

Computational semantics of terms rewriting systems

Gérard Boudol

▶ **To cite this version:**

Gérard Boudol. Computational semantics of terms rewriting systems. [Research Report] RR-0192, INRIA. 1983, pp.98. inria-00076366

HAL Id: inria-00076366
https://inria.hal.science/inria-00076366

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. 954 90 20

Rapports de Recherche

N° 192

COMPUTATIONAL SEMANTICS OF TERMS REWRITING SYSTEMS

Gérard BOUDOL

Février 1983

COMPUTATIONAL SEMANTICS OF TERMS REWRITING SYSTEMS

G.BOUDOL

I.N.R.I.A.
Sophia Antipolis
06560 VALBONNE
France

ABSTRACT

We study the structure of computations by term rewrite rules, allowing non-determinism and overlapping. Computations form, up to permutations of rewritings, a complete partial order, and semantics is defined as the set of results of terminating (i.e. maximal) computations. A call-by-need computation rule is then introduced and we prove, by means of a continuous projection over the complete partial order of call-by-need computations,that it is correct for sequential systems.

Résumé: Nous étudions la structure des calculs déterminés par systèmes de réécriture de Termes, autorisant le non-déterminisme et le chevauchement. L'ensemble de ces calculs forme, aux permutations de réécritures près, un ordre partiel complet, et la sémantique est définie comme l'ensemble des résultats des calculs terminés (ie maximaux). Une règle de calcul en appel par nécessité est introduite et nous pouvons, grâce à une projection continue sur l'ordre partiel des calculs en appel par nécessité, qu'elle est correcte pour les systèmes séquentiels.

PAPIER RECUPERÉ ET RECYCLÉ

Summary

1. Introduction

2. Domains, Algebras and Trees

2.1 Domains

2.2 Algebras

2.3 Trees and Term Rewriting Systems

3. Computations

3.1 Consistent sets of redexes

3.2 Computations

3.3 Equivalence : the permutation and simplification lemmas

3.4 The ordered space of computations

4. Semantics

4.1 Equational classes of algebras : the factorization theorem

4.2 Computational semantics

5. Implementation

5.1 Call-by-need computations

5.2 Sequential systems : the correctness theorem

6. Conclusion

To appear in "Proceedings of the US - French Joint Seminar on the Applications of Algebra to Language Definition and Compilation", Fontainebleau, June 9-15, 1982.

1. Introduction

May be the best way to introduce this work is to take a short walk on the historical side. It seems fair to say that the concept of computability is relevant to computer science. But here our concern is not in automata theory. Thus the starting point is recursion theory, as it can be found in the book of S.C. Kleene [26], or in the work of J. Mac Carthy [28]. With regard to computability, D. Scott [40,41] proposed a general semantical model, which recaptures the well-known fix-point theory of recursive definitions. Denotational semantics has gained a considerable popularity, owing to its mathematical elegance and fruitfulness (see for example the paper of R. Milner [29]), but also because it has an operational foundation : from a less abstract point of view on computability, denotational semantics is also adequate. For it has been shown by J.M. Cadiou [10], J. Vuillemin [44], P.J. Downey and R. Sethi [16] that there exist "correct" computation rules for recursive definitions. Efficiency is a less tractable problem,and indeed its precise formulation requires a careful analysis of the concept of computation, as in the work of G. Berry and J.J. Levy [4].

In the operational setting, computations are symbolic manipulations of syntactic objects, such as rewriting on trees studied by B.K. Rosen [39] and G. Huet [21] (thus computations have something to do with proofs in first-order equational theories -e.g. : abstract data types - see [22]). The confluence of denotational and operational approaches gives the so-called algebraic semantics of M. Nivat [31] and ADJ Group [1], which relies on the initiality of the symbolic semantics (see [32, 45, 17]). This semantics is "operationally adequate" in yet another acceptation : as shown by J.C. Raoult and J. Vuillemin [38], initial models are fully abstract.

Problems arise when we want to model non-determinism. Here it is not a matter of implicit non-determinism (resulting from hiding an argument to a function) but of explicit one, as introduced by adding a new operator of choice or to the recursive definitions (see for example [2,33]: we shall not quote all the numerous papers on this subject). Semantically this seems to be nothing more than set-theoretic union,

whereas operationally the behavior of this operator is given by

or (x,y) -> x

or (x,y) -> y

To extend denotational semantics then requires constructions of power-domains, such as the ones given by G. Plotkin [35] and M. Smyth [43], generalized by M.C.B. Hennessy and G. Plotkin [18] in a category-theoretic framework. But is there any operational counterpart to this denotational semantics? If we want to keep track of programs expressly designed to run forever, we have to carefully define a notion of acceptable computations; this has been done in [7] (and, with a distinct notion of "acceptable" , in [2,37]) where we show (see also [8]) that the operational semantics which we get for non-deterministic recursive definitions is intrinsically non-continuous. However, this "computational semantics" is consistent with the algebraic approach, and we retrieve an implicit non-determinism, since the semantics of a non-deterministic recursive program is actually a function from oracles to values.

Let us now more accurately investigate this computational approach. As we have said, in this setting computations are sequences of rewritings on terms, following a given finite set of rules (i.e. pairs of terms, subject to some restrictions about the occurrences of variables, see below). Usually a semantics is then defined by means of finitely terminating computations (those which end on a normal form), but when the term rewriting system is confluent (or has the Church-Rosser property, see [39,21]), and if we interpret terms in a complete partial order, one may have a more general definition of the denotation of a term (as the lub of the directed set of partial informations got along computations).

Confluence property is fulfilled when the term rewriting system is deterministic ("unequivocal" for B.K.Rosen [39], "non-ambiguous" in the terminology of G. Huet and J.J. Levy [23]), that is a partial function from left-hand sides to right-hand sides of the rules, with no overlapping between left-hand sides. This kind of term rewriting system, which

covers primitive recursive and recursive definitions, is a widely studied one ([39,34,5,20,23,11]). From the work of G.Huet and J.J. Levy [23], pursuing those of J. Vuillemin [44] and G. Berry and J.J. Levy [4], one may say much more about computations in deterministic term rewriting systems than confluence : one may define a very natural equivalence on computations (to be the same up to a permutation of the rewriting steps) and a preorder by means of "what remains to do" (the residual) of a computation after another one, up to permutations. With each term is thus associated a lattice of (classes of) computations. In this setting correctness and efficiency of computation rules can be precisely stated ([4,23]).

The key idea that a computational device determines an ordered space of "events" is not new : it underlies the work of G. Kahn and G. Plotkin [24] and of G. Berry and P.L. Curien [6] on concrete domains, and the concept of event structure of G. Winskel [46]. This ordered space is a set of classes of computations, up to permutations, not only in the context of deterministic term rewriting systems, but also for the lamda-calculus (see the nice paper of J.J. Levy [27]), for some kind of Petri nets ([30]) and in some sense for algebras of processes ([47]). This work is just an attempt to generalize (in two directions, relaxing the hypothesis of determinism and finite termination semantics) some of the results of G. Huet and J.J. Levy [23] (and of [4,7]), applying this computational approach. We give a partial answer to a question of [23] : "does one really need the non-ambiguity condition?", as it can be seen along the lines of the following presentation:

- In a second section, we briefly recall the necessary theoretical framework, and introduce term rewriting systems as finite sets of pair of terms (the rules) subject to the following restrictions :

(i) a variable cannot appears as the left-hand side of a rule (this is to prevent some semantical aberrations, see below)

(ii) a variable which occurs in the right-hand side of a rule must occurs in its left-hand side (by this way, the application of a rewriting rule at some of its occurrences gives a determinate result)

(iii) left-hand sides of the rules are linear, that is a variable may not occurs more than once in these terms (this seems to be only a technical restriction)

Thus we allow TRS's in which two distinct rules may have the same left-hand side, and more generally TRS's in which left-hand sides may overlap.

- Section 3 contains the definition of the equivalence of computations up to permutations. A slight difference with the deterministic case is that we have here to take into account the possible incompatibility of elementary computation steps. This appears in the permutation lemma (elsewhere called parallel moves lemma). Another crucial (and not easy to establish) property of this equivalence is its left simplifiability w.r.t. composition of computations (the simplification lemma). Then we order computation by "extension up to permutations" and, generalizing the deterministic case, prove that the ordered space of computations is a complete partial order. This allows to generalize also the concept of termination, by saying that a computation is terminating if it is maximal.

- In section 4 we first recall the main theorem of algebraic semantics, stated in a somewhat unusual form (but which is nothing more than the Birkhoff's completeness theorem, see[22]), which we call here the factorization theorem : the Herbrand interpretation of an equational theory is initial in the variety of its models, as an algebra of functions (not objects). Thus, thinking about TRS's as computational definitions of objects we define their interpretations : algebras in which no information is given on terms which have to be computed (this is consistent with the classical semantics of recursive definitions, see [45,14]). The semantics of terms under interpretation is then defined as the set of results of terminating computations (and here again we generalize the case of recursive definitions, see [4]).

- The last section 5 is an approach to the construction of interpreters, along the lines of [23]. The problem is to give effective and efficient correct computation rules. We first define the notion of needed redex; call-by-need computations are those which only rewrite needed redexes.

The main result is that the subspace of call-by-need computations is yet complete (and seems to have a better structure : it is a coherent algebraic domain - a "domaine de calcul" in the sense of [24] - and we conjecture that it is an event structure [46] and, in the non-ambiguous case, a concrete domain [24]) and that there is a continuous projection giving from any computation its (maximal) call-by-need subcomputation. Moreover when the system is non-ambiguous (which for us means that if left-hand sides overlap, then they are equal) this projection preserves termination.

Then we define sequential systems, which are non-ambiguous ones in which needed redex cannot be created by unnecessary rewritings, and we prove that the call-by-need is a correct (and effective) computation rule for these systems : the continuous projection preserves results in this case.

Note: For lack of space, most of the proofs are omitted or only sketched (thus we merely emphasize the intuitive view of objects).

2. Domains, Algebras and Trees

2.1 Domains

In this section we introduce some basic concepts widely used in programming languages semantics. The aim of this theory is to provide a notion of interpretation for languages, in order to give meaning to objects (such as programs). Thus a first question is : what can be a domain of interpretation? We must take into account the fact that functionnal objects can be defined, and that these "infinite" objects have to be computable in some sense. An answer has been given by D. Scott [40,41] : his idea is that infinite computable objects are limits of finite approximations; to be an approximation is nothing more than a partial order and in domains, an increasing sequence of objects must have a limit, its least upper bound. There are many possible variations about this idea (see [45,24]) but in this paper we shall have no need of

an elaborated notion of domain, thus we adopt the classical concept of complete partial order :

a <u>domain</u> is a complete partial order (cpo) that is a structure

$\langle D, \sqsubseteq, \perp \rangle$ where

(i) D is a set

(ii) $\sqsubseteq$ is a partial order on D

(iii) $\perp$ is the least ("undefined") element of D

such that any directed subset X of D has a least upper bound $\bigsqcup X$

(let us recall that X is directed iff two elements of X are bounded in X: $\forall x \in X \, \forall y \in X \, \exists z \in X : x \sqsubseteq z \& y \sqsubseteq z$)

A typical example is the set of partial mappings from a set to another, ordered by inclusion (or extension).

The corresponding notion of morphism is that of <u>strict continuous</u> functions, that is increasing functions which preserves the lub of directed subsets ($f(\bigsqcup X) = \bigsqcup f(X)$). With this notion (which is not exactly the usual one : generally one consider continuous functions, which are the increasing functions preserving the lub of non-empty directed subsets), there are free domains over partially ordered sets with least element : we say that a domain $\langle D', \sqsubseteq', \perp' \rangle$ is a <u>completion</u> of a such a $\langle D, \sqsubseteq, \perp \rangle$ (up to a mapping $j: D \to D'$) iff for any domain $\langle D'', \sqsubseteq'', \perp'' \rangle$ and strict increasing mapping $f: D \to D''$ (s.t. $f(\perp) = \perp''$ and

$x \sqsubseteq y \Rightarrow f(x) \sqsubseteq'' f(y)$.) there exists one and only one morphism

$f^{\infty} : D' \to D''$ such that $f = f^{\infty} \circ j$.

Two such completions are canonically isomorphic, and in this case j is an "embedding" :

$$x \sqsubseteq y \iff j(x) \sqsubseteq' j(y)$$

thus injective, so we may identify D and j(D) (and f^∞ is the restriction of f^∞ to D). There is a very well-know way to build a completion of a $\langle D, \sqsubseteq, \bot \rangle$: let us call ideal of D any non-empty directed cone (that is $X \subseteq D$ s.t. $x \in X \;\&\; y \sqsubseteq x \Rightarrow y \in X$) of D, let D^∞ be the set of these ideals and $j : D \to D^\infty$ be given by

$$j(x) = \{ y \;/\; y \in D \;\&\; y \sqsubseteq x \}$$

Then $\langle D^\infty, \subseteq, \{\bot\}\rangle$ is (up to j) a completion of $\langle D, \sqsubseteq, \bot \rangle$.

The definition of domains of interpretations as cpo's allows to build various domains from other ones. For example any product of domains, ordered by the product order (ie componentwise) is a domain. As a special case, one can see that the set D^X of mappings from X to D is a domain if D is a domain, and the product order is here the extensional one :

$$f \sqsubseteq g \iff \forall x \in X : f(x) \sqsubseteq g(x)$$

(and the limit of a directed subset F is a pointwise defined mapping : $\bigsqcup F(x) = \bigsqcup \{f(x) \;/\; f \in F\}$). If X is also a domain, we get as a subdomain of D^X the set $(X \to D)$ of the continuous functions from X to D.

2.2 Algebras

Pursuing our investigations about interpretations of languages, we remark that programming languages exhibit, at least at the level of abstract syntax, some algebraic structure. This is obvious for functional languages and also, for example, quoting again the work of D.Scott [42], we may built instructions from other ones by conditional branching, sequential composition and so on, so that control structures are functions over instructions.

Thus, as we have said that domains of interpretations are cpo, we may now precise : interpretations are complete algebras (or continuous algebras [1], complete magmas [13]). Since in programming languages we

have in general many syntactic categories, we should have consider heterogeneous algebras, but in this paper we shall not emphasize this point. Algebras are relative to a signature, which is a set F of function symbols and a mapping from F to $\mathbb{N}$ (the set of non-negative integers) which indicates the number of arguments (arity) of function symbols. We allways forget this mapping, denoting

$$F = F_o + F_1 + \ldots + F_n \ldots$$

where F_k is the set of function symbols of arity k. We shall use the following definitions (see [1,13]) :

(i) an F-algebra is a structure $A = \langle D, \{ f_A / f \in F\}\rangle$ where D is a set and for each $f \in F_k$, f_A is a mapping from D^k to D (an element of D if k = o since $D^o = \{ \emptyset \}$)

(ii) an ordered F-algebra is $A = \langle D, \sqsubseteq, \bot, \{f_A / f \in F \}\rangle$ which is an F-algebra where $\langle D, \sqsubseteq, \bot \rangle$ is an ordered set with least element, and the mappings f_A are increasing.

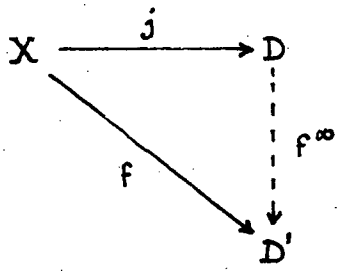
(iii) a complete F-algebra is an ordered F-algebra where $\langle D, \sqsubseteq, \bot \rangle$ is a domain and each f_A is continuous.

The respective notions of (strict) morphisms have an obvious definition: mappings (resp. strict increasing, strict continuous mappings) φ which preserves the algebraic structure :

$$\varphi (f_A (d_1,\ldots,d_k)) = f_A \cdot (\varphi(d_1),\ldots,\varphi(d_k))$$

This allows to define free structures generated by some set X : these are algebras on D and mapping $j : X \to D$ such that for all other algebra (of the same kind) on D' and mapping $f : X \to D$ there exists one and only one morphism $f^{\infty} : D \to D'$ such that $f = f^{\infty} \circ j$

In diagram :



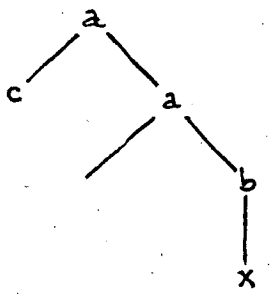
Our next aim is to provide constructions of such structures which are, up to isomorphism, unique.

2.3 Trees and Term Rewriting Systems.

It is well-known[1] that the sets of terms, finite trees and (finite or infinite) trees, on a set X, give the domain of the intended free structures. Terms can be considered as special cases of (finite) trees, so we formally define only this last concept and write terms as expressions (in polish notation with parentheses) or as tree-like pictures as well. The concept of tree has many possible definitions, as graph (may be labelled) for example. Our trees consists of a set of nodes, each equipped with a label in $F \cup X$, in such a way that if this label is in X, then the node is a leaf, and if it is in F_k, the node has at most k sons (we shall represent $\perp$ as the empty tree, which may be a subtree of another one).

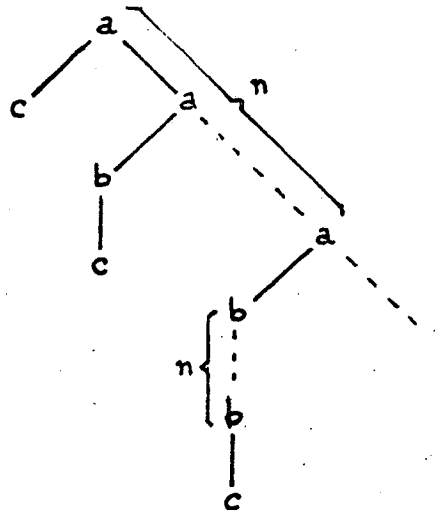
Example 2.1

if $F = \{c\}+\{b\}+\{a\}$ (which means $F_0 = \{c\}$, $F_1 = \{b\}$, $F_2 = \{a\}$ and $F_k = \emptyset$ for $k>2$) and $x \in X$, we may picture an example of finite tree by



(1) We shall emphasize here on very classical notions, in order to reduce our proofs to hopefully convincing intuitive arguments.

and an example of infinite one is suggested by



Usually (see [35]) nodes are coded by sequences of integers. We adopt here another convention : nodes are again words, in such a way that if in a tree a node has label $f \in F_k$, we code in its sons which argument of f they are. So we use to define nodes the "splitted alphabet" $W_F \subseteq F \times \mathbb{N}$ associated with F :

$$W_F = \{(f,i) \;/\; \exists\, k \in \mathbb{N} : f \in F_k \;\&\; 1 \leq i \leq k\}$$

(in the examples we often shorten (f,i) in f_i)

Example 2.1 (continued)

with $F = \{c\}+\{b\}+\{a\}$ we have

$W_F = \{(b,1),(a,1),(a,2)\}$ (or $\{b_1,a_1,a_2\}$)

As it is usual, W_F^* denotes the set of finite words over W_F. We shall denote by ε the empty word, and the concatenation of u and v by uv. We say that u is a prefix of v iff $\exists w : v = uw$ and note $u \leq_{pref} v$ (for it is obviouslv a partial order). Finally, for $L \subseteq W_F^*$ and $w \in W_F^*$ we denote by L/w the residual of L by w :

$$L/w = \{\, u \;/\; u \in W_F^* \text{ and } wu \in L \,\}$$

The definition is :

an F-Tree on X is a partial mapping t from W_F^* to $F \cup X$ such that, if dom (t) denotes its range of definition :

(i) $u \in dom(t)$ & $v \leq_{pref} u \Rightarrow v \in dom(t)$

(ii) $wf_i \in dom(t)$ for $f \in F_k \Rightarrow$

$$t(w) = f \;\&\; (dom(t) / w) \cap W_F \subseteq \{f_1, \ldots, f_k\}$$

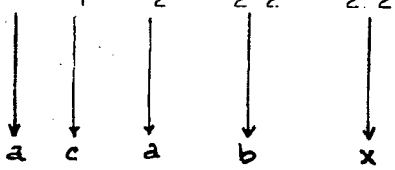
In this definition, $u \in dom(t)$ is a node of the tree t, while t(u) is its label. We may note that if this label is in $F_o \cup X$, then the node is a leaf, that is $dom(t)/u = \{\varepsilon\}$. If $dom(t) \neq \emptyset$ then $\varepsilon \in dom(t)$, and ε is the root of the tree.

