11 Computations in Orthogonal Rewriting Systems, I

Norman Ramsey

Gérard Huet and Jean-Jacques Lévy

We present in this chapter a study of derivations which formalize the computations of expressions in programming languages. Our formalism extends recursive programs with simplifications in [16, 20, 6, 27, 23, 25, 8, 2]. It permits to provide an operational semantics to programming languages such as LISP [19], LUCID [1], HOPE [5], and ML [9]. The derivations we study may also be considered as proofs in certain equational theories which are of use to study abstract data types [11, 22, 6]. Finally, our work may be used for the design of efficient simplifications in formula manipulation systems such as LCF [10].

The results of this paper may be sketched as follows. We consider finite systems of pairs of first-order terms: $\Sigma = \{\alpha_i \to \beta_i | i \leq n\}$. Computations consist of rewriting an occurrence of some instance of an α_i into the corresponding instance of β_i. We restrict every rule $\alpha_i \to \beta_i$ so that all variables of β_i occur in α_i and so that every variable of α_i has a unique occurrence in α_i. Furthermore, no two rules of Σ are nontrivially superposable. We believe that most computations in programming languages without parallelism can be formalized in this setting.

Section 1 contains the preliminary definitions. In section 2 we show how the parallel moves lemma induces a congruence ("permutation") on derivations issued from a given term. The usual notion of residuals of a redex is extended to derivations, yielding a sup-semilattice structure to the permutation classes of derivations. This generalizes Church-Rosser results from [26, 13, 24].

In section 3 we prove what is usually called the standardization theorem. The problem is to simulate any derivation by a "standard" derivation, i.e., one which computes in an outside-in way. For this we define the notion of a redex occurrence being external for a given derivation. Intuitively, an external redex is such that none of its residuals will be below any redex contracted in any permutation of the given derivation. This notion is thus invariant by permutation, and every nonempty derivation possesses at least one external redex occurrence. This allows us to define the notion of a standard derivation, and we show that every derivation class contains a unique standard derivation. We finally define the notion of a redex occurrence being external in a given term, which yields the notion of the normal derivation issued from a given term. When the term possesses a normal form, the normal derivation terminates in this normal form. This may be considered as an extension to our systems of the call by name computation rule for recursive program schemas, and of the normal derivations in λ-calculus. However, our formalism differs in essential ways from these two, because of the possibility of upward creation of redexes, and our call by name rule does not correspond to a simple lefmost-outermost strategy of computation. The results imply that all the nonambiguous linear term rewriting systems are *d*-outer in the sense of O'Donnell [24]. The existence of normal derivations entails the possibility of doing only needed computations in a top-down manner. However, it does not correspond to an effective computation rule, and at this stage the only correct effective interpreters use parallel strategies such as parallel outermost.

Part II of this paper (chapter 12) will deal with the problem of finding effective interpreters which compute in a sequential fashion.

Most of our concepts are adaptations to syntactic domains of denotational semantics notions. This algebraic approach permits us to state and prove our theorems in an abstract fashion, suggesting that our results can be extended to a more general theory of operational semantics.

1 Preliminary Definitions

We follow here the notations of Huet [13] and Berry-Lévy [2]. Let $\mathscr{F}_n$ be a set of function symbols of arity n, $\mathscr{F} = \bigcup \{\mathscr{F}_n | n \geq 0\}$, and $\mathscr{V}$ a denumerable set of variable symbols. Our expression language is the set $\mathscr{M}(\mathscr{F}, \mathscr{V})$ of first-order *terms* formed from $\mathscr{F}$ and $\mathscr{V}$, i.e.,

$\mathscr{V} \subseteq \mathscr{M}(\mathscr{F}, \mathscr{V})$,

$F \in \mathscr{F}_n \;\&\; \forall i \leq n\; M_i \in \mathscr{M}(\mathscr{F}, \mathscr{V}) \Rightarrow F(M_1, M_2, \ldots, M_n) \in \mathscr{M}(\mathscr{F}, \mathscr{V})$.

When $\mathscr{F}$ and $\mathscr{V}$ are fixed from the context, we shall usually denote $\mathscr{M}(\mathscr{F}, \mathscr{V})$ by $\mathscr{T}$.

For any term M we define its set of *occurrences* $\mathcal{O}(M)$ as a finite subset of the set N_+^* of finite sequences of positive integers as follows:

$\Lambda \in \mathcal{O}(M)$ (the empty occurrence)

$u \in \mathcal{O}(M_i) \Rightarrow iu \in \mathcal{O}(F(M_1, M_2, \ldots, M_n))$ for $1 \leq i \leq n$

Intuitively, an occurrence of M names a subterm of M by its access path. If $u \in \mathcal{O}(M)$, we define the *subterm of M at u* as the term M/u, defined inductively by

$M/\Lambda = M$,

$F(M_1, M_2, \ldots, M_n)/iu = M_i/u$.

Finally, if $u \in \mathcal{O}(M)$, we define for every term N the *replacement in M at u by N* as the term $M[u \leftarrow N]$, defined by

$M[\Lambda \leftarrow N] = M$,

$F(M_1, M_2, \ldots, M_n)[iu \leftarrow N] = F(M_1, M_2, \ldots, M_i[u \leftarrow N], \ldots, M_n)$.

We shall also use the notation $\bar{\mathcal{O}}(M) = \{u \in \mathcal{O}(M) | M/u \notin \mathscr{V}\}$.

Example Let $M = F(G(x), A)$. We have $\mathcal{O}(M) = \{\Lambda, 1, 1.1, 2\}$ and $\bar{\mathcal{O}}(M) = \{\Lambda, 1, 2\}$, $M/1 = G(x)$, and $M[1 \leftarrow H(B)] = F(H(B), A)$.

The set of occurrences $\mathcal{O}(M)$ is partially ordered by the prefix ordering $u \preceq v$ iff $\exists w$ $uw = v$. In this case we shall define v/u as w. If $u \not\preceq v$ and $v \not\preceq u$, we say that u and v are *disjoint*, and write $u|v$. Finally, $u \prec v$ iff $u \preceq v$ and $u \neq v$.

A *substitution* is any function σ from $\mathcal{T}$ to $\mathcal{T}$ satisfying

$$\sigma(F(M_1, M_2, \ldots, M_n)) = F(\sigma(M_1), \sigma(M_2), \ldots, \sigma(M_n)).$$

In other words, σ is a morphism in the algebra of terms and is therefore uniquely determined from its value on variables.

We call *term rewriting system* (abbreviated TRS) any set Σ of pairs of terms $\alpha_i \to \beta_i$ such that $\mathcal{V}(\beta_i) \subseteq \mathcal{V}(\alpha_i)$, where $\mathcal{V}(M)$ is the set of variables appearing in M. We denote by red_Σ the set of left-hand sides α_i of Σ, which we call *redex schemes*. For any substitution σ and $\alpha \in \mathrm{red}_\Sigma$, $\sigma(\alpha)$ is called a *redex* of Σ. We denote by $\mathcal{R}_\Sigma(M)$ the set of redex occurrences in the term M. A term M such that $\mathcal{R}_\Sigma(M) = \varnothing$ is called a Σ-*normal form*. We denote by $\mathcal{NF}_\Sigma$ the set of terms in Σ-normal form. From now on, we shall assume that TRS Σ is fixed and therefore drop the subscript Σ except when needed.

We say that the term M *reduces to* N *at occurrence* u *using rule* $\alpha_k \to \beta_k$ iff there exists a substitution σ such that $M/u = \sigma(\alpha_k)$ and $N = M[u \leftarrow \sigma(\beta_k)]$. The pair $A = \langle M, u\rangle$ is called an *elementary derivation*, and we shall write $A\colon M \xrightarrow{u}_k N$. In this notation, k, u or A may be omitted. Note that M, u and k determine unambiguously $\sigma(x)$ for every x in $\mathcal{V}(\alpha_k)$, and therefore $\sigma(\beta_k)$ and N.

A *derivation* is a sequence $A = A_1 A_2 \ldots A_n$ of elementary derivations $A_i\colon M_i \to M_{i+1}$. We use AB for the concatenation of A and B, 0 for the empty derivation, and $|A|$ for the length of derivation A. We shall use the notation $A\colon M \xrightarrow{*} N$ to indicate that derivation A starts in M and ends in N.

Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of mutually disjoint redex occurrences in term M, and let $M/u_i = \sigma_i(\alpha_i)$. We call *elementary multiderivation* the pair $A = \langle M, U\rangle$ and write $A\colon M \xrightarrow{U} N$, where

$$N = M[u_i \leftarrow \sigma_i(\beta_i) \mid i \le i \le n] = M[u_1 \leftarrow \sigma_1(\beta_1)] \ldots [u_n \leftarrow \sigma_n(\beta_n)].$$

(Of course, the order of the u_i's is irrelevant). We say that A *contracts* the set U.

We define *multiderivations* in the same way and use the same notation as for derivations. Furtermore, if we want to emphasize the system Σ we use, we can write $M \xrightarrow{*}_\Sigma N$. We denote $\mathcal{D}(M)$ the set of multiderivations issued from M, and $\mathcal{F}(A)$ the final term reached by the (multi)derivation A. We say that A and B are *coinitial* if A, $B \in \mathcal{D}(M)$, *cofinal* if $\mathcal{F}(A) = \mathcal{F}(B)$. The notation AB, for the concatenation of A and B, assumes that $B \in \mathcal{D}(\mathcal{F}(A))$. Concatenation being associative, we write ABC for $(AB)C$ or $A(BC)$. Every set $\mathcal{D}(M)$ contains an empty multiderivation which we shall denote 0, the term M being usually understood from the context. Thus we freely write $A0 = 0A = A$.

When considering a derivation $A\colon M_0 \xrightarrow{u_1} M_1 \xrightarrow{u_2} M_2 \xrightarrow{u_3} \cdots \xrightarrow{u_n} M_n$, it is convenient to denote by $A[i]$ the i first steps of A, with $0 \le i \le n$. The rest of A is denoted $A[i, n]$. Thus $A = A[i]A[i, n]$, with M_i as the final term of $A[i]$. Similarly for multiderivations.

We shall study in this paper the properties of derivations in TRSs which have two constraints:

Left linearity: for every α in red, every variable of α occurs only once:

$\alpha/u = \alpha/v \in \mathscr{V} \Rightarrow u = v$.

Nonambiguity: if $\alpha_i, \alpha_j \in$ red, for every u in $\bar{\mathscr{O}}(\alpha_i)$ there are no σ, σ' such that $\sigma(\alpha_i/u) = \sigma'(\alpha_j)$, except in the trivial case $i = j$ and $u = \Lambda$.

We also rule out the trivial TRSs which only consist of rules $\alpha \to \beta$ with $\alpha \in \mathscr{V}$. Therefore, we have red $\cap\, \mathscr{V} = \varnothing$ by condition (2).

DEFINITION 1.1 We call *orthogonal* any non-trivial TRS verifying conditions 1 and 2 above.

Our TRSs are similar to the schematic tree replacement systems of Rosen [26] and O'Donnell [24]. Condition 2 is called the nonoverlapping condition in these papers. Note that 1 and 2 imply together that our systems are outer, in the terminology of [24]. In the terminology of Knuth-Bendix [17], there is no critical pair. The derivation relation $\to$ is confluent (has the Church-Rosser property) [13, 26]. We shall study in this paper some stronger properties of derivation spaces, generalizing results in Berry-Lévy [2] obtained for recursive equations.

2 The Derivations Space

In this section we prove an important property of derivations in TRSs, the parallel moves lemma (see Curry & Feys [7]). This allows us to define a partial ordering on derivations, inducing a semi-lattice property on derivation spaces. This useful tool generalizes the one defined in Berry-Lévy [2], which can also be defined in the λ-calculus (Lévy [18]).

DEFINITION 2.1 Given an elementary derivation $A: M \xrightarrow{u}_k N$ and $v \in \mathscr{R}(M)$, we define the set $v \backslash A$ of *residuals* of v by A as a subset of $\mathscr{O}(N)$ as follows:

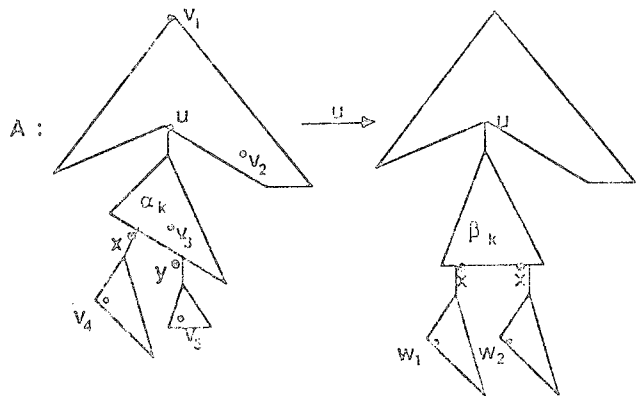
$$v \backslash A = \begin{cases} \varnothing & \text{if } v = u, \\ \{v\} & \text{if } v | u \text{ or } v \prec u, \\ \{uw_1v_1 \mid \beta_k/w_1 = x\} & \text{if } v = uwv_1 \text{ and } \alpha_k/w = x \in \mathscr{V}. \end{cases}$$

A pictorial explanation of residuals is given in figure 1. Note that all the relative positions of u and v have been considered because of nonambiguity. Furthermore, nonambiguity and left linearity imply that $v \backslash A \subseteq \mathscr{R}(N)$. Finally, for every w in $\mathscr{R}(N)$, there is at most one v such that $w \in v \backslash A$. When there is none, we say that w is *created* by A. Note that such ws may be above u, as well as in the substituted right-hand side β_k.

Example Let $\Sigma = \{F(G(x)) \to H(x, K(x)), F(H(x, y)) \to x, K(A) \to B, C \to A\}$.

We consider

$$A: M_0 = F(F(H(G(C), C))) \xrightarrow{\{1\}} M_1 = F(G(C)) \xrightarrow{\{\Lambda\}} M_2 = H(C, K(C))$$
$$\xrightarrow{\{1.2.1\}} M_3 = H(A, K(A)) \xrightarrow{\{2\}} H(A, B).$$



$v_1\backslash A = \{v_1\}, v_2\backslash A = \{v_2\}, v_3\backslash A = v_5\backslash A = u\backslash A = \emptyset, v_4\backslash A = \{w_1, w_2\}.$

Figure 1

The redex occurrence 1.1.1.1 of M_0 has one residual in M_1, namely 1.1, and two residuals in M_2, namely 1 and 2.1. It has no further residuals after the second step of A, which reduces these two redexes. The redex occurrence 1.1.2 of M_0 has no residual in M_1. The redex occurrence 2 of M_3 is created (upward) by redex occurrence 2.1 of M_2. Finally, note that the redex occurrence Λ of M_2 is created by the first step of A, even though the replacing term does not contribute to its redex, since it is a variable!

For any nonelementary derivation A, we define $v\backslash A$ by

$v\backslash 0 = \{v\},$

$v\backslash(AB) = \{w\backslash B \mid w \in v\backslash A\}.$

The residual mapping is extended to sets of redex occurrences by defining

$U\backslash A = \bigcup \{u\backslash A \mid u \in U\}.$

Finally, if A is an elementary multiderivation $A: M \xrightarrow{U} N$ with $U = \{u_1, u_2, \ldots, u_n\}$, we define $v\backslash A$ as $v\backslash(A_1 A_2 \ldots A_n)$, where $A_i: M_{i-1} \xrightarrow{u_i} M_i$, $M_0 = M$ and $M_n = N$. Of course, $v\backslash A$ is independent of the order of the u_i's. We extend $v\backslash A$ to nonelementary multiderivations and to sets of redex occurrences as above.

Let $A, B \in \mathscr{D}(M)$, with $|B| = 1$, contracting the set $U \subseteq \mathscr{R}(M)$. We define the *residual* $B\backslash A$ of B by A as the elementary derivation in $\mathscr{L}(\mathscr{F}(A))$, contracting the set $U\backslash A$. We also define $A \sqcup B = A(B\backslash A)$.

LEMMA 2.2: THE PARALLEL MOVES LEMMA Let $A, B \in \mathscr{D}(M)$, with $|A| = |B| = 1$. Then $\mathscr{F}(A \sqcup B) = \mathscr{F}(B \sqcup A)$, and for all u in $\mathscr{R}(M)$ we have $u\backslash(A \sqcup B) = u\backslash(B \sqcup A)$.

Proof Similar to the proof of theorem 6.5 in [23], lemma 11 in [17], lemma 2.1.6 in [3]. ∎

The parallel moves lemma is illustrated in figure 2.

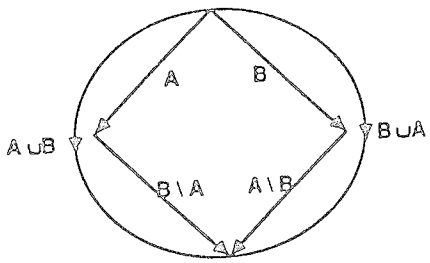


Figure 2

COROLLARY: THE CHURCH-ROSSER PROPERTY Since $|B\backslash A| = |A\backslash B| = 1$, lemma 2.2 shows that the multiderivation relation is strongly confluent (see [13]) and therefore that the relation $\rightarrow$ is confluent. Therefore, for every M, there is at most one normal form N such that $M \xrightarrow{*} N$.

The parallel moves lemma permits us to generalize the residual relation to arbitrary derivations, as follows.

DEFINITION 2.3 Let $A, B \in \mathscr{D}(M)$, with $|B| = 1$. We define $A\backslash B \in \mathscr{D}(\mathscr{F}(B))$ by induction on $|A|$:

$$0\backslash B = 0 \tag{1}$$

$$(A_1 A_2)\backslash B = (A_1\backslash B)(A_2\backslash(B\backslash A_1)) \quad \text{with } |A_1| = 1. \tag{2}$$

Note that $A_2\backslash(B\backslash A_1)$ is defined by induction, since $|B\backslash A_1| = 1$, and that it can be concatenated to $A_1\backslash B$, by the preceding lemma. Now for $A, B \in \mathscr{D}(M)$ of arbitrary lengths, we define $A\backslash B \in \mathscr{D}(\mathscr{F}(B))$ by induction on $|B|$:

$$A\backslash 0 = A \tag{3}$$

$$A\backslash(B_1 B_2) = (A\backslash B_1)\backslash B_2 \quad \text{with } |B_1| = 1 \tag{4}$$

We also extend the notation $A \sqcup B = A(B\backslash A)$ to derivations of any length. It is easy to show by induction that $|A\backslash B| = |A|$, that equations (1) through (4) are valid without length conditions, and to generalize the parallel moves lemma:

LEMMA 2.4: THE GENERALIZED PARALLEL MOVES LEMMA For all $A, B \in \mathscr{D}(M)$ we have $\mathscr{F}(A \sqcup B) = \mathscr{F}(B \sqcup A)$ and for all u in $\mathscr{R}(M)$, $u\backslash(A \sqcup B) = u\backslash(B \sqcup A)$.

The last part of lemma 2.4 can be generalized to an arbitrary multiderivation coinitial with A and B as follows.

LEMMA 2.5: THE CUBE LEMMA For all $A, B, C \in \mathscr{D}(M)$ we have $C\backslash(A \sqcup B) = C\backslash(B \sqcup A)$.

Proof By induction on $|A| + |B| + |C|$. If $C = 0$, use (1); otherwise, let $C = C_1 C_2$ with $|C_2| = 1$. We get

$$(A \sqcup B)\backslash C_1 = (A\backslash C_1)((B\backslash A)\backslash(C_1\backslash A)) \quad \text{by (2)}$$

$$= (A\backslash C_1)(B\backslash A \sqcup C_1)) \quad \text{by (4)}$$

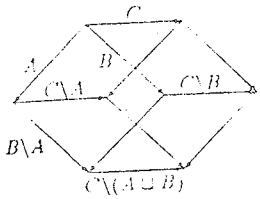


Figure 3

$$= (A \setminus C_1)(B\setminus(C_1 \sqcup A)) \qquad \text{by ind. hyp.}$$
$$= (A \setminus C_1)((B\setminus C_1)\setminus(A\setminus C_1)) \quad \text{by (4)}$$
$$= (A\setminus C_1)\sqcup(B\setminus C_1).$$

Thus

$$C_2\setminus((A\sqcup B)\setminus C_1) = C_2\setminus((A\setminus C_1)\sqcup(B\setminus C_1))$$
$$= C_2\setminus((B\setminus C_1)\sqcup(A\setminus C_1)) \quad \text{by lemma 2.4}$$
$$= C_2\setminus((B\sqcup A)\setminus C_1). \qquad \text{symmetrically}$$

Finally, $C_1\setminus(A\sqcup B) = C_1\setminus(B\sqcup A)$ by the induction hypothesis, which achieves the proof, using (2). ∎

The cube lemma is illustrated in figure 3.

