  is the set of *parameter* symbols. Intuitively, combinators name Böhm trees, whereas parameters name bound  $\lambda$ -variables.

We define *Böhm tree presentation* with respect to these two alphabets to be any denumerable system of guarded equations:  $\mathcal{E} = \{E_1, E_2, \dots\}$ , with

$$E_i : X_i u_1 u_2 \cdots u_{n_i} := u_{k_i}(M_1, \dots, M_{p_i})$$

where  $1 \leq k_i \leq n_i$ ,  $0 \leq p_i$ ,  $X_i \in \mathcal{X}$ , and for  $\forall j \leq p_i$   $M_j = X_{k_{i,j}}(v_1, \dots, v_{l_{i,j}})$  with  $1 \leq k_{i,j} \leq n_i$  and  $\{v_1, \dots, v_{l_{i,j}}\} \subseteq \{u_1, \dots, u_{n_i}\} \subseteq \mathcal{U}$ .

We assume, furthermore, that the system is *deterministic*, in the sense that every  $X \in \mathcal{X}$  possesses at most one defining equation in  $\mathcal{E}$ . We shall then define the *arity* of  $X_i$  in  $\mathcal{E}$  to be the natural number  $n_i$ , and we shall denote it by  $\text{ar}_{\mathcal{E}}(X_i)$ . We say that it is *total* when every  $X \in \mathcal{X}$  possesses exactly one defining equation in  $\mathcal{E}$ .

1.2. Completeness

Whatever your favorite formalism for Böhm trees, you ought to be able to convince yourself that any Böhm tree  $T$  containing only a finite number of free variables  $\{v_1, \dots, v_m\}$  is definable as  $X(v_1, \dots, v_m)$  in a suitable Böhm tree presentation. Indeed, if  $T$  is the undefined Böhm tree, define it as  $\perp$  in the presentation  $\emptyset$  over  $\mathcal{X} = \{\perp\}$ ,  $\mathcal{U} = \emptyset$ . If  $T$  is of the form  $\lambda u_1 u_2 \cdots u_n \cdot w(T_1, \dots, T_p)$  and its set of free variables is  $\{v_1, \dots, v_m\}$ , then let  $T_j$  be represented by  $X_j(v_{j,1}, \dots, v_{j,m})$  in  $\mathcal{E}_j$  for  $1 \leq j \leq p$ , where we assume that the parameter sets  $\mathcal{X}_j$  of the  $\mathcal{E}_j$ 's are disjoint. Note that  $v_{j,1}, \dots, v_{j,m}$  are included in  $v_1, \dots, v_m, u_1, \dots, u_n$ . Now choose a new parameter  $X_0$ , and consider the new equation

$$E_0 : X_0 v_1 \cdots v_m u_1 u_2 \cdots u_n := w(M_1, \dots, M_p)$$

with  $M_j = X_j(v_{j,1}, \dots, v_{j,m})$ . Now with  $\mathcal{E}_0 = \{E_0\}$  and  $\mathcal{X}_0 = \{X_0\}$ , we consider the presentation  $\bigcup_{i=0}^p \mathcal{E}_i$  over  $\mathcal{X} = \bigcup_{i=0}^p \mathcal{X}_i$ . In this presentation,  $T$  may indeed be defined as  $X_0(v_1, \dots, v_m)$ . This construction may be made rigorous as a limit construction, defining  $T$  as the ideal of its finite approximations, as usual.

Let us remark at this point that this shows that Böhm tree presentations are general enough to represent arbitrary families of finitely generated Böhm trees, which is enough, for instance, to represent the Böhm trees of any  $\lambda$ -term. But they permit us to do more, in that we may represent dags and looping structures.