Example 2.1 (continued)

The finite tree of this example is formally defined by

$dom(t) = \{ \varepsilon, a_1, a_2, a_2a_2, a_2a_2b_1 \}$

t :



while for the infinite one

$dom(t') = \{ u \;/\; \exists\, n \in \mathbb{N} : u \leq_{pref} a_2^n a_1 b_1^n \}$ with

$t'(a_2^p) = a$ (may be p=o)

$t'(a_2^n a_1) = b$

$t'(a_2^n a_1 b_1^p) = b$ if $p<n$ or c otherwise $(p=n)$

We shall denote by $M_F^\infty(X)$ the set of these trees, and by $M_F(X)$ the subset of finite ones (ie s.t. dom(t) is finite). Trees are naturally

ordered as partial mappings that is by inclusion (or restriction), but here we adopt for this relation the notation of M. Nivat :

$$t \preceq t' \iff \forall u \in W_F^* \; : \; u \in dom(t) \Rightarrow u \in dom(t') \;\&\; t'(u) = t(u)$$

We may also say that t is an initial subtree or a prefix of t'. For this order the empty tree $\perp$ (s.t. dom $(\perp) = \emptyset$) is obviously the least element. We shall say that a finite tree is a term if it is maximal w.r.t. $\preceq$, and we denote the set of terms by $\overline{M}_F(X)$. If we define :

(i) $j : X \to M_F(X)$ by :

$$\forall x \in X \quad dom(j(x)) = \{\varepsilon\} \;\&\; (j(x))(\varepsilon) = x$$

(ii) For all $f \in F_k$, $f_H : M_F^{\infty}(X)^k \to M_F^{\infty}(X)$ by :

$$\forall (t_1,\ldots,t_k) \in M_F^{\infty}(X)^k \qquad t = f_H(t_1,\ldots,t_k) \iff$$

$$dom(t) = \{\varepsilon\} \cup \bigcup_{1 \leq i \leq k} \{f_i\}\, dom(t_i)$$

$$t(\varepsilon) = f \;\&\; \text{for } u \in dom(t_i) : t(f_i\, u) = t_i(u)$$

then we can state the well-known (see [13]) result :

Proposition 2.1

The structures

$\langle \overline{M}_F(X) , \{f_H / f \in F\} \rangle$

$\langle M_F(X) , \preceq, \perp, \{f_H / f \in F\} \rangle$

$H = \langle M_F^{\infty}(X), \preceq , \perp , \{f_H / f \in F\} \rangle$

are resp. the free F-algebra, free ordered F-algebra, free complete F-algebra generated by X w.r.t. j. Moreover $\langle M_F^{\infty}(X), \preceq , \perp \rangle$ is a completion of $\langle M_F(X), \preceq , \perp \rangle$.

The last point allows to make free use of definitions and proofs on $M_F^\infty(X)$ by structural induction and continuity argument. As we have already mentioned, we identify x and j(x), thus $X \subseteq M_F^\infty(X)$, and we omit the suscript H in f_H, thus we can write finite trees as expressions such as

$$a(c,\ a(\ \perp\ ,b(x)))$$

for the finite tree of example 2.1.

In order to introduce computations on trees, we have to give more insight into these objects.

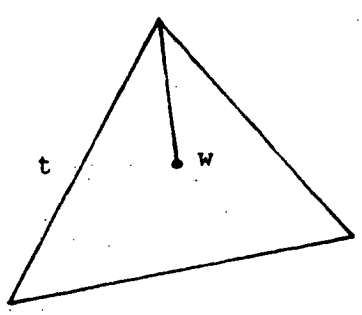
All the following definitions are classical (see [39,21]), but our notations may be unusual at some points. The first concept of use is that of <u>occurrence</u>: we shall make an ambiguous use of this word : occurrences in a tree will be occurrences of symbols, of subtree, of redexes and so on. There is a little difference between nodes and occurrences since we want for example to talk about the occurrences of the empty tree in a tree. So the definition is :

for all $t \in M_F^\infty(X)$ occ(t) is the subset of W_F^* given by

occ ($\perp$) = {ε} and for $t \neq \perp$:

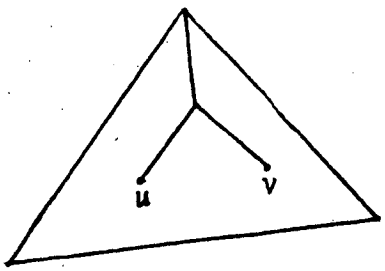
$$occ(t)=dom(t)\cup\{wf_i \ /\ w \in dom(t), t(w) = f \in F_k \ \ 1\leq i\leq k\}$$

We usually picture an occurrence w in a tree t by

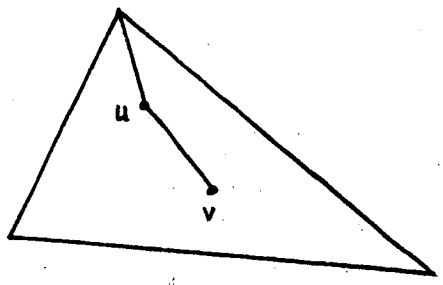


Since occ(t) is a subset of W_F^*, we may compare occurrences by means of $\leq_{pref}$: we say that two occurrences u and v in t are disjoint, and we denote u # v iff they are incompatible (ie there is no w in occ(t) s.t. u $\leq_{pref}$ w and v$\leq_{pref}$ w). On the other hand, two compatible occurrences are comparable and we say that u covers v if u$\leq_{pref}$v (the only possible respective positions are : u # v, u$\leq_{pref}$v or v$\leq_{pref}$u).

In picture :



u # v
(u and v disjoint)



u $\leq_{pref}$ v
(u covers v)

We may also note that, if we define a partial order on W_F by

$$f_i \leq g_j \iff f = g \;\&\; i \leq j$$

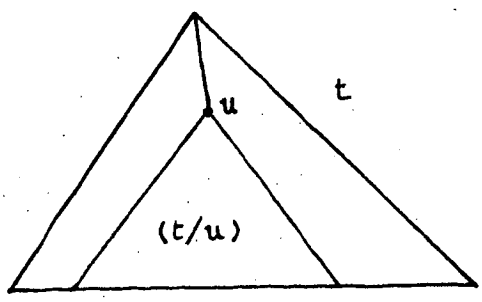
then for all tree t, occ(t) is totally ordered by the lexicographical ordering $\leq_{lex}$ on W_F^* associated with $\leq$ (u $\leq_{lex}$ v iff u covers v or u is "at the left" of v).

We have said that occurrences are occurrences of subtrees, thus for u $\in$ occ(t) we define the subtree (t/u) of t at occurrence u as

$$dom(t/u) = dom(t)/u$$

$$(t/u)(v) = t(uv)$$

and we draw :



We can say that $u \in occ(t)$ is an occurrence of (t/u) in t and more generally for $T \subseteq M_F^{\infty}(X)$, we denote by occ(T,t) the set of occurrences of elements of T in t :

$$occ(T,t) = \{u \ / \ u \in occ(t) \ \& \ (t/u) \in T\}$$

(this is not the last ambiguous use of "occ"!).

From now on, we shall take for X, the set of generators of the free algebras of trees, a denumerable set of <u>variables</u>:

$$X = \{x_n \ / \ n \in \mathbb{N}\}$$

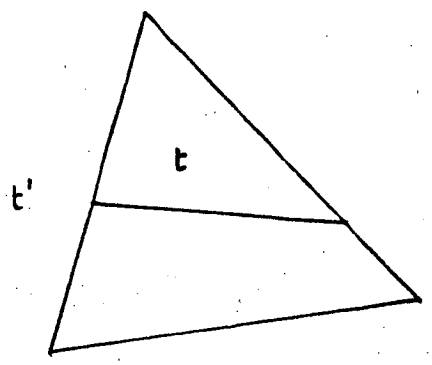
(we shall also denote variables by x, y, z ...)

so our terms are expressions denoting functions, and we denote by var(t) the set of variables which have some occurrence in t :

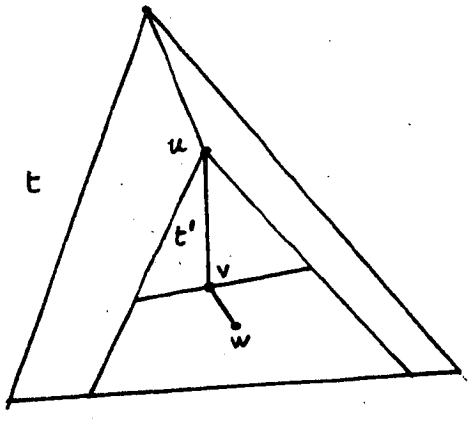
$$var(t) = \{x \ / \ x \in X \ \& \ \exists \ u \in occ(t) \ : \ (t/u) = x\}$$

A tree t is <u>linear</u> iff any variable has at most one occurrence in t. With this set of generators, it is usual to call <u>substitutions</u> the endomorphisms of H, for in this case such endomorphisms are s^{∞} for some $s : X \to M_F^{\infty}(X)$.

We say that a tree t' is an <u>instance</u> of t (or t matches t') iff their exists a substitution s such that $t' = s^{\infty}(t)$, and we draw :



This defines a preorder, the subsumption preorder: two trees are said to be unifiable iff they are compatible w.r.t. the subsumption preorder. We shall also say that $u \in occ(t)$ is an occurrence of t' in t if (t/u) is an instance of t', and we say that, in this case, the pair (u,t') covers an occurrence $w \in occ(t)$ iff $\exists\, v \in occ(X,t') : uv \leq_{pref} w$. In picture :

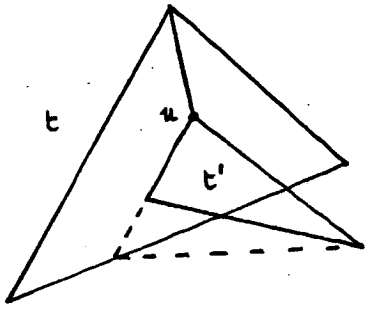


(u,t') covers w in t

We want to define now the overlapping relation between trees, and to avoid trivial cases we shall denote by int(t) the "internal" occurrences in t, that is those which are not occurrences of variables :

$$int(t) = occ(t) - occ(X,t)$$

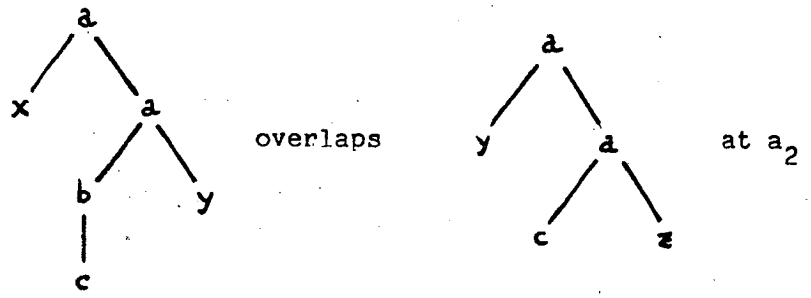
We then say that a tree t overlaps t' iff $\exists\, u \in int(t)$ s.t. (t/u) and t are unifiable (we may precise that t overlaps t' at occurrence u) and we draw :



t overlaps t' at u

Example

with F = {c}+{b}+{a}



In fact, we shall consider overlapping as a symetric relation :

$t \mathbin{\Delta} t'$ iff t overlaps t' or t' overlaps t

The last definition about trees is that of replacement of the subtree at occurrence u in t by t' resulting in a tree denoted $[t'/u]t$ (usually this is written $t[u<-t']$, but there is some similarity with substitution) defined by

$$dom([t'/u]t) = (dom(t)-\{u\}W_F^*) \cup \{u\}dom(t')$$

$$[t'/u]t(v) = t(v) \text{ if } v \in dom(t) \;\&\; u \not\leq_{pref} v$$

$$[t'/u]t(uw) = t'(w) \text{ if } w \in dom(t')$$

We obviously have :

$$[(t/u)/u]t = t \quad , ([t'/u]t/u) = t' \quad \text{and}$$

$$\text{if } u \# v : [t_1/u]([t_2/v]t) = [t_2/v]([t_1/u]t)$$

while $[t_2/v]([t_1/u]t) = [(t_1/(u/v))t_2/u]t$ if $u \leq_{pref} v$

and $[t_2/v]([t_1/u]t) = [t_2/v]t$ if $v \leq_{pref} u$.

Now we introduce rewriting relations on trees which are subsets of the cartesian square of $M_F^\infty(X)$.

Such an $R \subseteq M_F^\infty(X) \times M_F^\infty(X)$ is said to be

(i) <u>compatible</u> (with the algebraic structure) <u>iff</u> $\forall k \in \mathbb{N}\ \forall f \in F_k$ <u>if</u> $\forall i\ (1 \leq i \leq k)\ (t_i, t'_i) \in R$ <u>then</u> $(f(t_1,\ldots,t_k)\ ,\ f(t'_1,\ldots,t'_k)) \in R$

(ii) <u>stable</u> (by substitutions) <u>iff</u> for all substition s

$$(t,t') \in R \Rightarrow (s^\infty(t),\ s^\infty(t')) \in R$$

We shall only use stable relations on trees, thus we call <u>precongruence</u> any compatible and stable preorder, and <u>congruence</u> any symetric precongruence. If $\leq$ is a precongruence, we denote by $M_F^\infty(X)/\leq$ the quotient of $M_F^\infty(X)$ by the congruence associated with $\leq$ (that is : $t \leq t' \ \&\ t' \leq t$), which is canonically an F-algebra. Given $R \subseteq M_F^\infty(X) \times M_F^\infty(X)$, the <u>rewriting relation</u> determined by R is the reflexive and transitive closure $\overset{*}{\underset{R}{\rightarrow}}$ of the relation $\underset{R}{\rightarrow}$ defined by

$$t \underset{R}{\rightarrow} t' \iff \exists\ (\theta,\theta') \in R\ \ \exists s \text{ substitution } \exists u \in occ(t) :$$

$$(t/u) = s^\infty(\theta)\ \&\ t' = [s^\infty(\theta')/u]t$$

If we denote by $\underset{R}{\leftarrow}$ the inverse of $\underset{R}{\rightarrow}$ and $\underset{R}{\leftrightarrow} = (\underset{R}{\leftarrow} \cup \underset{R}{\rightarrow})$, it is a well-known fact that $\overset{*}{\underset{R}{\rightarrow}}$ and $\overset{*}{\underset{R}{\leftrightarrow}}$ are resp. the precongruence and congruence generated by R (they are the least ones which contain R). In fact we only consider $\underset{R}{\rightarrow}$ as a true rewriting relation when R is a term rewriting system :

a <u>term rewriting system</u> (TRS) is a relation $R \subseteq M_F^\infty(X) \times M_F^\infty(X)$ such that :

(i) R is a finite subset of $\overline{M}_F(X) \times \overline{M}_F(X)$

(ii) $(\theta,\theta') \in R \Rightarrow$ - $\theta \notin X$

- $var(\theta') \subseteq var(\theta)$

- θ is linear.

Every element $r = (\theta,\theta')$ of a TRS R, written $\theta \to \theta'$, is a rule of R, θ is its left-hand side (lhs(r)) and θ' its right-hand side (rhs(r)). A TRS R is said to be non-ambiguous (or non overlapping) if overlapping left-hand sides of its rules are equal and deterministic if moreover no two distinct rules have the same left-hand side (thus R is a partial function).

From the definition of $\underset{R}{\to}$, it will be usefull to introduce some termonology : we say that an occurrence u in t is an occurrence of application of R in t , and we write $u \in occ_R(t)$, if (t/u) is an instance of the left-hand side of some rule of R :

$$occ_R(t) = \{u/u \in occ(t) \;\&\; \exists\, \theta \in lhs(R)\ \exists\, s : (t/u) = s^{\infty}(\theta)\ \}$$

and we may even precise : u is an occurrence of the rule r, writing $u \in occ(r,t)$, and we call redex of R in t any pair (u,r) such that u is an occurrence of r in t. We denote by $red_R(t)$ the set of redexes of R in t :

$$red_R(t) = \{(u,r) \;/\; u \in occ(r,t) \;\&\; r \in R\ \}$$

(if R is deterministic, the occurrence $u \in occ_R(t)$ determines a unique redex, and a unique left-hand side if R is non-ambiguous). Thus if $t \underset{R}{\to} t'$ we can say that t' results from t by application of a rule r of R at some of its occurrences u in t which is written $t \xrightarrow[R]{(u,r)} t'$. Since R is a term rewriting system, the relations $\xrightarrow[R]{(u,r)}$ are partial mappings (for $var(rhs(r)) \subseteq var(lhs(r))$ for any rewriting rule r), while the notation $\underset{R}{\to}$ is generally ambiguous since for example if we have r : a(x) -> x as a

rule of R : $a(a(x)) \xrightarrow[R]{(\varepsilon,r)} a(x)$ and $a(a(x)) \xrightarrow[R]{(a_1,r)} a(x)$.