If we now define, for $A, B \in \mathscr{D}(M)$, $A \equiv B$ iff for all $C \in \mathscr{D}(M)$, $C\setminus A = C\setminus B$, we get the following corollary:

COROLLARY $\forall A, B \in \mathscr{D}(M)$, $A \sqcup B \equiv B \sqcup A$.

We shall call $\equiv$ *permutation* equivalence.

LEMMA 2.6: $\sqcup$ IS ASSOCIATIVE $\forall A, B, C \in \mathscr{D}(M)$, $(A\sqcup B)\sqcup C = A\sqcup(B\sqcup C)$.

Proof

$$(A\sqcup B)\sqcup C = (A\sqcup B)(C\setminus(A\sqcup B))$$
$$= (A\sqcup B)(C\setminus(B\sqcup A)) \qquad \text{by lemma 2.5}$$
$$= A(B\setminus A)((C\setminus B)\setminus(A\setminus B)) \quad \text{by (4)}$$
$$= A((B\sqcup C)\setminus A) \qquad \text{by (2)}$$
$$= A\sqcup(B\sqcup C)$$

∎

The empty multiderivation 0 in $\mathscr{D}(M)$ should not be confused with the elementary multiderivation starting from M and contracting an empty set of redex occurrences, which we shall denote $\varnothing$. In particular, $|\varnothing| = 1$. However, it follows from the definition of residual that for every A in $\mathscr{D}(M)$ we have $\varnothing\setminus A = \varnothing$. By an easy induction on $|A|$ we get that $A\setminus\varnothing = A$, and thus $0 \equiv \varnothing$, and also that $A\setminus A = \varnothing^n$, with $n = |A|$. This and lemma 2.4 easily imply that if $A \equiv B$, then $\mathscr{F}(A) = \mathscr{F}(B)$.

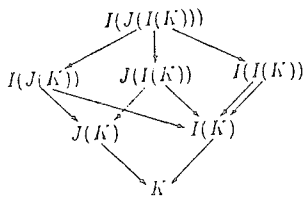


Figure 4

LEMMA 2.7: ≡ IS A CONGRUENCE Let $A, B \in \mathscr{D}(M)$, with $A \equiv B$. We have

$\forall C \in \mathscr{D}(M)$, $A \backslash C \equiv B \backslash C$

$\forall C \in \mathscr{D}(M)$, $A \sqcup C \equiv B \sqcup C$

$\forall C \in \mathscr{D}(\mathscr{F}(A))$, $AC \equiv BC$

$\forall C$ such that $\mathscr{F}(C) = M$, $CA \equiv CB$.

Proof Easy consequences of the definitions. □

COROLLARIES $\forall A \in \mathscr{D}(M)$, $A \backslash A \equiv 0$. $\forall A \in \mathscr{D}(M)$, $A \sqcup A \equiv A$.

We now define a relation $\sqsubseteq$ between coinitial multiderivations as follows:

$\forall A, B \in \mathscr{D}(M)$, $A \sqsubseteq B$ iff $A \sqcup B \equiv B$

THEOREM 2.8: LATTICE OF DERIVATIONS THEOREM $\langle \mathscr{D}(M)/\equiv, \sqsubseteq, \sqcup \rangle$ is an upper semi-lattice.

Proof Easy algebraic manipulations from the preceding lemmas. □

Phrased in categorical terms, this result means that the category whose objects are terms and whose morphisms are the equivalence classes of derivations admits pushout.

Caution! The lattice structure given by the parallel moves theorem is on *derivations* and *not* on terms. For instance, if we consider the system R consisting solely of the rules $I(x) \to x$ and $J(x) \to x$, figure 4 shows that the terms $I(J(K))$ and $J(I(K))$ do not possess a least upper bound. Note that this phenomenon may be traced to the existence of two non-equivalent derivations between $I(I(K))$ and $I(K)$. This shows that the categorical viewpoint is the right one here: we need to talk in terms of arrows, not just relations between terms. And now the confluence diagrams can be replaced by more informative commuting diagrams expressing permutation equivalences of derivations. For instance, in figure 4 certain sub-diagrams are confluent only for unimportant syntactic coincidences (due to the double occurrence of I in $I(J(I(K)))$), but the others are commuting diagrams expressing a strong equivalence of computations.

Using all that precedes, it is easy to prove the following relations (with the appropriate conditions on initial and final terms of the following derivations).

$$(AB)\backslash A \equiv B \tag{5}$$

$$(CA)\backslash(CB) \equiv A \backslash B \tag{6}$$

$$A \sqsubseteq B \text{ iff } A \setminus B = \varnothing^m, \text{ with } m = |A| \tag{7}$$

$$A \equiv B \text{ iff } A \sqsubseteq B \sqsubseteq A \tag{8}$$

$$A \equiv 0 \text{ iff } A = \varnothing^n \text{ for some } n \geq 0 \tag{9}$$

$$A \sqsubseteq B \text{ iff } AC \equiv B \text{ for some } C \tag{10}$$

$$A \sqsubseteq B \text{ implies } A \setminus C \sqsubseteq B \setminus C \tag{11}$$

$$A \sqsubseteq B \text{ implies } CA \sqsubseteq CB \tag{12}$$

Note that (7) and (8) show that $\sqsubseteq$ and $\equiv$ are easily decidable.

We could also have defined $\equiv$ as the smallest congruence relation on $\mathcal{P}(M)$ such that $\varnothing \equiv 0$ and $A \sqcup B \equiv B \sqcup A$. This justifies our terminology of permutation equivalence.

Finally, we may extend relations $\sqsubseteq$ and $\equiv$ to derivations by confusing the derivation

$$M_0 \xrightarrow{u_1} M_1 \xrightarrow{u_2} M_2 \xrightarrow{u_3} \cdots \xrightarrow{u_n} M_n$$

with the multiderivation

$$M_0 \xrightarrow{U_1} M_1 \xrightarrow{U_2} \cdots \xrightarrow{U_n} M_n,$$

where $\forall i \leq n$, $U_i = \{u_i\}$.

3 Standardization, Call by Name, Call by Need

We are going to show in this section that every derivation is equivalent by permutations to a certain derivation computing redexes in an outside-in manner. In analogy to what happens for recursive definitions or λ-calculus, we shall call these special derivations *standard*. However, it should be remarked that, unlike in these two formalism, the leftmost outermost derivations are usually not standard in TRS. For instance, consider

$$\Sigma = \{F(x, A) \to B, C \to C, D \to A\}.$$

The standard derivation starting from the term $F(C, D)$ and ending in its normal form is

$$F(C, D) \to F(C, A) \to B,$$

whereas the leftmost outermost rule leads to the infinite derivation

$$F(C, D) \to F(C, D) \to \cdots$$

3.1 Outside-In and Standard Computations

Let us first give some preliminary definitions and technical lemmas.

DEFINITION 3.1 Let $u \in \mathcal{R}(M)$, α the redex scheme at u in M. We define the *contractum in M at u* as the set $C_M(u)$ of occurrences of M that are inside α:

$C_M(u) = \{uv \in \mathcal{O}(M) | v \in \overline{\mathcal{O}}(\alpha)\}$.

We also define

$C_M(u) = \{uv \in \mathcal{O}(M) | v \notin \mathcal{O}(\alpha)\}$.

Note that $u \in C_M(u)$, since $\alpha \notin \mathcal{V}$, and thus $\forall v \in \overline{C_M}(u)$, $u \prec v$.

Let $u, v \in \mathcal{R}(M)$, with $u \prec v$. The nonambiguity condition imposes $v \in \overline{C_M}(u)$. Furthermore, for any elementary A: $M \xrightarrow{V} N$ such that $v \in V$, we get $u \backslash A = \{u\}$.

LEMMA 3.2 Let $u, v \in \mathcal{R}(M)$. If $C_M(u) \cap \overline{C_M}(v) \neq \emptyset$, then $u \in \overline{C_M}(v)$.

Proof Easy corollary of nonambiguity. □

DEFINITION 3.3 Let A: $M_0 \xrightarrow{U_1} M_1 \xrightarrow{U_2} M_2 \xrightarrow{U_3} \cdots \xrightarrow{U_n} M_n$, and $u \in \mathcal{O}(M_0)$. We say that A *preserves* u iff A does not contract a redex above u, that is, $\forall i \preceq n \nexists v \in U_i$, $v \prec u$.

LEMMA 3.4 Let $A \in \mathcal{D}(M)$, preserving v. For every u in $\mathcal{R}(M)$ such that $u \prec v$, we have $u \backslash A = \{u\}$.

Proof Easy induction on $|A|$. □

We can easily refine the parallel moves lemma to derivations preserving an occurrence u, using:

LEMMA 3.5 Let A, B be coinitial, both preserving u. Then $A \backslash B$ preserves u.

Proof Easy induction on $|A| + |B|$, using the following observation. Let A_1: $P \xrightarrow{v} Q$. For all $w \in \mathcal{R}(P)$, $w \backslash A_1$ does not contain any occurrence above both v and w. □

LEMMA 3.6 Let A, B be coinitial, with $A \sqsubseteq B$. If B preserves u, then A too preserves u.

Proof Assume that $A \sqsubseteq B$, B preserves u, and A does not preserve u. That is, $A = A_1 A_2 A_3$, where A_1 preserves u and A_2 contracts set W such that $\exists w \in W$. By lemma 3.5, $B \backslash A_1$ preserves u, and by lemma 3.4, $w \backslash (B \backslash A_1) = \{w\}$, whence $A \backslash B \not\equiv 0$, contrary to hypothesis. □

We are now ready to present the important definition of an occurrence being external for a (multi)derivation.

DEFINITION 3.7 Let $A \in \mathcal{D}(M)$, u in $\mathcal{O}(M)$. We say that u is *external* for A, and write $u \in \mathcal{X}(A)$, iff either A preserves u or $A = A_1 A_2 A_3$, and there exists $v \prec u$ such that A_1 preserves u, A_2: $P \xrightarrow{V} Q$, with $v \in V$ and $u \in C_P(v)$ and $v \in \mathcal{X}(A_3)$.

Note that in the second case v and the decomposition of A into $A_1 A_2 A_3$ are unique: A_2 is the first step in A that does not preserve u, and v is the unique redex occurrence above u contracted at this step. Intuitively, u is external for A iff A does not contract at some step a redex above u for which the symbol at u in M did not contribute.

Example Let $\Sigma = \{F(x, B) \to G(x, x), A \to B\}$ and consider

$A: M_0 = F(C, A) \to M_1 = F(C, B) \to M_3 = G(C, C).$

We have $\mathscr{X}(A) = \{\Lambda, 2\}$.

DEFINITION 3.8 Let $A: M_0 \xrightarrow{U_1} M_1 \xrightarrow{U_2} M_2 \xrightarrow{U_3} \cdots \xrightarrow{U_n} M_n$. We say that u is an *initial redex occurrence contributing* to A, and write $u \in \mathscr{I}(A)$, iff $\exists i \leq n\ U_i \cap (u \backslash A[i-1]) \neq \varnothing$. Finally, we define the *external redex occurrences* of A as $\mathscr{E}(A) = \mathscr{I}(A) \cap \mathscr{X}(A)$.

Note that outermost redex occurrences in the starting term of A may be excluded from $\mathscr{E}(A)$ either because they are not external for A or else because they are not contracted in A.

We shall first extend lemmas 3.4, 3.5 and 3.6 above to $\mathscr{X}(A)$. For lemma 3.4 we need an extra hypothesis: v must be below the redex scheme at u.

LEMMA 3.9 Let $A \in \mathscr{D}(M)$, u in $\mathscr{R}(M)$. If $\mathscr{X}(A) \cap \overline{C_M}$, we have $u \backslash A = \{u\}$.

Proof By induction on $|A|$. Let $v \in \mathscr{X}(A) \cap \overline{C_M}(u) \neq \varnothing$. If A preserves v, use lemma 3.4. Otherwise, $A = A_1 A_2 A_3$, and there exists $w \prec v$ such that the following hold:

a. A_1 preserves v, and therefore $u \backslash A_1 = \{u\}$ by lemma 3.4.

b. $A_2: P \xrightarrow{W} Q$ with $w \in W$ and $v \in C_P(w)$. Furthermore, by (a), $v \in \overline{C_P}(u)$, and thus $w \in \overline{C_P}(u)$, by lemma 3.2. Therefore, $u \backslash A_2 = \{u\}$, and $w \in \overline{C_Q}(u)$.

c. Also, $w \in \mathscr{X}(A_3)$ and therefore, using $w \in \overline{C_Q}(u)$, we get $u \backslash A_3 = \{u\}$, by the induction hypothesis, and finally, $u \backslash A = \{u\}$. ▨

We now extend lemma 3.5 as follows.

LEMMA 3.10 Let A and B be coinitial and B preserve $u \in \mathscr{X}(A)$. Then $u \in \mathscr{X}(A \backslash B)$.

Proof By induction on $|A|$.

Case 1: A preserves u. Then by lemma 3.5, $A \backslash B$ preserves u, and therefore $u \in (A \backslash B)$.

Case 2: $\exists v \prec u$ such that $A = A_1 A_2 A_3$, as in the definition above. As A_1 and B preserve u, $A_1 \backslash B$ preserves u, by lemma 3.5. Similarly, $B \backslash A_1$ preserves u, and therefore $v \backslash (B \backslash A_1) = \{v\}$, by lemma 3.4. Finally, $B \backslash A_1$ preserves v (since $v \prec u$). A_2 preserves v trivially, and by lemma 3.5, $B \backslash (A_1 A_2)$ preserves v. Now $v \in \mathscr{X}(A_3)$ implies $v \in \mathscr{X}(A_3 \backslash (B \backslash (A_1 A_2)))$, by the induction hypothesis, and therefore,

$A \backslash B = (A_1 \backslash B)(A_2 \backslash (B \backslash A_1))(A_3 \backslash (B \backslash (A_1 A_2)))$

is a decomposition, which shows that $u \in \mathscr{X}(A \backslash B)$. ▨

Note also that it follows from the definition of $\mathscr{X}$ that if A preserves $u \in \mathscr{X}(B)$, we have $u \in \mathscr{X}(AB)$. We shall freely use this property below. We are now able to prove the extension of lemma 3.6.

LEMMA 3.11 Let A and B be coinitial, with $A \sqsubseteq B$. We have $\mathscr{X}(B) \subseteq \mathscr{X}(A)$.

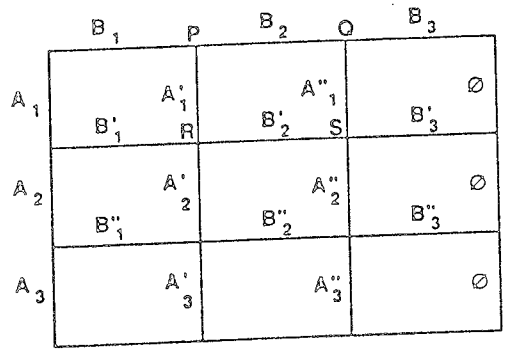


Figure 5

Proof Let $A \sqsubseteq B$ and $u \in \mathscr{X}(B)$. We show that $u \in \mathscr{X}(A)$ by induction on $|A|$.

Case 1: B preserves u. Then A preserves u, by lemma 3.6, and thus $u \in \mathscr{X}(A)$.

Case 2: Otherwise, $B = B_1 B_2 B_3$, with B_2: $P \xrightarrow{V} Q$; $\exists v \in V$, $v \prec u$; B_1 preserves u; $u \in C_P(v)$; and $v \in \mathscr{X}(B_3)$. Since A does not preserve u, we have $A = A_1 A_2 A_3$, where A_1 preserves u and A_2: $R \xrightarrow{W} S$ such that $\exists w \in W$ $w \prec u$. Let $A'_1, A'_2, \ldots, B'_1, B'_2, \ldots$ be as shown in figure 5, which expresses the parallel moves lemma for A and B. By lemma 3.5, A'_1 preserves u and therefore v, B_2 preserves v trivially, and by lemma 3.5, again A''_1 preserves v. By lemma 3.10 we get $v \in \mathscr{X}(B'_3)$. As A'_1 preserves u, v is contracted by B'_2, by lemma 3.4. Similarly, B'_1 preserves u and therefore w, according to the relative positions of v, $w \in \mathscr{R}(R)$.

Case 2.1: $v \neq w$. Since our systems are nonambiguous, $u \in C_R(v)$ and $w \prec u$ imply $v \in \overline{C_R}(w)$, and thus $w \backslash B'_2 = \{w\}$. Therefore, $w \in \mathscr{R}(S)$. Using $v \in \mathscr{X}(B'_3)$, we get by lemma 3.9 that $w \backslash B'_3 = \{w\}$, and therefore $A_2 \backslash (B'_1 B'_2 B'_3) \neq \varnothing$, a contradiction with $A \sqsubseteq B$.

Case 2.2: $v = w$. Since A'_2 and B'_2 preserve v, so do A''_2 and B''_2, by lemma 3.5. By lemma 3.10 we get $v \in \mathscr{X}(B''_3)$. As $B''_1 B''_2$ preserve v, we have $v \in \mathscr{X}(B''_1 B''_2 B''_3)$. Since $A_3 \sqsupseteq B''_1 B''_2 B''_3$, we get $v \in \mathscr{X}(A_3)$, by the induction hypothesis, and that $A_1 A_2 A_3$ is a decomposition of A, which shows that $u \in \mathscr{X}(A)$. ▨

COROLLARY $A \equiv B$ implies $\mathscr{X}(A) = \mathscr{X}(B)$.

External redex occurrences are their own residual until they are contracted:

LEMMA 3.12 Let A: $M \xrightarrow{*} N$ and $u \in \mathscr{X}(A) \sqcup \mathscr{R}(M)$. We have either $u \in \mathscr{R}(A)$ and $u \backslash A = \varnothing$ or $u \notin \mathscr{R}(A)$ and $u \backslash A = \{u\}$.

Proof Straightforward from the definitions of $\mathscr{X}(A)$ and $\mathscr{R}(A)$. ▨

LEMMA 3.13 If $A \equiv B$, we have $\mathscr{E}(A) = \mathscr{E}(B)$.

Proof Let $A \equiv B$ and $u \in \mathscr{E}(A)$. We get $u \in \mathscr{X}(B)$ by the corollary to lemma 3.11.

Lemma 3.12 implies $u\backslash A = \varnothing$ and therefore, by the parallel moves lemma, $u\backslash B = \varnothing$. Now lemma 3.12 applied to B gives $u \in \mathscr{R}(B)$, and thus $u \in \mathscr{E}(B)$. ▨

If u is preserved by A, then any prefix occurrence $v \preceq u$ is also preserved by A, obviously. This property is also true of members of $\mathscr{X}(A)$.

LEMMA 3.14 If $u \in \mathscr{X}(A)$, then $\forall v \preceq u\ v \in \mathscr{X}(A)$.

Proof Induction on $|A|$.

Case 1: A preserves u. Then A preserves v, and $uv \in \mathscr{X}(A)$.

Case 2: $A = A_1 A_2 A_3$ with A_2: $P \xrightarrow{W} Q$ and $\exists w \prec u$ such that $w \in W$, $u \in C_P(w)$ and $w \in \mathscr{X}(A_3)$. Moreover, A_1 preserves u and therefore v.

Case 2.1: $u \preceq w$. Then $v \in \mathscr{X}(A_3)$ by the induction hypothesis, and therefore, since $A_1 A_2$ preserve v, we get $v \in \mathscr{X}(A)$.

Case 2.2: $w \prec v \preceq u$. Then $v \in C_P(w)$, and therefore the decomposition $A_1 A_2 A_3$ shows that $v \in \mathscr{X}(A)$.

