Math. Struct. in Comp. Science (1998), *vol.* 00, *pp.* 1–11

Regular Böhm Trees

Gérard HUET[†]

INRIA-Rocquencourt

BP 105, 78153 Le Chesnay Cedex, France

Received December 1996

Introduction

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 λ-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_∞ 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). We shall adopt here the point of view that Böhm trees are potentially infinite computational objets 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 given two disjoint denumerable alphabets of symbols: $\mathcal{X} = \{X_1, X_2, ...\}$ is the set of *combinator* symbols, $\mathcal{U} = \{u_1, u_2, ...\}$ is the set of *parameter* symbols. Intuitively, combinators name Böhm trees, whereas parameters name bound λ-variables.

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

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

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

Email: `Gerard.Huet@inria.fr`

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

1.2. *Completeness*

Whatever is 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 $\{v_1, ..., v_m\}$ is definable as $X(v_1, ..., v_m)$ 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 $\lambda\ u_1\ u_2 ... u_n \cdot w(T_1, ..., T_p)$ and its set of free variables is $\{v_1, ..., v_m\}$, then let T_j be represented by $X_j(v_{j,1}, ..., v_{j,m})$ 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 $v_{j,1}, \ldots, v_{j,m}$ are included in $v_1, \ldots, v_m, u_1, \ldots, u_n$. Now choose a new parameter X_0, and consider the new equation:

$$E_0 : X_0\ v_1 ... v_m\ u_1\ u_2 ... u_n := w(M_1, ..., M_{p_i})$$

with $M_j = X_j(v_{j,1}, ..., v_{j,m})$. Now with $\mathcal{E}_0 = \{E_0\}$ and $\mathcal{X}_0 = \{X_0\}$, we consider the presentation $\bigcup_{i=0}^{p} \mathcal{E}_j$ over $\mathcal{X} = \bigcup_{i=0}^{p} \mathcal{X}_j$. In this presentation, T may indeed be defined as $X_0(v_1, ..., v_m)$. 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 Böhm trees of any λ-term. But they permit to do more, in that we may represent dags and looping structures.

For instance, the λ-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

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

with sharing of combinator I. Whereas the single equation $Z\ u := u(Z)$ defines as Z the infinite tree $\lambda u_1 \cdot u_1(\lambda u_2 \cdot u_2(...))$. Another example is the fixpoint combinator Y, with $Y\ f := f(Y(f))$. Still 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 λ-term $(\mathbf{Y}\ \lambda j\ \lambda x\ \lambda y\ (x\ (j\ y)))$, for $\mathbf{Y}$ any fixpoint combinator such as Curry's.

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

Definition. We call *regular* any finite Böhm tree presentation. Such presentations define Böhm trees which 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 just saw that arbitrary finitely generated Böhm trees were 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

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

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

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

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 $w(T_1, ..., T_p, x)$. Finally, when T is the undefined Böhm tree $\perp$, we define $(T\ x)$ as $\perp$.

The *top node* of the defined Böhm tree

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

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 which is invariant by 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' as 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:

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

with $M_j = X_j(v_1, ..., v_l)$ then

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

with $T_j = (T_{X_j} v_1, ..., v_l)$ 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.

Remark 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 $X(v_1, ..., v_l)$ 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 $u(M_1, ..., M_p)$ where the M_j's are either formed with a combinator applied to parameters (as above), or else are themselves head expression.

Another variation is to be on the contrary more restrictive, by demanding that each M_j is a linear pattern $X(v_1, ..., v_m)$ 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 extra difficulty. In terms of λ-calculus, as usual, one gets a notion of *weak* reduction, i.e. a reduction which does not occur below λs. In order to get a more interesting equational theory, one has to allow η-expansion, leading to a notion of extensional equality.

3. Extensional equality

In the following we assume given alphabets $\mathcal{X}$ and $\mathcal{U}$ as well as a regular Böhm tree presentation $\mathcal{E}$. We saw that Böhm trees could be represented as linear patterns of the form $X(v_1, ..., v_m)$ 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 $X(v_1, ..., v_m)$ 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 $m \geq \mathrm{ar}_{\mathcal{E}}(X)$.