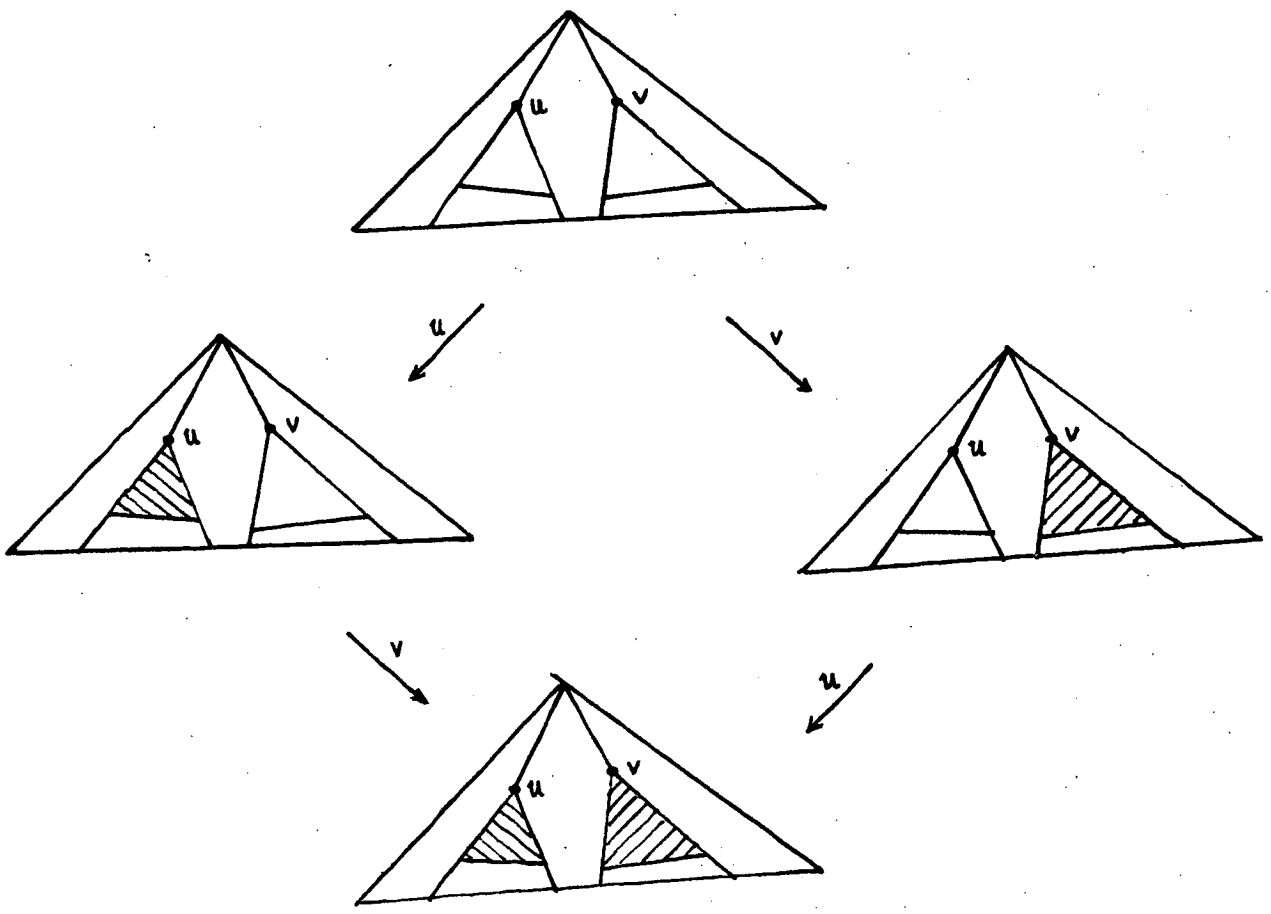
3. Computations

In a TRS R, a relation $\xrightarrow[R]{(w,r)}$ constitutes an elementary step of computation, and computations will be any sequence of such steps. But this definition is too fine, and we want to identify such sequences when they rewrite the "same" redexes, perhaps in different order. For example, if r : b(x) -> a(x,x) is a rule of R, then the two sequences

$$a(b(x),b(x)) \xrightarrow[R]{(a_1,r)} a(a(x,x),b(x)) \xrightarrow[R]{(a_2,r)} a(a(x,x),a(x,x))$$

$$a(b(x),b(x)) \xrightarrow[R]{(a_2,r)} a(b(x),a(x,x)) \xrightarrow[R]{(a_1,r)} a(a(x,x),a(x,x))$$

are the "same" computation. In this example the situation is clear since the applied redexes are disjoint. Formally two redexes (u,r_1) and (v,r_2) in $red_R(t)$ are <u>disjoint</u> iff $u \# v$. In picture :



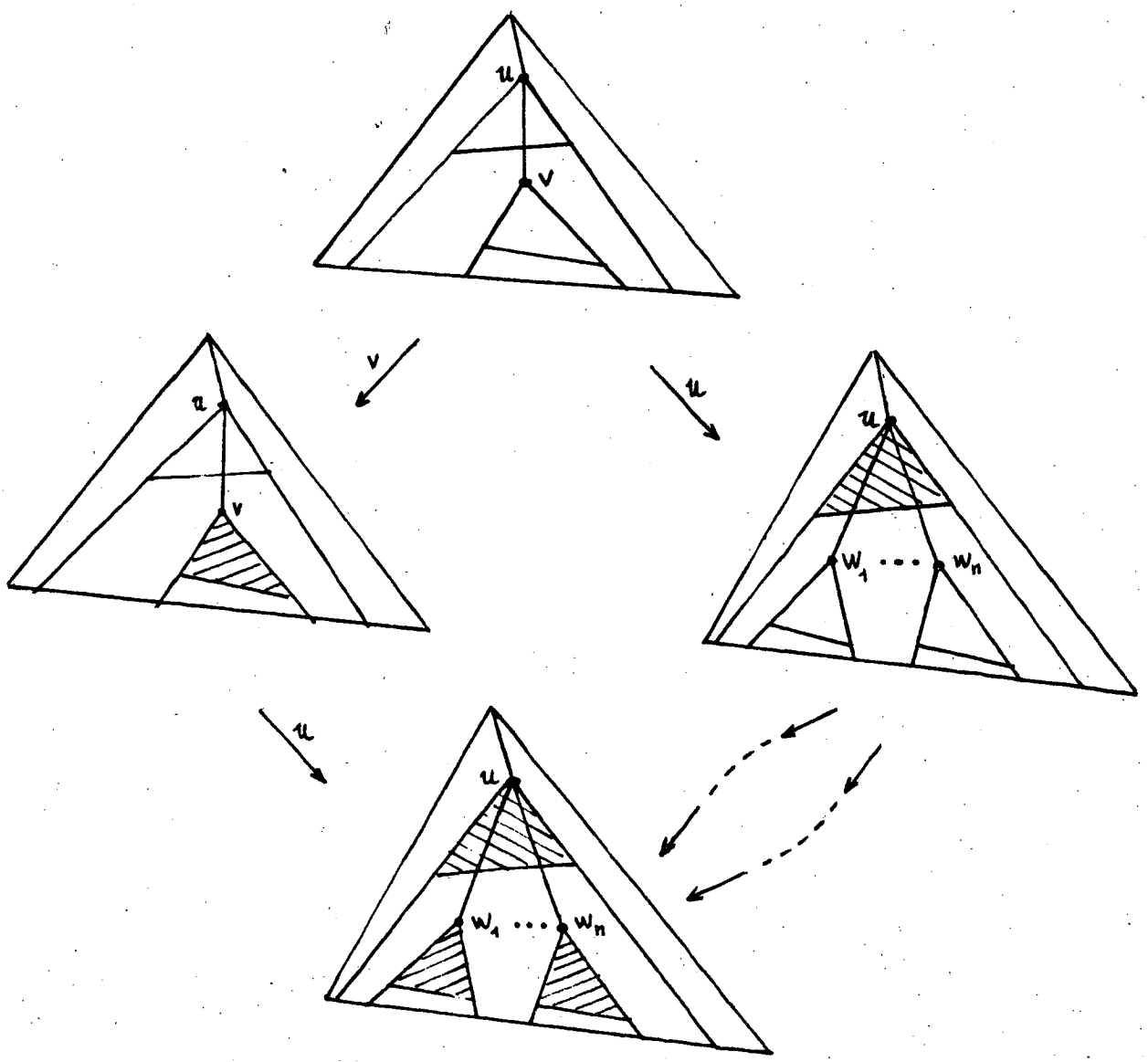
This is less clear if one <u>covers</u> the other (ie $(u,lhs(r_1))$ covers v for example), as in the example of rewritings

$$b(b(x)) \xrightarrow[R]{(\mathcal{E},r)} a(b(x),b(x)) \xrightarrow[R]{(a_1,r)} a(a(x,x),b(x)) \xrightarrow[R]{(a_2,r)} a(a(x,x),a(x,x))$$

$$b(b(x)) \xrightarrow[R]{(\mathcal{E},r)} a(b(x),b(x)) \xrightarrow[R]{(a_2,r)} a(b(x),a(x,x)) \xrightarrow[R]{(a_1,r)} a(a(x,x),a(x,x))$$

$$b(b(x)) \xrightarrow[R]{(b_1,r)} b(a(x,x)) \xrightarrow[R]{(\mathcal{E},r)} a(a(x,x),a(x,x))$$

which, intuitively, are again the "same" computation. To explain this, we have to say that the set $\{(a_1,r),(a_2,r)\}$ is, after the application of $(\mathcal{E},r)$, the "same redex" that (b_1,r). In picture :



Thus we have to

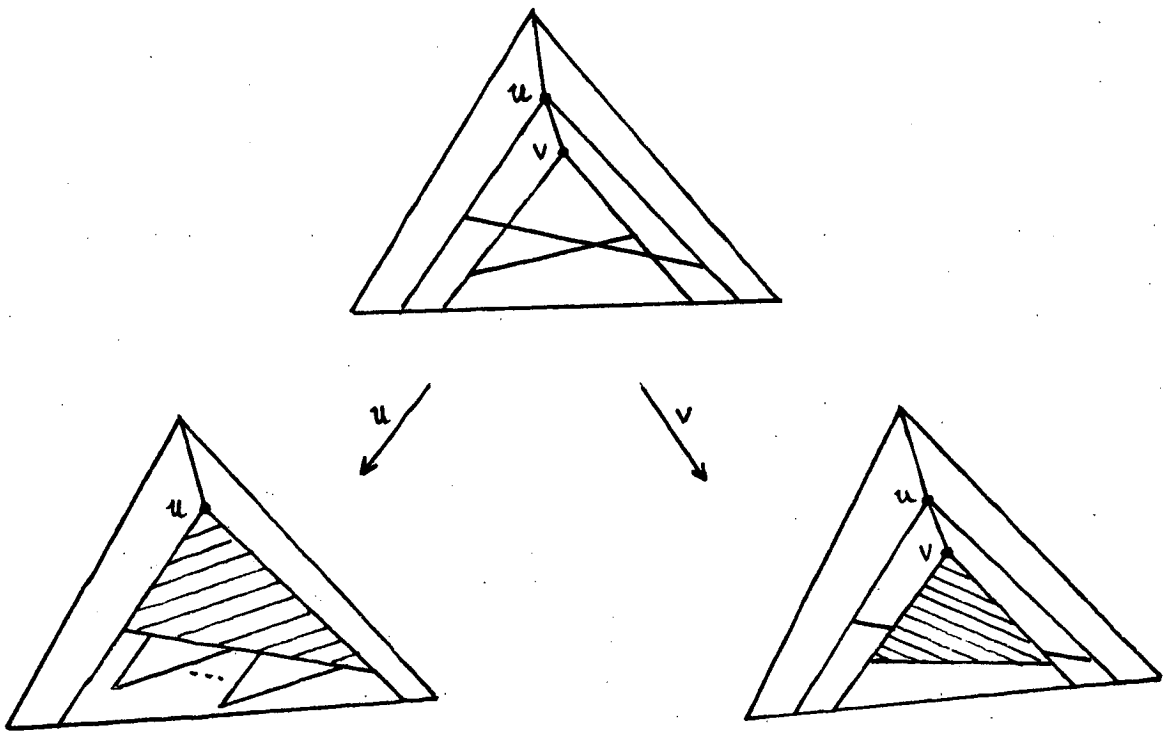
(i) define in what sense some sets of redexes can be considered as a single step of computation.

(ii) define the notion of "to be the same step of computation" along rewriting sequences.

This has been done by G. Berry and J.J Levy [4] in the case of recursive definitions and by G. Huet and J.J. Levy [23] in the more general setting of deterministic TRS's. We shall take exactly the same way.

3.1 Consistent set of redexes.

A set of redexes can be viewed as defining a single step of computation if we can apply simultaneously (or : in parallel) all its elements. Clearly, this is not allways possible : let us say that a redex (u,r_1) overlaps a redex (v,r_2) in a tree t iff $\exists$ w : v = uw and $lhs(r_1)$ overlaps $lhs(r_2)$ at w.



In this case, even if this diagram is confluent, there is no natural way

to define a computation which represent the simultaneous rewriting of (u,r_1) and (v,r_2). On the other hand, the examples of the begining of this section suggest that the other possibilities (disjoint redexes, or one covering the other) lead to more tractable situations. This is partially explained by :

lemma 3.1

let (u,r_1) and (v,r_2) be in $red_R(t)$ where R is a TRS

If (u,r_1) covers (v,r_2) or $u \# v$ and $t \xrightarrow[R]{(v,r_2)} t'$

then $(u,r_1) \in red_R(t')$

proof: we only have to check (the case $u \# v$ is trivial) that if (u,r_1) covers (v,r_2) then (t'/u) is again an instance of $lhs(r_1)$. But this is true since $lhs(r_1)$ is linear ∎

We can now define what sets of redexes can be simultaneously rewritten (from now on we only consider relations R which are TRS's, and computation of terms) :

let t be a term. A set $\sigma \subseteq red_R(t)$ of redexes of R in t

is consistent iff any two (distinct) of its elements do not overlap.

A consistent set of redexes is thus a partial mapping from occ(t) to R (which indicates what rule is to be applied at some occurrence) whose range of definition is

$$dom(\sigma) = \{u/u \in occ(t) \;\&\; \exists\, r \in R : (u,r) \in \sigma\}$$

(for two distinct redexes in t at the same occurrence must overlap, since $lhs(R) \cap X = \emptyset$)

Remark: in a deterministic TRS, any set of redexes (in a term t) is consistent (and $\emptyset$ is allways consistent).

Let us now define what is the result of the application of a consistent set of redexes (the simultaneous rewriting of redexes) σ to a term: from the preceding lemma, it can be defined by induction on the size $|\sigma|$ (the number of redexes in σ) as the term $t.\sigma$ given by :

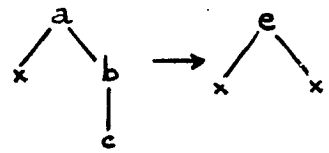
(i) $t.\emptyset = t$

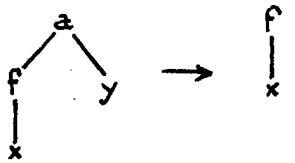
(ii) if $|\sigma| > 0$ then $t.\sigma = t'.\sigma'$ where $\sigma' = \sigma - \{(w,\sigma(w))\}$,
w is the last element w.r.t. $\leq_{lex}$ of $dom(\sigma)$ and
$t \xrightarrow[R]{(w,\sigma(w))} t'$

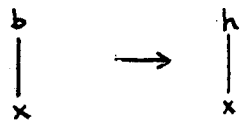
We shall write $t \xrightarrow[R]{\sigma} t'$ (or sometimes $t \xrightarrow[R]{\sigma} t'$) when σ is a consistent set of redexes of R in t and $t' = t.\sigma$.

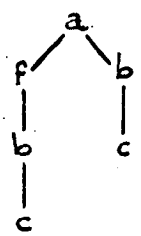
Example

Let R be the set of rules

r_1 : 

r_2 : 

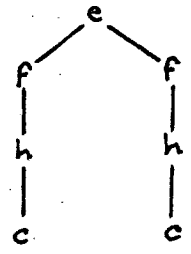
r_3 : 

then for $t =$ 

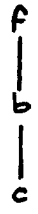
$\{(\varepsilon, r_1), (\varepsilon, r_2)\}$ and $\{(\varepsilon, r_1), (a_2, r_3)\}$

are inconsistent, while

$$t \xrightarrow[R]{(\varepsilon,r_1),(a_1f_1,r_3)}$$



$$t \xrightarrow[R]{(\varepsilon,r_2),(a_2,r_3)}$$



In this example, one can see that, applying (ε,r_1) in the first one step computation, the redex (a_1f_1,r_3) has been duplicated, while applying (ε,r_2) in the second consistent set of redexes, (a_2,r_3) have disappear. This will be formalized latter.

3.2 Computations

From now on, the steps of a computation are consistent sets of redexes (we might better say : derivation instead of computation, since we want to consider computations as equivalence classes of sequences of rewriting). Let R be a TRS and $t \in \overline{M}_F(X)$ a term :

a <u>computation</u> of t in R is a finite or infinite sequence $\gamma = (\sigma_i)_{i \in I}$ (where I is an initial segment of $\mathbb{N}$, the length of the computation) of sets of redexes (the steps of γ) such that : $\forall\, i \in I$ σ_i is a consistent set of redexes of R in $t.\sigma_o \ldots \sigma_{i-1}$

(implicitely if i=o $\sigma_o \ldots \sigma_{i-1}$ is the empty sequence and σ_o is a consistent set of redexes in t). We may also say that the length of a

finite computation $\gamma = (\sigma_i)_{i \in I}$ is the integer $|\gamma|$ such that $I = \{i/i \in \mathbb{N} \ \& \ i<|\gamma| \}$, and in this case we say that γ is a computation of t in t' if $t' = t.\ \sigma_o \ldots \sigma_{|\gamma|-1}$ and we write :

$$\gamma : t \xrightarrow[R]{*} t' \quad (\gamma \text{ is a computation of } t \text{ in } t' \text{ in } R)$$

We shall also denote a computation $\gamma = (\sigma_i)_{i \in I}$ of t in R by :

$$\gamma : t = t_o \xrightarrow[R]{\sigma_o} t_1 \to \ldots \to t_{i-1} \xrightarrow[R]{\sigma_i} t_i \to \ldots$$

$\mathcal{C}_R^\infty(t)$ is the set of computations of the term t in R, $\mathcal{C}_R(t)$ is the subset of finite ones. Sometimes we use $\mathcal{C}_R$ if we do not want to know what term is computed.

For every term t, $\mathcal{C}_R(t)$ contains the empty sequence of consistent set of redexes, that is the empty computation ε. If $\gamma : t \xrightarrow[R]{*} t'$ and $\gamma' \in \mathcal{C}_R^\infty(t')$ the concatenation $\gamma\gamma'$ is obviously a computation of t, which is called the <u>composition</u> of γ and γ', and denoted by $\gamma;\gamma'$.