The set $\mathscr{X}(A)$ is therefore closed by prefix. When it contains some $u \in \mathscr{R}(M)$, with $A \in \mathscr{C}(M)$, is also contains all the members of $C_M(u)$. $\mathscr{X}(A)$ is obviously non-empty, since it always contains Λ, preserved by every derivation. We shall now show that $\mathscr{E}(A)$ is also non-empty whenever $A \neq \varnothing^n$.

LEMMA 3.15 If $A \not\equiv 0$, we have $\mathscr{E}(A) \neq \varnothing$.

Proof By induction on $|A|$. If $A = 0$, the proof is trivial. Otherwise, let $A = A_1 A_2$, with A_1: $M \xrightarrow{U} N$ elementary. If $A_2 \equiv 0$, then $U \neq \varnothing$, and any member of U is obviously in $\mathscr{E}(A)$. Otherwise, let us consider $v \in \mathscr{E}(A_2)$, which exists by the induction hypothesis.

Case 1: There exists u in U such that $u \preceq v$. Then $u \in \mathscr{X}(A_2)$ by lemma 3.14 and thus $u \in \mathscr{X}(A)$ since A_1 preserves u. Therefore, $u \in \mathscr{E}(A)$.

Case 2: Otherwise, v is preserved by A_1, and therefore $v \in \mathscr{X}(A)$.

Case 2.1: There exists u in $U \cap C_M(v)$. Then by definition, $u \in \mathscr{X}(A)$, and therefore $u \in \mathscr{E}(A)$.

Case 2.2: Otherwise, $v \in \mathscr{R}(A)$, and therefore $v \in \mathscr{E}(A)$. ▨

We are now ready to use $\mathscr{E}(A)$ to construct an outside-in equivalent of A. The following lemma is useful to show that this construction will always terminate.

LEMMA 3.16 There is no infinite chain $A_1 \rhd A_2 \rhd A_3 \rhd \cdots$, where $A \rhd B$ iff $A \equiv B$ and $B = A\backslash u$, with $u \in \mathscr{E}(A)$.

Proof Let A: $M_0 \xrightarrow{U_1} M_1 \xrightarrow{U_2} M_2 \xrightarrow{U_3} \cdots \xrightarrow{U_n} M_n$. If $A \equiv 0$, there is no B such that $A \rhd B$. Otherwise, let $u \in \mathscr{E}(A)$. According to lemma 3.12, there exists $k \leq n$ such that $u\backslash A[k-1] = \{u\}$ and $u \in U_k$. Therefore, $(A\backslash u)[k-1, n] = A[k-1, n]\backslash u$ contracts the sets $U_k - \{u\}, U_{k+1}, \ldots, U_n$. Thus $A\backslash u$ is less than A in the lexicographic ordering on the tuples $\langle |U_n|, |U_{n-1}|, \ldots, |U_1| \rangle$. ▨

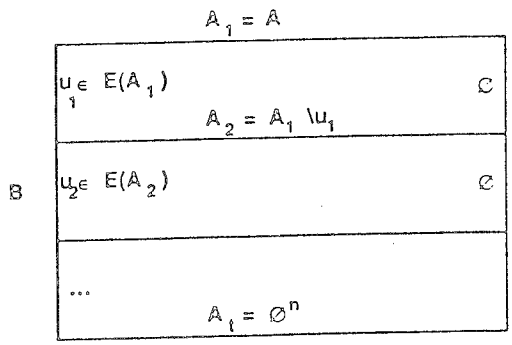


Figure 6

DEFINITION 3.17 The derivation A is said to be *outside-in* iff either $A = 0$ or $A = A_1 A_2$, where A_1 is the elementary derivation contracting some u in $\mathcal{E}(A)$ and A_2 is outside-in.

In other words, in an outside-in derivation, we contract at every step some redex occurrence external for the rest of the derivation.

Let A be any multiderivation. We construct an outside-in equivalent B of A as follows. If $A \equiv 0$, we stop with $B = 0$. Otherwise, let us pick some u in $\mathcal{E}(A)$. We define B as the elementary derivation contracting u, followed by an outside-in equivalent of $A \backslash u$. By Nœtherian induction, using lemma 3.16, the construction always terminates. By construction, $B \sqsubseteq A$. But according to lemma 3.15, the construction can only stop when $A \sqsubseteq B$, and therefore $B \equiv A$, which justifies our terminology. The construction is illustrated in figure 6.

DEFINITION 3.18 Let A be any derivation. We say that A is *standard* iff A is outside-in and at every step the redex occurrence u is the *leftmost* in $\mathcal{E}(A)$.

Similarly to the above, we define *the standard derivation* B in the class of A, and write $B = \mathrm{st}(A)$. Using lemma 3.13, we easily get by Nœtherian induction that $\mathrm{st}(A_1) = \mathrm{st}(A_2)$ whenever $A_1 \equiv A_2$.

To conclude, we have shown the following:

THEOREM 3.19: STANDARDIZATION THEOREM Every derivation class contains a unique standard derivation.

Note that the leftmost condition is unimportant here: any choice function over sets of disjoint occurrences would guarantee the uniqueness of the standard derivation.

Example Let $\Sigma = \{F(x, B) \to G(x, x), A \to B, C \to D\}$. Among the derivations starting at $F(C, A)$, all the ones that do not use the reductions marked with an X in figure 7 are standard.

Finally, we remark that it is possible to express standard derivations in terms of residuals, because $\mathcal{E}(A)$ can be characterized as the set of redex occurrences of A that stay outermost modulo permutations in the following sense.

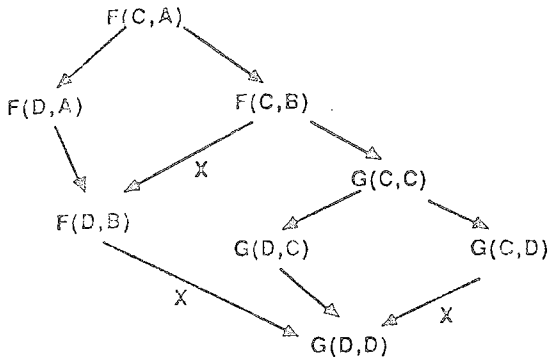


Figure 7

DEFINITION 3.20 Let $A: M_0 \overset{u_1}{\to} M_1 \overset{u_2}{\to} M_2 \overset{u_3}{\to} \cdots \overset{u_n}{\to} M_n$ and $u \in \mathscr{R}(A)$. We say that u is *outermost* in A, and write $u \in \mathrm{Out}(A)$, iff $\forall i \leq n\ \forall v \in u \backslash A[i-1]\ \exists u_i \in U_i\ u_i \prec v$.

In other words, $u \in \mathrm{Out}(A)$ iff $u \in U$ for some $i \leq n$, and $\forall j \prec i, \exists u_j \in U_j, u_j \preceq u$. Therefore, $\forall j \prec i, u \backslash A[j] = \{u\}$ and $u \backslash A[i] = u \backslash A = \varnothing$. Note that lemma 3.12 implies that $\mathscr{E}(A) \subseteq \mathrm{Out}(A)$. The converse is true for outside-in derivations.

LEMMA 3.21 If A is outside-in, we have $\mathscr{E}(A) = \mathrm{Out}(A)$.

Proof Let $A: M_0 \overset{u_1}{\to} M_1 \overset{u_2}{\to} M_2 \overset{u_3}{\to} \cdots \overset{u_n}{\to} M_n$ be an outside-in derivation and $u \in \mathrm{Out}(A)$. By definition, $\exists i \leq n$, $u_i = u$ and $\forall j \prec i$, $u_j \not\preceq u$. Since A is outside-in, we have $u \in \mathscr{E}(A[i-1, n])$ and, since $A[i-1]$ preserves u, we get $u \in \mathscr{E}(A)$. The converse follows from lemma 3.12. ▨

LEMMA 3.22 $\mathscr{E}(A) = \bigcap_{B \equiv A} \mathrm{Out}(B)$.

Proof Let $B \equiv A$. We have, by lemma 3.13, $\mathscr{E}(A) = \mathscr{E}(B) \subseteq \mathrm{Out}(B)$, which shows that $\mathscr{E}(A) \subseteq \bigcap_{B \equiv A} \mathrm{Out}(B)$. For the converse, take $B = \mathrm{st}(A)$ and use the preceding lemma. ▨

That is, $\mathscr{E}(A)$ is the set of redex occurrences which are outermost in every permutation of A, or equivalently, which are outermost in some outside-in equivalent of A.

The results obtained so far do not allow us to say how to compute in a standard way, since $\mathscr{E}(A)$ may depend on the whole of A. We are now going to relativize these notions to the starting term of derivation A, so that we may define an outside-in computation rule, generalizing the call-by-name computation rule for recursive program schemes.

3.2 Normal Derivation, Call by Name

DEFINITION 3.23 We define the set of *external occurrences* in term M as $\mathscr{X}(M) = \bigcap_{A \in \mathscr{Q}(M)} \mathscr{X}(A)$. Similarly to above, we define the set of *external redex occurrences* in M as $\mathscr{E}(M) = \mathscr{R}(M) \cap \mathscr{X}(M)$, or equivalently, $\mathscr{E}(M) = \bigcap_{A \in \mathscr{Q}(M)} \mathscr{E}'(A)$, where for A in $\mathscr{Q}(M)$ we define $\mathscr{E}'(A) = \mathscr{X}(A) \cap \mathscr{R}(M)$.

It is easy to show, similarly to lemma 3.22, that $\mathscr{E}'(A) = \bigcap_{B \sim A} \mathrm{Out}'(B)$, where

$$\mathrm{Out}'(A) = \{u \in \mathscr{R}(M_0) | \forall i \leq n\ \forall v \in u \backslash A[i-1] \exists u_i \in U_i\ u_i \prec v\}$$

with $A: M_0 \xrightarrow{U_1} M_1 \xrightarrow{U_2} \cdots \xrightarrow{U_{n-1}} M_{n-1} \xrightarrow{U_n} M_n$.

We may therefore characterize alternatively $\mathscr{E}(M)$ as the set of redex occurrences of M which are outermost in every derivation issued from M:

$$\mathscr{E}(M) = \bigcap_{A \in \mathscr{D}(M)} \mathrm{Out}'(A).$$

We shall now show that $\mathscr{E}(M)$ is not empty if M is not a normal form. But first we need a technical lemma.

NOTATION We write $\rightarrow_{\mathrm{int}}$ for $\rightarrow - \xrightarrow{\Lambda}$ (internal reduction). We say that derivation A is *internal* iff it is composed of internal reductions. That is, A in $\mathscr{D}(M)$ is internal iff $M = FM_1 \ldots M_p$ and A preserves $1, 2, \ldots, p$ (and therefore $\mathscr{F}(A) = FN_1 \ldots N_p$). Of course, if A is internal, every permutation of A is internal.

LEMMA 3.24 Let $A: M \rightarrow_{\mathrm{int}}^* \xrightarrow{\Lambda} N$. For any B in $\mathscr{D}(N)$ we have $\mathscr{X}(AB) = \mathscr{X}(A)$.

Proof Obvious from definitions, since $\Lambda \in \mathscr{X}(B)$. ▨

LEMMA 3.25 $M \in \mathscr{NF}$ iff $\mathscr{E}(M) = \varnothing$.

Proof If M is a normal form, $\mathscr{R}(M) = \varnothing$ and thus $\mathscr{E}(M) = \varnothing$. Conversely, by induction on M. Assume $M \in \mathscr{NF}$. We show $\mathscr{E}(M) \neq \varnothing$.

Case 1: Every A in $\mathscr{D}(M)$ is internal. Thus $M = F(M_1, \ldots, M_p)$ and for some $k \leq p$, $M_k \notin \mathscr{NF}$. By induction hypothesis there exists u_k in $\mathscr{E}(M_k)$, and therefore ku_k is in $\mathscr{E}(M)$.

Case 2: There is some $A: M \rightarrow_{\mathrm{int}} \xrightarrow{\Lambda}$. Let us show that $\mathscr{E}(A) \subseteq \mathscr{E}(M)$. Contrariwise, assume that there exists B in $\mathscr{D}(M)$ and u in $\mathscr{E}(A) - \mathscr{X}(B)$. Since $B \sqsubseteq A \sqcup B$, we have $u \notin \mathscr{X}(A \sqsubseteq B)$ by lemma 3.11. But $\mathscr{X}(A \sqcup B) = \mathscr{X}(A)$ by lemma 3.24, a contradiction with $u \in \mathscr{E}(A)$. Therefore, $\mathscr{E}(A) \subseteq \mathscr{E}(M)$, and since $A \not\equiv 0$, we get $\mathscr{E}(M) \neq \varnothing$ by lemma 3.15. ▨

This permits us to define the notion of *normal derivation issued from a term* M, similarly to what happens in λ-calculus:

DEFINITION If $M \in \mathscr{NF}$, the *normal derivation issued from* M is 0. Otherwise, let u be the leftmost occurrence in $\mathscr{E}(M)$ and A the elementary derivation: $M \xrightarrow{u} N$. The *normal derivation issued from* M is A followed by the normal derivation issued from N.

THEOREM 3.26: NORMAL DERIVATION THEOREM If M admits a normal form N, the normal derivation issued from M will end in N; it is the standard in the class of all derivations going from M to N.

Proof Let M be a term admitting a normal form N and A be some derivation from M to N. Let now B be any derivation issued from M. We have $B \sqsubseteq A$ and thus $\mathscr{X}(A) \subseteq \mathscr{X}(B)$ by lemma 3.11. This shows that $\mathscr{X}(M) = \mathscr{X}(A)$: the external occurrences of a term admitting a normal form are the occurrences external for any derivation going to this normal form. Assume now there is some u in $(\mathscr{X}(A) \cap \mathscr{R}(M)) - \mathscr{R}(A)$, i.e., u is a redex occurrence in M external to A but not contributing to A. According to lemma 3.12 we have $u \backslash A = \{u\}$, contrary to the hypothesis that N is in normal form. This shows that $\mathscr{E}(M) = \mathscr{E}(A)$. With a simple induction on $|A|$, it is now easy to show that $\mathrm{st}(A)$ is the normal derivation issued from M. ▨

The notion of normal derivation may be considered as the generalisation to term rewriting systems of the call by name computation rule for recursive equations [2, 6, 8, 27]. In the terminology of [24], we have that all our systems are d-outer, using the dominance ordering d defined by "$u(dM)v$ iff either $u \preceq v$ or $u|v$, $u \in \mathscr{X}(M)$, and $v \notin \mathscr{X}(M)$."

In the next section, we are going to extend these results to define a call-by-need computation rule for terms possessing a normal form. Before that, let us remark that it is possible to extend lemma 3.24 to external redex occurrences, yielding a canonical form for the standard of external derivations issued from a given term.

LEMMA 3.27 Let $A: M \to^*_{\mathrm{int}} \xrightarrow{\Lambda} N$. For any B in $\mathscr{C}(N)$ we have $\mathscr{E}(AB) = \mathscr{E}(A)$.

Proof Let $u \in \mathscr{R}(AB) - \mathscr{R}(A)$. We have $u \notin \mathrm{Out}(AB)$ and therefore $u \notin \mathscr{E}(AB)$. ▨

LEMMA 3.28 Let $A: M \to^*_{\mathrm{int}} \xrightarrow{\Lambda} N$. For any B in $\mathscr{D}(N)$, AB is standard iff A and B are standard.

Proof An easy induction on $|A|$, using lemma 3.27 above. ▨

COROLLARY Let $A: M \to^*_{\mathrm{int}} \xrightarrow{\Lambda} N$ be standard. For every noninternal B in $\mathscr{D}(N)$ we have $\mathrm{st}(B) = AC$, with $C = \mathrm{st}(B \backslash A)$.

Proof If B is not internal, it is easy to show that $A \sqsubseteq B$. Consider $C = \mathrm{st}(B \backslash A)$. Since A is standard, AC is standard by lemma 3.28. But $C \equiv B \backslash A$ implies then that $AC \equiv B$. By unicity of standard derivations, we get $\mathrm{st}(B) = AC$. ▨

3.3 Call by Need

We are interested in defining an interpreter for any TRS Σ in our class. Such an interpreter will be defined by a *computation rule*, where we choose in any term M some redex occurrence to contract next. The interpreter will compute correctly if, started on any term M possessing a normal form N, it defines a derivation that terminates in N after a finite number of steps.

Note that this question is non-trivial only if for some rule $\alpha \to \beta$ in Σ we have $\mathscr{V}(\alpha) - \mathscr{V}(\beta) \neq \varnothing$, since otherwise any computation rule is correct; we leave this easy proof to the reader.

In the general case, we saw in the last section that the normal computation rule (leftmost of $\mathcal{E}(M)$) is correct. Actually, with the help of lemma 3.16 we could easily have proven correct a more general computation rule: contract any redex occurrence in $\mathcal{E}(M)$. Intuitively, the redexes named by $\mathcal{E}(M)$ *need* to be contracted in order to get to the normal form of M. We shall here formalize this concept, and generalize the results of 3.2 by defining precisely what is a correct call-by-need interpreter.

DEFINITION 3.29 If M admits a normal form N, then $u \in \mathcal{R}(M)$ is a redex occurrence *needed for the normal form*, in symbols $u \in \mathcal{N}(M)$ iff $u \in \mathcal{R}(A)$ for every A: $M \xrightarrow{*} N$.

In order to prove further properties of needed redexes, consider first a few technical lemmas about outside-in derivations. In the following, we abbreviate outside-in as "oi."

LEMMA 3.30 If $A \equiv B$ with A oi, then $\mathcal{R}(A) \subseteq \mathcal{R}(B)$.

Proof Let $A = A_1 A_2$ with A_1: $M \xrightarrow{u} M'$. Then $u \in \mathcal{E}(A) = \mathcal{E}(B)$. Thus $u \in \mathcal{R}(B)$. Furthermore, $\mathcal{R}(A_2) \subseteq \mathcal{R}(B \backslash A_1)$, by the induction hypothesis. Therefore, $\mathcal{R}(A) \subseteq \mathcal{R}(B)$. ▨

COROLLARY If M has the normal form N, then $\mathcal{N}(M) = \mathcal{R}(A)$, where A is any oi derivation from M to N.

LEMMA 3.31 If A is oi and $B \sqsubseteq A$, then $A \backslash B$ is oi (except for some empty steps).

Proof By induction on $|A|$. Let $A = A_1 A_2$, with A_1: $M \xrightarrow{u} N$. By the induction hypothesis, we already know that $A_2 \backslash (B \backslash A_1)$ is oi. Consider now $A_1 \backslash B$, i.e., $u \backslash B$. If $u \backslash B = \varnothing$, then $A \backslash B$ is obviously oi. Suppose now $u \backslash B \neq \varnothing$. Since A is oi and $A \equiv B \sqcup A$, we know that $u \in \mathcal{E}(A) = \mathcal{E}(A \backslash B)$. Thus $u \in \mathrm{Out}(B \sqcup A)$, by lemma 3.22, and $u \backslash B = \{u\}$. Now for all $C \equiv A \backslash B$, one has $u \in \mathrm{Out}(C)$, since $BC \equiv A$ and $u \in \mathcal{E}(A)$. Again by lemma 3.22, one has $u \in \mathcal{E}(A \backslash B)$. ▨

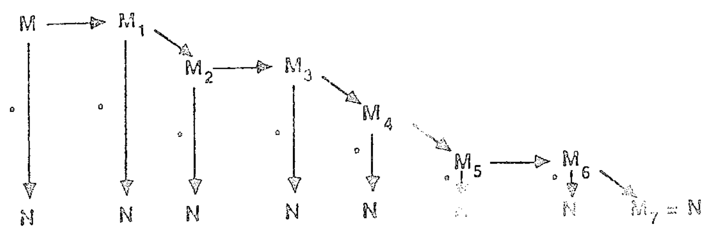
LEMMA 3.32 If A, B are oi and $A \equiv B$, then $|A| = |B|$.