For instance, the  $\lambda$ -term in normal form  $\lambda u_1 u_2 \cdot (u_1 \lambda v \cdot (v v) \lambda w \cdot w)$  may be presented as  $X$  in the system

$$\begin{aligned} X u_1 u_2 &:= u_1(D, I) \\ D v &:= v(I(v)) \\ I w &:= w \end{aligned}$$

with sharing of the combinator  $I$ . Whereas the single equation  $Z u := u(Z)$  defines as  $Z$  the infinite tree  $\lambda u_1 \cdot u_1(\lambda u_2 \cdot u_2(\dots))$ . Another example is the fixpoint combinator  $Y$ , with  $Y f := f(Y(f))$ . Yet another example, also denoting an infinite Böhm tree, is  $J$  presented by the system:  $J x y := x(J(y))$ . It is the Böhm tree of the  $\lambda$ -term  $(\mathbf{Y} \lambda j \lambda x \lambda y (x (j y)))$ , for  $\mathbf{Y}$  any fixpoint combinator such as Curry's.

Indeed, we will call any finitely definable Böhm tree regular (in analogy with regular languages).

**Definition.** We define any finite Böhm tree presentation to be *regular*. Such presentations define Böhm trees that are regular in the sense of admitting only a finite number of distinct subtrees, up to variable renaming. Of course not every Böhm tree is regular.

2. Semantics

We have just seen that arbitrary finitely generated Böhm trees are definable by Böhm tree presentations. Conversely, let us show that any Böhm tree presentation defines a unique Böhm forest.

2.1. Parameterization of Böhm trees

We start with a few auxiliary technical notions.

If  $T$  is the (defined) Böhm tree

$$\lambda u_1 u_2 \cdots u_n \cdot w(T_1, \dots, T_p)$$

and  $x$  is a variable, we define  $(T \ x)$  to be the Böhm tree

$$\lambda u_2 \cdots u_n \cdot w'(T'_1, \dots, T'_p)$$

if  $n > 0$ , with  $w' = x$  if  $w = u_1$  and  $w' = w$  otherwise, and  $T'_j$  is obtained from  $T_j$  by substituting every free occurrence of  $u_1$  by  $x$ . If  $n = 0$ , it is just the Böhm tree  $w(T_1, \dots, T_p, x)$ . Finally, when  $T$  is the undefined Böhm tree  $\perp$ , we define  $(T \ x)$  to be  $\perp$ .

The top node of the defined Böhm tree

$$T = \lambda u_1 u_2 \cdots u_n \cdot w(T_1, \dots, T_p)$$

is the triple  $(n, w, p)$ . The head variable  $w$  may be encoded as a de Bruijn index, or as a pair of indexes as in Huet (1993), or in any way that is invariant under variable renaming. A path  $u$  in tree  $T$  is a list of integers addressing a node  $T/u$  in the tree. If  $u$  is the empty list,  $T/u$  is the top node of  $T$ . If  $u = q;v$  with  $q \leq p$ , then  $T/u$  is  $T_q/v$ .

Two trees  $T$  and  $T'$  are different if and only if there exists a common path  $u$  such that  $T/u \neq T'/u$ . In this case we define the distance  $d(T, T')$  between  $T$  and  $T'$  to be  $2^{-h}$ , where  $h$  is the minimum of the lengths of such differentiating paths, otherwise  $d(T, T') = 0$ . The set of (finite and infinite) Böhm trees equipped with distance  $d$  has the structure of a complete metric space.

2.2. Constructing the Böhm forest

Now every  $\mathcal{X}$ -indexed family of Böhm tree  $T$  is mapped into an  $\mathcal{X}$ -indexed family  $\mathcal{E}(T)$  as follows:

- If  $X$  has no defining equation in  $\mathcal{E}$ , then  $\mathcal{E}(T)_X = \perp$ .
- If it has the defining equation:

$$X u_1 u_2 \cdots u_n := u_k(M_1, \dots, M_p)$$

with  $M_j = X_j(v_1, \dots, v_l)$ , then

$$\mathcal{E}(T)_X = \lambda u_1 u_2 \cdots u_n \cdot u_k(T_1, \dots, T_p)$$

with  $T_j = (T_{X_j v_1, \dots, v_l})$  as defined in the previous section.

This defines a contracting map  $T \mapsto \mathcal{E}(T)$ , in the sense that if for all trees  $T, T'$ , we have  $d(\mathcal{E}(T), \mathcal{E}(T')) \leq \frac{1}{2}d(T, T')$ . We leave this easy proof to the reader. By the Banach fixpoint theorem, it has a unique fixpoint associating a Böhm tree  $T_X$  to every combinator  $X$  in  $\mathcal{X}$ . These trees are all total when  $\mathcal{E}$  is total.