For $\gamma = (\sigma_i)_{i \in I}$ and $j \in I$, we define the <u>restrictions</u> of γ at j :

$$\gamma|^j = (\sigma_i)_{i < j} \quad \text{and}$$

$$\gamma|_j = (\sigma'_k)_{k \in J} \quad \text{where } J = \{i-j/i \in I \ \& \ i \geq j\} \text{ and}$$

$$\sigma'_k = \sigma_{j+k}$$

and for $j \in I$ and $i \leq j$:

$$\gamma|_i^j = (\gamma|^j)|_i$$

The size $||\gamma||$ of a computation is the number (may be infinite) of the non-empty steps of γ.

In order to define equivalent computations, we have to define, as we have already pointed out, the notion of "to be the same occurrence" along a computation. This is done as in [4,23] by means of the concept of residual due to A. Church [12] (see also [27]), to what we add the notion of trace of an occurrence through a computation :

given a term $t \in \overline{M}_F(X)$, a finite computation $\gamma \in \mathcal{C}_R(t)$ and a set $W \subseteq occ(t)$ of occurrence in t, the sets $W[\gamma]$ of residuals of W by γ and $W\langle\gamma\rangle$ of traces of W by γ are defined by :

(i) $W[\gamma] = \bigcup_{w \in W} w[\gamma]$ and $W\langle\gamma\rangle = \bigcup_{w \in W} w\langle\gamma\rangle$

(ii) $w[\varepsilon] = \{w\} = w\langle\varepsilon\rangle$

(iii) $w[\sigma;\gamma] = (w[\sigma])[\gamma]$ and $w\langle\sigma;\gamma\rangle = (w\langle\sigma\rangle)\langle\gamma\rangle$

(iv) $w[\emptyset] = \{w\} = w\langle\emptyset\rangle$

(v) if $|\sigma| > o$ then $w[\sigma] = (w[u,r])[\sigma']$ and $w\langle\sigma\rangle = (w\langle u,r\rangle)\langle\sigma'\rangle$

where $(u,r) \in \sigma$, u is the last (w.r.t. $\leq_{lex}$) element of $dom(\sigma)$

and $\sigma' = \sigma - \{(u,r)\}$

It just remains to define residuals and traces of an occurrence $w \in occ(t)$ by an elementary one step computation of t, that is $(u,r) \in red_R(t)$:

(i) if u does not cover w (i.e $w \# u$ or $w <_{pref} u$)

then $w[u,r] = \{w\} = w\langle u,r\rangle$. Otherwise ($u \leq_{pref} w$) :

(ii) if $\exists\, v \in int(lhs(r))$: $w = uv$

then $w[u,r] = \emptyset$ and $w\langle u,r\rangle = \{u\}$ else

(iii) if $\exists$ $v \in occ(X,lhs(r))$ $\exists$ w' : $w = uvw'$ (i.e (u,r) covers w)

then $w[u,r] = w\langle u,r\rangle$

$= u.\ \{v'/v' \in occ(rhs(r)) \ \&\ (rhs(r)/v') = (lhs(r)/v)\}.w'$

Example: let R be the TRS

r_1 : b(x) -> a(x,x)

r_2 : a(x,y) -> f(x)

and γ the finite computation

$$\gamma : b(c) \xrightarrow[R]{(\varepsilon,r_1)} a(c,c) \xrightarrow[R]{(\varepsilon,r_2)} f(c)$$

Then :

- $\varepsilon[\gamma] = \emptyset$ and $\varepsilon\langle\gamma\rangle = \{\varepsilon\}$: the occurrence ε

is consumed by the (first step of the) computation

- $b_1[\gamma] = b_1\langle\gamma\rangle = \{f_1\}$ and $b_1[\varepsilon,r_1] = \{a_1,a_2\}$:

the occurence b_1 is duplicated by $\gamma|^1$ while a_2 is deleted by $\gamma|_1$ since $a_2[\varepsilon,r_2] = a_2\langle\varepsilon,r_2\rangle = \emptyset$

- the redex (ε,r_2) is created by $\gamma|^1$: it was not a redex of t.

We even may have backward (or bottom-up) creation of redexes as in :

r_1 : k(x,a) -> c

r_2 : b -> a

and γ : $k(x,b) \xrightarrow[R]{(k_2,r_2)} k(x,a) \xrightarrow[R]{(\varepsilon,r_1)} c$

(to follow residuals and traces along computation, it should have been better to have colours, or labels as in [44]).

It is easy to verify that if γ is a computation of t in t' then $w[\gamma]$ and $w\langle\gamma\rangle$ are subsets of occ(t'). These concepts give a first tool to distinguish computations :

let γ and γ' be two finite computations. γ and γ' are <u>elementarily equivalent</u> , $\gamma \simeq \gamma'$ <u>iff</u> $\exists t \in \bar{M}_F(X)$: γ and γ' are two computations of t in t' and for all $w \in occ(t)$:

$$w[\gamma] = w[\gamma'] \;\&\; w\langle\gamma\rangle = w\langle\gamma'\rangle$$

This is a congruence (w.r.t. composition) for we obviously have

$$w[\gamma_1 ; \gamma_2] = (w[\gamma_1])[\gamma_2] \text{ and } w\langle\gamma_1 ; \gamma_2\rangle = (w\langle\gamma_1\rangle)\langle\gamma_2\rangle$$

However this equivalence is not fine enough. For example if we have

$$r_1 : f(x) \to b(b(x))$$

$$r_2 : b(x) \to x$$

and

$$\gamma_1 : f(c) \xrightarrow[R]{\varepsilon} b(b(c)) \xrightarrow[R]{\varepsilon} b(c)$$

$$\gamma_2 : f(c) \xrightarrow[R]{\varepsilon} b(b(c)) \xrightarrow[R]{b_1} b(c)$$

then $\gamma_1 \simeq \gamma_2$ but we shall not consider that γ_1 and γ_2 are the "same" computation (in the intuitive meaning : "to perform the same rewriting perhaps in different order").

We may note that :

- since R is left-linear, every occurrence which is a residual of something through a computation is a residual of a unique occurrence :

$$\forall\, y \in \mathcal{C}_R(t)\ \ \forall u \in occ(t)[y]\ \ \exists!\ \ w \in occ(t) : u \in w[y]$$

and we call w the ancestor of u through y

(this property is generally false for traces)

- one-step computations do not destroy the compatible redexes they do not use. To state this more formally, let us say that :

> for σ and σ' consistent sets of redexes of R in t, σ and σ' are <u>compatible</u>, $\sigma \uparrow \sigma'$ <u>iff</u> $\sigma \cup \sigma'$ is a consistent set of redexes in t.

For such consistent sets of redexes, one can define the crucial notion of residual of one by the other, intuitively what remains to do of the first after applying the other :

<u>lemma 3-2</u>

> for σ and σ' compatible consistent set of redexes of R in the term t :
>
> $$\sigma[\sigma'] = \{(w',r)/\exists w : (w,r)\in\sigma\ \&\ w'\in w[\sigma']\}$$
>
> is a consistent set of redexes of R in $t.\sigma'$, the <u>residual</u> of σ by σ'

<u>proof</u>: by induction on $(|\sigma'|,|\sigma|)$ and by cases when $|\sigma|=1=|\sigma'|$, omitted.
<u>Remark</u>: if the TRS R is deterministic, one can allways define $\sigma[\sigma']$ since in this case two consistent sets of redexes are compatible.

For the rest of this section, our aim is to state (without proofs, which are rather long and technical, see [4,7], but straightforward) two well-known facts about computations : an elementary form of the "parallel moves lemma" (here called the permutation lemma) and the "finite developments theorem".

Let us sav that a step of computation σ is elementary if $|\sigma| = 1$, and denote by $e(\sigma)$ the (elementary) computation

$$e(\sigma) : t \xrightarrow[R]{(w_n,r_n)} t_1 \to \ldots \xrightarrow[R]{(w_1,r_1)} t_n$$

if $\sigma = \{(w_1,r_1),\ldots,(w_n,r_n)\}$ is a consistent set of redexes in t where $w_1,\ldots,w_n$ is lexicographically ordered, and $e(\emptyset) = \mathcal{E}$. Obviously $\sigma \simeq e(\sigma)$.

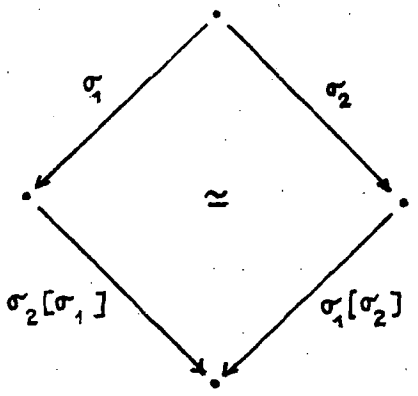
permutation lemma 3-3 : first (elementary) form

let σ_1 and σ_2 be two elementary steps of computation of t :

$$\sigma_1 \uparrow \sigma_2 \Rightarrow \sigma_1 ; e(\sigma_2[\sigma_1]) \simeq \sigma_2 ; e(\sigma_1[\sigma_2])$$

(one may remark that, under the hypothesis of this lemma, $dom(\sigma_1[\sigma_2])$ and $dom(\sigma_2[\sigma_1])$ are sets of disjoint occurrences).

In picture :



if $\sigma_1 \uparrow \sigma_2$

Now we want to define, by means of the notion of residuals, the set of computations which "realize" a given consistent set of redexes σ on a term t. Such computations have to apply only redexes of σ, or some of their residuals through a first step which partially realizes σ:

a computation $\gamma \in \mathcal{C}_R(t)$ is relative to a consistent set of redexes σ in t iff either $\gamma = \varepsilon$ or $\gamma = \sigma' : \gamma'$ for some $\sigma' \subseteq \sigma$, $\sigma' \neq \emptyset$ and γ' relative to $\sigma[\sigma']$.

Then a computation realize - or : is a development of - σ if it is relative to σ and performs all the rewritings indicated by σ, that is it remains nothing to do of σ after it :

$\gamma \in \mathcal{C}_R(t)$ is a development of σ on t iff γ is relative to σ and $dom(\sigma)[\gamma] = \emptyset$

This definition is not empty : for all σ, there exists a development of σ on t : ε if $\sigma = \emptyset$, otherwise σ itself (or $e(\sigma)$).

Example

Let R be given by the set of rules

r_1 : h(a,x) ->x

r_2 : f(a,x,y) -> h(x,x)

Then the following computations (where we unambiguously forget some indications) are developments of

$\sigma = \{(\varepsilon, r_2), (f_2, r_1), (f_3, r_1)\}$ on

t = f(a,h(a,a),h(a,a))

γ_1 : $t \xrightarrow[R]{\varepsilon} h(h(a,a),h(a,a)) \xrightarrow[R]{h_1} h(a,h(a,a)) \xrightarrow[R]{h_2} h(a,a)$

γ_2 : $t \xrightarrow[R]{\varepsilon} h(h(a,a),h(a,a)) \xrightarrow[R]{h_1,h_2} h(a,a)$

γ_3 : $t \xrightarrow[R]{f_2} f(a,a,h(a,a)) \xrightarrow[R]{\varepsilon} h(a,a)$

γ_4 : $t \xrightarrow[R]{f_3} f(a,h(a,a),a) \xrightarrow[R]{\varepsilon} h(h(a,a),h(a,a)) \xrightarrow[R]{h_1,h_2} h(a,a)$

and so on .

The finite development theorem 3-1

For all consistent set of redexes σ of R in a term t

(i) there exists $n \in \mathbb{N}$ such that if γ is a development of σ on t then $|\gamma| \leq n$

(ii) for two such developments γ and γ' : $\gamma \simeq \gamma'$

The termination argument for the first point is not so easy to find , see [4].

3.3 Equivalence : the permutation and simplification lemmas

In this paragraph we consider only finite computations. We have seen that we can define residuals for compatible consistent sets of redexes : without the compatibility hypothesis, this in general does not make sense, and a fortiori for computations. Thus in our TRS's (may be ambiguous, non-deterministic) we cannot say as in [4,27,23] that two computations are the same if it remains nothing to perform of one after the other. Nevertheless, we have already define in some particular cases computations which are the same one : the developments of a given consistent set of redexes. This is the basis for the definition of the equivalence by permutations of finite computations of a given term :

For computations γ , γ' in $\mathcal{C}_R$

(i) $\gamma \sim \gamma' \Leftrightarrow_{def} ||\gamma|| = o = ||\gamma'||$ or γ and γ' are developments of some consistent set of redexes σ on t

(ii) the <u>equivalence by permutation</u> is the congruence (w.r.t. composition) $\equiv$ generated by $\sim$, that is the transitive closure of $\approx$ given by :

$$\gamma \approx \gamma' <=>_{def} \exists\ \gamma_1, \gamma_2, \delta, \delta' : \gamma = \gamma_1 ; \delta ; \gamma_2 \ , \ \gamma' = \gamma_1 ; \delta' ; \gamma_2$$

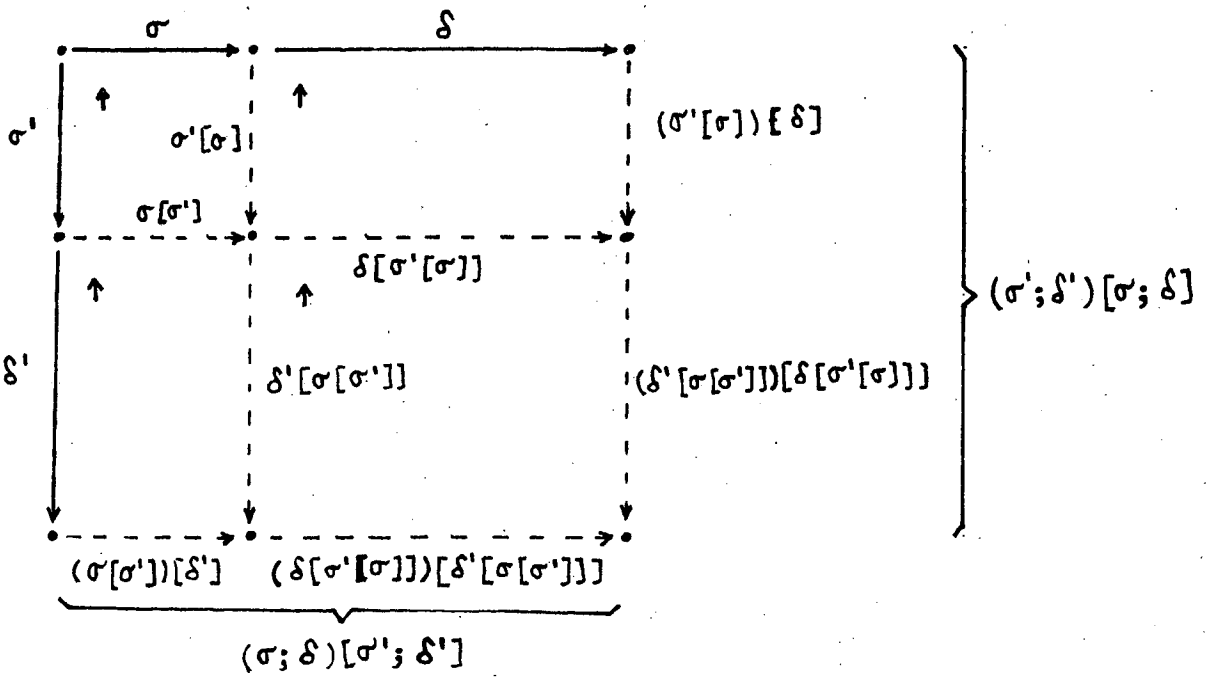
$$\text{and } \delta \sim \delta'$$

We may read $\gamma \equiv \gamma'$ as « γ' is a permutation of γ ». An immediate consequence of the definition and of the finite developments theorem is :

$$\text{For } \gamma, \gamma' \text{ in } \mathcal{C}_R(t) : \gamma \equiv \gamma' \Rightarrow \gamma \simeq \gamma'$$

which we shall freely use .

We want to show another property of this equivalence : its left simplifiability (w.r.t. composition). This needs some intermediate results, the first being a more elaborated form of the permutation lemma: the formulation of this last requires the definition of a compatibility relation on computations (which, from now on, are finite computations in a given TRS R of a given term t), which allows to simultaneously define the residual of a computation by another compatible one. This definition is by induction on the length, thus we picture the induction step to support the intuition :



two computations γ and γ' are <u>compatible</u>: $\gamma \uparrow \gamma'$ and under this assumption $\gamma[\gamma']$ (resp. $\gamma'[\gamma]$) is the <u>residual</u> of γ by γ' (resp. of γ' by γ) <u>iff</u>:

(i) $\gamma = \varepsilon$ and $\gamma[\gamma'] = \varepsilon$ and $\gamma'[\gamma] = \gamma$ or

(ii) $\gamma' = \varepsilon$ and $\gamma[\gamma'] = \gamma$ and $\gamma'[\gamma] = \varepsilon$ or

(iii) $\gamma = \sigma;\delta$ and $\gamma' = \sigma';\delta'$ and (see the figure):

- $\sigma \uparrow \sigma'$
- $\sigma[\sigma'] \uparrow \delta'$ and $\sigma'[\sigma] \uparrow \delta$
- $\delta[\sigma'[\sigma]] \uparrow \delta'[\sigma[\sigma']]$
- $\gamma[\gamma'] = (\sigma[\sigma'])[\delta'] ; (\delta[\sigma'[\sigma]])\,[\delta'[\sigma[\sigma']]]$

$\gamma'[\gamma] = (\sigma'[\sigma])[\delta] ; (\delta'[\sigma[\sigma']])[\delta[\sigma'[\sigma]]]$

The compatibility relation is reflexive and symetric, and γ and $\gamma[\gamma']$ have the same length. We obviously have :

$\gamma \uparrow \gamma_1;\gamma_2 \iff \gamma \uparrow \gamma_1 \;\&\; \gamma[\gamma_1] \uparrow \gamma_2$ and

$\gamma \uparrow \gamma_1;\gamma_2 \iff \gamma[\gamma_1;\gamma_2] = (\gamma[\gamma_1])[\gamma_2] \;\&$

$(\gamma_1;\gamma_2)[\gamma] = \gamma_1[\gamma] ; \gamma_2[\gamma[\gamma_1]]$

<u>Remark</u>: equivalent computations may be incompatible. For example if R is the TRS

r_1: k(x,a) -> c

r_2: b -> a

r_3: b -> c

then

$$\gamma_1: k(b,a) \xrightarrow[R]{(k_1,r_2)} k(a,a) \underset{R}{\xrightarrow{\varepsilon}} c \quad \text{and}$$

$$\gamma_2: k(b,a) \xrightarrow[R]{(k_1,r_3)} k(c,a) \underset{r}{\xrightarrow{\varepsilon}} c$$

are equivalent (and equivalent to $\gamma: k(b,a) \underset{r}{\xrightarrow{\varepsilon}} c$) but incompatible.