Proof If $|A| = 0$, then $B \equiv 0$ and $B = 0$, since B is oi. Similarly if $|B| = 0$. Now let $A = A_1 A_2$ and $B = B_1 B_2$ with A_1: $M \xrightarrow{u} N$ and B_1: $M \xrightarrow{v} P$. Denote temporarily by C' the derivation C, where the empty steps are suppressed. As $\mathcal{E}(A) = \mathcal{E}(B)$ and $u \in \mathcal{E}(A)$, $v \in \mathcal{E}(A)$, we have u and v disjoint. Thus if $B_3 = B_1 \backslash A_1$ and $A_3 = A_1 \backslash B_1$, one has B_3: $N \xrightarrow{v} Q$ and A_3: $P \xrightarrow{u} Q$. Therefore, using the previous lemma and an induction on $|A| + |B|$, we get $|A| = 1 + |A_2| = 1 + |(B_2 \backslash A_3)'|$. Again by induction, $|(B_2 \backslash A_3)'| = |(A_2 \backslash B_3)'|$. But one also has $|B| = 2 + |(A_2 \backslash B_3)'|$. ▨

NOTATION Let $d(A) = |\mathrm{st}(A)|$.

By the previous lemma, since $\mathrm{st}(A)$ is oi, one has $d(A) = |B|$ for any B oi such that $B \equiv A$. The fact that external redexes are preserved until they are reduced gives us properties of this notion of distance.

LEMMA 3.33 If $A \sqsupseteq B$, then $d(A) \geq d(A \backslash B)$.



$$d(M) = d(M_1) > d(M_2) = d(M_3) > d(M_4) > d(M_5) = d(M_6) > d(M_7) = 0.$$

Figure 8

Proof Obvious from the two previous lemmas. ▫

LEMMA 3.34 If $A \sqsupseteq B$ and $|B| = 1$, then $d(A) > d(A\backslash B)$ iff $\mathscr{R}(\mathrm{st}(A)) \cap \mathscr{R}(B) \neq \emptyset$.

Proof Let $C = \mathrm{st}(A)$ and $C = C_1 C_2$, with $C_1: M \xrightarrow{U} N$ and $C_2: N \xrightarrow{V} P$. Then with $C \equiv A \sqsubseteq B$, we get $C \equiv B \sqcup C$ and $u \in \mathrm{Out}(B \sqcup C)$, since $u \in \mathscr{B}(C) = \mathscr{B}(B \sqcup C)$. This means that $u\backslash B = \emptyset$ iff $u \in V$. Now the lemma follows by induction on $|C|$, since $A\backslash B \equiv C\backslash B$ and by lemma 3.31 and lemma [illegible]. ▫

For the following corollary we denote by $d(M)$ the length of the standard derivation of M to its normal form when it exists.

COROLLARY Let M have the normal form N. Then $d(M) = 0$ iff $M = N$. In addition, let $M \xrightarrow{U} M'$. Then if $U \cap \mathscr{N}(M) = \emptyset$, one has $d(M) = d(M')$. Otherwise, $d(M) > d(M')$.

This is summarized in figure 8, where the slanted steps are needed and the vertical derivations are standard.

This concludes the proof of the correctness of the call-by-need computation rule. Actually, any interpreter which is fair for needed redexes, in the sense that it will never postpone forever the contraction of needed redexes, is correct for computing normal forms. Note that the standard derivation is the longest derivation contracting only needed redexes.

The next problem we shall tackle is how to effectively compute a needed redex in a given term. We know from the corollary to lemma 3.30 that $\mathscr{N}(M) = \mathscr{R}(A)$ for A any outside-in derivation going from M to its normal form. But this is useless practically. Intuitively we want our interpreter to compute an element of $\mathscr{N}(M)$ without looking ahead. This problem will be stated precisely and solved in part II of this paper (chapter 12).

References

[1] E. Ashcroft, W. Wadge. LUCID—A formal system for writing and proving programs. *SIAM J. on Computing* 5, no. 3 (1976): 336–354.

[2] G. Berry, J. J. Lévy. Minimal and optimal computations of recursive programs. *JACM* 26, no. 1, 1979.

[3] R. Burstall. Design considerations for a functional programming language. Infotech State of the Art Conf., Copenhagen, 1977.

[4] R. Burstall, J. A. Goguen. Putting theories together to make specifications. 5th IJCAI Conf., Boston, 1977.

[5] R. M. Burstall, D. B. Macqueen, D. T. Sannella. HOPE: An experimental applicative language. Report CSR-62-80. Computer Science Dept., University of Edinburgh, Feb. 1981.

[6] J. M. Cadiou. Recursive definitions of partial functions and their computations. Ph.D. thesis, Stanford, March 1972.

[7] H. B. Curry, R. Feys. *Combinatory Logic*, vol. 1. North-Holland, 1958.

[8] P. J. Downey, R. Sethi. Correct computation rules for recursive languages. *SIAM Journal* 5, no. 3, Sept. 1976.

[9] M. Gordon, R. Milner, L. Morris, M. Newey, C. Wadsworth. A metalanguage for interactive proof in LCF. Report CSR-16-77, Computer Science Dept., University of Edinburgh, Sept. 77.

[10] M. J. Gordon, A. J. Milner, C. P. Wadsworth. Edinburgh LCF. Springer-Verlag Lecture Notes in Computer Science, no. 78, 1979.

[11] J. V. Guttag, E. Horowitz, D. R. Musser. Abstract data types and software validation. Research report 76-48, ISI, Aug. 1976.

[12] R. Hindley. An abstract Church-Rosser theorem. Pt. 1, *J. Symbolic Logic* 34 (1969): 545–560; Pt. 2, *J. Symbolic Logic* 39 (1974): 1–21.

[13] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *JACM* 27, no. 4 (1980): 797–821.

[14] G. Huet, J. J. Lévy. Computations in regular rewriting systems, II. Chapter 12, this volume.

[15] G. Kahn, G. Plotkin. Domaines concrets. Rapport Laboria no. 336, IRIA, December 1978.

[16] S. C. Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

[17] D. E. Knuth, P. B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, ed. J. Leech. Pergamon Press, 1970, pp. 263–297.

[18] J. J. Lévy. Reductions correctes et optimales dans le lambda-calcul. Thesis, Université Paris 7, Jan. 1978.

[19] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *CACM* 3 (April 1960): 184–195.

[20] J. McCarthy. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*, ed. P. Braffort and D. Hirschberg. North-Holland, 1963, pp. 33–70.

[21] R. Milner. Implementation and application of Scott's logic for computable functions. *Proc. ACM Conf. on Proving Assertions about Programs*. SIGPLAN notices 7.1, Las Cruces, Jan. 1972.

[22] D. R. Musser. A data type verification system based on rewrite rules. 6th Texas Conf. on Computing Systems, Austin, Nov. 1978.

[23] M. Nivat. On the interpretation of recursive polyadic program schemes. *Symposia Mathematica*, vol. 15. Instituto Nazionale di Alta Matematica, Italy, 1975, pp. 225–281.

[24] M. O'Donnell. *Computing in Systems Described by Equations*. LNCS no. 58, Springer-Verlag, 1977.

[25] J. C. Raoult, J. Vuillemin. Operational and semantic equivalence between recursive programs. 10th SIGACT Conf., San Diego, 1978.

[26] B. K. Rosen. Tree manipulation systems and Church-Rosser theorems. *JACM* 20, no. 1, 1973.

[27] J. Vuillemin. Correct and optimal implementation of recursion in a simple programming language. *JCSS* 9, no. 3 (1974): 332–354.

12 Computations in Orthogonal Rewriting Systems, II

Gérard Huet and Jean-Jacques Lévy

Nonambiguous linear term rewriting systems were introduced and studied in the first part of this paper (chapter 11). In particular, it was shown that the space of derivations is a pushout category, and a standardization theorem was shown, leading to the existence of a correct *call by name* computation rule. Finally, *call by need* computations were defined. However, this does not lead in general to an effectively computable computation rule.

We shall in this part attack the problem of effective call-by-need computation rules.

The first problem we address is the definition of sequential interpreters for our systems. Using a sequentiality criterion derived from the semantic notion of sequentiality in concrete data types [11], we characterize an important subclass of our systems which admit a correct sequential strategy of interpretation. Intuitively, Σ is sequential iff it is possible to compute in call by need without looking ahead. That is, for any term M not in normal form, there exists a place in M which we need to compute in order to get to the normal form of M (if any), and furthermore, we can determine this place without looking at the subparts of M which are not computed yet. Unfortunately, it is undecidable in general whether Σ is sequential or not.

We finally consider a restriction of sequential systems which allows us to define an effective sequential strategy. Intuitively, a system Σ is strongly sequential if the computation effected to determine the needed positions is independent of the right hand sides of Σ. For a strongly sequential system Σ it is possible to represent the left-hand sides of Σ in a trie structure which we call matching dag. The computation of the needed subpart, together with the actual pattern-matching of the left-hand sides, is driven by the matching dag. We show that this computation is linear in the size of the term we compute on, extending to trees the technique of the Knuth-Morris-Pratt string-pattern-matching algorithm. Our technique uses a new notion of top-down tree automaton with a markers stack.

Strong sequentiality is decidable and is easily verified in many important subcases, under which fall, for instance, combinatory logic, primitive recursive schemes and recursive programming languages with constructors such as HOPE [5] and ML.

4 Sequential and Strongly Sequential Systems

4.1 A Definition of Sequentiality

Obviously, some rewriting systems require parallel evaluation strategies in order to compute normal forms. Take, for instance, the well-known example of the *parallel or*, defined by "true or $x \rightarrow$ true," "x or true $\rightarrow$ true," where we use an infix notation for the binary operator *or*. For computing any term $T = T_1$ or T_2, one needs a parallel evaluation of T_1 and T_2, since it is not generally decidable whether T_1 or T_2 produces the value *true*. Furthermore, in case both T_1 and T_2 evaluate to *true*, one easily shows

that needed redexes (in the sense of section 3) could not exist. This is because the above system is ambiguous. Fortunately, the nonambiguity condition prevents us from considering such systems. And we showed (in section 3) that call-by-need evaluation strategies exist in our systems. Hence, in order to get a normal form, one has just to contract needed redexes.

However, even if we know the existence of these critical needed redexes, there still remains to find them in any given term. Unfortunately, this can also demand some parallelism, as illustrated in the following example inspired from Berry [2] and which is obtained by a slight modification of *parallel or*. Consider any system Σ containing the three rules

$$\{F(A, B, x) \to C, F(x, A, B) \to C, F(B, x, A) \to C\}.$$

Note first that these rules are indeed nonambiguous, since none of these three left-hand sides can be unified with any other one. We know from section 3 that in any term $T = F(T_1, T_2, T_3)$ there is some needed redex. But as for the previous example of the *parallel or*, it is impossible to avoid parallel evaluation, since one cannot in general decide whether T_1, T_2, T_3 evaluate to A or B.

Thus, the nonambiguity condition is not sufficient for insuring a sequential evaluator. It seems necessary to consider the way in which terms are traversed in order to find some needed redex. We shall restrict our attention to top-down searches and develop some useful notations for prefixes of terms.

We represent prefixes by Ω-terms, i.e., by terms where a new nullary function symbol Ω can occur. Intuitively, Ω means the part of the term which has not been yet traversed. Let T_Ω be the set of these Ω-terms. Let us also consider the prefix ordering $\preceq$ on T_Ω obviously defined as

$\Omega \preceq M$ for any $M \in T_\Omega$,

$F(M_1, M_2, \ldots, M_n) \preceq F(N_1, \ldots, N_n)$ if $M_i \preceq N_i$ for $1 \leq i \leq n$,

$x \preceq x$ for any variable x.

All the previously defined operations on terms are obviously extended to Ω-terms. One just adds a very few new notions.

If $M \preceq P$ and $N \preceq P$, then M and N are said to be *compatible*, written $M \uparrow N$. The greatest lower bound (glb) of two Ω-terms M and N, written $M \sqcap N$, and the least upper bound (lub) of two compatible Ω-terms M and N, written $M \sqcup N$, are defined as follows:

$$F(M_1, \ldots, M_n) \sqcap F(N_1, \ldots, N_n) = F(M_1 \sqcap N_1, \ldots, M_n \sqcap N_n),$$

$$x \sqcap x = x,$$

$M \sqcap N = \Omega$ in all other cases

$$\Omega \sqcup M = M \sqcup \Omega = M,$$

$$F(M_1,\ldots,M_n)\sqcup F(N_1,\ldots,N_n)=F(M_1\sqcup N_1,\ldots,M_n\sqcup N_n),$$

$$x\sqcup x=x$$

It will be convenient to write M_Ω for $\sigma(M)$, where σ is the substitution such that $\sigma(x)=\Omega$ for all variables x. We define a predicate *nf* on Ω-terms by $nf(M)$ iff M has a normal form, i.e., $\exists N\in\mathcal{WF}$ such that $M\xrightarrow{*}N$. Note that we require N to be a term, i.e., without Ωs. An Ω-term N such that $\mathcal{R}(N)=\varnothing$ will be called an Ω-normal form. Suppose that the set $B=\{\mathrm{tt},\mathrm{ff}\}$ of boolean values is ordered by $\mathrm{ff}\sqsubseteq\mathrm{tt}$. Then it is straightforward to check that nf is a monotonic predicate on the set of Ω-terms:

LEMMA 4.1 If $M\preceq N$, then $\mathrm{nf}(M)\sqsubseteq\mathrm{nf}(N)$.

Proof First one shows that if $M\xrightarrow{*}N$ and $M\preceq M'$, there is an N' such that $M'\xrightarrow{*}N'$ and $N\preceq N'$. This is because our systems are linear. Second, if N is in normal form and $N\preceq N'$, then N' is also in normal form, since then $N=N'$. ▨

For the time being, we forget the problem of defining sequential evaluators and show the relevance for our systems of a notion of stability defined in Berry [2]. In [2] it is claimed in an axiomatic way that this notion is median between sequential and parallel functions, and a stable model of the lambda-calculus is constructed. (Note that the problem is still open for sequential models.) We show that the predicate *nf* is stable, i.e., conditionally multiplicative, in our systems. For this purpose, consider temporarily a new kind of rewriting rules, the ω-rules, associated with any system Σ and defined as follows: $M\to_\omega N$ iff, for some $u\in C(M)$ and redex scheme α in Σ, one has $M/u\uparrow\alpha_\Omega$ but not $\alpha_\Omega\preceq M/u$ and $N=M[u\leftarrow\Omega]$ (i.e., M/u is compatible with a redex scheme but is not a redex).

LEMMA 4.2 $(\to\cup\to_\omega)^*$ is confluent.

Proof We know already that $\xrightarrow{*}$ is confluent. Now if we temporarily write $M\xrightarrow{\varepsilon}N$ for $M\to N$ or $M=N$, and similarly for $M\xrightarrow{\varepsilon}_\omega N$, the nonambiguity condition of our system enables us to prove the lemma via the two properties summarized in the two diagrams of figure 9. ▨

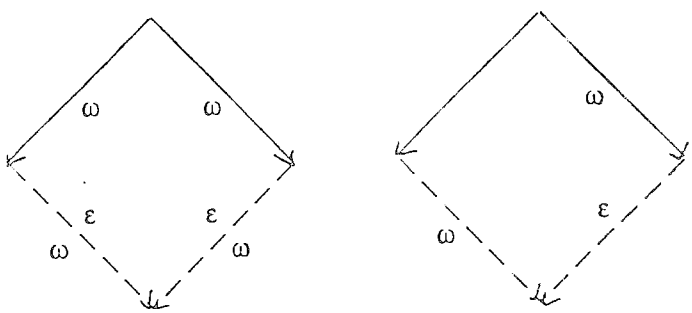


Figure 9
Church-Rosser diagrams for ω-rules.

The ω-rules will be also useful for the semantics of our systems. (Note that they correspond exactly to the ones defined by Wadsworth [23] in the λ-calculus.) Finally, let $\wedge$ also denote the trivial glb operation in truth values domain B (i.e., the usual *and*). We now prove Berry's stability property: nf is conditionally multiplicative.

LEMMA 4.3 If $M \uparrow N$, then $\mathrm{nf}(M \sqcap N) = \mathrm{nf}(M) \wedge \mathrm{nf}(N)$.

Proof By monotonicity, one easily has $\mathrm{nf}(M \sqcap N) \sqsubseteq \mathrm{nf}(M) \wedge \mathrm{nf}(N)$. Conversely, the only problem is when $\mathrm{nf}(M) \wedge \mathrm{nf}(N) = \mathrm{tt}$, i.e., when M and N have a normal form. By monotonicity, we know that $\mathrm{nf}(M \sqcup N) = \mathrm{tt}$. We work by induction on the length $d(M \sqcup N)$ of any outside-in derivation A from $(M \sqcup N)$ to its normal form. Let $A = A_1 A_2$, with $A_1 : (M \sqcup N) \xrightarrow{u} P$. Then $u \in \mathscr{E}(M \sqcup N)$. We claim that $u \in \mathscr{R}(M \sqcap N)$. Otherwise we have 3 cases.

Case 1: $u = u_1 u_2$, with $(M \sqcap N)/u_1 = \Omega$ and $u \in \mathscr{R}(M)$. Then since

$$M \preceq (M \sqcup N)[u \leftarrow \Omega],$$

we know by monotonicity that $(M \sqcup N)[u \leftarrow \Omega]$ has a normal form. This contradicts $u \in \mathscr{E}(M \sqcup N)$.

Case 2: $u = u_1 u_2$, with $(M \sqcap N)/u_1 = \Omega$ and $u \in \mathscr{R}(N)$. Same as in case 1.

Case 3: $u \in \mathscr{C}(M \sqcap N)$, but $u \notin \mathscr{R}(M \sqcap N)$. This means that if α is the redex scheme such that $(M \sqcup N)/u = \sigma\alpha$, one has $M/u \preceq (M \sqcup N)/u \uparrow \alpha_\Omega$, and similarly $N/u \uparrow \alpha_\Omega$. Now since $u \notin \mathscr{R}(M \sqcap N)$, one cannot have $\alpha_\Omega \preceq M/u$ and $\alpha_\Omega \preceq N/u$. Assume, for instance, that not $\alpha_\Omega \preceq M/u$. Then $M \to_\omega M[u \leftarrow \Omega]$. By the previous confluence property, $M[u \leftarrow \Omega]$ has a normal form. Let $M_1 = M[u \leftarrow \Omega]$. We conclude that $(M_1 \sqcup N)[u \leftarrow \Omega]$ has a normal form by reasoning as in the first two cases, and by monotonicity, $(M \sqcup N)[u \leftarrow \Omega]$ has a normal form. This contradicts $u \in \mathscr{E}(M \sqcup N)$.

Thus $u \in \mathscr{R}(M \sqcap N)$. Now, let $M \xrightarrow{u} M'$ and $N \xrightarrow{u} N'$. Since $\mathrm{nf}(M) = \mathrm{nf}(N) = \mathrm{tt}$, one has also that $\mathrm{nf}(M') = \mathrm{nf}(N') = \mathrm{tt}$. Furthermore, $M \sqcup N \xrightarrow{u} M' \sqcup N'$. Finally, by induction we know that $M' \sqcap N'$ has a normal form, and thus $\mathrm{nf}(M \sqcap N) = \mathrm{tt}$. □

COROLLARY For any Ω-term M having a normal form there is a (unique) minimum prefix N of M having a normal form.

This is not true in general in ambiguous systems. Consider again the *parallel or* example. Let $M =$ true or true. Then, both "true or Ω" and "Ω or true" have a normal form, but not "Ω or Ω." Now we can also consider the already mentioned example of Berry [2], which is linear and nonambiguous. Let $T = F(T_1, T_2, T_3)$. We know by the previous result that there is a minimum prefix of T having a normal form if $\mathrm{nf}(T) = \mathrm{tt}$. But nothing ensures us that for all Ts of this form, we need to compute always a fixed argument of F. Here this is false, since $\mathrm{nf}(F(\Omega, \Omega, \Omega)) = \mathrm{ff}$, but $\mathrm{nf}(F(A, B, \Omega)) = \mathrm{tt}$, $\mathrm{nf}(F(\Omega, A, B)) = \mathrm{tt}$, and $\mathrm{nf}(F(B, \Omega, A)) = \mathrm{tt}$. (Note that this does not contradict our last result, since these last three terms are not compatible with respect to the prefix ordering). Using our new notations for prefixes of terms, we can state what is going wrong in the previous example. In $M = F(\Omega, \Omega, \Omega)$ it is possible to get normal forms

by increasing some Ωs. But there is not a uniform one to consider. One can always increase the two other Ωs in order to get a normal form.