Note that when  $X$  has no defining equation in  $\mathcal{E}$  we could alternatively define  $\mathcal{E}(T)_X = X$ . We would get a slight generalisation of Böhm trees with (parameterized) constructors, whose leaves are of the form  $X(v_1, \dots, v_l)$  for  $X \in \mathcal{X}$  with no definition in  $\mathcal{E}$ .

### 2.3. Alternative formulations

Many variations on the formulation of systems of recursive combinators are possible. What is essential is that our combinator definitions are *guarded*, in the sense that the head variable is explicit in the definition of each combinator, which guarantees that every defined combinator is solvable, in the sense of having a head normal form. It is the guard indeed that guarantees that the defining map is contracting.

Our definition is minimal in the sense that we define the Böhm tree one layer at a time: each immediate subterm  $M_j$  starts with a combinator symbol. We may relax this condition by simply requiring that a right-hand side is a head expression  $u(M_1, \dots, M_p)$  where the  $M_j$ 's are either formed with a combinator applied to parameters (as above), or are themselves head expression.

In contrast, another variation is to be more restrictive, by demanding that each  $M_j$  is a linear pattern  $X(v_1, \dots, v_m)$  where the  $v_k$ 's are distinct variables. Obviously, both variations have the same expressive power, but differ in the number of combinators needed to define a given Böhm tree.

### 2.4. Operational semantics

Systems of recursive combinators admit a straightforward operational semantics, by viewing each definition as a rewrite rule. This is similar to combinatory reduction – the recursive nature of our systems does not introduce any extra difficulty. In terms of  $\lambda$ -calculus, as usual, one gets a notion of *weak* reduction, *i.e.*, a reduction that does not occur below  $\lambda$ s. In order to get a more interesting equational theory, one has to allow  $\eta$ -expansion, leading to a notion of extensional equality.

## 3. Extensional equality

In the following we will assume that we are given alphabets  $\mathcal{X}$  and  $\mathcal{U}$  as well as a regular Böhm tree presentation  $\mathcal{E}$ . We have seen that Böhm trees can be represented as linear patterns of the form  $X(v_1, \dots, v_m)$  with  $X \in \mathcal{X}$  and the  $v_j$ 's distinct variables in  $\mathcal{U}$ . Let us now relax the linearity requirement.

**Definition.** A *pattern* is any expression  $X(v_1, \dots, v_m)$  with  $X \in \mathcal{X}$  and  $v_j \in \mathcal{U}$  for  $1 \leq j \leq m$ . The pattern is said to be *saturated* if  $m \geq \text{ar}_{\mathcal{E}}(X)$ .

A *head expression* is an expression  $u(M_1, \dots, M_p)$  where  $u \in \mathcal{U}$  and the  $M_j$ 's are simple expressions for  $1 \leq j \leq p$ , where a *simple expression* is either a pattern or a head expression.

We define the notion of *shape* of a simple expression as follows. The shape of the head expression  $u(M_1, \dots, M_p)$  is  $(-p, u)$ . The shape of pattern  $X(v_1, \dots, v_m)$ , with  $X$  defined by the equation

$$E : X \ u_1 \ u_2 \ \cdots \ u_n := u_k(M_1, \dots, M_p)$$

is defined as  $(n - p - m, v)$  where  $v$  is  $v_k$  if  $k \leq m$ , and  $w_j$  if  $k = m + j$ , where  $w_1, w_2, \dots$  is some denumerable set disjoint from  $\mathcal{U}$ . Finally, the shape of pattern  $X(v_1, \dots, v_m)$ , with  $X$  undefined, is defined as some default value  $\perp$ .

We say that two simple expressions are *similar* when they have the same shape, and dissimilar otherwise.