The permutation lemma 3-3 (second form)

$$\gamma \uparrow \gamma' \Rightarrow \gamma;\gamma'[\gamma] \equiv \gamma';\gamma[\gamma']$$

proof: trivial induction on the length of computations. We only have to check that $\sigma \uparrow \sigma' \Rightarrow \sigma;\sigma'[\sigma] \equiv \sigma';\sigma[\sigma']$, but these are two developments of $\sigma \cup \sigma'$ ∎

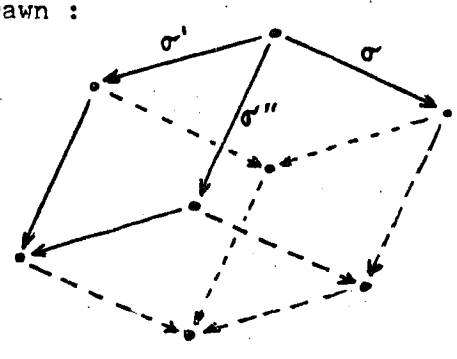
We now state without proof (straightforward, but rather tedious) another well-known fact about computations :

the cube lemma 3-4 (first form)

Let σ,σ',σ'' be consistent set of redexes (in t) pairwise compatible.

Then $\sigma \uparrow \sigma';\sigma''[\sigma']$ and $\sigma \uparrow \sigma'';\sigma'[\sigma'']$ & $\sigma[\sigma';\sigma''[\sigma']] = \sigma[\sigma'';\sigma'[\sigma'']]$

which can be drawn :



As a technical consequence of this property, we get :

corollary 3-1

let σ and σ' be consistent sets of redexes

$$\sigma \uparrow \sigma' \Rightarrow \text{for all development } \alpha \text{ of } \sigma : \sigma' \uparrow \alpha ,$$

$$\sigma'[\alpha] = \sigma'[\sigma] \;\&\; \alpha[\sigma'] \equiv \sigma[\sigma']$$

proof: induction on $|\alpha|$

(i) The property is obvious if $|\alpha| = 0$ ($\Leftrightarrow \sigma = \emptyset$)

(ii) $\alpha = \sigma'' : \alpha'$, where σ'' is a non empty subset of σ and α' a development of $\sigma[\sigma'']$.

Since $\sigma \uparrow \sigma'$ and $\sigma'' \subseteq \sigma$, obviously $\sigma' \uparrow \sigma''$ and

$$\sigma'[\sigma''] \cup \sigma[\sigma''] = (\sigma \cup \sigma')[\sigma''] \Rightarrow \sigma'[\sigma''] \uparrow \sigma[\sigma'']$$

thus by induction hypothesis $\sigma'[\sigma''] \uparrow \alpha'$, whence $\sigma' \uparrow \alpha$

By induction hypothesis :

$(\sigma'[\sigma''])[\alpha'] = (\sigma'[\sigma''])[\sigma[\sigma'']]$ thus

$$\sigma'[\alpha] = (\sigma'[\sigma''])[\sigma[\sigma'']] = \sigma'[\sigma'' : \sigma[\sigma'']]$$

But by the cube lemma 3-4 :

$\sigma'[\alpha] = \sigma'[\sigma]$ (since $\sigma''[\sigma] = \emptyset$)

We have : $\alpha[\sigma'] = \sigma''[\sigma'] : \alpha'[\sigma'[\sigma'']]$

and by the induction hypothesis :

$\alpha'[\sigma'[\sigma'']] \equiv (\sigma[\sigma''])[\sigma'[\sigma'']]$

But $(\sigma[\sigma''])[\sigma'[\sigma'']] = \sigma[\sigma'':\sigma'[\sigma'']]$

and by the cube lemma 3-4 :

$(\sigma[\sigma''])[\sigma'[\sigma'']] = \sigma[\sigma':\sigma''[\sigma']] = (\sigma[\sigma'])[\sigma''[\sigma']]$.

If $\sigma''[\sigma'] = \emptyset$ then

$\alpha[\sigma'] = \sigma''[\sigma']:\alpha'[\sigma'[\sigma'']] \equiv \alpha'[\sigma'[\sigma'']] \equiv (\sigma[\sigma''])[\sigma'[\sigma'']] = \sigma[\sigma']$

Otherwise

$\alpha[\sigma'] = \sigma''[\sigma'];\alpha'[\sigma'[\sigma'']] \equiv \sigma''[\sigma'];(\sigma[\sigma''])[\sigma'[\sigma'']] =$

$\sigma''[\sigma'];(\sigma[\sigma'])[\sigma''[\sigma']]$

but $\sigma''[\sigma'];(\sigma[\sigma'])[\sigma''[\sigma']]$ is here a development of $\sigma[\sigma']$ ▬

We can infer a second form of the cube lemma, if we define :

δ is <u>strongly compatible</u> with γ , $\delta \Uparrow \gamma \iff \forall \gamma' : \gamma' \equiv \gamma \Rightarrow \delta \uparrow \gamma'$

<u>the cube lemma</u> 3-5 (second form)

<u>if</u> , for σ consistent set of redexes and γ finite computation

$\sigma \Uparrow \gamma$ <u>then</u> $\gamma' \equiv \gamma \Rightarrow \sigma[\gamma'] = \sigma[\gamma]$ & $\gamma'[\sigma] \equiv \gamma[\sigma]$

<u>proof</u>: by transitivity ($\equiv = \approx^+$) it suffices to prove that for $\gamma \approx \gamma'$, and this is an easy consequence of corollary 3-1 ▬

Let us now define the external occurrences for a computation of [23] :

For $\gamma \in \mathcal{C}_p(t)$, $w \in occ(t)$ is <u>external</u> for γ , $w \in ext_R(\gamma)$ iff

(i) $\gamma = \mathcal{E}$ or

(ii) $\gamma = \sigma : \gamma'$ and

(iii) $u \in dom(\sigma)$ & $u \leq_{pref} w \Rightarrow \exists v \in int(lhs(\sigma(u))) : w = uv$

(iv) $w\langle\sigma\rangle \subseteq ext_R(\gamma')$

(external occurrences cannot be covered by a redex of the first step, and their traces remain external).

Some obvious facts about external occurrences for a computation are :

- if $w \in ext_R(\gamma)$ => and $u \leq_{pref} w$ =>$u \in ext_R(\gamma)$

- $w \in ext_R(\gamma)$ => $\exists u \leq_{pref} w : w\langle\gamma\rangle = \{u\}$

(external occurrences for a computation cannot disappear along this computation).

- $ext_R(\gamma_1 ; \gamma_2) = \{w / w \in ext_R(\gamma_1)$ & $w\langle\gamma_1\rangle \subseteq ext_R(\gamma_2)\}$

thus $ext_R(\gamma_1 ; \gamma_2) \subseteq ext_R(\gamma_1)$

External occurrences are also preserved by permutations :

<u>lemma</u> 3-6

$$\gamma \equiv \gamma' \Rightarrow ext_R(\gamma) = ext_R(\gamma')$$

<u>proof</u> : by transitivity, we only have to show this when $\gamma \approx \gamma'$ and in this case, by means of the above remarks, we only have to prove : <u>if</u> α

is a development of σ <u>then</u> $ext_R(\alpha) = ext_R(\sigma)$

by induction on $|\alpha|$ and by cases, straightforward ∎

The initial redexes of a computation are the redexes of t applied by γ (possibly not at the first step) :

For $\gamma \in \mathcal{C}_R(t)$, $(u,r) \in red_R(t)$

is an <u>initial redex</u> of γ

$(u,r) \in initred_R(\gamma)$

<u>iff</u>

$\gamma = \sigma;\gamma'$ and

(i) $(u,r) \in \sigma$ or

(ii) $(u,r) \uparrow \sigma$, $(u,r)[\sigma] \neq \emptyset$ and $(u,r)[\sigma] \subseteq initred_R(\gamma')$

One may verify that $initred_R(\gamma)$ is a consistent set of redexes of t such that

$$initred_R(\gamma) \uparrow \gamma \ \& \ \gamma \equiv initred_R(\gamma) ; \gamma[initred_R(\gamma)]$$

(for $initred_R(\gamma)[\gamma] = \emptyset$, whence the terminology) and for all development α of σ :

$$initred_R(\alpha) \subseteq \sigma$$

(for R is left-linear)

Now we may call <u>external redexes</u> of a computation its initial redexes at an external occurrence :

$$extred_R(\gamma) = \{(u,r)/(u,r) \in initred_R(\gamma) \ \& \ u \in ext_R(\gamma)\}$$

In fact $extred_R(\gamma)$ (which obviously is a consistent set of redexes) is strongly compatible with γ and

$$\gamma \equiv extred_R(\gamma);\gamma[extred_R(\gamma)]$$

One may also show that :

$(u,r) \in extred_R(\gamma_1:\gamma_2)$ <=>

(i) $(u,r) \in extred_R(\gamma_1)$ and $u\langle\gamma_1\rangle \subseteq ext_R(\gamma_2)$ or

(ii) $(u,r) \uparrow \gamma_1$, $u \in ext_R(\gamma_1)$ and $(u,r)[\gamma] \subseteq extred_R(\gamma_2)$

From this fact it follows :

lemma 3-7

$$\gamma \equiv \gamma' \Rightarrow extred_R(\gamma) = extred_R(\gamma')$$

proof: by transitivity, we have to check this for $\gamma \approx \gamma'$ and for this to establish

if α is a development of σ then $extred_R(\alpha) = extred_R(\sigma)$

which is easy, from what have been asserted up to this point ∎

lemma 3-8

$$||\gamma|| = o \Leftrightarrow extred_R(\gamma) = \emptyset$$

proof: obviously

$$||\gamma|| = o \Rightarrow initred_R(\gamma) = \emptyset \Rightarrow extred_R(\gamma) = \emptyset$$

To prove the converse implication we can suppose, by cancelling the empty steps (which preserve the size and equivalence class), that

$\gamma = (\sigma_i)_{i<k}$ where $k>o$ and $\forall i<k : \sigma_i \neq \emptyset$

Then $initred_R(\gamma) \neq \emptyset$ since $\sigma_o \subseteq initred_R(\gamma)$

and we could prove, by induction on k that if u is the first w.r.t. $\leq_{lex}$ element of $dom(initred_R(\gamma))$ then $u \in ext_R(\gamma)$ ∎

As a corollary, we may assert that :

$$\delta;\delta' \equiv \varepsilon \Rightarrow \delta \equiv \delta' \equiv \varepsilon$$

lemma 3-9

let $\rhd$ be the relation given by :

$$\gamma \rhd \gamma' <=>_{def} \exists (u,r) \in extred_R(\gamma) : \gamma' = \gamma[u,r]$$

Then $\rhd$ is noetherian (that is there is no infinite sequence of computations $(\gamma_n)_{n \in \mathbb{N}}$ s.t. $\forall n \in \mathbb{N} : \gamma_n \rhd \gamma_{n+1}$)

proof : let us first remark that (from lemma 3-8) if $\exists \gamma' : \gamma \rhd \gamma'$ then $|\gamma'|=|\gamma|>o$. We prove that

$$\gamma \rhd \gamma' \Rightarrow \underline{n}(\gamma') <_{lex} \underline{n}(\gamma)$$

where, if $k=|\gamma|$, $<_{lex}$ is the strict lexicographical order on $\mathbb{N}^k$ and, for $\delta = (\sigma_i)_{i<k}$

$$\underline{n}(\delta) = (\underline{n}(\sigma_{k-1}),\ldots,\underline{n}(\sigma_o))$$

where, for σ consistent set of redexes

$$\underline{n}(\sigma) = |dom(\sigma)<\sigma>|$$

It can be checked that $\gamma = \sigma_o:\ldots:\sigma_{k-1}$ and $\gamma \rhd \gamma'$ implies

$$\exists i<k\ \exists (u,r) \in \sigma_i \quad \text{(and in fact } (u,r) \in \text{extred}_R(\gamma))$$

$$\gamma' = \sigma'_i : \ldots : \sigma_j[u,r] : \ldots : \sigma_{k-1}$$

and that

$$(u,r) \in \sigma \ \&\ (u,r)\langle\sigma\rangle \neq \emptyset \Rightarrow \underline{n}(\sigma[u,r]) < \underline{n}(\sigma) \quad \blacksquare$$

From all these technical results, we finally get the announced

simplification lemma 3-10 (first form)

$$\delta;\gamma \equiv \delta;\gamma' \Rightarrow \gamma \equiv \gamma'$$

proof: by induction on $|\delta|$: it suffice to prove this for $|\delta|=1$ (one step computation, $\delta = \sigma$), and here we proceed by noetherian induction on $\rhd$ (see [21]) :

- if there is no δ s.t. $\sigma;\gamma \rhd \delta$ then, by lemma 3-8

$$||\sigma;\gamma|| = o \text{ thus } (\sigma=\emptyset)\ \gamma \equiv \gamma' \text{ trivially.}$$

- otherwise, let $(u,r) \in \text{extred}_R(\sigma;\gamma)$. Since $(u,r) \Uparrow \sigma;\gamma$, by the permutation lemma 3-3

$$\sigma;\gamma \equiv (u,r) : (\sigma;\gamma)[u,r] \equiv (u,r):(\sigma;\gamma')[u,r] \equiv \sigma;\gamma'$$

and by the cube lemma 3-5

$$\sigma[u,r];\ \gamma[(u,r)[\sigma]] = (\sigma;r)[u,r] \equiv (\sigma;\gamma')[u,r] = \sigma[u,r];\gamma'[(u,r)[\sigma]]$$

Since $\sigma;\gamma \rhd (\sigma;\gamma)[u,r]$, by induction hypothesis :

$$\gamma[(u,r)[\sigma]] \equiv \gamma'[(u,r)[\sigma]]$$

..if $(u,r) \in \sigma$ then $(u,r)[\sigma] = \emptyset$ and thus we have $\gamma \equiv \gamma'$

..if $(u,r) \notin \sigma$ then (since $(u,r) \in \text{extred}_R(\sigma;\gamma)$ $\Rightarrow (u,r)[\sigma] = (u,r)$ and $(u,r) \in \text{extred}_R(\gamma)$)

$\gamma \equiv (u,r)\ ;\ \gamma[u,r] = (u,r)\ ;\ \gamma[(u,r)[\sigma]]$

Since (lemma 3-7) $\text{extred}_R(\sigma;\gamma) = \text{extred}_R(\sigma;\gamma')$ we also have :

$\gamma' \equiv (u,r);\gamma'[u,r] = (u,r);\gamma'[(u,r)[\sigma]]$

Thus, from (induction hypothesis) $\gamma[(u,r)[\sigma]] \equiv \gamma'[(u,r)[\sigma]]$:

$\gamma \equiv \gamma'$ ∎

This property will be of constant (and often implicit) use in the following. We may remark that the technical developments of this paragraph has another consequence : we may find (exactly as in [23]) in any equivalence class of finite computation a canonical representative, which is the standard (external) form of the computations of this class. Let us say that :

γ is <u>standard</u> <u>iff</u>

$\gamma = \varepsilon$ or $\gamma = \text{extred}_R(\gamma);\gamma'$ where γ' is standard

Then one can easily prove (by noetherian induction, routine) the

<u>standardization theorem</u> :

For all $\gamma \in \mathcal{G}_R(t)$ there exists a (unique) standard computation

$\overline{\gamma} \in \mathcal{G}_R(t)$ s.t. $\gamma \equiv \overline{\gamma}$, $\gamma \equiv \gamma' \Leftrightarrow \overline{\gamma} = \overline{\gamma'}$ (and $\overline{\overline{\gamma}} = \overline{\gamma}$)

This result allows a reformulation of the property of left-simplifiability of the equivalence by permutations :

<u>the simplification lemma</u> 3-11 (second form)

$\gamma \equiv \gamma' \ \&\ \gamma;\delta \equiv \gamma';\delta' \Rightarrow \delta \equiv \delta'$

3-4 The ordered space of computations

Our aim in this paragraph is to define and study an ordering relation on computations which formalizes the intuitive idea that a computation is greater than another if it remains nothing of the second after it. As we already have pointed out however, we cannot exactly take this way, that is by means of residuals as in [4,27,23] since our TRS's are not necessarily deterministic : two computations may be incompatible. But we can rephrase the intuitive idea : a computation γ' is greater than γ if it makes the same rewritings as γ and perhaps other ones, up to permutations, or else : γ' begins by γ , up to permutations.

Formally :

For finite computations γ, γ' in $\mathcal{C}_R(t)$:

(i) $\gamma \sqsubseteq \gamma' \iff_{def} \exists \delta : \gamma' = \gamma ; \delta$

(γ is an initial restriction, or a prefix of γ')

(ii) $\gamma \leq \gamma' \iff_{def} \exists \delta : \gamma \sqsubseteq \delta \ \& \ \delta \equiv \gamma'$

(γ is a subcomputation of γ')

Example 3-1

Let R be given by

r_1 : h(a) -> b r_2 : h(a) -> c

r_3 : k(x,a) -> b r_4 : k(x,a) -> c

γ_1 : $k(h(a),a) \xrightarrow[R]{(k_1,r_1)} k(b,a)$ γ_2 : $k(h(a),a) \xrightarrow[R]{(k_1,r_2)} k(c,a)$

$\delta_1 : k(h(a),a) \xrightarrow[R]{(\varepsilon,r_3)} b$ $\qquad\qquad$ $\delta_2 : k(h(a),a) \xrightarrow[R]{(\varepsilon,r_4)} c$

then $\gamma_i \leq \delta_j$ for $i,j \in \{1,2\}$

Some obvious properties of these relations are expressed in

lemma 3-12

(i) $\sqsubseteq$ is a partial order

(ii) $\leq$ is a preorder such that :

- $\leq$ contains $\sqsubseteq$: $\gamma \sqsubseteq \gamma' \Rightarrow \gamma \leq \gamma'$

- $\leq$ is compatible with composition : $\gamma \leq \gamma' \Rightarrow \delta;\gamma \leq \delta;\gamma'$

- the equivalence associated with $\leq$ is $\equiv$: $\gamma \equiv \gamma' \Leftrightarrow \gamma \leq \gamma' \;\&\; \gamma' \leq$

(for the last point we use the simplification lemma and $\delta;\delta' \equiv \varepsilon \Rightarrow \delta \equiv \delta' \equiv \varepsilon$)

Thus the quotient $\mathcal{C}_R(t)/\equiv$ is ordered by the quotient order for which we keep the same notation $\leq$. Let us recall that if R is deterministic then for γ,γ' in $\mathcal{C}_R(t)$ $\gamma \uparrow \gamma'$ is allways true and thus γ and γ' are compatible w.r.t. $\leq$ since

$$\gamma \leq \gamma ; \gamma'[\gamma] \quad \text{and} \quad \gamma' \leq \gamma ; \gamma'[\gamma]$$

Moreover in this case we have the equivalent definition of $\leq$:

$$\gamma \leq \gamma' \Leftrightarrow \gamma[\gamma'] \equiv \varepsilon$$

Thus $\mathcal{C}_R(t)/\equiv$ is, when R is deterministic, an upper semi-lattice: the least upper bound of the classes of γ_1 and γ_2 is the class of $\gamma_1;\gamma_2[\gamma_1]$

This is the result of [23].