For short, let a sequential system be a term rewriting system which has a sequential evaluator. (Thus the two previously considered systems are not sequential.) In a sequential system it seems required that in any Ω-term in Ω-normal form (having at least one Ω occurrence) there is an occurrence of Ω which it is necessary to increase in order to make the nf predicate true. This means, when we forget Ω-terms, that for all terms not in normal form there is a redex occurrence u such that if we consider redexes as black boxes which might produce any term, it is impossible to get a normal form without computing the redex of occurrence u. In other words, there is always one needed redex occurrence which can be found without any look-ahead computation.

Technically, we want the predicate *M has a normal form* to be sequential in the sense of Kahn-Plotkin [17]. So let us first recall what a sequential predicate is in their sense.

DEFINITION 4.4 Let P be a monotonic predicate on T_Ω (with the truth values domain B ordered as before). An occurrence u of M is said to be an *index* of P in M iff $M/u = \Omega$ and $\forall N$ such that $N \geq M$, $P(N) = \text{tt}$ implies $N/u \neq \Omega$. Then P is *sequential* at M iff whenever $P(M) = \text{ff}$ and $\exists N \geq M$ such that $P(N) = \text{tt}$, it follows that there exists an index of P in M. Finally, P is said to be sequential iff it is sequential at every M in T_Ω.

Thus, if P is a sequential predicate and if there is an Ω in M and $P(M) = \text{ff}$, there is a critical Ω in M, namely an index, to increase in order to make the predicate true.

DEFINITION 4.5 Σ is a sequential system iff nf is a sequential predicate.

Note that this definition is independent of the linearity and nonambiguity assumptions. However, when these assumptions are true, it is possible to show that sequentiality is only required at Ω-normal forms. So let $\omega'(M)$ be M, where all (outermost) redexes are replaced by the constant Ω.

LEMMA 4.6 Σ is a sequential system iff the predicate $\text{nf}(M)$ is sequential at any Ω-normal form M (i.e., such that $M = \omega'(M)$).

Proof By induction on $n = \text{Min}\{d(N) | M \preceq N, \text{nf}(N) = \text{tt}\}$. Let $n = 0$. Then there is N in normal form such that $M \preceq N$. Thus $M = \omega'(M)$ and this case is trivial. Let $n > 0$ and u be an index of nf in $\omega'(M)$. We have two possibilities. First $M/u = \Omega = \omega'(M)/u$. As $\omega'(M) \preceq M$, the occurrence u is also an index in M. Second, $M/u \neq \Omega$, which means M/u is a redex. Then $\text{nf}(N[u \leftarrow \Omega]) = \text{ff}$ for all N such that $M \preceq N$, and $\text{nf}(N) = \text{tt}$. Thus $u \in \mathcal{N}(N)$. Let $M \xrightarrow{u}_k M'$ and $N \xrightarrow{u}_k N'$. Then $d(N') < d(N)$, by lemma 3.34. Therefore, by induction there is an index v' of nf in M'. Now let v be the occurrence of M defined by $v = v'$ if u and v' are disjoint, $v = uwu'$ if $v' = uw'u'$, where $\alpha_k/w = \beta_k/w' \in \mathcal{V}$. One then easily checks that v is an index of nf in M. ∎

This last lemma is not true for ambiguous systems. Take

$$\Sigma = \{F(A, B, x) \to K, F(B, x, A) \to K, C \to A, C \to B\}$$

and consider $M = F(C, \Omega, \Omega)$. Then there is no index in M, but Σ is sequential at any Ω-normal form.

Going back to the evaluation strategy problem, an easy terminating strategy can be designed: for any term M, contract a redex at occurrence u which is an index of nf in $\omega'(M)$. (Then $u \in \mathcal{N}(M)$ if M has a normal form.) But, in general, these indexes are not computable. For instance, take Σ containing the following four rules:

$F(G(A, x), B) \to K$

$F(G(x, A), C) \to K$

$F(D, x) \to K$

$G(E, E) \to \cdots$

Then if $M = F(G(\Omega, \Omega), \Omega)$, the occurrence $u = 2 \in \mathcal{O}(M)$ is an index iff $G(E, E)$ cannot derive to D, which is not decidable, since Σ can have many other rules. Therefore, neither indexes, nor sequentiality are decidable. In order to make these two questions effective, we consider more restricted systems. The idea is to forget right hand sides of rules and to make sequentiality be determined only from the left hand sides.

4.2 Strongly Sequential Systems

In order to forget right hand sides of rules, we introduce a new derivation relation; the intuition behind it is to permit any redex to produce any term.

NOTATION Let M and N be two Ω-terms. We say that M possibly reduces to N, and write $M \to' N$, iff $N = M[u \leftarrow P]$ for some redex occurrence $u \in \mathcal{R}(M)$ and Ω-term P. Furthermore, let $\mathrm{nf}'(M) = \mathrm{tt}$ iff $M \overset{*}{\to}' N$ for some term N in normal form.

DEFINITION 4.7 Σ is a *strongly sequential system* iff the predicate nf′ is sequential at any M in Ω-normal form. We denote by $\mathcal{S}(M)$ the set of indexes of nf′ in M.

The restriction to Ω-normal forms is necessary here because lemma 4.6 is no longer true for strong sequentiality. (Take $\Sigma = \{F(A, B, x) \to K, F(B, x, A) \to K\}$ and consider $M = F(R, \Omega, \Omega)$, where R is a redex. Then $\omega'(M) = F(\Omega, \Omega, \Omega)$ has an index for nf′, but M has not.) However, since $\mathrm{nf}'(M) = \mathrm{ff}$ iff M is in Ω-normal form and $\mathrm{nf}(M)$ implies $\mathrm{nf}'(M)$ for any M, a strongly sequential system is sequential by lemma 4.6.

DEFINITION 4.8 A redex occurrence $u \in \mathcal{R}(M)$ is strongly needed iff $u \in \mathcal{S}(\omega'(M))$.

Clearly, if $u \in \mathcal{R}(M)$ is strongly needed, then u is needed, i.e., $u \in \mathcal{N}(M)$. We want to show that strongly needed redex occurrences in M can be found effectively and that the strong sequentiality of any system Σ can be decided. A priori, this is not too straightforward. Consider for instance Σ having the following redex schemes:

$F(G(A, x), F(B, y)) \to \cdots$

$F(G(x, A), F(C, y)) \to \cdots$

$G(D, D) \to \cdots$

Then the smallest Ω-term without indexes for nf′ is

$M = F(G(\Omega, \Omega), F(G(\Omega, \Omega), \Omega))$.

(Notice that M is not a prefix of any redex scheme.)

4.3 Decision Procedures for Strongly Needed Redexes

We show that strongly needed redexes can be computed with the help of some *direct approximation* function. (An alternative characterization will be also given in the next section). This function will represent for any term the maximum prefix which is guaranteed to remain after any derivation. Before introducing this function, we need to introduce some notation.

If E is any set of Ω-terms, we write $M \preceq E$ iff $\exists N \in E$ such that $M \preceq N$. Similarly for $M \succeq E$ and $M \uparrow E$. Finally, with $\text{red}_\Omega = \{\alpha_\Omega | \alpha \in \text{Red}\}$, we write $M \uparrow$ for $M \uparrow \text{red}_\Omega$.

DEFINITION 4.9 For any Ω-term M, we define its *direct approximation* $\omega(M)$ and *internal direct approximation* $\bar{\omega}(M)$ by

$\omega(x) = \bar{\omega}(x) = x$,

$\omega(\Omega) = \bar{\omega}(\Omega) = \Omega$,

$\bar{\omega}(F(M_1, M_2, \ldots, M_n)) = F(\omega(M_1), \omega(M_2), \ldots, \omega(M_n))$,

$\omega(F(M_1, M_2, \ldots, M_n)) = \Omega$ if $\bar{\omega}(F(M_1, M_2, \ldots, M_n))\uparrow$,

$= \bar{\omega}(F(M_1, M_2, \ldots, M_n))$ otherwise.

Remark We can see $\omega(M)$ as the $\to_\omega$-normal form of $\omega'(M)$. It follows that $\omega(M) = \bigsqcap \{N | M \overset{*}{\to}' N\}$. Intuitively, $\omega(M)$ is the maximum prefix of M guaranteed to exist in every N which can be derived from M, whatever could be the right-hand sides of the rules in Σ. Our terminology is consistent with [23] for λ-calculus and [18, 20] for recursive program schemes.

We note that in our TRSs, subparts of redex schemes are their own direct approximation:

LEMMA 4.10 Let $\alpha \in \text{red}$. For every nonempty u in $\mathcal{O}(\alpha)$, we have $\omega(\alpha_\Omega/u) = \alpha_\Omega/u$.

Proof Induction on the size of α_Ω/u. If $\alpha_\Omega/u = \Omega$, it is trivial. Otherwise, by induction hypothesis we get $\bar{\omega}(\alpha_\Omega/u) = \alpha_\Omega/u$. By nonambiguity, we cannot have $\alpha_\Omega/u\uparrow$, and thus $\omega(\alpha_\Omega/u) = \alpha_\Omega/u$. □

It follows easily from its definition that ω is monotonic increasing, i.e., $M \preceq N$ implies $\omega(M) \preceq \omega(N)$, idempotent, i.e., $\omega(M) = \omega^2(M)$, and a projection, i.e., $\omega(M) \preceq M$. More important, if $M \overset{*}{\to} N$, then $\omega(M) \preceq \omega(N)$. Namely, approximation is increasing with derivation (In fact, this is true also with $\overset{*}{\to}'$). Finally, an Ω-term M is said to be *ω-maximum* iff for every N such that $N \succeq M$ we have $\omega(N) = \omega(M)$.

Now we connect strongly sequential systems and the direct approximation function by showing that $\mathrm{nf}'(M) = \mathrm{tt}$ if M is ω-maximum.

NOTATION Let $x \in \mathcal{V}$. For any $M \in \mathcal{T}_\Omega$ we define M_x as M in which all Ωs are replaced by xs, i.e., $M_x = M[u \leftarrow x | M/u = \Omega]$.

LEMMA 4.11 $M \xrightarrow{*}{}' N$ iff $\omega(M_x) \preceq N_x$ for some $x \notin \mathcal{V}(M) \cup \mathcal{V}(N)$.

Proof By induction on the size of M. The substitution of variables for the Ωs is a technical trick. ▨

LEMMA 4.12 Let $x \in \mathcal{V}$ and $u \in \mathcal{O}(M)$ such that $M/u = \Omega$. Then $\omega(M) = \omega(M[u \leftarrow x])$ iff $\omega(M) = \omega(M[u \leftarrow N])$ for all Ω-terms N.

Proof Obvious, once one remarks that $M[u \leftarrow x]\uparrow$ implies $M[u \leftarrow N]\uparrow$ for every N. ▨

LEMMA 4.13 We have $\mathrm{nf}'(M) = \mathrm{tt}$ iff M is ω-maximum iff $\omega(M) = \omega(M_x)$, where x is any variable.

Proof Let $\mathrm{nf}'(M) = \mathrm{tt}$. Then $M \xrightarrow{*}{}' N$ for some normal form N. Thus $\omega(M_y) \preceq N_y$ by lemma 4.11 if $y \notin \mathcal{V}(M) \cup \mathcal{V}(N)$. But $N_y = N$ as N is a normal form (without Ω's). Thus $y \notin \mathcal{V}(\omega(M_y))$ and $\omega(M_y) \preceq M$. By monotonicity one gets $\omega(\omega(M_y)) \preceq \omega(M)$. Hence $\omega(M_y) \preceq \omega(M)$, since ω is idempotent. Finally, $\omega(M) = \omega(M_y)$ because $M \preceq M_y$. By lemma 4.12, the Ω-term M is ω-maximum. Conversely, let M be ω-maximum. Then $\omega(M) = \omega(M_y)$ for some $y \notin \mathcal{V}(M)$. Thus $\omega(M_y) \preceq M \preceq M_z$ for some $z \neq y$. Therefore, $M \xrightarrow{*}{}' M_z$, since $y \notin \mathcal{V}(M) \cup \mathcal{V}(M_z)$ by lemma 4.11, and $\mathrm{nf}'(M) = \mathrm{tt}$. Now M is ω-maximum iff $\omega(M) = \omega(M_x)$, by lemma 4.12. ▨

Therefore, one can easily decide if $\mathrm{nf}'(M) = \mathrm{tt}$ by testing $\omega(M_x) = \omega(M)$. Similarly, an index of the nf' predicate can also be effectively found with the help of the following lemma.

LEMMA 4.14 Let $x \in \mathcal{V}$ and $u \in \mathcal{O}(M)$ such that $M/u = \Omega$. Then $u \in \mathcal{I}(M)$ iff $\omega(M) \neq \omega(M)[u \leftarrow x])$ iff $u \in \mathcal{O}(\omega(M[u \leftarrow x]))$.

Proof Let $u \notin \mathcal{I}(M)$. That is, there is some $N \succeq M$ such that $\omega(N) = \omega(N_x)$ and $N/u = \Omega$. Then $M[u \leftarrow x] \preceq N_x$, and $\omega(M[u \leftarrow x] \preceq \omega(N_x) = \omega(N)$. Without loss of generality, by lemma 4.12 we may assume $x \notin \mathcal{V}(N)$. Thus $u \notin \mathcal{O}(\omega(M[u \leftarrow x]))$, and $\omega(M[u \leftarrow x]) \preceq M$, which implies that $\omega(M[u \leftarrow x]) \preceq \omega(M)$. Therefore, $\omega(M) = \omega(M[u \leftarrow x])$. For the converse, assume $\omega(M) = \omega(M[u \leftarrow x])$. This implies that Σ is not empty. Let α be any redex scheme, and consider

$$N = M[v \leftarrow \alpha | M/v = \Omega \text{ and } v \neq u].$$

We have $N \succeq M$ and $\omega(N_x) = \omega(M[u \leftarrow x]) = \omega(M) \preceq \omega(N)$. Thus $\omega(N_x) = \omega(N)$, and since $N/u = \Omega$, we have $u \notin \mathcal{I}(M)$. Finally, $\omega(M) = \omega(M[u \leftarrow x])$ implies $u \notin \mathcal{O}(\omega(M[u \leftarrow x]))$, which concludes the proof. ▨

In order to find a strongly needed redex occurrence u in any given term M, one has just to test $u \in \mathscr{C}(\omega(M[u \leftarrow x]))$ for some redex occurrence u and any given variable x. Thus a strongly needed redex is a redex which may increase, after its reduction, the direct approximation.

Let us now give a few properties of indexes needed in the following.

LEMMA 4.15 If $uv \in \mathscr{I}(M)$ and $\omega(M/u) = \Omega$, then $u \in \mathscr{I}(M[u \leftarrow \Omega])$.

Proof Since $\omega(M[uv \leftarrow x]) \neq \omega(M) = \omega(M[u \leftarrow \Omega])$, $\omega(M[u \leftarrow x]) \neq \omega(M[u \leftarrow \Omega])$, by lemma 4.12 with $N = M/u[v \leftarrow x]$. ▨

LEMMA 4.16 If $uv \in \mathscr{I}(M)$, then $v \in \mathscr{I}(M/u)$.

Proof Let $uv \in \mathscr{I}(M)$ and $v \notin \mathscr{I}(M/u)$. Then $\omega(M/u) = \omega(M/u[v \leftarrow x])$, by lemma 4.14, and therefore

$\omega(M) = \omega(M[u \leftarrow \omega(M/u)] = \omega(M[u \leftarrow \omega(M/u[v \leftarrow x])]) = \omega(M[uv \leftarrow x])$,

which contradicts $uv \in \mathscr{I}(M)$. ▨

LEMMA 4.17 If $\omega(M/u) = \Omega$, then for every v such that $v|u$, we have $v \in \mathscr{I}(M)$ equivalent to $v \in \mathscr{I}(M[u \leftarrow \Omega])$.

Proof Obvious, since $\omega(M/u) = \Omega$ implies

$\omega(M) = \omega(M[u \leftarrow \Omega])$

$\omega(M[v \leftarrow x]) = \omega(M[u \leftarrow \Omega][v \leftarrow x])$. ▨

We are now left with the problem of deciding whether a TRS Σ is strongly sequential or not. Since this property is closely related to the organization of matching of redexes, we shall first consider this problem.

5 Deterministic Automata for Strongly Sequential Systems

In this section we show how the previous definitions permit to derive algorithms for computing strongly needed redexes. First we consider matching of redexes and see that this operation is sequential in case of strongly sequential systems.

5.1 Sequential Sets, Directions

DEFINITION 5.1 Let E be any set of Ω-terms. The predicate *match* is defined by $\text{match}(M, E) = \text{tt}$ iff $M \geq E$. We say that E is *sequential* iff $\text{match}(M, E)$ is sequential at every M. The set $\text{Dir}(M, E)$ of the indexes of this predicate in M is called the set of *directions* from M to E. As a particular case we shall consider the predicate $\text{isredex}(M) = \text{match}(M, \text{red}_\Omega)$ and will abbreviate its set of indexes $\text{Dir}(M, \text{red}_\Omega)$ as $\text{Dir}(M)$.

We have, of course, $\mathscr{I}(M) \subseteq \text{Dir}(M)$, and we shall show below that if Σ is strongly sequential, isredex is also sequential. Furthermore, the decidability of sequentiality of isredex is straightforward.

NOTATION We write $M \# N$ iff M and N are incompatible, i.e., not $M \uparrow N$, and also $M \# E$ iff not $M \uparrow N$, i.e., $M \# N$ for all N in E, and $M \#$ iff $M \# \mathrm{red}_\Omega$.

LEMMA 5.2 Let M be an Ω-term and E be a set of Ω-terms. Then the following are equivalent:

$u \in \mathrm{Dir}(M, E)$

$M/u = \Omega$, and for all $N \in E$ such that $N \uparrow M$, one has $u \in \mathcal{O}(N)$ and $N/u \neq \Omega$

$M/u = \Omega$ and $M[u \leftarrow x] \# E$, where x is not a variable of any Ω-term in E.

Proof Obvious. ▨

LEMMA 5.3 If $N \in E$ and $N \uparrow M$, then $\mathrm{Dir}(M, E) = \mathrm{Dir}(M \sqcap N, E')$, where $E' = \{P \in E | P \uparrow M\}$.

Proof Suppose $u \in \mathrm{Dir}(M, E)$. Then $M/u = \Omega$. But, since $N \in E$ and $N \uparrow M$, lemma 5.2 says that $u \in \mathcal{O}(N)$. Thus $u \in \mathcal{O}(M \sqcap N)$ and $(M \sqcap N)/u = \Omega$. Now if $P \in E'$ and $P \uparrow (M \sqcap N)$, then $P \uparrow M$ and $u \in \mathcal{O}(P)$, with $P/u \neq \Omega$. Thus $u \in \mathrm{Dir}(M \sqcap N, E')$. Conversely, since $(M \sqcap N)/u = \Omega$ and $N \in E'$, with $N \uparrow (M \sqcap N)$, one has $u \in \mathcal{O}(N)$ and $N/u \neq \Omega$. Thus $M/u = \Omega$ because $N \uparrow M$. Now take any $P \in E$ such that $P \uparrow M$. Then $P \in E'$ and $P \uparrow (M \sqcap N)$. Thus $u \in \mathcal{O}(P)$ and $P/u \neq \Omega$. That is, $u \in \mathrm{Dir}(M, E)$. ▨

LEMMA 5.4 For any finite E, one can decide if E is sequential.

Proof One just checks that $\mathrm{match}(M, E)$ is sequential at every M such that $M \preceq E$. ▨

LEMMA 5.5 If Σ is a strongly sequential system, then isredex is a sequential predicate.