We now define extensional equality as the largest bisimulation consistent with  $\mathcal{E}$  that separates dissimilar expressions. More precisely, we mutually define an inductive relation  $\equiv_{\mathcal{E}}$  and a co-inductive relation  $\sim_{\mathcal{E}}$  between simple expressions by the following closure conditions:

$$\text{Consistency : } M_i \equiv_{\mathcal{E}} N_i \ (1 \leq i \leq p) \implies u(M_1, \dots, M_p) \sim_{\mathcal{E}} u(N_1, \dots, N_p)$$

$$\text{Inclusion : } M \sim_{\mathcal{E}} N \implies M \equiv_{\mathcal{E}} N$$

where  $M$  and  $N$  are patterns.

$$\text{Extensionality : } M(x) \equiv_{\mathcal{E}} N(x) \implies M \equiv_{\mathcal{E}} N.$$

Here  $M$  and  $N$  are unsaturated patterns in which parameter  $x$  does not occur.

$$\text{Definition : } M := N \in \mathcal{E} \implies M \equiv_{\mathcal{E}} N$$

$$\text{Renaming : } M \equiv_{\mathcal{E}} N \implies \sigma(M) \equiv_{\mathcal{E}} \sigma(N)$$

where  $M$  and  $N$  are patterns and  $\sigma$  is a parameter substitution (not necessarily one–one).

The renaming rule could be dispensed with if we imposed the linearity of patterns (at the cost of increasing the number of combinators); it would then be reduced to  $\alpha$ -conversion, which may be made implicit given a proper canonical representation of parameter variables.

If  $\mathcal{E}$  is not total, and we want to consider undefined combinators as representing the undefined tree rather than free constructors, we could single out one such undefined combinator, say  $\perp$ , and add the following extra closure conditions:

$$\text{Undefined : } X \equiv_{\mathcal{E}} \perp$$

where  $X \in \mathcal{X}$  has no defining clause in  $\mathcal{E}$ ,

$$\text{Saturation : } \perp(x) \equiv_{\mathcal{E}} \perp.$$

Finally, we take the closure conditions for  $\equiv_{\mathcal{E}}$  to be an equivalence relation (Reflexivity, Symmetry, Transitivity), and a congruence with respect to application:

$$\text{Congruence : } M \equiv_{\mathcal{E}} N \implies M(x) \equiv_{\mathcal{E}} N(x)$$

where  $M$  and  $N$  are patterns and  $x$  a parameter.

This completes the mutual definition of relations  $\sim_{\mathcal{E}}$  and  $\equiv_{\mathcal{E}}$ .

**Lemma 3.1.**  $M \equiv_{\mathcal{E}} N$  only if  $M$  and  $N$  are two similar expressions.

*Proof.* Similarity is an equivalence relation. It is straightforward to check that it is preserved by the Extensionality, Renaming, and Congruence rules. By definition, the Definition, Undefined and Saturation rules introduce only similar expressions as equivalent. Finally, since Consistency only relates similar expressions, and is the only introduction rule for  $\sim_{\mathcal{E}}$ , similarity is also enforced by the Inclusion rule. Assume that  $M \equiv_{\mathcal{E}} N$ . Then, by induction on its proof, we get that  $M$  and  $N$  are similar. Inversely,  $M \sim_{\mathcal{E}} N \implies \text{False}$  for any two dissimilar simple expressions.  $\square$

**Remarks.** Note that  $\sim_{\mathcal{E}}$  is *not* defined inductively. It is, rather, a co-inductive definition in the sense of Pitts (1994). It has the flavour of defining truth as consistency, in the spirit of inductionless induction (Huet and Hullot 1982). For instance, with

$$\mathcal{E}_{\mathcal{Y}} = \{Y f := f(Y(f)); Z f := f(Z'(f)); Z' f := f(Z(f))\},$$

we may prove by co-induction that  $Y \sim_{\mathcal{E}_{\mathcal{Y}}} Z$ .

On the other hand,  $\equiv_{\mathcal{E}}$  is an inductively defined equivalence relation.  $\sim_{\mathcal{E}}$  and  $\equiv_{\mathcal{E}}$  are mutually defined in the same recursion. Such mixtures of inductively and co-inductively defined objects are explained in Giménez (1995); a corresponding proof package, allowing such definitions and the mechanical checking of formal proofs about such objects, is available in the Coq proof assistant (Giménez 1996).

From the point of view of semantics,  $M \sim_{\mathcal{E}} N$  corresponds to equality in  $D_{\infty}$  of the Böhm trees defined by  $M$  and  $N$  in  $\mathcal{E}$  (Wadsworth 1976).