A *head expression* is an expression $u(M_1, ..., M_p)$ 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 $u(M_1, ..., M_p)$ is $(-p, u)$. The shape of pattern $X(v_1, ..., v_m)$, with X defined by equation:

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

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

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

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

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

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

where M and N are patterns.

$$Extensionality : M(x) \equiv_{\mathcal{E}} N(x) \Longrightarrow M \equiv_{\mathcal{E}} N$$

Here M and N are unsaturated patterns where parameter x does not occur.

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

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

where M and N are patterns and σ is a parameter substitution (not necessarily one-one).

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

If $\mathcal{E}$ is not total, when 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:

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

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

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

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:

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

where M and N are patterns and x a parameter.

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

Lemma 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 rules Extensionality, Renaming, and Congruence. By definition, rules Definition, Undefined and Saturation introduce as equivalent only similar expressions. Finally, since Consistency relates only similar expressions, and is the only introduction rule for $\sim_{\mathcal{E}}$, similarity is also enforced by rule Inclusion. Assume that $M \equiv_{\mathcal{E}} N$; by induction on its proof, we get that M and N are similar. Inversely, $M \sim_{\mathcal{E}} N \Longrightarrow False$ for any two dissimilar simple expressions.

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

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

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 semantics point of view, $M \sim_{\mathcal{E}} N$ corresponds to equality in D_∞ 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, remark that we cannot dispense with transitivity, for instance 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. It is decidable whether $M \equiv_{\mathcal{E}} N$ for any regular total $\mathcal{E}$ and simple expressions M and N.

We shall now prove this theorem, by exhibiting a completion algorithm which completes a finite set of equations into another one which either equates two dissimilar head expressions, or else is closed by the closure conditions above. The theorem follows from its termination proof. This algorithm is inspired from 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:

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

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, in case we do not have them already in head variable form, by using the defining equation for their governing combinator. This may need replacing M by $(M\ z_1...\ z_k)$ and N by $(N\ z_1...\ z_k)$, where $z_1, ..., z_k$ are new variables not occurring already in M or N, in order to have the X's have enough arguments to match their arity. This is the analogue of η expansion. We thus get two applicative forms $x(M_1, ..., M_l)$ and $y(N_1, ..., N_m)$. Now if either $x \neq y$ or $l \neq m$ the algorithm stops with answer NO. Otherwise, we iterate, with $HYP := REST \cup \{E_1, ..., E_l\}$ where $E_q : M_q = N_q$ and $CON := CON \cup \{E\}$.

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

4.2. *Its proof*

Remark that, without loss of generality, we may assume that HYP and CON contain only 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 2. The algorithm always terminates.
Proof. There is a finite number of candidates for CON, since all patterns $X(v_1, ..., v_m)$ 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 right hand sides of the system $\mathcal{E}$.

Lemma 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 ρ between simple expressions, and let σ be the closure of ρ by rules Extensionality, Definition, Renaming, Equivalence and Congruence (plus Undefined and Saturation if we allow undefined combinators). By construction of the algorithm, the rule Consistency holds when we replace $\equiv_{\mathcal{E}}$ by σ and $\sim_{\mathcal{E}}$ by ρ. 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 $M = N$.

Lemma 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 1.

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

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, η-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 η-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 $I \equiv_{\mathcal{E}} J$, i.e. that $\lambda x \cdot x = (\mathbf{Y}\ \lambda j\ \lambda x\ \lambda y\ (x\ (j\ y)))$ in D_∞.

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

4.4. *Distances, separability, apartness*

We recognize as equal combinators defining Böhm trees which are indeed quite different as trees, since I has a finite Böhm tree, whereas J's is infinite. And note that these Böhm trees do not correspond to equivalent λ-terms in the sense of $\beta\eta$-conversion; intuitively an infinite number of η 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 which is here in question corresponds to trees being hereditarily of the same shape, where the shape of $\lambda\ u_1\ u_2 ... u_n \cdot u(T_1, ..., T_p)$ is $(n-p, u)$, and 'hereditarily' means recursing in the T_i's, after possible η-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 η-expansions of the two trees, where the corresponding subterms are of different shapes. Böhm's theorem, in the slightly different context of λ-calculus, shows that a separating path permits to construct a uniform context which separates the two original λ-terms, in the sense of β-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:

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

Remark that if d is an ultrametric, i.e. 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 (i.e. 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 accomodates (mutually recursive) definitions. It has the flavor 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 flavor: 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 programmming language for communicating processes: overall computation may be infinite, but no process may loop without producing information, in sharp contrast to pure λ-calculus, or non-guarded recursion. For instance, it would be interesting to investigate closely in what way it relates to 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 practical such an application. For instance, there are several alternative ways to represent data structures or more complex control structures. An exemple 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 which 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 η-long normal form).

Acknowlegment. We thank Martin Abadi and Gilles Dowek for their judicious remarks.

References

H. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics.* North-Holland (1980).

C. Böhm. *Alcune proprietà delle forme $\beta-\eta-$normali nel $\lambda-K-$calcolo.* Pubblicazioni dell'Istituto per le Applicazioni del Calcolo N. 696, Roma, 1968.

C. Böhm, M. Dezani-Ciancaglini, P. Peretti and S. Ronchi della Rocca. *A discrimination algorithm inside $\lambda-\beta$-Calculus.* Theoretical Computer Science **8** (1979), 271–291.

C. Böhm, A. Piperno and E. Tronci. *Solving equations in Lambda-Calculus.* In Logic Colloquium'88, Eds Ferro, Bonotto, Valentini and Zanardo, North-Holland (1989).

P. Caspi and M. Pouzet. *A functional extension to Lustre.* International Symposium on Languages for Intentional Programming, Sydney, May 1995.

M. Coppo, M. Dezani-Ciancaglini and S. Ronchi della Rocca. *(Semi-)separability of finite sets of terms in Scott's D_∞ models of the λ-calculus.* In Proc. 5th ICALP, Eds G. Ausiello and C. Böhm, LNCS **62** (1978), 142–164.

C. Coquand and T. Coquand. *On the definition of reduction for infinite terms.* C. R. Acad. Sci. Paris, t. 323, Série I, p. 553–558 (1996).

B. Courcelle, G. Kahn et J. Vuillemin. *Algorithmes d'équivalence et de réduction à des expressions minimales dans une classe d'équations récursives simples.* Proceedings ICALP 74, Springer-Verlag.

R. David and K. Nour. *Une preuve syntaxique de l'équivalence opérationnelle de deux λ-termes.* Private communication.

E. Giménez. *Codifying guarded definitions with recursive schemes.* Proceedings of the 1994 Workshop on Types for Proofs and Programs, LNCS 996 (1995) 39–59. Extended version of the paper available by ftp at `lip.ens-lyon.fr:/pub/Rapports/RR/RR95/RR95-07.ps.Z`.

E. Giménez. *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`.

H. Herbelin. *Séquents qu'on calcule : de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes.* Thèse, U. Paris 7, 1995.

J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ-Calculus.* Cambridge University Press.

G. Huet. *An analysis of Böhm's Theorem*. In To C. Böhm: Essays on Lambda-Calculus and Functional Programming. S. Ronchi della Rocha, M. Dezani-Ciancaglini and M. V. Zilli (eds.). Also Theoretical Computer Science, 121 (1993) pp. 145–167.

G. Huet and J. M. Hullot. *Proofs by Induction in Equational Theories With Constructors.* JCSS **25,2** (1982) 239–266.

G. Huet and H. Laulhère. *Finite-state Transducers as Regular Böhm Trees.* Proceedings of TACS'97, Sendai, Japan (Sept. 1997).

J. J. Lévy. *Böhm trees and Extensionality.* Private communication (1993).

A. Pitts. *A Co-induction Principle for Recursively Defined Domains.* Theoretical Computer Science **124** (1994), 195–219.

von Plato. *The axioms of constructive geometry*. Annals of Pure and Applied Logic **76** (1995) 169–200.

C. Wadsworth. *The relation between computational and denotational properties for Scott's D_∞-models of the lambda calculus.* SIAM J. Comput. **5** (1976), 488-521.