Now we extend these relations to the infinite case. We keep the same notation $\sqsubseteq$ for the relation "is a prefix of" on infinite computations :

(i) For $\gamma, \gamma' \in \mathcal{C}^{\infty}_{R}(t)$:

$$\gamma \sqsubseteq \gamma' \iff_{def} \gamma' = \gamma \text{ or } \gamma \in \mathcal{C}_{R}(t) \;\&\; \exists \delta : \gamma' = \gamma;\delta$$

(ie : $\exists k : \gamma = \gamma'|^{k}$)

This is obviously a partial order on $\mathcal{C}^{\infty}_{R}(t)$ for which ε is the least element and s.t. $< \mathcal{C}^{\infty}_{R}(t), \sqsubseteq, \varepsilon >$ is a complete partial order, in fact a completion of $< \mathcal{C}_{R}(t), \sqsubseteq, \varepsilon >$. This allows to define an algebraic extension of $\leq$ (w.r.t. $\sqsubseteq$) on infinite computations : $\gamma \leq^{\infty} \gamma'$ if for each finite approximation - w.r.t. $\sqsubseteq$ - of γ, there is a greater - w.r.t. $\leq$ - finite approximation of γ' :

(ii) For $\gamma, \gamma' \in \mathcal{C}^{\infty}_{R}(t)$:

$$\gamma \leq^{\infty} \gamma' \iff_{def} \forall \delta \in \mathcal{C}_{R}(t) : \delta \sqsubseteq \gamma \Rightarrow \exists \delta' \in \mathcal{C}_{R}(t) : \delta' \sqsubseteq \gamma' \;\&\; \delta \leq \delta'$$

Example 3-2

Let R (deterministic) given by

r_1 : a -> h(a)

r_2 : k(x,b) -> c

Then, if for $n \in \mathbb{N}$ we let $t_n = k(h^n(a),b)$ and $(\gamma_n)_{n \in \mathbb{N}}$ is the ($\sqsubseteq$-increasing) sequence of computations of t_o :

$\gamma_o = \varepsilon$

$\gamma_{n+1} = \gamma_n : (k_1 h_1^n, r_1)$

if γ is the lub (w.r.t. $\sqsubseteq$) of $(\gamma_n)_{n \in \mathbb{N}}$, that is

$$\gamma : k(a,b) \xrightarrow[R]{} k(h(a),b) \to \ldots \to k(h^n(a),b) \xrightarrow[R]{} k(h^{n+1}(a),b) \to \ldots$$

we have $\gamma \leq^\infty \delta$ where

$$\delta : k(a,b) \xrightarrow[R]{\varepsilon} c$$

since for each n : $\gamma_n \leq \delta$

Clearly $\leq^\infty$ is a preorder which contains $\sqsubseteq$, whose restriction to finite computations is $\leq$. Relating $\leq^\infty$ to composition, we can assert that :

$$\gamma \equiv \gamma' \;\&\; \delta \leq^\infty \delta' \Rightarrow \gamma;\delta \leq^\infty \gamma';\delta' \quad (\gamma,\gamma' \text{ finite})$$

and give the last formulation of the simplification property :

simplification lemma 3-13 (third form)

$$\gamma \equiv \gamma' \;\&\; \gamma;\delta \leq^\infty \gamma';\delta' \Rightarrow \delta \leq^\infty \delta'$$

We shall denote by $\equiv^\infty$ the equivalence relation associated with $\leq^\infty$ and keep the same notation for the quotient order. Our main result about $(\mathcal{C}^\infty_R(t)/\equiv^\infty, \leq^\infty)$ is that it is complete. An intuitive way to see that is to consider an increasing (w.r.t. $\leq$) sequence $(\gamma_n)_{n \in \mathbb{N}}$ of finite computations; we build the lub of this sequence, up to $\equiv^\infty$, as follows : we choose an increasing sequence (w.r.t. $\sqsubseteq$) $(\overline{\gamma}_n)_{n \in \mathbb{N}}$ of finite computations such that $\forall n \in \mathbb{N} : \overline{\gamma}_n \equiv \gamma_n$

Such a sequence exists by definition of $\leq = \sqsubseteq \circ \equiv$, which contains $\equiv \circ \leq$ $(\overline{\gamma}_n \equiv \gamma_n \leq \gamma_{n+1} \Rightarrow \exists\, \overline{\gamma}_{n+1} : \overline{\gamma}_n \sqsubseteq \overline{\gamma}_{n+1} \equiv \gamma_{n+1})$

Then the lub (for $\sqsubseteq$) of the sequence $(\overline{\gamma}_n)_{n \in \mathbb{N}}$ is, up to $\equiv^\infty$, a lub (w.r.t. $\leq^\infty$) of $(\gamma_n)_{n \in \mathbb{N}}$

Formally, if we denote the equivalence class of a computation (resp. finite computation) γ w.r.t. $\equiv^\infty$ (resp. w.r.t. $\equiv$) by $\ulcorner\gamma\urcorner^\infty$ (resp. $\ulcorner\gamma\urcorner$) :

Theorem 3-2

the partial order $\langle \mathcal{C}_R^\infty(t)/\equiv^\infty , \leq^\infty , \ulcorner\varepsilon\urcorner^\infty \rangle$ is complete :

it is a completion of $\langle \mathcal{C}_R(t)/\equiv , \leq , \ulcorner\varepsilon\urcorner \rangle$. Moreover, if R is deterministic, it is a complete lattice.

Proof: let us first remark that $\langle \mathcal{C}_R^\infty(t) , \sqsubseteq , \varepsilon \rangle$ is algebraic, of denumerable base $\mathcal{C}_R(t)$. This simply means that, w.r.t. $\sqsubseteq$, any computation is the lub of the increasing sequence of its finite prefixes.

Obviously $\ulcorner\varepsilon\urcorner$ is the least element of $\langle \mathcal{C}_R(t)/\equiv , \leq \rangle$.

Let ψ be the mapping from $\mathcal{C}_R^\infty(t)$ to the power set of $\mathcal{C}_R(t)/\equiv$ given by

$$\psi(\gamma) = \{\ulcorner\delta\urcorner \;/\; \delta \in \mathcal{C}_R(t) \;\&\; \delta \leq^\infty \gamma\}$$

One easily verifies that :

(i) for all γ, $\psi(\gamma)$ is an ideal and $\psi(\varepsilon) = \{\ulcorner\varepsilon\urcorner\}$

(ii) for all γ, γ' : $\gamma \leq^\infty \gamma' \iff \psi(\gamma) \subseteq \psi(\gamma')$ (this is the definition of $\leq^\infty$).

Thus we can define :

$$\varphi : \mathcal{C}_R^\infty(t)/\equiv^\infty \;\to\; (\mathcal{C}_R(t)/\equiv)^\infty$$

by $\varphi(\ulcorner\gamma\urcorner^\infty) = \psi(\gamma)$

and this mapping is such that

$$\forall e\ ,\ e' \in \mathcal{G}_R^{\infty}(t)/\equiv^{\infty} \quad : e \leq^{\infty} e' \iff \varphi(e) \subseteq \varphi(e')$$

Thus φ is increasing and injective. To establish that it is the desired isomorphism, it only remains to show that it is surjective.

Let $Y \in (\mathcal{G}_R(t)/\equiv)^{\infty}$ and $\overline{Y} = \{\gamma \ / \ \ulcorner\gamma\urcorner \in Y\}$

We want to exhibit $\delta \in \mathcal{G}_R^{\infty}(t)$ such that $\varphi(\delta) = Y$. For γ and γ' in $\mathcal{G}_R(t)$ we let

$$f(\gamma,\gamma') = \{\gamma'' \ / \ \gamma'' \in \overline{Y} \ \& \ \gamma \subseteq \gamma'' \ \& \ \gamma' \leq \gamma''\}$$

Then $\gamma, \gamma' \in \overline{Y} \Rightarrow f(\gamma,\gamma') \neq \emptyset$ since Y is an ideal

Now let $(\gamma_n)_{n \in \mathbb{N}}$ be an enumeration of $\overline{Y}$ and h a choice function over $\mathcal{G}_R(t)$ (which for each non-empty subset gives one of its elements). We define :

$$\overline{\gamma}_o = \gamma_o$$

$$\overline{\gamma}_{n+1} = h(f(\overline{\gamma}_n, \gamma_{n+1}))$$

It is easy to verify that :

(i) $\forall n \in \mathbb{N} \quad \overline{\gamma}_n \in \overline{Y}$ and $\gamma_n \leq \overline{\gamma}_n$

(ii) $\forall n \in \mathbb{N} \quad \overline{\gamma}_n \subseteq \overline{\gamma}_{n+1}$

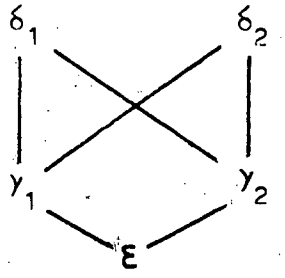
Thus, if δ is the limit (for $\subseteq$) of the increasing sequence $(\overline{\gamma}_n)_{n \in \mathbb{N}}$

then $\psi(\delta) = \gamma$ ∎

We cannot say much more about this order structure, which seems to generally have no "good" properties (even in deterministic TRS's)

Example 3-1 (continued)

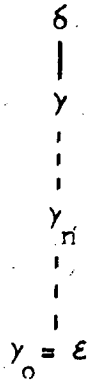
For $\mathcal{C}_R^{\infty}(t)$ where $t = k(h(a),a)$ we get



in which two compatible points (w.r.t. the order, not in the technical sense of $\uparrow$) do not have a lub.

Example 3-2 (continued)

Here $\mathcal{C}_R^{\infty}(t)$ where $t = k(a,b)$ is



where a finite computation is greater than an infinite one (that is greater than an infinite number of finite ones). Thus we do not investigate further the properties of the partial order $\leq^{\infty}$, but merely

indicates the only consequence of the theorem that we shall use :

since $\mathcal{C}_R^\infty(t)$ is complete, it is inductive (every $\leq^\infty$ chain is bounded) thus by Zorn's lemma, every computation is bounded by a maximal one :

Corollary 3-2

For all $\gamma \in \mathcal{C}_R^\infty(t)$ there exists $\gamma' \in \mathcal{C}_R^\infty(t)$ such that γ' is $\leq^\infty$ maximal (ie $\gamma' \leq^\infty \delta \Rightarrow \delta \equiv^\infty \gamma'$) and $\gamma \leq^\infty \gamma'$

From our intuitive point of view of $\leq^\infty$, we can say that a computation is maximal if "it remains nothing to do after it" (at least this is the exact formulation in the deterministic case). This is why we call terminating the maximal computations, and we denote by $\overline{\mathcal{C}}_R^\infty(t)$ the set of such computations (of the term t). We must say that finite termination is just what is usually meant :

for $\gamma : t \xrightarrow[R]{*} t'$, γ is maximal iff t' is a normal form (that is $red_R(t') = \emptyset$, or equivalently $\mathcal{C}_R^\infty(t') = \ulcorner\mathcal{E}\urcorner^\infty$)

One may use the third form of the simplification lemma to see this. It is less obvious to get an intuition about infinite terminating computations.

Example

Let R (non-ambiguous) be given by

$\varphi(x)$ -> $f(x,\varphi(x))$

ψ -> a

ψ -> b

Then

$$\varphi(\psi) \to f(\psi,\varphi(\psi)) \to f(a,f(\psi,\varphi(\psi))) \to f(a,\ldots,f(a,\varphi(\psi))\ldots) \to \ldots$$

is not maximal. It is strictly majorated by

$$\varphi(\psi) \to \varphi(a) \to f(a,\varphi(a)) \to f(a,f(a,\varphi(a))) \to \ldots$$

which is maximal. On the other hand :

$$\varphi(\psi) \to f(\psi,\varphi(\psi)) \to f(a,f(\psi,\varphi(\psi))) \to f(a,f(b,f(\psi,\varphi(\psi)))) \to \ldots$$

(where we alternatively use the rules $\psi \to a$ and $\psi \to b$) is terminating.

We may also note that if R is deterministic, then there is only one (up to $\equiv^{\infty}$) terminating computation of t. We can describe it as $K_R(t) = (\sigma_n)_{n \in \mathbb{N}}$, the <u>full</u> computation of t (we recapture here the case of recursive definitions) given by :

$$t = t_o \xrightarrow[R]{\sigma_o} t_1 \to \ldots \to t_n \xrightarrow[R]{\sigma_n} t_{n+1} \to \ldots$$

where $\forall i \in \mathbb{N} \quad \sigma_i = red_R(t_i)$.

In this case $\overline{\mathcal{C}}^{\infty}_R(t) = \ulcorner K_R(t) \urcorner^{\infty}$

4. Semantics

In order to define the semantics of a term in a TRS, we have to define what can be the result of its computations. But computations are symbolic manipulations, thus we must define interpretations (of a TRS). As a preliminary step, let us revisit algebras.

4-1 Equational classes of algebras : the factorization theorem

In paragraph 2-2, we have seen that the set $M_F^\infty(X)$ of trees is "the" free complete F-algebra generated by X, and later we have said that, since X is a set of variables, our terms are expressions denoting functions. This may be asserted in fact for arbitrary trees :

if $A = \langle D, \sqsubseteq, \bot, \{f_A / f \in F\} \rangle$ is a

complete F-algebra, any mapping $e : X \to D$ may be viewed as an evaluation of variables in A. Since H is free, such an evaluation uniquely extends in a morphism $e^\infty : M_F^\infty(X) \to D$. Thus we can now consider a tree $t \in M_F^\infty(X)$ as a mapping $t_A : D^X \to D$ which, for any evaluation of the variables gives a value (the value of the "expression" t) : $t_A(e) = e^\infty(t)$. This mapping is the interpretation of t in the algebra A, and we abstractly define the semantics (of the free language given by F,X, for which $M_F^\infty(X)$ is the abstract syntax, see [1]) as the mapping

$$\lambda_A : M_F^\infty(X) \to D^{(D^X)} \quad \text{given by } \lambda_A(t) = t_A$$

In fact, we can say more about the image of λ_A :

lemma 4-1

For each $t \in M_F^\infty(X)$ t_A is continuous

proof: by structural induction on $M_F(X)$ and continuity argument, omitted.

Thus λ_A is actually a mapping from $M_F^\infty(X)$ to $(D^X \to D)$.

This last is also the domain of an algebra, if we define

for $f \in F_k$, $\hat{f}_A : (D^X \to D)^k \to (D^X \to D)$ by

$$\hat{f}_A(h_1,\ldots,h_k)(e) = f_A(h_1(e),\ldots,h_k(e))$$

and we may now precise the properties of λ_A :

proposition 4-1

For all complete F-algebra $A = \langle D, \sqsubseteq, \bot, \{f_A \mid f \in F\}\rangle$, the

structure $\hat{A} = \langle (D^X \to D), \sqsubseteq, \bot, \{\hat{f}_A \mid f \in F\}\rangle$ is a complete F-algebra,

and λ_A is a morphism from H to $\hat{A}$

(the proof is straightforward).

The new point of view about semantics is thus now : $M_F^\infty(X)$ is the abstract syntax, an interpretation is a complete F-algebra A and the semantics under this interpretation is λ_A, thus we actually interpret in the ("second order") structure $\hat{A}$ determined by A. A question is : is there any "initial object" w.r.t. this definition ? We shall answer (affirmatively) to a more general question, by giving a slight modification of the central result of the so-called "algebraic semantics" due to I. Guessarian (see [13,17]). To express formally the problem, let us give some definitions :

a tree t' is semantically greater than t, under the interpretation A,

$$t \leq_A t' \iff_{def} t_A \sqsubseteq t'_A$$

This definition can be extended to classes of interpretations, if $\mathcal{A}$ is such a class :

$$t \leq_{\mathcal{A}} t' \iff_{def} \forall A \in \mathcal{A} : t \leq_A t'$$

These preorder $\leq_{\mathcal{A}}$ are continuous precongruences containing $\leq$, see [14].

Given an interpretation A, we denote by $\mathcal{F}_A$ the canonical structure of ordered F-algebra on $M_F^\infty(X) / \leq_A$. This structure is obviously isomorphic to $\langle \lambda_A(M_F^\infty(X)), \sqsubseteq, \bot, \{\hat{f}_A / f \in F\}\rangle$; thus we can say that its domain is the domain of definable functions (from D^X to D), and consider that λ_A is a morphism from H to $\mathcal{F}_A$ (remark : in general $\mathcal{F}_A$ is not complete, see [17]). We are now ready to give the definition of "initial objects" :

> let $\mathcal{A}$ be a class of interpretations; a complete F-algebra A is said to be $\mathcal{A}$-universal iff for all $A' \in \mathcal{A}$ there exists one and only one morphism (of ordered F-algebras)
>
> $$\lambda_{\langle A,A'\rangle} \text{ from } \mathcal{F}_A \text{ to } \mathcal{F}_{A'} \text{ s.t. } \lambda_{A'} = \lambda_{\langle A,A'\rangle} \circ \lambda_A$$

We could have said "$\mathcal{A}$-initial" or "$\mathcal{A}$-free" instead of $\mathcal{A}$-universal, but this definition is slightly different from what is usually found in the litterature [1,32,13,17], where one devises more on algebras of objects than on algebras of functions. We could also have said that A factorizes $\mathcal{A}$, since A $\mathcal{A}$-universal means that the semantics in every interpretation in $\mathcal{A}$ is factorized, univocally and homomorphically, by the semantics in A.