Many equivalent variations are possible. For instance, we do not need the closure by reflexivity, which may be proved to be an admissible rule. Similarly, Symmetry may be dispensed with if we add the symmetric rules of Definition and Undefined. On the other hand, notice that we cannot dispense with transitivity, for instance, we need it to chain applications of Extensionality. And thus we are obliged to have an interplay between an inductive and a co-inductive relation (the co-inductive closure by transitivity being trivial).

If, on the other hand, we were interested in intensional equality (*i.e.* equality of the underlying Böhm trees), we would need to restrict the Extensionality rule to unsaturated patterns, in the spirit of Section 9-C of Hindley and Seldin (1986).

#### 4. Decidability of regular systems

The main result of this paper is to show that extensional equality is decidable for regular systems.

**Theorem 4.1.** The question as to whether  $M \equiv_{\mathcal{E}} N$  for any regular total  $\mathcal{E}$  and simple expressions  $M$  and  $N$  is decidable.

We shall now prove this theorem, by exhibiting a completion algorithm that completes a finite set of equations into another one that either equates two dissimilar head expressions, or is closed by the closure conditions above. The theorem follows from its termination proof. This algorithm is inspired by a similar one in recursive program schemas (Courcelle *et al.* 1974).

##### 4.1. The algorithm

Each *recursion combinator*  $X_i$  is either undefined or it has a unique recursion equation  $E_i$  defining it in a given regular system:

$$E_i : X_i u_1 u_2 \cdots u_{n_i} := u_{k_i}(M_1, \dots, M_{p_i}).$$

We decide sets of equations of the form  $E : M = N$ , where  $M$  and  $N$  are simple expressions.

The algorithm manages two sets  $CON$  and  $HYP$  of such formulas. Initially, the set of conjectures is put in  $HYP$ , and  $CON$  is initialised as empty. The algorithm terminates with YES when  $HYP$  is empty.

Here is one step of the algorithm when  $HYP$  is non-empty: let  $HYP = REST \cup \{E\}$ , with  $E$  of the form above. If  $E$  is already in  $CON$  modulo equivalence and renaming, we just iterate with  $HYP := REST$  and  $CON$  unchanged. Otherwise, let us unfold  $M$  and  $N$  to head normal form, when we do not already have them in head variable form, by using the defining equation for their governing combinator. This may need replacing  $M$  by  $(M z_1 \cdots z_k)$  and  $N$  by  $(N z_1 \cdots z_k)$ , where  $z_1, \dots, z_k$  are new variables not already occurring in  $M$  or  $N$ , in order that the  $X$ 's have enough arguments to match their arity. This is the analogue of  $\eta$  expansion. We thus get two applicative forms  $x(M_1, \dots, M_l)$  and  $y(N_1, \dots, N_m)$ . Now, if either  $x \neq y$  or  $l \neq m$ , the algorithm stops with answer NO. Otherwise, we iterate, with  $HYP := REST \cup \{E_1, \dots, E_l\}$  where  $E_q : M_q = N_q$  and  $CON := CON \cup \{E\}$ .

When the governing combinator of  $M$  is undefined, if the governing combinator of  $N$  is also undefined, we iterate with  $HYP := REST$  and  $CON$  unchanged, otherwise we stop with answer NO.

#### 4.2. The proof

Note that, without loss of generality, we may assume that  $HYP$  and  $CON$  only contain equations between patterns: an initial query containing a head expression may be reduced to pattern queries by one initial pass; thereafter head expressions occur only in temporary conjectures in the processing step.