Proof If E is sequential, then $\mathrm{match}(M, E)$ is sequential at every M and in particular at every Ω-term prefix of some element of E. Conversely, E is sequential iff $\mathrm{Dir}(M, E) \neq \varnothing$ for all M not in normal form such that $\mathrm{match}(M, E) = \mathrm{ff}$. Now if $M \# E$, then

$\mathrm{Dir}(M, E) = \{u \in \mathcal{O}(M) | M/u = \Omega\} \neq \varnothing$,

since M is not in normal form. Assume that $M \uparrow N$ for some $N \in E$. Then $\mathrm{Dir}(M, E) \supseteq \mathrm{Dir}(M \sqcap N, E)$, by lemma 5.3. But $M \sqcap N \preceq N$ and $\mathrm{match}(M \sqcap N, E) = \mathrm{ff}$, since $\mathrm{match}(M, E) = \mathrm{ff}$. Thus $\mathrm{Dir}(M \sqcap N, E) \neq \varnothing$, which implies $\mathrm{Dir}(M, E) \neq \varnothing$.

Now let Σ be strongly sequential. Then nf′ is sequential at every M in Ω-normal form (i.e., $\omega'(M) = M$), by definition. Thus isredex is sequential at every Ω-normal form M. But since Σ is not ambiguous, every N such that $N \preceq \mathrm{red}_\Omega$ is in Ω-normal form. Thus $\mathrm{isredex}(M) = \mathrm{match}(M, \mathrm{red}_\Omega)$ is sequential at every M according to the first part of the proof. ▨

Intuitively, a set E of Ω-terms is sequential iff we can factor the matching of any term against E. As in Curien [6], for the general case of sequential functions, this factorization can be achieved here by a tree gathering all possible patterns. For instance, take

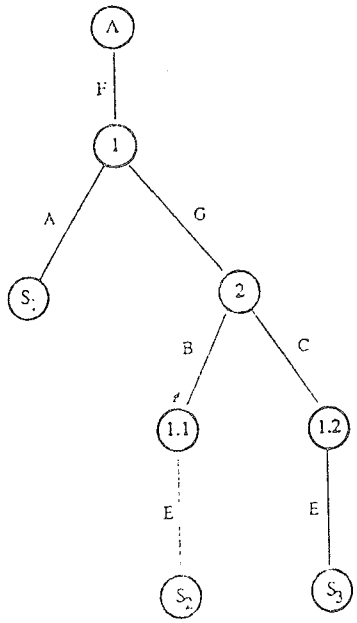


Figure 10

$$E = \{F(A,\Omega), F(G(E,\Omega),B), F(G(\Omega,E),C)\},$$

which is sequential. The corresponding *matching tree* is shown in figure 10, where a node u corresponds to a query of the function symbol at occurrence u in the term M, branches from u are the different successful alternatives for this function symbol query, and leaves are conventionally written as success nodes S_i. These matching trees generalize the trie structures used to store dictionaries of strings.

This tree can easily be built up from Ω-terms M such that $M \preceq E$ by considering the $\mathrm{Dir}(M,E)$ sets. We shall see in the next section how, in the case of strongly sequential systems, we may construct matching dags which permit to factorize local and global matches, yielding an efficient (linear) match algorithm.

5.2 Matching Dags

For the time being, we forget strong sequentiality. We only define matching dags and shall show in the next section that these dags exist for a rewriting system iff it is strongly sequential.

NOTATION We shall write $M \prec \mathrm{red}_\Omega$ for $M \preceq \mathrm{red}_\Omega$ and $M \notin \mathrm{red}_\Omega$, and $M\uparrow_+$ for $M\uparrow$ and $M \neq \Omega$.

HYPOTHESIS We assume the existence of a function Q mapping every Ω-term M such that $M \prec \mathrm{red}_\Omega$ into a non-empty set of its directions, $\varnothing \neq Q(M) \subseteq \mathrm{Dir}(M)$, and such that $\forall u \in Q(M), \forall N \prec \mathrm{red}_\Omega$, if $M[u \leftarrow N] \prec \mathrm{red}_\Omega$, then

$\exists v\ uv \in Q(M[u \leftarrow N])$ (Q_1)

$\forall v\ uv \in Q(M[u \leftarrow N])$ implies $v \in Q(N)$. (Q_2)

As previously stated, the existence of such a mapping will be shown in the case of strong sequentiality. Intuitively, an occurrence in $Q(M)$ is a direction for matching redexes and is also a direction for internal matches (by condition Q_2). Furthermore, it is possible to find some (strongly needed) redex without any dangling partial matches (condition Q_1).

CONSTRUCTION To construct a matching dag associated with Q and verifying Q_1 and Q_2, let $\text{red} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. We consider $\text{Occ} = \bigcup \{\bar{\mathcal{C}}(\alpha_i) | i \leq n\}$ and a set $\text{Succ} = \{S_1, S_2, \ldots, S_n\}$ of success tokens disjoint from Occ. We construct a graph whose nodes are all

a. pairs $\langle M, v\rangle$ such that $M \prec \text{red}_\Omega$, $v \in Q(M)$ and $\forall u\ M/u\uparrow_+ \Rightarrow u \prec v$,
b. pairs $\langle \alpha_{i\Omega}, S_i\rangle$, with $1 \leq i \leq n$ (the success nodes).

Let $\langle M, v\rangle$ be a node, F a function symbol in $\mathcal{F}$, and $M' = M[v \rightarrow F\vec{\Omega}]$. If $\langle M', *\rangle$ is a node, where $*$ denotes any element of Occ $\cup$ Succ, we draw an arc: $\langle M, v\rangle \xrightarrow{F} \langle M', *\rangle$ It is clear that our graphs are directed acyclic. We shall therefore call them *matching dags*.

DIFINITION 5.6 The node $\langle M, *\rangle$ is said to be *accessible* iff either it is the origin node $\langle \Omega, \Lambda\rangle$ or there is some accessible node $\langle M_1, v_1\rangle$ and $F \in \mathcal{F}$ such that $\langle M_1, v_1\rangle \xrightarrow{F} \langle M, *\rangle$

LEMMA 5.7 If $\langle M, v\rangle$ is accessible, then for every u such that $M/u\uparrow_+$ the pair $\langle M[u \leftarrow \Omega], u\rangle$ is an accessible node and there exists in the matching dag a path

$$\langle M[u \leftarrow \Omega], u\rangle = \langle M[u \leftarrow N_1], uw_1\rangle \xrightarrow{F_1} \langle M[u \leftarrow N_2], uw_2\rangle$$
$$\xrightarrow{F_2} \cdots \xrightarrow{F_{k-1}} \langle M[u \leftarrow N_k], uw_k\rangle = \langle M, v\rangle.$$

Furthermore, for every such path there exists in the matching dag an identically labeled path

$$\langle \Omega, \Lambda\rangle = \langle N_1, w_1\rangle \xrightarrow{F_1} \langle N_2, w_2\rangle \xrightarrow{F_2} \cdots \xrightarrow{F_{k-1}} \langle N_k, w_k\rangle = \langle M/u, v/u\rangle.$$

Proof Induction on the definition of accessible. If $M = \langle \Omega, \Lambda\rangle$, the proof is trivial. Otherwise, assume $\langle M, v\rangle$ is an accessible node satisfying the condition above such that $\langle M, v\rangle \xrightarrow{F} \langle M', v'\rangle$. Now let u be such that $M'/u\uparrow_+$. Since $\langle M', v'\rangle$ is a node, we have $u \prec v'$. Similarly, as $\langle M, v\rangle$ is a node, the case $u|v$ is impossible, and therefore $u \preceq v$. $M'[u \leftarrow \Omega] = M[u \leftarrow \Omega]$, and $M'/u = M/u[v/u \leftarrow F\vec{\Omega}]$, and thus $M/u\uparrow$.

Case 1: $u = v$. Then $\langle M'[u \leftarrow \Omega], u\rangle = \langle M, v\rangle$ is accessible by hypothesis. Also $M'/u = F\vec{\Omega}$ and $M'/u\uparrow$ implies $M'/u \preceq \text{red}_\Omega$.

Case 2: Otherwise, we get $M/u\uparrow_+$, and by the induction hypothesis we get $\langle M'[u \leftarrow \Omega], u\rangle = \langle M[u \leftarrow \Omega], u\rangle$ accessible, and $\langle M/u, v/u\rangle$ is a node. In particular $M/u \prec \text{red}_\Omega$, and $v/u \in Q(M/u) \subseteq \text{Dir}(M/u)$. Therefore in this case too $M'/u \preceq \text{red}_\Omega$.

In both cases we have shown that $\langle M'[u \leftarrow \Omega], u\rangle$ is an accessible node, and the existence of the first path for $\langle M', v'\rangle$ follows, since $\forall j \leq k\ M[u \leftarrow N_j] = M'[u \leftarrow N_j]$, assuming the existence of the path for $\langle M, v\rangle$, as in the statement of the lemma. Let us now show that the corresponding path for $\langle M'/u, v/u\rangle$ can be extended. We have shown above that $M'/u \preceq \mathrm{red}_\Omega$. But since $\langle M', v'\rangle$ is a node, $M' \prec \mathrm{red}_\Omega$. We easily get $M'/u \prec \mathrm{red}_\Omega$, because either $u = \Omega$ or else we use the nonambiguity condition. Also, for all u', $M'/uu'\uparrow_+$ implies $u' \prec v'/u$. Finally, using $u \in Q(M[u \leftarrow \Omega]) = Q(M'[u \leftarrow \Omega])$, we get by condition Q_2 that $v'/u \in Q(M', u)$, and therefore $\langle M/u, v/u\rangle \xrightarrow{F} \langle M'/u, v'/u\rangle$. ▨

LEMMA 5.8 Suppose $\langle M, v\rangle$ is an accessible node, and $N = M[v \leftarrow F\vec{\Omega}]$. If $N \uparrow \mathrm{red}_\Omega$, then $N \preceq \mathrm{red}_\Omega$.

Proof Suppose that $N \uparrow \alpha_\Omega$. $M \prec N \uparrow \alpha_\Omega$ and $v \in \mathrm{Dir}(M)$, so $\alpha_\Omega(v) = F$. If $\bar{\mathcal{O}}(N) \subset \bar{\mathcal{O}}(\alpha)$, then clearly $N \prec \alpha_\Omega$. So assume $\exists u \in \bar{\mathcal{O}}(N)$ with $u \notin \bar{\mathcal{O}}(\alpha)$. Let $u' \in \bar{\mathcal{O}}(N)$ such that $\alpha_\Omega/u' = \Omega$. Suppose $N(u') = H$. Since $\langle M, v\rangle$ is accessible, we thus have a path in the matching dag (notice that $u' \in \bar{\mathcal{O}}(M)$):

$$\begin{aligned}\langle \Omega, \Lambda\rangle &= \langle N_1, w_1\rangle \to \cdots \to \langle N_k, w_k\rangle \\ &= \langle N_k, u'\rangle \xrightarrow{H} \langle N_{k+1}, w_{k+1}\rangle \to \cdots \to \langle N_l, w_l\rangle \\ &= \langle M, v\rangle.\end{aligned}$$

Hence $N_k \prec M \prec N \uparrow \alpha_\Omega$ and $u' \in \mathrm{Dir}(N_k)$. So $\alpha_\Omega/u' \neq \Omega$. A contradiction. ▨

COROLLARY When we remove inaccessible nodes from the matching dag, the terminal nodes are exactly all the success nodes.

This allows us to get rid of inaccessible nodes, which we shall assume from now on.

We use our matching dags to store the redex schemes of Σ. The search for a strongly needed redex is driven by the matching dag, in which are factored local and global matches. We need some extra information to know how to continue the search when the global match has failed but some local ones may be still successful. The idea is similar to the one used in the extension to multiple patterns of the Knuth-Morris-Pratt string-pattern-matching algorithm [13].

DEFINITION 5.9 Let $\langle M, v\rangle$ be a nonsuccess accessible node different form $\langle \Omega, \Lambda\rangle$, and u be the minimum non-null prefix of v such that $M/v\uparrow$. We define $\mathrm{Fail}(\langle M, v\rangle)$ as $\langle M/u, v/u\rangle$, which is an accessible node by lemma 5.7.

We shall indicate in our matching dags $w' = \mathrm{Fail}(w)$ by a dotted arc going from w to w'. Also, we shall drop the first component of the nodes, which is redundant since it is uniquely determined from a path accessing it. This first component is only convenient during the construction and in proofs concerning our dags.

Example Let

$$\Sigma = \{F(F(x, A), B) \to F(D, x), F(C, F(D, x)) \to F(x, A)\}.$$

The matching dag associated with red_Σ is given in figure 11.

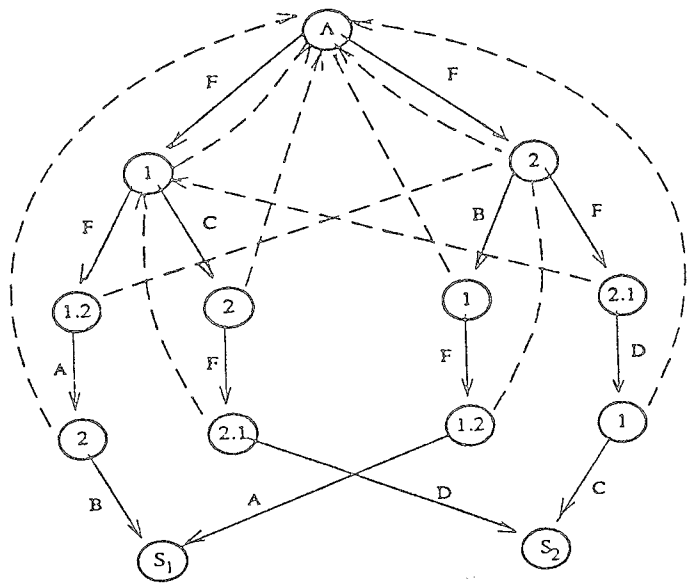


Figure 11

We shall defer for the moment the computation of sets Q verifying our hypothesis. However, the next lemma shows us that they must be chosen among indexes.

LEMMA 5.10 Let M, u and v be such that $\langle M/u, v\rangle$ is an accessible node, and $\forall w\ M/w\uparrow_\perp \Rightarrow w \geq u$. Then $uv \in \mathscr{I}(M)$.

Proof First of all we remark that for any $w \in \mathscr{O}(M)$ such that $w|uv$, we have $\omega(M/w) = M/w$, by lemma 4.10. Now for any w such that $w \preceq uv$, let $M_w = M[uv \leftarrow x]/w$. We show that $\omega(M_w) = M_w$, by induction on $|uv/w|$. For $w = uv$, this is obvious. Otherwise, we get $\bar{\omega}(M_w) = M_w$, from above and the induction hypothesis. Let us show that $M_w\#$.

Case 1: $M/w\#$. Then obviously $M_w\#$.

Case 2: Otherwise, $M/w\uparrow_+$. By hypothesis, $w \geq u$. By lemma 5.7, $w/u \in Q(M[w \leftarrow \Omega]/u)$ and $M/w \prec \mathrm{red}_\Omega$. Using property Q_2, we get $uv/w \in \mathrm{Dir}(M/w)$, and thus $M_w\#$.

In all cases we get $\omega(M_w) = M_w$. When $w = \Lambda$ this shows that

$$\omega(M[uv \leftarrow x]) = M[uv \leftarrow x] \neq \omega(M) \preceq M[u \leftarrow \Omega],$$

and therefore $uv \in \mathscr{I}(M)$. □

COROLLARY If $\langle M, v\rangle$ is accessible, we have $v \in \mathscr{I}(M)$.

Assuming the existence of a matching dag, we shall now give an algorithm that computes, for any term M not in normal form, a strongly needed redex occurrence of M. We assume, without loss of generality, that M has no variables (Otherwise, treat them as constants). We present algorithm A annotated with assertions, $\{\ldots\}$.

ALGORITHM A

begin $\{M \in \mathscr{T}, \mathscr{V}(M) \neq \varnothing\}$
  $u \leftarrow \Lambda$; $P \leftarrow \Omega$; $v \leftarrow \Lambda$; $w \leftarrow \langle \Omega, \Lambda \rangle$;
getsymbol: $\{w = \langle P/u, v \rangle, w$ accessible, $uv \in \mathscr{I}(P), P \prec M,$
    $P[u \leftarrow \Omega] \preceq \omega'(M), \forall u'\ P/u'\uparrow_+ \Rightarrow u' \succeq u\}$.
    Let F be the head symbol of M at uv; $P^- \leftarrow P$; $P \leftarrow P[uv \leftarrow F\vec{\Omega}]$;
search: $\{w = \langle P^-, u, v \rangle, w$ accessible, $uv \in \mathscr{I}(P^-), P \preceq M,$
    $P[u \leftarrow \Omega] \preceq \omega'(M), \forall u'\ P/u'\uparrow_+ \Rightarrow u' \succeq u\}$
  . By cases on the matching dag at node w:
  case 1: $w \xrightarrow{F} w' = \langle \alpha_{i\Omega}, S_i \rangle$: $\{u \in \mathscr{I}(P[u \leftarrow \Omega]), P/u = \alpha_{i\Omega}\}$
    exit with answer "redex of type i at u"
  case 2: $w \xrightarrow{F} w' = \langle P', v' \rangle$: $\{uv' \in \mathscr{I}(P), P/u = P'\}$
    $v \leftarrow v'$: $w \leftarrow w'$; go to getsymbol;
  otherwise $\{P/u \neq\}$
    case 3: $w \neq \langle \Omega, \Lambda \rangle$:
      Let $w' = \text{Fail}(w) = \langle v', P' \rangle$;
      Let u' such that $v = u'v'$;
      $\{u'$ min such that $\Lambda \neq u' \preceq v$ and $P^-/uu'\uparrow$, $P' = P^-/uu'\}$
      $u \leftarrow uu'$; $v \leftarrow v'$; $w \leftarrow w'$; go to *search*;
    case 4: otherwise: $\{w = \langle \Omega, \Lambda \rangle, P^-/u = \Omega, v = \Lambda, \nexists u'\ P/u'\uparrow_+\}$
      choose u such that $P/u = \Omega$;
      if no such u then $\{P = M, \omega'(M) = M\}$
      exit with answer "normal form"
      else go to getsymbol;
end.

THEOREM 5.11: MATCHING DAG THEOREM Let M be a term without variables. If M is in normal form, algorithm A applied to M terminates with answer "normal form." Otherwise, it terminates with answer "redex of type i at u" with $u \in \mathscr{I}(\omega'(M))$ and $M/u \succeq \alpha_{i\Omega}$.

Proof We use Floyd's method to prove the weak correctness of algorithm A, annotated with its assertions. We leave it to the reader to show the invariance of these assertions, using lemmas 4.15, 5.7, 5.8 and 5.10. The termination of the algorithm is easily established, since at getsymbol, $|M| - |P|$ decreases, and at search, $|v|$ decreases, with $|M| - |P|$ constant. When A terminates at exit "normal form," we get $\omega'(M) = M$, i.e., $M \in \mathscr{NF}$. When it terminates at exit "redex of type i at u," we get $M/u \succeq P/u = \alpha_\Omega$. Therefore, $\omega'(M)/u = \Omega$. Furthermore, $P[u \leftarrow \Omega] \preceq \omega'(M)$, and $u \in \mathscr{I}(P[u \leftarrow \Omega])$. Therefore, by definition of index we get $u \in \mathscr{I}(\omega'(M))$. ∎

Now that we have proved the correctness of our algorithm, we give a more readable version of it. We remove the redundant first component of the node in the matching dag and implement P by marking the occurrences explored in M. Also, variable u' now stands for uv of algorithm A, and we use the notation $u' \div v$ to denote u such that $u' = uv$.

In the following algorithm u is the place in M where we are trying to match the root of a redex, u' is the descendant of u where we are pursuing the match, w is pointing to the node of the matching dag which represents the current state of matching.

ALGORITHM B

begin Input: term M, with $\mathcal{V}(M) = \emptyset$.
  $u \leftarrow \Lambda$; $w \leftarrow \Lambda$;
init: $u' \leftarrow u$;
getsymbol: let F be the head symbol of M at u';
    mark occurrence u' in M;
search: By cases on the matching dag at node w:
    case 1: $w \xrightarrow{F} S_i$: exit "redex of type i at u"
    case 2: $w \xrightarrow{F} w' = v'$: $u' \leftarrow uv'$; $w \leftarrow w'$; go to getsymbol;
    otherwise
      case 3: $w \neq \Lambda$: let $w' = \mathrm{Fail}(w) = v'$;
        $u \leftarrow u' \div v'$; $w \leftarrow w'$; go to search;
      case 4: otherwise
        if all occurrences in M marked then exit "normal form"
        else choose u unmarked and go to init;
end.