We now show the existence of such objects for equational classes of algebras. To restrict ourselves to such classes correspond to the idea that our abstract syntax is too abstract : we often have to assume some properties of the functions in F (given by an interpreter, for example). Here these properties are expressed as first-order equational theories, that is simply subsets S of $M_F(X) \times M_F(X)$, written as equalities, and the class of their model are equational classes of interpretations $\mathcal{A}_S$:

$$A \in \mathcal{A}_S \iff_{def} (t=t') \in S \Rightarrow t_A = t'_A$$

To build an $\mathcal{A}_S$-universal interpretation, let us define $\preceq_S$ as the reflexive and transitive closure of $\preceq \cup \overleftrightarrow{S}$ (thus $\preceq_S$ is a precongruence containing $\preceq$ and S), and denote by π_S the canonical surjection from $M_F(X)$ to $M_F(X) / \preceq_S$ (which is an ordered F-algebra). The structure :

$$H_S = \langle (M_F(X)/\preceq_S)^\infty , \subseteq, \perp_S, \{f_S / f \in F\}\rangle$$

is given by :

- $\perp_S$ is the ideal generated by $\pi_S(\perp)$
- For $f \in F_k$ and $T_1,\ldots,T_k$ in $(M_F(X) / \preceq_S)^\infty$:

$$\pi_S(t) \in f_S(T_1,\ldots,T_k) \iff \exists (t_1,\ldots,t_k) \in T_1 \times \ldots \times T_k :$$

$$t \preceq_S f(t_1,\ldots,t_k)$$

The algebraic extension of $\preceq_S$ to infinite trees is :

$$\theta \preceq_S^\infty \theta' \iff \forall t \in M_F(X) : t \preceq \theta \Rightarrow \exists\, t' \in M_F(X) : t' \preceq \theta' \;\&\; t \preceq_S t'$$

Denoting by $\bar{\pi}_S$ the canonical surjection from $M_F^\infty(X)$ to $M_F^\infty(X) / \preceq_S^\infty$ we get

the factorization theorem 4-1

For all relation $S \subseteq M_F(X) \times M_F(X)$, H_S is a complete F-algebra which is $\mathcal{A}_S$-universal. Moreover $\mathcal{F}_{H_S}$ is isomorphic to $M_F^\infty(X) / \preceq_S^\infty$

Since this result is only a variation of a classical one, we only sketch the proof, leaving the details to the reader.

(i) the first step is to establish that :

$$A \in \mathcal{A}_S \Rightarrow \preceq_S^\infty \subseteq \leq_A$$

from what we may infer that λ_A is factorized in a unique morphism (of ordered F-algebras) :

$$\psi_A : M_F^\infty(X) / \preceq_S^\infty \to \mathcal{F}_A$$

such that $\psi_A \circ \bar{\pi}_S = \lambda_A$

(ii) in a second step, one proves that $\preceq_S^\infty = \leq_{H_S}$ and thus λ_{H_S} factorizes in a unique isomorphism

$$\varphi_S : M_F^\infty(X) / \preceq_S^\infty \to \mathcal{F}_{H_S}$$

Now if we let $\lambda_{\langle H_S, A\rangle} = \psi_A \circ \varphi_S^{-1}$ we get the result ∎

Remarks : one has $H_S \in \mathcal{A}_S$, thus $\leq_{\mathcal{A}_S} = \leq_{H_S} = \preceq_S^\infty$ that is H_S is also $\mathcal{A}_S$-Herbrand (see [13,17]). We could have define the domain of H_S as the set of $\preceq_S$-ideals of $M_F(X)$, that is to say non-empty subsets T of $M_F(X)$ such that :

(i) $t_1 \in T \;\&\; t_2 \in T \Rightarrow \exists\, t \in T : t_1 \preceq_S t \;\&\; t_2 \preceq_S t$

(ii) $t \in T \;\&\; t' \preceq_S t \Rightarrow t' \in T$

One can try to improve this result, finding conditions on S under which we can describe more accurately H_S, for example as a quotient of $M_F^\infty(X)$. This has been done by B. Courcelle [15] with the concept of "monotone reduction" :

a subset S of $M_F(X) \times M_F(X)$ is a <u>reduction</u> iff S is a finite relation such that

(i) $\perp \notin dom(S)$

(ii) $\xrightarrow[S]{}$ is noetherian and confluent, which means (see [21])

(iii) there is no infinite sequence $(t_n)_{n \in \mathbb{N}}$ in $M_F(X)$ such that $\forall n \in \mathbb{N}\ t_n \xrightarrow[S]{} t_{n+1}$

(iv) if $t \xrightarrow[S]{*} t_1$ & $t \xrightarrow[S]{*} t_2$ then $\exists \bar{t} : t_1 \xrightarrow[S]{*} \bar{t}$ & $t_2 \xrightarrow[S]{*} \bar{t}$

<u>Remark</u> : since $\xrightarrow[S]{}$ is noetherian, we must have $dom(S) \cap X = \emptyset$, $(t,t') \in S \Rightarrow var(t') \subseteq var(t)$, and confluence is equivalent to local confluence ([21]) :

(iv)' if $t \xrightarrow[S]{} t_1$ & $t \xrightarrow[S]{} t_2$ then $\exists \bar{t} : t_1 \xrightarrow[S]{*} \bar{t}$ & $t_2 \xrightarrow[S]{*} \bar{t}$

It is well-known that if S is a reduction, then every finite tree $t \in M_F(X)$ has a unique <u>S-normal form</u> which we denote $\lambda_S(t)$ (with some consistence with the previous notation λ_A, as we shall see), that is an S-irreducible tree $\lambda_S(t)$ such that :

$$t \xrightarrow[S]{*} \lambda_S(t) \ \&\ t \xleftrightarrow[S]{*} t' \iff \lambda_S(t) = \lambda_S(t')$$

Now we say that (see [15])

a reduction S is <u>monotone</u> iff λ_S is increasing :

$$t \preceq t' \Rightarrow \lambda_S(t) \preceq \lambda_S(t')$$

In this case, one can extend by continuity λ_S to $M_F^\infty(X)$ in λ_S^∞ (with some ambiguity in the notation, since λ_S^∞ is the extension of <u>id</u> : X->X) :

$$\lambda_S^\infty(\theta) = \bigsqcup \{\lambda_S(t) \ /\ t \in M_F(X) \ \&\ t \preceq \theta\}$$

Trees of the form $\lambda_S^\infty(\Theta)$ are said to be in S-canonical form, and λ_S^∞ is a continuous projection. The result of B.Courcelle [15] is :

proposition 4-2

if S is a monotone reduction, then H_S is isomorphic to the canonical structures of F-algebras on $(M_F^\infty(X)/\preceq_S^\infty)$,

$(\lambda_S(M_F(X)))^\infty$ and $\lambda_S^\infty(M_F^\infty(X))$.

From the factorization theorem, we get the fact that the semantics of a tree in the equational class of interpretations of a monotone reduction is represented by a tree in canonical form, its image by λ_S^∞.

4-2 Computational semantics

To define the semantics of terms in a TRS R, we have to decide what is an interpretation for such a specification. In our mind, a TRS is intended to define the denotation of some terms (e.g. the left-hand sides of the rules) which are to be computed. Thus we assume that an algebra A is an interpretation of R if it takes no decision, or more technically gives no information about the value of these terms. Following the ideas of D. Scott [40] this means that in these algebras the denotation of the (prefix of) terms which are to be computed is the least element :

an interpretation of a TRS R (on the set F of function symbols) is a complete F-algebra A s.t. for all rule t -> t'in R : $t_A = \bot$

Thus the class $\mathcal{J}_R$ of interpretation of R is an equational one.

Remark: if we should have allow TRS R such that $dom(R) \cap X \neq \emptyset$ then for such an R , an interpretation would have been trivial (ie with domain {⊥}).

Let us immediatly note a consequence of this definition : if t->t' is a rule of R, then under any interpretation A of R we shall obviously have $t_A \sqsubseteq t'_A$ and, since $\leq_A$ is a precongruence, the information given by

λ_A increase along computations (see [311) :

$$\underline{\text{if}}\ t \xrightarrow[R]{*} t' \ \ \underline{\text{then}}\ t_A \sqsubseteq t'_A$$

thus we can define the result, under the interpretation A, of a computation of a term t

$$\gamma : t = t_o \xrightarrow[R]{\sigma_o} t_1 \to \ldots \to t_n \xrightarrow[R]{\sigma_n} t_{n+1} \to \ldots$$

as the lub of the increasing partial information that we get along the computation :

$$res_A(\gamma) = \bigsqcup \{\lambda_A(t_n) \ / \ n \in \mathbb{N}\}$$

which is an element of $(D^X \to D)$ if D is the domain of A. Obviously :

$$\gamma \leq^{\infty} \gamma' \Rightarrow res_A(\gamma) \sqsubseteq res_A(\gamma')$$

and thus res_A is also defined (and we keep the same notation) on $\mathcal{C}^{\infty}_R(t) / \equiv^{\infty}$, and it is straightforward to verify that :

$$res_A : \mathcal{C}^{\infty}_R(t) / \equiv^{\infty} \ \to (D^X \to D)$$

is continuous.

The computational semantics of a term t in a TRS R under an interpretation A is then the relation (since R is non-deterministic) obtained as the union of the functions which are the results of terminating computations :

$$Comp_{\langle R,A\rangle}(t) = \bigcup \{res_A(\gamma) \ / \ \gamma \in \bar{\mathcal{C}}^{\infty}_R(t)\}$$

where we consider that an element of $(D^X \to D)$ is a subset of $D^X \times D$, thus $Comp_{\langle R,A\rangle}(t) \subseteq D^X \times D$ or equivalently $Comp_{\langle R,A\rangle}(t)$ is a mapping from D^X (evaluations of the variables) to the power-set of D (sets of values).

When R is confluent (e.g. R deterministic, and especially in the case of recursive definition [31,45,41]) we may have another possible definition of the semantics since in this case for each t the set

$$\{t' \;/\; t \overset{*}{\underset{R}{\rightarrow}} t'\} \text{ is directed (w.r.t. } \overset{*}{\underset{R}{\rightarrow}})$$

thus we can define :

$$Val_{<R,A>}(t) = \bigsqcup \{\lambda_A(t') \;/\; t \overset{*}{\underset{R}{\rightarrow}} t'\}$$

and this semantics satisfies R :

$$t \to t' \in R \quad \Rightarrow \quad Val_{<R,A>}(t) = Val_{<R,A>}(t')$$

We know that these two definitions of the semantics of confluent TRS's coincide in at least two cases (which are in fact the only kinds of confluent TRS's about which something is known, see [21]) :

(i) when R is noetherian (more accurately : $\overset{+}{\underset{R}{\rightarrow}}$ noetherian) since here any term t has a unique normal form $\bar{t}$ and a computation of t is maximal iff it is, up to $\equiv^{\infty}$, a computation of t in $\bar{t}$

(ii) when R is deterministic (TRS's defined and studied by G.Huet and J.J. Levy [23]) since there is, up to $\equiv^{\infty}$, only one terminating computation in each $\mathcal{C}^{\infty}_{R}(t)$.

In the general case of confluent TRS's, the adequation of the two definitions is an open problem.

We know from what precedes that there is a universal interpretation of R, since the class of interpretations is an equational one. We now show that this universal interpretation has a nice characterization, each element of its domain being canonically represented by a tree. Let us first remark that, since $\leq_A$ allways contains $\preceq$ any interpretation of R is also a model of :

$$\overline{R} = \{(t, \bot) \;/\; t \neq \bot \;\&\; \exists r \in R : t \preceq lhs(r)\}$$

It is trivial that, since R is a TRS, $\overline{R}$ is a finite subset of $M_F(X) \times M_F(X)$, left-linear and, by construction, noetherian. Let t be such that (t/u_1) and (t/u_2) are instance of rules of $\overline{R}$, for some occurrences u_1, u_2, and let $t_1 = [\bot/u_1]t$ and $t_2 = [\bot/u_2]t$. If $u_1 \# u_2$, then clearly $t_1 \underset{\overline{R}}{\rightarrow} [\bot/u_2]t_1 = [\bot/u_1]t_2 \underset{\overline{R}}{\leftarrow} t_2$ Otherwise, if for example $u_1 <_{pref} u_2$ then (t_2/u_1) is an instance of another rule of $\overline{R}$ and $t_2 \underset{\overline{R}}{\rightarrow} t_1$. This proves that $\overline{R}$ is locally confluent, thus a reduction. We shall here denote the $\overline{R}$-normal form of a finite tree t (ie $\lambda_{\overline{R}}(t)$) by $\pi_R(t)$, the (symbolic) <u>immediate information</u> about t (what is guaranteed to be an approximation of any term computed from t, whatever could be the right-hand sides of the rules, see [45,23]).

B. Courcelle has proved ([15]) that if a left-linear reduction such as $\overline{R}$ satisfies

(i) $\quad t \preceq t' \;\&\; t' \to t'' \in \overline{R} \Rightarrow \pi_R(t) \preceq \pi_R(t')$.

(ii) $\quad t \to t' \in \overline{R} \Rightarrow \pi_R(t) \preceq \pi_R(\hat{t})$ where we get $\hat{t}$ from t by renaming each $\bot$ in t by distinct variables (not in var(t).)

then it is monotone.

But these two points are here trivial since

$t' \to t'' \in \overline{R} \;\&\; t \preceq t' \Rightarrow t'' = \bot = \pi_R(t')$ and $t = \bot$ or

$t \to \bot \in \overline{R}$ thus in any case $\pi_R(t) = \bot = \pi_R(t')$

and $t \to t' \in \overline{R} \Rightarrow \pi_R(t) = \bot \preceq \pi_R(\hat{t})$ for any $\hat{t}$

From what have been said in the preceeding paragraph, we may

consider the results of computations in the $\mathcal{J}_R$-universal interpretation H_R as canonical trees belonging to $\Pi_R^\infty(M_F^\infty(X))$: these are trees which do not contain any (non-empty) prefix of a left-hand side of a rule of R. We call the semantics in H_R the symbolic semantics and we shall denote (*) , and $Comp_{<R,H_R>}$ by $Comp_R$. One may view $Comp_R(t)$ as the set of (canonical) trees generated by t in TRS R. The symbolic semantics characterizes (or more adequately : factorizes) all the possible semantics of a term t since H_R is $\mathcal{J}_R$-universal :

for all interpretation $A \in \mathcal{J}_R$ of R :

$$Comp_{<R,A>}(t) = \bigcup \{\lambda_A(\theta) \;/\; \theta \in Comp_R(t)\}$$

and this is the exact generalization of the algebraic definition of the semantics of recursive program schemes, see [33].

If we define a semantic equivalence of terms in R by :

$$t \equiv_R t' \iff \forall\, A \in \mathcal{J}_R \quad Comp_{<R,A>}(t) = Comp_{<R,A>}(t')$$

we thus get the fact (since $H_{\overline{R}} \in \mathcal{J}_R$) that two terms are semantically equivalent iff they are symbolically equivalent :

$$t \equiv_R t' \iff Comp_R(t) = Comp_R(t')$$

Example: as a typical example of non-deterministic computable functions, one may consider (see [25])

(i) the function which splits a file (first in - first out) of data, buffered from a shared common ressource, for example, into two files.

(ii) the "inverse" function which merges two files of data into a common channel.

Assuming that the set of data is D, if we denote by D^∞ the set of finite or infinite sequences of data, these two functions have for recursive definitions :

(*) $res_{H_{\overline{R}}}$ by res_R

split(ε) = {(ε,ε)}

split(du) = {(d,ε),(ε,d)}split(u)

(the first data of the file is at the left) and

merge(u,ε) = merge(ε,u) = {u}

merge(du,d'v) = {d} merge(u,d'v) $\cup$ {d'}merge(du,v)

Here we shall not take care about the definition of the type file, but merely consider that these functions are defined on lists, for which the constructors are :

cons : data x list -> list

nil : list

Then the merge function can be described by the (ambiguous) TRS :

merge (x,nil) -> x

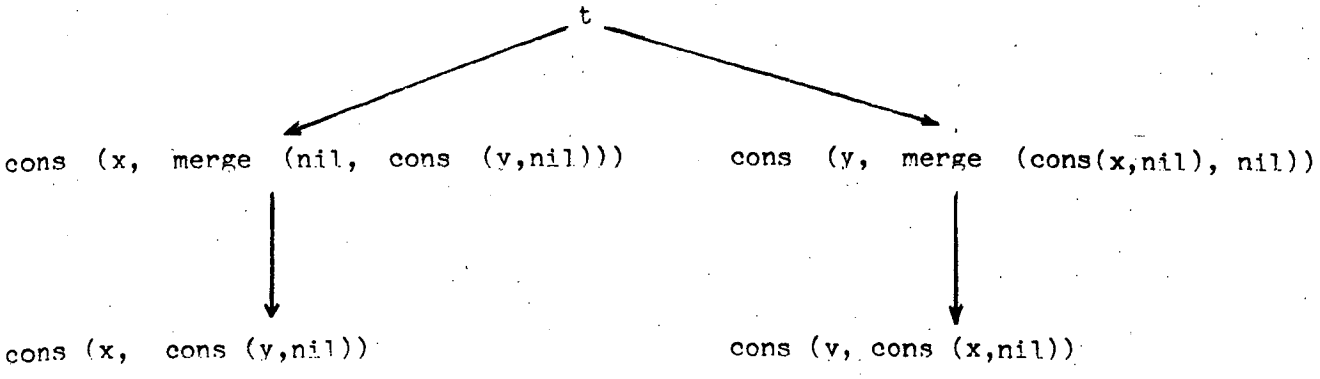
merge (nil,y) -> y

merge (cons (x,y), cons (u,v)) -> cons (x, merge (y, cons (u,v)))

merge (cons (x,y), cons (u,v)) -> cons (u, merge (cons (x,y), v))

Then for example the terminating computations of

t = merge (cons (x, nil), cons (y, nil)) are



If we have two processes which repeatdly produce data a and b resp. we can write, as equations for these processes :

ψ -> cons (a,ψ)

φ -> cons (b,φ)

and the reader may convince himself that the symbolic denotation of merge(ψ,φ) is the set of all infinite sequences on {a,b}.

For the split function, we have to define the cartesian product list x list and this can be done with a pairing function pair and two projections proj 1 and proj 2, which satisfies

proj 1 (pair (x,y)) -> x

proj 2 (pair (x,y)) -> y

(and this is the only way to get data from a pair of lists: these projections are the names of two channels). We also have to define on this product a left cons and a right cons, for which

proj 1 (consl (x,y)) -> cons (x, proj 1 (y))

proj 2 (consl (x,y)) -> proj 2 (y)

proj 1 (consr (x,y)) -> proj 1 (y)

proj 2 (consr (x,y)) -> cons (x, proj 2 (y))

Thus the desired definition of split is :

split (nil) -> pair (nil, nil)

split (cons (x,y)) -> consl (x, split (y))

split (cons (x,y)) -> consr (x, split (y))

If for example we have a process which alternatively produce data a and b, that is :

ψ -> cons (a, cons(b, ψ))

Then split (ψ) has for symbolic results (through the projections) any pair of sequences on {a,b} with at least one infinite. For example :

proj 1 (split (ψ)) -> proj 1 (split (cons (a, cons (b, ψ))))

-> proj 1 (consl (a, split (cons (b, ψ))))

-> cons (a, proj 1 (cons r(b, split (ψ))))

-> cons (a, proj 1 (split (ψ)))

-> cons (a, cons (a,..., cons (a, proj 1 (split (ψ)))...))