**Lemma 4.2.** The algorithm always terminates.

*Proof.* There is a finite number of candidates for  $CON$ , since all patterns  $X(v_1, \dots, v_m)$  stored in  $CON$  (except possibly the ones given in the initial query) have a number of parameter arguments  $m$  bound by the maximum of such arguments in all  $X$ -patterns used in the right-hand sides of the system  $\mathcal{E}$ .  $\square$

**Lemma 4.3.** When the algorithm stops with YES,  $M \equiv_{\mathcal{E}} N$  for every initial conjecture  $M = N$ .

*Proof.* Every initial conjecture ends up in  $CON$  ultimately. Let us consider the set of pairs  $CON$  as a relation  $\rho$  between simple expressions, and let  $\sigma$  be the closure of  $\rho$  by the Extensionality, Definition, Renaming, Equivalence and Congruence rules (plus the Undefined and Saturation rules if we allow undefined combinators). By construction of the algorithm, the Consistency rule holds when we replace  $\equiv_{\mathcal{E}}$  by  $\sigma$  and  $\sim_{\mathcal{E}}$  by  $\rho$ . Thus, by bisimulation/coinduction, we have that  $\rho \subseteq \sim_{\mathcal{E}}$  and  $\sigma \subseteq \equiv_{\mathcal{E}}$ . Thus, in particular,  $M \equiv_{\mathcal{E}} N$  for every initial conjecture  $M = N$ .  $\square$

**Lemma 4.4.** When the algorithm stops with NO, the set of initial conjectures is inconsistent.

*Proof.* The steps of the algorithm correspond to inversion schemas of the various closure conditions. Thus all the formulas placed in  $CON$  are logical consequences of the initial conjectures. When the algorithm stops with NO, one such formula equates two dissimilar expressions, from which a contradiction may be derived by Lemma 3.1.  $\square$

Equivalently, the algorithm may be interpreted in this case as a successful search for a separating path (in the sense of Section 4.4 below) in the Böhm trees denoted by two members of one initial equation.

**Note.** We only provide informal proofs in this paper. It is hoped that fully formal proofs, mechanically verified by the Coq proof assistant, will be available soon.

#### 4.3. Example: $I=J$

Here is a simple, but generic, example of the procedure.

Let  $\mathcal{E} = \{J \ x \ y := (x \ (J \ y)); I \ x := x\}$ . We show that  $I \equiv_{\mathcal{E}} J$ .

Initially,  $CON_0 = \{\}$ ,  $HYP_0 = \{I = J\}$ .

We select  $E_0 : I = J$ . We introduce new variables  $x$  and  $y$ ,  $\eta$ -expand to  $(I \ x \ y) = (J \ x \ y)$ , and substitute  $I$  and  $J$  by their definitions, obtaining  $(x \ y) = (x \ (J \ y))$ . Since shapes fit, we generate the subgoal  $y = (J \ y)$  (Note that we get rid of the useless  $x$ , this is important.) Thus we get:  $CON_1 = \{I = J\}$ ,  $HYP_1 = \{y = (J \ y)\}$ .

We now select  $E_1 : y = (J \ y)$ . We  $\eta$ -expand to  $(y \ z) = (J \ y \ z)$ , substitute  $J$ , and get  $(y \ z) = (y \ (J \ z))$ . Since shapes fit, we generate the subgoal  $z = (J \ z)$ , and get  $CON_2 = \{I = J, y = (J \ y)\}$ ,  $HYP_2 = \{z = (J \ z)\}$ .

We now select  $E_2 : z = (J \ z)$ . But this equation is equivalent by renaming to one in  $CON_2$ , and thus we stop with  $CON_3 = \{I = J, y = (J \ y)\}$ ,  $HYP_3 = \{\}$ . Thus we have shown that  $I \equiv_{\mathcal{E}} J$ , that is, that  $\lambda x \cdot x = (\mathbf{Y} \ \lambda j \ \lambda x \ \lambda y \ (x \ (j \ y)))$  in  $D_{\infty}$ .

As an exercise for the reader, we suggest trying the algorithm for proving  $Y \sim_{\mathcal{E}_{yy}} Z$  in the presentation  $\mathcal{E}_{yy}$  above.

#### 4.4. Distances, separability, apartness

We recognize as equal the combinators defining Böhm trees that are in fact quite different as trees:  $I$  has a finite Böhm tree, whereas  $J$ 's is infinite. Moreover, note that these Böhm trees do not correspond to equivalent  $\lambda$ -terms in the sense of  $\beta\eta$ -conversion; intuitively an infinite number of  $\eta$  expansions is necessary to transform  $I$  into  $J$ . However, these trees are not *separable* in the sense of Böhm's theorem.