Algorithm B is nondeterministic for two reasons. The first one comes from the choice of u in case 4. It is natural to implement this choice by taking for u the first unmarked occurrence in M, say in preorder. By keeping a pointer (Free) to the last chosen occurrence, one pass in M will suffice for all the necessary choices. We assume below the existence of the function Searchfree(Free), which returns the first unmarked occurrence of M following Free in preorder and the special value Done if there is none.

The second cause of nondeterminism is that there may be several nodes w' in the matching dag satisfying case 2. This can be easily eliminated by making our matching dags deterministic, keeping only one arc labeled F from any node w. This does not mean that we can restrict $Q(M)$ to be a singleton, since the extremity of a deleted arc may be reacheable through a failure arc. For instance, in figure 11, we may delete one of the two arcs issued from the origin node, but we must keep all the nodes.

When the matching dag is made deterministic, it is natural to store it in a double array Next(w, F), indexed firstly by the nodes of the dag, and secondly by the function symbols in $\mathcal{F}$. This way the computation of which case to choose at search is constant, i.e., independent of the size of red.

For a given Σ algorithm B is linear in the size of M, since an occurrence of M is marked at every passage at getsymbol, and Searchfree makes one pass in M. However, the computations done at search depend on the length of v', i.e., on red, and algorithm B has a running time $O(h \times |M|)$, where h is the maximum height of a redex scheme. We shall now describe one more refinement, in which we get rid of this h factor.

We need pointer u' only to read the occurrence in M at getsymbol. The new value of u' updated at case 2 may be at some distance from the old value, but in any case it

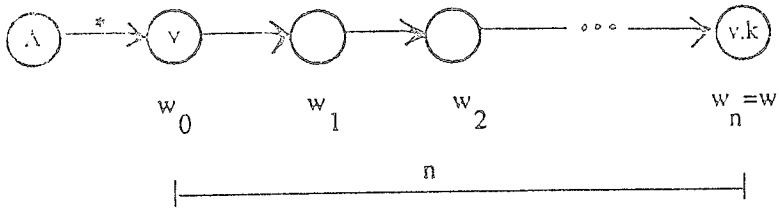


Figure 12

is the son of some node in M that has been already explored. If we keep in a stack all accesses of M at getsymbol, we shall be able to get directly at this father in a way that depends only on the path chosen in the matching dag. We can precompute the corresponding displacement in the matching dag as follows.

Let $v \cdot k$ be the label of node w, and let w_0 be the node with label v on some access path leading to w. See figure 12. We shall keep as information for node w the *virtual address* $\langle -n, k \rangle$, indicating that a query corresponding to node w is effected by accessing the kth son of the node of M accessed at time $-n$. If we keep these accesses in a stack Display with current index Top, we have direct access to this node through Display(Top $- n$).

We must now make sure that our virtual addressing mechanism is correct, independently of the path chosen to access w. The only difficulty comes from the sharing in the dag. More precisely, we may have two paths of different lengths leading from a node labeled with v to node w. Whenever this is the case, we shall duplicate the common nodes of these two paths so as to associate a unique virtual address with every node.

After this unsharing has been effected, we compute the fail arcs. More precisely, let $w = \langle M, c \rangle$ and let u be the minimum nonempty prefix of v such that $M/u\uparrow$. Let w_1, F_1, w_2, F_2, ..., w_k be the sequence whose existence is given by lemma 5.7. Lemma 5.8 tells us that by following this sequence from the origin, we arrive at some node w'. We must precisely take Fail(w) = w', and we shall be sure in this way that our virtual addressing will be correct, even when backtracking through the fail arcs. In particular, note that w and Fail(w) have the same virtual address.

Example Let us consider the system Σ

$G(F(A, x)) \to \cdots$

$F(F(x, H(A)), B) \to \cdots$

$G(F(F(x, H(y)), C)) \to \cdots$

with corresponding matching dag shown in figure 13. The node marked X has two distinct virtual addresses, and there is no way to get rid of the conflicting paths by making the dag deterministic. Figure 14 shows the corresponding *virtual matching dag*, after duplication of node X, computation of virtual addresses, computation of fail arcs and suppression of nondeterminism.

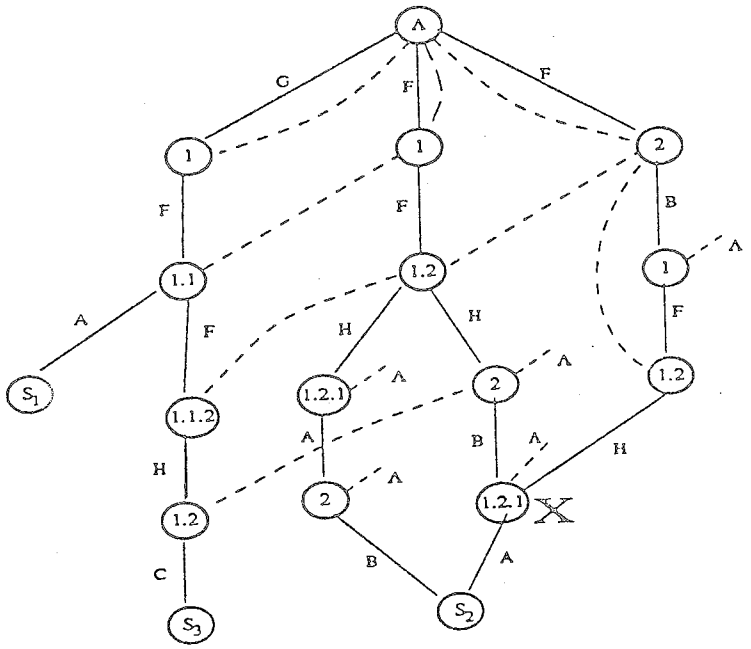


Figure 13

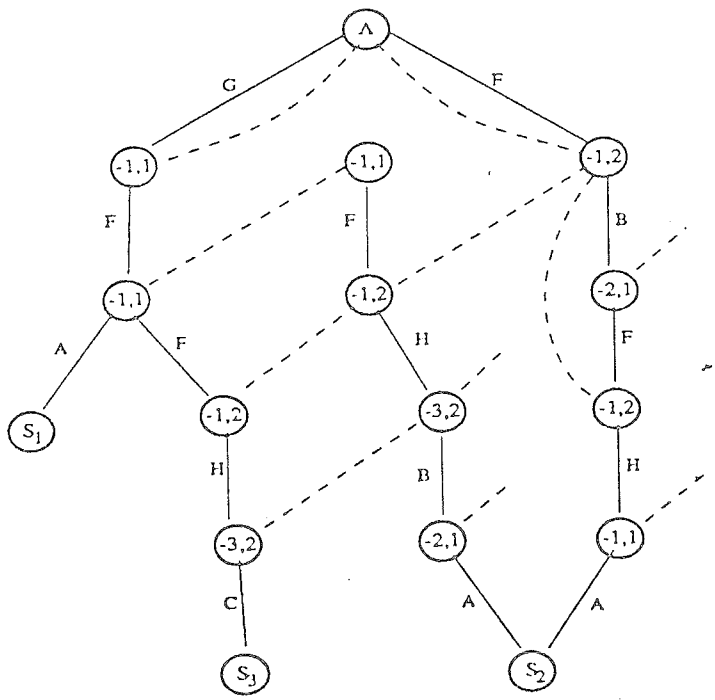


Figure 14

We are now ready to present the next refinement of our algorithm. We assume that the nonsuccess nodes of the dag are coded with successive integers, with the origin coded 0. For w a dag address and F a function symbol, Next(w, F) contains the address of its F-successor, if it exists, or S_i, or 0 to indicate failure. For $w \neq 0$, Node(w) contains a virtual address $\langle -n, k \rangle$. Finally, for every success node S_i, we keep in Size(i) the size of $\mathrm{red}_{i\Omega}$ (i.e., the length of any path leading from the origin in the dag to S_i); this is used as virtual address of the redex in case of success. We use pointer P (represented in algorithm B by occurrence u') to access term M, and $M(P)$ denotes the top function symbol of M at P.

ALGORITHM C

```
begin Free ← origin of M;
init: Top ← 0; w ← 0; P ← Free;
getsymbol: F ← M(P); Mark P;
           Display(Top) ← P; Top ← Top + 1;
search: w' ← Next(w, F); by cases on w':
        S_i: exit "redex of type i at Display(Top − Size(i))"
        0: if w ≠ 0 then w ← Fail(w); go to search;
           else Free ← Searchfree(Free);
              if Free = Done then exit "normal form"
              else go to init;
        otherwise: ⟨−n, k⟩ ← Node(w');
           P ← kth son at Display(Top − n);
           w ← w'; go to getsymbol;
end.
```

Algorithm C is a generalization to trees of the extension of the Knuth-Morris-Pratt fast string-matching algorithm to several patterns. It is straightforward to show that C has a running time of $O(|M|)$, independently of *red*.

Our final refinement consists of getting rid of the inner loop at search by iterating sequences of global failures at compile time. The array Next is now computed as follows. Let us consider the chain of Fail arcs issued from node w in the dag: $w_1 = w$, $w_{i+1} = \mathrm{Fail}(w_i)$ for $1 \leq i \leq p$, $w_p = 0$. Let $F \in \mathcal{F}$. There are two cases:

- For some i, $1 \leq i \leq p$, there is an arc $w_i \xrightarrow{F} w'_i$. We set $\mathrm{Next}(w, F) \leftarrow w'_m$ for m the minimum such i.
- Otherwise, $\mathrm{Next}(w, F) \leftarrow 0$, indicating failure of all local and global searches.

We can now get rid of Fail(w) altogether, and the final version of our algorithm is given below. This corresponds to the technique of [13] for elimination of failure transitions.

ALGORITHM FINDREDEX

```
begin Free ← origin of M;
init: Top ← 0; w ← 0; P ← Free;
getsymbol: F ← M(P); Mark P;
```

```
Display(Top) ← P; Top ← Top + 1;
w' ← Next(w, F);
By cases on w':
S_i: exit "redex of type i at Display(Top − Size(i))"
0: Free ← Searchfree(Free)
   if Free = Done then exit "normal form"
   else go to init;
otherwise:
   ⟨−n, k⟩ ← Node(w');
   P ← kth son at Display(Top − n);
   w ← w'; go to getsymbol;
```

end

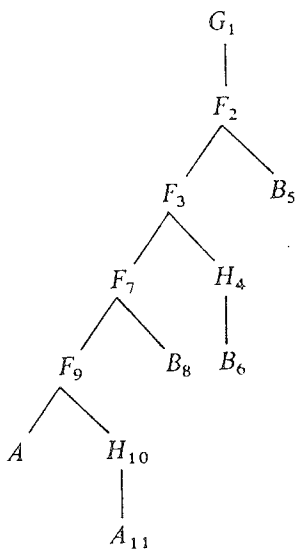
Note that more nodes may now be garbage-collected as inaccessible. The resulting structure Next is no longer a dag but rather a finite automaton transition graph. It is not clear whether this graph should be implemented as lists rather than arrays. This space-time trade-off depends on the structure of the particular Σ.

Example With Σ as above, we construct from figure 14 the information given in table 1. With input term

$M = G(F(F(F(F(A, H(A)), B), H(B)), B)),$

algorithm findredex will explore M in the order shown by the numbers in (1) and will stop with answer "redex of type 2 at x," with x the address of the node labeled 7.

(1)



The only nonconstant space used by findredex (apart from the mark bits in M) is the stack Display. This stack is popped at init, i.e., every time all current searches fail. Actually, we could do much better, since Display is accessed only at a maximum distance of h from its top, where h the maximum size of a redex scheme. We could

Table 1

w	Node(w)	Next					
		F	G	H	A	B	C
0		2	1	0	0	0	0
1	$\langle -1, 1\rangle$	3	1	0	0	0	0
2	$\langle -1, 2\rangle$	2	1	0	0	4	0
3	$\langle -1, 1\rangle$	5	1	0	S_1	0	0
4	$\langle -2, 1\rangle$	6	1	0	0	0	0
5	$\langle -1, 2\rangle$	2	1	7	0	4	0
6	$\langle -1, 2\rangle$	2	1	9	0	4	0
7	$\langle -3, 2\rangle$	2	1	0	0	8	S_3
8	$\langle -2, 1\rangle$	2	1	0	S_2	0	0
9	$\langle -1, 1\rangle$	2	1	0	S_2	0	0

Notes: Size(1) = 3. Size(2) = Size(3) = 5.

therefore implement Display as a circular ring of length h. The instruction Top $\leftarrow$ Top + 1 would become Top $\leftarrow$ (Top + 1) mod h, and our algorithm would work in constant space (plus one bit for each node in M for the marks). However, this might not be desirable in practice, since if we want to implement a lazy interpreter using algorithm findredex, we should be able to dynamically restart it after effecting one step of reduction, and in that case we would need to keep the whole Display.

Note that in order to effect a reduction step, we must have unambiguous addresses for the variable occurrences in the redex scheme. For instance, in the example above we would distinguish between the two possible successes s_2, since variable x has virtual address $-3, 1$ in one case and $-2, 1$ in the other. This unsharing problem could be simplified by completely developing our dags as trees. Actually, a slightly more general construction would be to directly construct matching trees with nodes $w = \langle M, v\rangle$ such that $v \in \mathcal{Q}(w)$, i.e., have Q depend on the access path to w and not only on M.

We summarize this section with the following theorem:

THEOREM 5.12: FAST TREE PATTERN MATCHING THEOREM Let Σ be such that there exists a matching dag for red_Σ. For any term M not in normal form, the algorithm findredex will find a strongly needed redex occurrence of M in time $O(M)$.

5.3 Deciding Strong Sequentiality

We shall now show that strong sequentiality is equivalent to the conditions Q_1 and Q_2 of section 5.2.

DEFINITION 5.13 Let M be an Ω-term. We define its set $\mathcal{T}(M)$ of *increasing indexes* as

$$\mathcal{T}(M) = \{u \in \mathcal{I}(M) | \forall N\ \omega(N) = \Omega\ \&\ \omega'(N) = N \Rightarrow \exists v \geq u\ v \in \mathcal{I}(M[u \leftarrow N])\}.$$

Intuitively, an increasing index can be increased into an index whenever we replace it by a term which may become a redex after some reductions.

Example With
$\Sigma = \{F(G(A,x),B) \to \cdots, F(G(x,A),C) \to \cdots, G(E,E) \to \cdots\}$,
the first occurrence of Ω in $F(\Omega,\Omega)$ is a nonincreasing index.

LEMMA 5.14 Let Σ be strongly sequential. Then for any M in Ω-normal form we have $\mathscr{T}(M) \neq \emptyset$.

Proof Assume Σ strongly sequential and $M = \omega'(M)$. Let $\mathscr{I}(M) = \{u_1, u_2, \ldots, u_n\}$. Assume that $\mathscr{T}(M) = \emptyset$, i.e., for every $i \leq n$ there exists N_i such that $\omega(N_i) = \Omega$, $\omega'(N_i) = N_i$, and $\nexists v \geq u_i$, $v \in \mathscr{I}(M[u_i \leftarrow N_i])$. Now consider

$P = M[u_i \leftarrow N_i | 1 \leq i \leq n]$

We first show $\omega'(P) = P$. Since M and the N_is are in Ω-normal form, the only possible case of a redex in P is when there is some α in red and u in $\bar{\mathscr{O}}(\alpha)$ such that $\alpha_\Omega/u \preceq N_k$ for some $k \leq n$. But then $\omega(\alpha_\Omega/u) \preceq \omega(N_k) = \Omega$, i.e., $\omega(\alpha_\Omega/u) = \Omega$, which is impossible by lemma 4.10. Thus $\omega'(P) = P$.

Σ being strongly sequential, there exists v in $\mathscr{I}(P)$. There are two cases:

Case 1: $\exists k \leq n$, $u_k \preceq v$. Then by repeated application of lemma 4.17 we get $v \in \mathscr{I}(M[u_k \leftarrow N_k])$, contrary to the hypothesis on N_k.

Case 2: $\forall k \leq n$, $u_k | v$. Then by repeated application of lemma 4.17 we get $v \in \mathscr{I}(M)$, which is impossible.

This completes the proof that $\mathscr{T}(M) \neq \emptyset$. ▨

Let us now extend lemma 4.16 to $\mathscr{T}$:

LEMMA 5.15 $uv \in \mathscr{T}(M) \Rightarrow v \in \mathscr{T}(M/u)$.

Proof Assume that $uv \in \mathscr{T}(M)$ and $v \notin \mathscr{T}(M/u)$. That is, there exists N such that $\omega'(N) = N$, $\omega(N) = \Omega$, and $\nexists v' \geq v$, $v' \in \mathscr{I}(M/u[v \leftarrow N])$. Now let $P = M[uv \leftarrow N]$. Since $uv \in \mathscr{T}(M)$, there exists $w \geq uv$ such that $w \in \mathscr{I}(P)$. By lemma 4.16, $w/u \in \mathscr{I}(M/u[v \leftarrow N])$, a contradiction. ▨

COROLLARY If Σ is strongly sequential, $\mathscr{T}(M)$ satisfies Q_2.

Proof $M \prec \mathrm{red}_\Omega$ implies $\omega'(M) = M$ and $\mathscr{T}(M) \subseteq \mathscr{I}(M) \subseteq \mathrm{Dir}(M)$. ▨

An increasing index may, by definition, be increased into an index. Actually, it may even be increased into an increasing index:

LEMMA 5.16 Let N be such that $\omega(N) = \Omega$ and $\omega'(N) = N$. For every $u \in \mathscr{T}(M)$, there exists $v \geq u$ such that $v \in \mathscr{T}(M[u \leftarrow N])$.

Proof Assume $u \in \mathscr{T}(M)$, $\omega(N) = \Omega$, $\omega'(N) = N$, and $\nexists v \geq u$ $v \in \mathscr{T}(M[u \leftarrow N])$. Let $\{u_1, \ldots, u_n\} = \{u_i \in \mathscr{I}(M[u \leftarrow N]) | u_i \geq u\}$. For every $i \leq n$, $u_i \notin \mathscr{T}(M[u \leftarrow N])$, by hypothesis; that is, there exists N_i such that $\omega(N_i) = \Omega$, $\omega'(N_i) = N_i$ and $\nexists w \geq u_i$, $w \in \mathscr{I}(M[u \leftarrow N[v_i \leftarrow N_i]])$, with $v_i = u_i/u$. Consider now $P = N[v_i \leftarrow N_i | 1 \leq i \leq n]$.

We get $\omega(P) = \Omega$ and $\omega'(P) = P$, like in the proof of lemma 5.14, and from $u \in \mathcal{T}(M)$ we know that there exists $w \geq u$ such that $w \in \mathcal{I}(M[u \leftarrow P])$. From above and repeated application of lemma 4.17 we must have $w|u_i$ for all $i \leq n$. Using lemma 4.17 again we get $w \in \mathcal{I}(M[u \leftarrow N])$, which is impossible by definition of the u_is. ▨

COROLLARY If Σ is strongly sequential, $\mathcal{T}(M)$ satisfies Q_1.

THEOREM 5.17: DECIDABILITY OF STRONG SEQUENTIALITY Σ is strongly sequential iff there exists a matching dag for red_Σ.