The equality between Böhm trees in question here corresponds to trees being hereditarily of the same shape, where the shape of  $\lambda \ u_1 \ u_2 \ \dots \ u_n \ \cdot \ u(T_1, \dots, T_p)$  is  $(n - p, u)$ , and 'hereditarily' means recursing in the  $T_i$ 's, after possible  $\eta$ -expansion to the same prefix. Equivalently, we may define equality as non-separability, with two trees being separable if their distance is greater than 0, where now the distance between two Böhm trees is  $2^{-h}$  where  $h$  is the length of a minimum *separating path* for the two trees in the sense of Huet (1993). Intuitively, a separating path is a virtual path through  $\eta$ -expansions of the two trees, where the corresponding subterms are of different shapes. Böhm's theorem, in the slightly different context of  $\lambda$ -calculus, shows that a separating path permits us to construct a uniform context that separates the two original  $\lambda$ -terms, in the sense of  $\beta$ -reducing to, respectively,  $\lambda \ x \ y \cdot x$  and  $\lambda \ x \ y \cdot y$ .

The idea of defining equality as non-separability has a long history. This notion is already implicit in Leibniz' equality. More recently, the idea was systematically applied



to von Plato's treatment of constructive geometry (von Plato 1995). This conforms to the view of mathematical modelling of reality up to the precision of measuring instruments. It is thus quite natural to define separability with some measure  $d$ :

$$X \neq Y \quad =_{\text{def}} \quad dXY > 0.$$

Note that if  $d$  is an ultrametric, that is, if  $dXX = 0$  and  $dXZ \geq \max\{dXY, dYZ\}$ , then separability is an *apartness* relation, that is,  $\neg X \neq X$  and  $X \neq Z \Rightarrow \forall Y X \neq Y \vee Y \neq Z$  and its opposite (that is, equality) is, by construction, an equivalence relation. This gives a general methodology for constructive mathematical modelling, from measure to separability to equality.

## 5. Applications and further investigations

The formalism of guarded combinators is extremely simple, but powerful, since it combines in one notion combinatory logic and recursion (as opposed to indirectly coding up recursion by a fixpoint combinator). Furthermore, it accommodates (mutually recursive) definitions. It has the flavour of machine code, with combinators playing the role of program addresses and parameterization the role of register transfer. The notion of guard gives to its execution a dataflow flavour: at each combinator invocation, when enough arguments are provided for it to fire, one grain of information is computed.

This formalism is thus a good candidate for a sort of basic programming language for communicating processes: overall computation may be infinite, but no process may loop without producing information, in sharp contrast to pure  $\lambda$ -calculus, or non-guarded recursion. For instance, it would be interesting to investigate closely in what way it relates to the applicative programming languages proposed for describing reactive processes. Lustre is a particularly good candidate. Recently, Caspi and Pouzet have shown that a functional extension to Lustre could be implemented in a kernel of recursively defined primitives for stream manipulations (Caspi and Pouzet 1995). This kernel can be represented in a rather direct way as a set of regular combinators.

Many further investigations are needed to make such an application practical. For instance, there are several alternative ways of representing data structures or more complex control structures. An example is given in Huet and Laulhère (1997), which considers the encoding of finite-state transducers as regular Böhm trees.

The algorithmic aspects of the decision procedure remain to be investigated. If no constraint is put on the way combinators mutually recurse, in the worst case the number of parameters of such calls may be of the same order as the size of the system, in which case the algorithm may have exponential behaviour. If mutual recursion is checked with further devices, such as local sections with hierarchical scoping we may hope to improve the bounds and obtain an algorithm that will scale up to realistic sizes. Sharing techniques from BDD technology may also prove useful in this context.

Finally, application of this formalism to typed systems, in particular to proof assistants, where Böhm trees may represent sequent calculus partial proofs, in the manner of Herbelin (1995), remains to be investigated. In particular, the Extensionality rule needs to be constrained (for instance with a notion of  $\eta$ -long normal form).

### Acknowledgment

We thank Martin Abadi, Gilles Dowek and Vaughan Pratt for their judicious remarks.