Proof If Σ is strongly sequential, we have just seen that $\mathcal{T}(M)$ satisfies Q_1 and Q_2. For the converse, consider a matching dag as constructed in 5.2. Let M be any Ω-term such that $\omega'(M) = M$ and $\omega(M_x) \neq \omega(M)$. We shall show that there exists an index in M by processing M with algorithm A slightly modified as follows. There are two more cases at getsymbol for the symbol read. If it is a variable, do as in "otherwise" (i.e., failure of global match). If it is an Ω, stop. We know from the assertions given for algorithm A, which are all still valid except that $P \prec M$ at getsymbol should become $P \preceq M$, that in this last case we have $uv \in \mathcal{I}(P)$. Since $M/uv = \Omega$, it follows that $uv \in \mathcal{I}(M)$, by definition of index. Note that this is the only way we may stop, since an exit at "normal form" is impossible by condition $\omega(M_x) \neq \omega(M)$, which implies $\exists w \in M$, $M/w = \Omega$, and an exit at "redex at u" is impossible by condition $\omega'(M) = M$. Therefore, $\mathcal{I}(M) \neq \varnothing$, which concludes the proof that Σ is strongly sequential. ▨

Note that the statement of the theorem gives effectively a decision procedure: to test whether Σ is strongly sequential, try constructing a matching dag as defined in 5.2. We may restrict conditions Q_1 and Q_2 to accessible nodes, and in that case, $\mathcal{T}$ is actually the maximum solution for Q. This is the solution we obtain if we start with $Q(M) = \text{Dir}(M)$, suppressing progressively elements with Q_2 and checking Q_1. In practice we might want to generate a smaller solution, starting with $Q(M) = \varnothing$ and adding an element with Q_1, checking Q_2. In either case, several passes may be needed, and we do not know of an efficient algorithm (i.e., linear in the size of red) to build a matching dag for a strongly sequential Σ. Once a dag is found verifying Q_1 and Q_2, we effect the necessary unsharing and compute the fail arcs. (This process could be simplified by completely developing our dags as trees). We then make the dag deterministic, complete the graph by iterating failures, garbage-collect inaccessible nodes, and finally get an automaton table such as the one given in table 1. Some ingenuity may be needed in this process if we want to minimize the size of the automaton. Of course, this is done once and for all for a given Σ, and therefore we do not care very much about the cost of these operations. In practical terms, this is the cost of building a compiler for the programming language defined by Σ. However, as pointed out by one of the referees, the size of the tables can blow up exponentially. Fortunately, this problem disappears in the practically relevant cases of the next two sections.

6 Applications

6.1 Recursive Functions with Constructors

We now consider a special case of our rewriting systems, corresponding to HOPE programs [5]. We assume that the set $\mathscr{F}$ of function symbols is partitioned into two set $\mathscr{F}_R$ and $\mathscr{F}_C$. $\mathscr{F}_R$ is the set of recursive functions symbols and $\mathscr{F}_C$ the set of constructors. Then any redex scheme $\alpha \in \text{red}$ is of the form $F(\alpha_1, \alpha_2, \ldots, \alpha_n)$, where $F \in \mathscr{F}_R$ and, for every i, the term $\alpha_i \in \mathscr{M}(\mathscr{F}_C, \mathscr{V})$ contains only constructor function symbols and variables. These particular systems will be called *systems with constructors*.

LEMMA 6.1 A system Σ with constructors is strongly sequential iff the set red_Ω is sequential.

Proof It is easy to check that for every $M \prec \text{red}_\Omega$ the set $\text{Dir}(M)$ of directions of M satisfies Q_1 and Q_2. ▨

6.2 Simple Systems

In the general case of strongly sequential systems, it is not easy to find a corresponding matching dag. We consider now a restricted case (which often meets in TRS) when it is possible to factor the matching for all subexpressions of redexes. We call these systems *simple systems*.

DEFINITION 6.2 Let $\text{red}^*_\Omega = \{\alpha_\Omega/u | \alpha \in \text{red}, u \in \bar{\mathscr{C}}(\alpha)\}$. Then Σ is a *sequential simple system* iff all subsets of red^*_Ω are sequential sets.

The set red is a simple forest in the sense of Hoffman and O'Donnell [7], since if α and β are two compatible Ω-terms of red^*_Ω, they must be comparable in order to make the subset $\{\alpha, \beta\}$ sequential. But the algorithm defined in [7, appendix C] does not work if non-sequential simple systems are considered. In the terminology of [7], the subsumption graph (i.e., the covering relation of $\preceq$ in red^*_Ω) is a tree where at each node there is an occurrence of Ω in the corresponding Ω-term where the different branches are differenciated.

Let us define $\text{Dir}^*(M) = \text{Dir}(M, \text{red}^*_\Omega)$, $\mathscr{I}^*(\Omega) = \{\Lambda\}$, and

$$\mathscr{I}^*(M) = \{uv | u \in \text{Dir}^*(\bar{\omega}(M)), v \in \mathscr{I}^*(M/u)\}$$

when $M \neq \Omega$.

LEMMA 6.3 Let $\omega(M) = \Omega$. Then $\mathscr{I}^*(M) \subseteq \mathscr{I}(M)$.

Proof Let $u \in \mathscr{I}^*(M)$, and let x be any variable. We prove first by induction that $\omega(M[u \leftarrow x]) \# \text{red}^*_\Omega$. If $M = \Omega$, then $u = \Lambda$ and $\omega(M[u \leftarrow x]) = x \# \text{red}^*_\Omega$, since $\Omega \notin \text{red}^*_\Omega$. Now suppose $M \neq \Omega$ and $\omega(M) = \Omega$. Then $u = vw$ with $v \in \text{Dir}^*(\bar{\omega}(M))$ and $w \in \mathscr{I}^*(M/v)$. Let $M_v = \omega((M/v)[w \leftarrow x])$. By induction hypothesis $M_v \# \text{red}^*_\Omega$. Thus $M_v \# \text{red}_\Omega$ and $\bar{\omega}(M[u \leftarrow x]) = \bar{\omega}(M)[v \leftarrow M_v]$. Now suppose that there is $\alpha \in \text{red}^*_\Omega$ such that $\alpha \uparrow \bar{\omega}(M[u \leftarrow x])$. Since $\bar{\omega}(M) \preceq \bar{\omega}(M[u \leftarrow x])$, we get $\alpha \uparrow \bar{\omega}(M)$. But

$v \in \mathrm{Dir}^*(\bar{\omega}(M))$. Thus $v \in \mathcal{C}(\alpha)$ and $\alpha/v \neq \Omega$. Therefore $\alpha/v \in \mathrm{red}_\Omega^*$ and $\alpha/v \uparrow M_v$. A contradiction. Hence $\omega(M[u \leftarrow x]) \neq \mathrm{red}_\Omega^*$. In particular, $\bar{\omega}(M[u \leftarrow x]) \neq \mathrm{red}_\Omega$, which implies $\omega(M[u \leftarrow x]) = \omega(M[u \leftarrow x])$ and $\omega(M[u \leftarrow x]) \neq \mathrm{red}_\Omega^*$. Now since $\omega(M) = \Omega$ and $\omega(M[u \leftarrow x]) \neq \Omega$, the occurrence u is in $\mathscr{I}(M)$, by lemma 4.14. ▨

LEMMA 6.4 Any sequential simple system is strongly sequential.

Proof Assume that Σ is a sequential simple system. We show first by induction that $\mathscr{I}^*(M) \neq \varnothing$ for every M such that $\omega(M) = \Omega$ and $\omega'(M) = M$. If $M = \Omega$, then $\mathscr{I}^*(M) = \{\Lambda\} \neq \varnothing$. Suppose now that $M \neq \Omega$, $\omega(M) = \Omega$, and $\omega'(M) = M$. Then $\bar{\omega}(M)\uparrow_+$. Thus because of the nonambiguity condition and since $\omega'(M) = M$, one cannot have $\alpha \preceq \bar{\omega}(M)$ for some $\alpha \in \mathrm{red}_\Omega^*$. Therefore $\mathrm{Dir}^*(\omega(M)) \neq \varnothing$ and by induction $\mathscr{I}^*(M) \neq \varnothing$. Now if $M \prec \mathrm{red}_\Omega$, then $\omega(M) = \Omega$ and $\omega'(M) = M$ (again by nonambiguity). Thus $\mathscr{I}^*(M) \neq \varnothing$, and $\mathscr{I}^*(M)$ obviously satisfies Q_1 and Q_2. ▨

The matching dag of a sequential simple system can be built up easily from $\mathscr{I}^*(M)$ by induction on the size of M as follows. Suppose $\langle N, v\rangle$ is a node of the matching dag, $M = N[v \leftarrow F\dot{\Omega}]$ and $M \prec \mathrm{red}_\Omega$. Then search, along the chain of failure arcs starting at $\langle N, v\rangle$, the first node $\langle N', v'\rangle$ with an outgoing arc labelled by F. If there is no such node, then choose any $u \in \mathrm{Dir}^*(M)$, create node $\langle M, u\rangle$ with an F-arc from $\langle N, v\rangle$ to $\langle M, u\rangle$ and a failure arc from $\langle M, u\rangle$ to $\langle \Omega, \Lambda\rangle$. Otherwise, let $v = wv'$ and $\langle M', u'\rangle$ be the extremity of the F-arc outgoing from $\langle N', v'\rangle$. Then, taking $u = wu'$, create node $\langle M, u\rangle$ with an F-arc from $\langle N, v\rangle$ to $\langle M, u\rangle$ and a failure arc from $\langle M, u\rangle$ to $\langle M', u'\rangle$. Thus the matching dag (which is then a tree) can be efficiently built up in one top-down pass starting from the origin (not bottom-up as in [7]).

Example Let Σ be such that

$$\mathrm{red} = \{F(A, G(x, y, C)), F(B, G(D, x, C)), G(E, x, D)\}.$$

Then the two possible matching dags associated to the sequential simple system Σ are given in figure 15.

Finally, O'Donnell [19] gave a sufficient condition for insuring the termination of the leftmost outermost derivation sequence. This condition is as follows. Remember that when $u, v \in \mathcal{C}(M)$, then u is said to be before v in preorder, written $u \leq_M v$, iff either $u \preceq v$ or $u|v$ with u to the left of v. Let Σ be a *left system* iff

$$\forall \alpha \in \mathrm{red}_\Sigma,\ \forall u, v \in \mathcal{C}(\alpha),\ \alpha/u \in \mathscr{V} \ \&\ u \leq_\alpha v \Rightarrow \alpha/v \in \mathscr{V}.$$

Thus left systems are characterised by redex schemes in which after a variable there could be only variables in preorder. One can easily prove that left systems are particular sequential simple systems and that they correspond exactly to systems where the leftmost outermost redex of any term is strongly needed. One nice example of left systems, given in [19], is the case of combinatory logic i.e., when

$$\Sigma = \{A(I, x) \to x,\ A(A(K, x), y) \to x,\ A(A(A(S, x), y), z) \to A(A(x, z), A(y, z))\}.$$

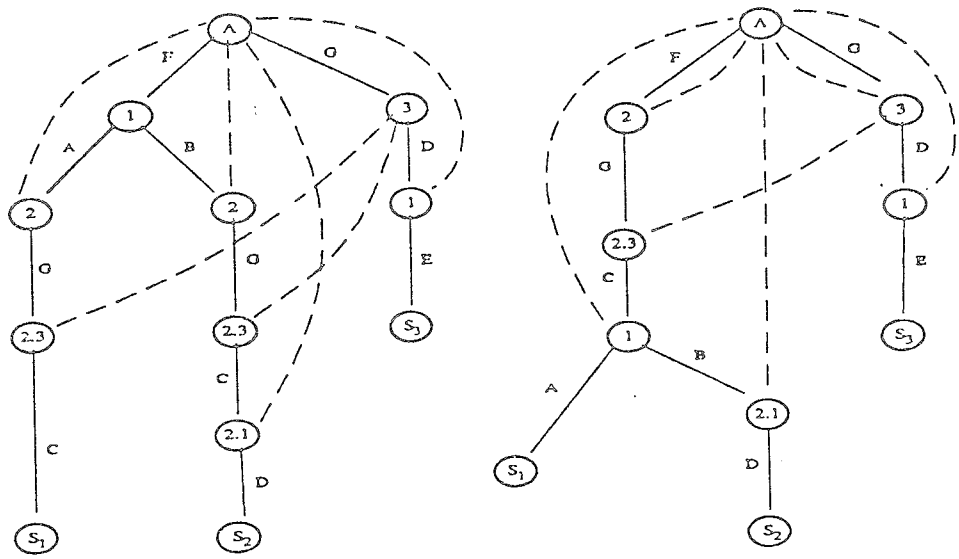


Figure 15

6.3 An Example

The left-linear non-overlapping term rewriting systems define a rather general family of applicative programming languages. The restriction to strongly sequential programs permits to define efficient interpreters for these languages. Important subcases are recursive equations with if-then-else expressions [16, 3], HOPE [5], and combinatory logic, as shown above. Several examples of such "programming with equations" are given in [8], including the definitions for a LISP and a LUCID interpreters.

We shall illustrate our methods on a small example of call-by-need computations in a highly functional programming language. This example defines the set of prime numbers, Primes, implemented as an infinite stream and computed along the lines suggested by [10]. First, we generalize the above coding of combinatory terms to arbitrary curried operators, using the following abbreviation:

$fM_1 M_2 \ldots M_N$ stands for $\mathrm{App}(\ldots \mathrm{App}(\mathrm{App}(f, M_1), M_2), \ldots M_n)$.

Second, we assume given a minimum set of predefined arithmetic operations, that is a constant $\underline{n}$ for every integer n, and the standard operations $+$ and $\times$.

6.3.1 General combinators We use an infix "$\cdot$" for list construction, and an infix "$\circ$" for function composition.

$\mathrm{hd}(x \cdot y) = x$

$\mathrm{tl}(x \cdot y) = y$

$(f \circ g)x = f(g(x))$

$\mathrm{map}\ f\ (x \cdot y) = (f\ x) \cdot (\mathrm{map}\ f\ y)$

Note that *all* our function symbols are (zero-ary) constants, with the sole exception of the binary App implicit from our notation.

6.3.2 Arithmetic operators

$N_2 = \underline{2} \cdot \text{map}\,(+\underline{1})\,N_2$

$\text{mult}\,x = \text{map}\,(\times x)\,N_2$

$\text{filter}(\underline{n} \cdot x)(\underline{m} \cdot y) = \underline{n} \cdot \text{filter}\,x\,(\underline{m} \cdot y) \quad \text{if } n \neq m$

$\text{filter}(\underline{n} \cdot x)(\underline{n} \cdot y) = \text{filter}\,x\,y$

These equations should present no problem: N_2 is the stream of integers starting from 2, mult x is the stream of all multiples of x, and filter x y removes elements of y from x, assuming x is a sorted stream, and y a sub-stream of x. The following function removes from a stream all multiples of its first element:

$\text{remult}(x \cdot y) = \text{filter}\,y\,(\text{mult}\,x)$

6.3.3 Computing primes It is now easy to compute the various stages of Erathostenes' sieve in a stream of streams:

Sieve = N_2.((map ∘ map) remult Sieve).

At every stage, one new prime number is produced, i.e.

Primes = map hd Sieve.

We leave it to the reader to check that it is possible to use the program rules to evaluate, say, hd(tl Primes) to $\underline{3}$. In doing so, he should convince himself that the correct computation strategy is not quite obvious, since non-terminating computations lurk everywhere.

Actually, it is not hard to show that this example is strongly sequential (it falls under the conditions of simple systems). A part of the matching dag, showing all non-trivial fail ares, can be seen in figure 16.

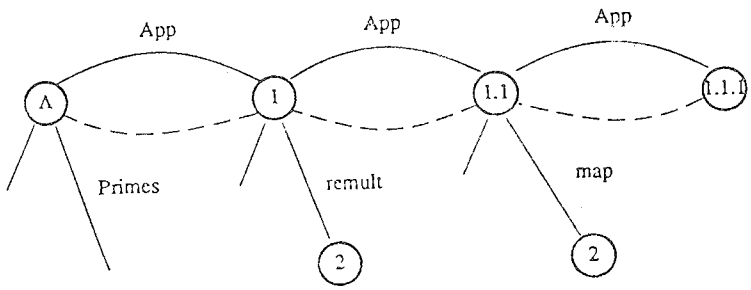


Figure 16

7 Conclusion

In our paper we established some mathematical foundations for computing with the call-by-need technique in TRS. As usual, optimal evaluation strategies are expected by adding some sharing mechanism. Duplications of subterms can be avoided by implementing terms in dags. However, the dags formalism need to be neatly defined. Another issue is to generalize propositions of our paper to more complicated rewriting systems. For instance, does one really need the nonambiguity condition? Can we treat systems with bound variables, such as the lambda calculus? Or with imperative features? Moreover, for the semantics of our systems, is it possible to get some full abstraction result? For which class of functions? Finally, it seems possible to consider the general problem of tree pattern matching along the lines of this paper and thus to define sets of patterns where a linear top-down matching algorithm could exist. However, this problem differs from finding strongly needed redexes, since then one has not to consider replacement of matching subterms.

Addendum

The results of this paper were obtained in 1979. Since then, extensions appeared in [4] for a non-deterministic TRS and in [17] for parallelism. A sophisticated implementation has been done by Laville [14] for ML. Efficient compilation of a TRS are also in Strandh [22]. Corrections and improvements in some of the proofs are reported in [12]. We especially thank Aart Middeldorp for giving us a correct proof of lemma 5.8.

References

[1] G. Berry. Séquentialité de l'évaluation formelle des lambda-expressions. In *Proc. 3rd International Colloquium on Programming*. Paris, March 1978, Dunod.

[2] G. Berry. Stable models of typed lambda-calculi. In *Proc. 5th ICALP Conf.* Udine, Italy, 1978.

[3] G. Berry, J.-J. Lévy. Minimal and optimal computations of recursive programs. In *Proc. 3rd POPL Conf.* Santa Monica, Jan. 1977. Also *JACM* 26, no. 1, 1979.

[4] G. Boudol. Computational semantics of terms rewriting systems. Rapport de recherche INRIA, Feb. 1983.

[5] R. M. Burstall, D. B. Macqueen, and D. T. Sannella. HOPE: An experimental applicative language. Report CSR-62-80, Computer Science Dept., University of Edinburgh, Feb. 1981.

[6] P. L. Curien. Algorithmes séquentiels sur structures de données concrètes. Third cycle thesis, Université de Paris March 1979.

[7] C. Hoffmann, M. O'Donnell. Interpreter generation using tree pattern matching. In *Proceedings 6th POPL*. San Antonio, Jan. 1979.

[8] C. Hoffmann, M. O'Donnell. Programming with equations. *ACM Transactions on Programming Languages and Systems* 4, no. 1:83–112, Jan. 1982.

[9] G. Huet, J.-J. Lévy. Computations in orthogonal rewriting systems, I. Chapter 11, this volume.

[10] G. Kahn and D. Macqueen. Coroutines and networks of parallel processes. In *Proc. IFIP Congress 77*. North-Holland, pp. 993–998.

[11] G. Kahn, G. Plotkin. Domaines concrets. IRIA-Laboria report no. 336, Dec. 1978.

[12] J. W. Klop, A. Middeldorp. Strongly sequential term rewriting systems. Rep. IR-128, Free Univ. Amsterdam, Jun 1987.

[13] D. E. Knuth, J. Morris, and V. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing* 6, no. 2:323–350, 1977.

[14] A. Laville. Evaluation paresseuse des filtrages avec priorité: Application au langage ML. Thesis, Univ. of Paris 7, Feb 1988.

[15] J.-J. Lévy. Réductions correctes et optimales dans le lambda-calcul. Thesis, Paris 7, Jan. 1978.

[16] J. McCarthy. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*, ed. P. Braffort and D. Hirschbert. North-Holland, 1963, pp. 33–70.

[17] F. Muller. Entwurf und Implementierung eines Interpreter-Generators unter Verwendung von Baumtransformatoren und schnellen Verfahren zur Einbettung von Baumen. Diplomarbeit, Abteilung Informatik, Universitat Dortmund, June 1980.

[18] M. Nivat. On the interpretation of recursive polyadic program schemes. In *Symposia Mathematica*, vol. 15. Istituto Nazionale di Alta Mathematica, Italy, 1975, pp. 225–281.

[19] M. O'Donnell. *Computing in Systems Described by Equations.* LNCS no. 58. Springer-Verlag, 1977.

[20] J. C. Raoult, J. Vuillemin. Operational and semantic equivalence between recursive programs. 10th SIGACT Conf., San Diego, 1978.

[21] J. Staples. Computation on graph-like expressions. Report no. 2/77, Math & Comp. Science, Queensland Institute of Technology, Brisbane, Australia, 1977.

[22] R. I. Strandh. Compiling equational programs into efficient machine code. Ph.D. thesis, Johns Hopkins Univ., 1988.

[23] C. P. Wadsworth. Semantics and pragmatics of the λ-calculus. Ph.D. thesis, Oxford, 1971.