### References

- Barendregt, H. (1980) *The Lambda-Calculus: Its Syntax and Semantics*, North-Holland.
- Böhm, C. (1968) Alcune proprietà delle forme  $\beta - \eta$ -normali nel  $\lambda - K$ -calcolo. Pubblicazioni dell'Istituto per le Applicazioni del Calcolo **696**, Roma.
- Böhm, C., Dezani-Ciancaglini, M., Peretti, P. and Ronchi della Rocca, S. (1979) A discrimination algorithm inside  $\lambda - \beta$ -Calculus. *Theoretical Computer Science* **8** 271–291.
- Böhm, C., Piperno, A. and Tronci, E. (1989) Solving equations in Lambda-Calculus. In: Ferro, Bonotto, Valentini and Zanardo (eds) *Logic Colloquium'88*, North-Holland.
- Caspi, P. and Pouzet, M. (1995) A functional extension to Lustre. International Symposium on Languages for Intentional Programming, Sydney.
- Coppo, M., Dezani-Ciancaglini, M. and Ronchi della Rocca, S. (1978) (Semi-)separability of finite sets of terms in Scott's  $D_\infty$  models of the  $\lambda$ -calculus. In: Ausiello, G. and Böhm, C. (eds) Proc. 5th ICALP. *Springer-Verlag Lecture Notes in Computer Science* **62** 142–164.
- Coquand, C. and Coquand, T. (1996) On the definition of reduction for infinite terms. *C. R. Acad. Sci. Paris* **323** Série I 553–558.
- Courcelle, B., Kahn, G. and Vuillemin, J. (1974) Algorithmes d'équivalence et de réduction à des expressions minimales dans une classe d'équations récursives simples. *Proceedings ICALP 74*, Springer-Verlag.
- David, R. and Nour, K. (1996) Une preuve syntaxique de l'équivalence opérationnelle de deux  $\lambda$ -termes. (Private communication.)
- Giménez, E. (1995) Codifying guarded definitions with recursive schemes. Proceedings of the 1994 Workshop on Types for Proofs and Programs. *Springer-Verlag Lecture Notes in Computer Science* **996** 39–59. (Extended version of the paper available by ftp at [lip.ens-lyon.fr:/pub/Rapports/RR/RR95/RR95-07.ps.Z](http://lip.ens-lyon.fr:/pub/Rapports/RR/RR95/RR95-07.ps.Z).)
- Giménez, E. (1996) Co-inductive types in Coq. Documentation included in the release of Coq V6.1. (Available by ftp at [ftp.inria.fr:INRIA/Projects/coq/coq/V6.1.beta](http://ftp.inria.fr:INRIA/Projects/coq/coq/V6.1.beta).)
- Herbelin, H. (1995) *Séquents qu'on calcule : de l'interprétation du calcul des séquents comme calcul de  $\lambda$ -termes et comme calcul de stratégies gagnantes*, Thèse, U. Paris 7.
- Hindley, J. R. and Seldin, J. P. (1986) *Introduction to Combinators and  $\lambda$ -Calculus*, Cambridge University Press.
- Huet, G. (1993) An analysis of Böhm's Theorem. In: Ronchi della Rocca, S., Dezani-Ciancaglini, M. and Zilli, M.V. (eds.) *To C. Böhm: Essays on Lambda-Calculus and Functional Programming*. (Also *Theoretical Computer Science* **121** 145–167.)
- Huet, G. and Hullot, J. M. (1982) Proofs by Induction in Equational Theories With Constructors. *JCSS* **25,2** 239–266.
- Huet, G. and Lahlouche, H. (1997) Finite-state Transducers as Regular Böhm Trees. *Proceedings of TACS'97, Sendai, Japan*.
- Lévy, J. J. (1993) Böhm trees and Extensionality. (Private communication.)
- Pitts, A. (1994) A Co-induction Principle for Recursively Defined Domains. *Theoretical Computer Science* **124** 195–219.
- von Plato, J. (1995) The axioms of constructive geometry. *Annals of Pure and Applied Logic* **76** 169–200.
- Wadsworth, C. (1976) The relation between computational and denotational properties for Scott's  $D_\infty$ -models of the lambda calculus. *SIAM J. Comput.* **5** 488–521.