---

5. Implementation

This section is just a first approach (and may be the title will seem to be unfair !) of the following problem : can we give an effective way to compute $Comp_{<R,A>}(t)$, at least in the symbolic case ? Up to this point, we have not made very severe restrictions on our TRS, but the price of this generality is that our definition of the semantics is hightly ineffective : by no way, except for the finite case, we can decide if a computation is terminating. As we shall see, the situation drastically changes (that is we have to considerably restrict the class of TRS's) if we want to have, stating more formally the problem, for each term t a recursively enumerable set $\mathcal{K}_R(t)$ of computations of t which is universally correct, that is :

for all interpretation $A \in \mathcal{I}_R$

$$Comp_{<R,A>}(t) = \bigcup \{res_A(\gamma) \;/\; \gamma \in \mathcal{K}_R(t)\}$$

One usually is even more precise about this question : the algorithm which compute the set $\mathcal{U}_R(t)$ is a computation rule.

Formally in our non-deterministic case a computation rule is a mapping ρ which for each term t gives a subset of $occ_R(t)$, the set of occurrences of rules of R in t. Then a computation is relative to this rule if at each step it only apply redexes at some occurrences selected by ρ, and is full w.r.t. ρ (a special kind of development, related to effective termination) if at each step, the set of occurrences where a rule is applied is the set given by ρ. Then the rule ρ is said to be universally correct if for all t the set of full computations of t w.r.t. ρ is correct. Obviously to be correct a rule ρ has to be such that $occ_R(t) \neq \emptyset \Rightarrow \rho(t) \neq \emptyset$

From this point of view, let us examine the situation of two rules well-known to be correct in the case of deterministic recursive definitions : the "full substitution" one, which is occ_R, and the "parallel outermost" one which gives for t :

$$outocc_R(t) = \{u \;/\; u \in occ_R(t) \;\&\; \forall v \in occ_R(t) : v \underline{<}_{pref} u \Rightarrow v=u\}$$

In our non-deterministic case, these rules are not correct, as shown by the

Example 5-1

Let R be given by

$k\ (x,c) \rightarrow f\ (x,x)$

$\psi \rightarrow a \qquad \psi \rightarrow b$

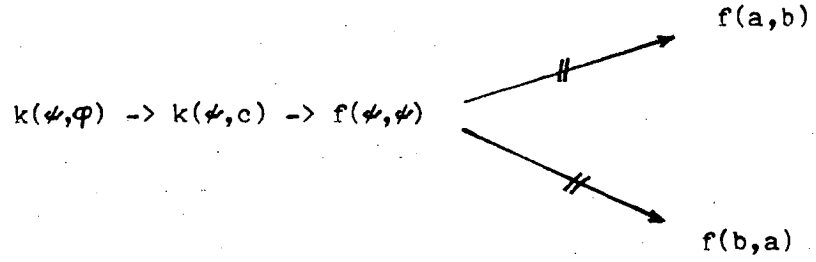
$\varphi \rightarrow c$

Then for the term $t = k\ (\psi, \varphi)$ these two rules give the same full computations :

$k(\psi,\varphi) \not\#> k(a,c) \rightarrow f(a,a)$

$k(\psi,\varphi) \not\#> k(b,c) \rightarrow f(b,b)$

but we cannot get the results of



This example also shows that the "left-most computations" (called normal in [23]) are no longer correct. We may remark also that for the non-deterministic recursive definitions ([2,33]) the full substitution rule is incorrect, for the same reason as in the above example : to choose and then duplicate is not correct. For example if :

$\varphi(x) \rightarrow f(x,x)$

$\underline{or}\ (x,y) \rightarrow x\ ,\ \underline{or}\ (x,y) \rightarrow y$

then the full substitution rule cannot allow to get the results

$f(a,b)$ and $f(b,a)$ for $\varphi(\underline{or}\ (a,b))$.

5-1 Call-by-need computations

In fact there is another exigence that one has about computation rules : they have to be efficient, and this means at least that they do not allow to made unnecessary rewritings. In the deterministic case, G. Huet and J.J. Levy [23] define a notion of needed redex, one which is unavoidable to compute, in order to terminate, and show the correctness of the corresponding "call-by-need" computation rule - w.r.t. finite termination, which thus does not generalize the case of recursive definitions, see [4]. But this correctness property is no longer true in

the non-deterministic case, even with finite termination; if we take a non-deterministic variant of the example of G. Berry [3] (of stable non sequential function) :

f(a,b,x) -> c $\quad\quad$ $\perp$ -> a

f(b,x,a) -> c $\quad\quad$ $\perp$ -> b

f(x,a,b) -> c

then one may check that there is no needed occurrence of redex in the term ($f(\perp,\perp,\perp)$: for each redex, there is a terminating finite computation which do not rewrite at this occurrence.

Moreover, as G. Huet and J.J. Levy [23] point out, the TRS must have a restricted form if we want to get an effective call-by-need : one has to find syntactical restrictions which are expressed only on the left-hand sides of the rules. We shall take this way, and we already have a standard manner to express this kind of restriction : by means of the immediate information about a term (which does not depend on the right-hand sides of the rules). Thus we define <u>strongly needed</u> (or <u>necessary</u>) occurrences in a term, by saying that an occurrence is unnecessary if the immediate information about the term does not depend of the subterm at this occurrence :

for $t \in M_F(X)$ & $w \in occ(t)$:

$$w \notin nec_R(t) \iff \forall t' : \pi_R([t'/w]t) = \pi_R(t)$$

And our <u>call-by-need rule</u> of computation will be :

$$necocc_R(t) = occ_R(t) \cap nec_R(t)$$

(the strongly needed occurrences of redexes).

Example 5-1 (continued)

For the term $t = k(\psi,\varphi)$ we have

$$nec_R(t) = \{\varepsilon, k_2\}$$

since $\pi_R(k(\psi,a)) = k(\perp,a) \neq \pi_R(t) = \perp$ and $k(x,\perp) \xrightarrow[R]{} \perp$

It should be intuitively clear that an unnecessary occurrence is one which is covered by an instance of a tree t such that $\pi_R(t) = \perp$. One can prove that it is true :

Let $E_R = \bigcup_{n\in\mathbb{N}} E_n$ where

$t \in E_o \iff \exists \theta \in dom(\overline{R})\ \exists$ s permutation of X s.t. $t = s^{\infty}(\theta)$

$t \in E_{n+1} \iff \exists \theta \in dom(\overline{R})\ \exists t' \in E_n\ \exists\ u \in occ(t') :$

$$var(\theta) \cap var(t') = \emptyset\ ,\ (t'/u) = \perp\ \&\ t = [\theta/u]t'$$

Obviously $t \in E_R \Rightarrow$ t is linear and $\pi_R(t) = \perp$

lemma 5-1

for $t \in M_F(X)$ and $w \in occ(t)$:

$w \notin nec_R(t) \iff \exists \theta \in E_R\ \exists w \in occ(X,\theta)\ \exists u\ \exists$ s substitution :

$$(t/u) = s^{\infty}(\theta)\ \&\ uv <_{pref} w$$

proof : if the right-hand side of the statement is true, then since θ is linear, for all substitution s' : $\pi_R([s'^{\infty}(\theta)/u]t) = \pi_R([\perp/u]t) = \pi_R(t)$

and thus $w \notin nec_R(t)$.

We prove the converse implication by $\underset{\overline{R}}{\rightarrow}$noetherian induction on t :

(i) $t = \pi_R(t)$. Let us suppose that $w \notin nec_R(t)$ for some $w \in occ(t)$.

Then we have, if x is a variable such that $x \neq (t/w)$:

$$\pi_R([x/w]t) = \pi_R(t) = t$$

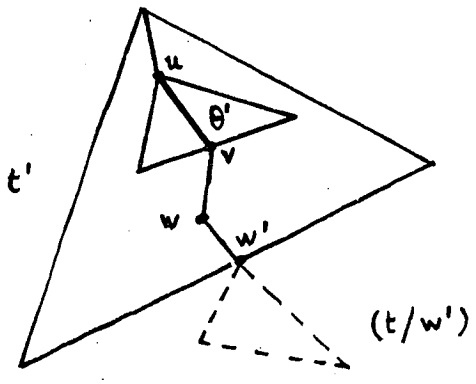
Thus, since $[x/w]t \neq t$, we must have $[x/w]t \overset{+}{\underset{\overline{R}}{\rightarrow}} t$, but this contradicts the fact that t is in $\overline{R}$-normal form.

In this case we have $nec_R(t) = occ(t)$, and the implication is vacuously true.

(ii) let us now suppose that $t \xrightarrow[\overline{R}]{(w',r)} t' = [\perp/w']t$ where $r = \Theta \rightarrow \mathbf{1}$.

By cases :

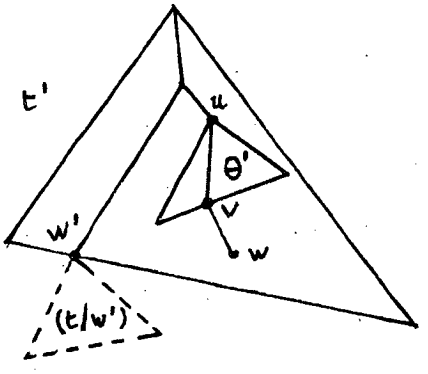
(iii) if $w \underset{pref}{\leq} w'$ then for all $t'' : [t''/w]t' = [t''/w]t$, thus $w \notin nec_R(t')$ and we apply the induction hypothesis, as shown by

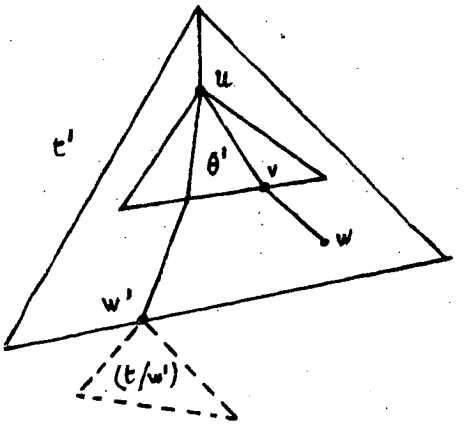


(since Θ' is linear)

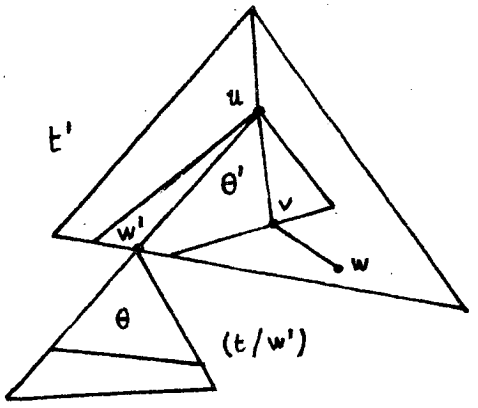
(iv) if $w' \# w$ then for all $t'' : [t''/w]t \xrightarrow[\overline{R}]{(w',r)} [t''/w]t'$ and thus

$w \notin nec_R(t')$. We left to the reader the details of application of the induction hypothesis in the possible cases :





(θ' is linear)



(here with a suitable change of variables of θ')

(v) if $w' <_{pref} w$ then, for $x \in X$:

$$\pi_R([x/w]t) = \pi_R(t) = \pi_R(t') \preceq t' \text{ , and } t' \preceq [x/w]t$$

thus we must have : $[x/w]t \xrightarrow[\overline{R}]{(w'',r)} t''$ with

- $w'' \# w$, and in this case we return to (iv) (obviously $w \notin nec_R([x/w]t)$).

- or $w'' \leq_{pref} w$ and in this case $lhs(r) \in E_R$ and there exists $v \in occ(X, lhs(r))$ such that $w''v \leq_{pref} w$ ■

As a corollary one may see that

$$w \notin nec_R(t) \iff \text{for } x \in X,\ x \neq (t/w) : \pi_R([x/w]t) = \pi_R(t)$$

and thus our necessary occurrences are exactly the strongly needed ones in the sense of [23]. Another consequence of this characterization is that one may prove that necessity is preserved along computations :

corollary 5-1

$$\text{For } \gamma : t \xrightarrow[R]{*} t' \ ,\ w \in nec_R(t) \Rightarrow w\langle\gamma\rangle \subseteq nec_R(t')$$

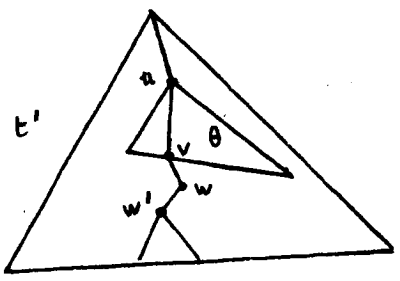
(thus $w[\gamma] \subseteq nec_R(t)$ since $w[\gamma] \subseteq w\langle\gamma\rangle$)

proof: induction on $|\gamma|$. It suffices to prove this fact for $\gamma = (w',r)$

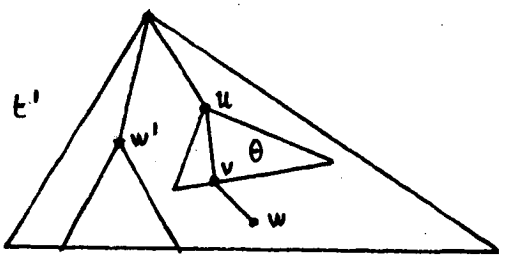
(i) If $w' \leq_{pref} w$ and $w \in nec_R(t)$ then $w\langle\gamma\rangle = \{w'\}$ (since (w',r) cannot covers w, by lemma 5-1). But $w' \notin nec_R(t') \Rightarrow w' \notin nec_R(t)$. Thus the assumption $w' \notin nec_R(t')$ contradicts $w \in nec_R(t)$ in this case, where $w' \leq_{pref} w$.

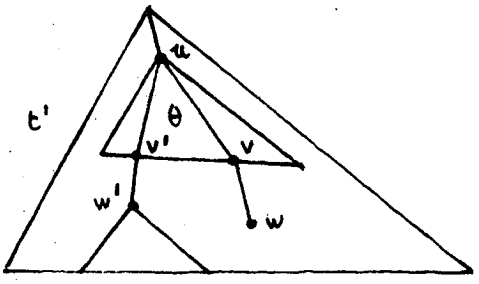
(ii) If $w' \not\leq_{pref} w$ then $w\langle\gamma\rangle = \{w\}$

Assuming that $w \notin nec_R(t')$ and using lemma 5-1, the reader may find a contradiction in each case :

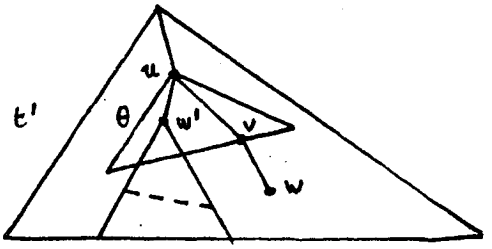


(θ linear)





(θ linear)



(here with a suitable change of variables of θ : $[lhs(r)/(w'/u)]\,\theta \in F_R$) ■

From this corollary, one may see that strongly needed occurrences are external for all (finite) computations. If we define the "absolutely external" occurrences of a term t as

$$ext_R(t) = \bigcap_{\gamma \in \mathcal{C}_R(t)} ext_R(\gamma)$$

then $nec_R(t) \subseteq ext_R(t)$

Another trivial consequence is :

For $\gamma : t \xrightarrow[R]{*} t'$ and σ consistent set of redexes in R :

$dom(\sigma) \subseteq nec_R(t)$ & $\sigma \uparrow \gamma \Rightarrow dom(\sigma[\gamma]) \subseteq nec_R(t')$

The call-by-need computations are computations which only rewrite necessary redexes, that is

$$\gamma : t_o \xrightarrow[R]{\sigma_o} t_1 \to \ldots \to t_n \xrightarrow[R]{\sigma_n} t_{n+1} \to \ldots$$

such that $\forall n \in N : dom(\sigma_n) \subseteq nec_R(t_n)$

If moreover γ is $\subseteq$-maximal such that $\forall n \in N\ dom(\sigma_n) = necocc_R(t)$ then we say that γ is a full call-by-need computation. We shall denote by $\mathcal{N}_R^\infty$ (resp. $\mathcal{N}_R$, $\mathcal{S}_R$) the set of call-by-need (resp. finite call-by-need, full call-by-need) computations. Let us mention some obvious facts :

$$\gamma \in \mathcal{N}_R^\infty \Rightarrow \forall n \in N\ \gamma|_n \in \mathcal{N}_R^\infty \ \&\ \gamma|^n \in \mathcal{N}_R$$

$$\gamma \in \mathcal{N}_R \ \&\ \delta \in \mathcal{N}_R^\infty \Rightarrow \gamma;\delta \in \mathcal{N}_R^\infty \text{ (if the composition is defined)}$$

$$\gamma, \gamma' \in \mathcal{N}_R \ \&\ \gamma \uparrow \gamma' \Rightarrow \gamma[\gamma'] \in \mathcal{N}_R \text{ (from the above corollary)}$$

We denote for $\gamma \in \mathcal{N}_R(t)$ by $necred_R(\gamma)$ the necessary initial redexes of γ :

$$necred_R(\gamma) = \{(u,r)/(u,r) \in initred_R(\gamma) \ \& \ u \in nec_R(t)\}$$

It is easy to see that

$$necred_R(\gamma) \subseteq extred_R(\gamma)$$

(thus $necred_R(\gamma)$ is consistent set of redexes strongly compatible with γ) and :

$$dom(\sigma) \subseteq nec_R(t) \Rightarrow \sigma \subseteq necred_R(\sigma;\delta) \text{ for all } \delta$$

Moreover (by lemma 3-7) :

$$\gamma \equiv \gamma' \Rightarrow necred_R(\gamma) = necred_R(\gamma')$$

thus $\gamma \leq \gamma' \Rightarrow necred_R(\gamma) \subseteq necred_R(\gamma')$

We now study the structure $< \mathcal{N}_R^{\infty} , \leq^{\infty} >$ of ordered call-by-need computations. The first property to note is that in $\mathcal{N}_R$ the compatibility relation ($\uparrow$) coincide with the usual-w.r.t. ordering-one, and that two compatible call-by-need finite computations have a lub :

lemma 5-2

For γ, γ' in $\mathcal{N}_R(t)$:

$\exists\, \gamma'' \in \mathcal{N}_R(t) : \gamma \leq \gamma'' \ \& \ \gamma' \leq \gamma'' \Rightarrow \gamma \uparrow \gamma'$ and $\gamma;\gamma'[\gamma]$ is a least upper bound of $\{\gamma, \gamma'\}$

(it is allways true that $\delta \uparrow \delta' \Rightarrow \exists\, \delta'' : \delta \leq \delta'' \ \& \ \delta' \leq \delta''$, by the permutation lemma)

proof: by induction on $\min(|\gamma|, |\gamma'|)$. The lemma is trivial if this is o.

Now let $\gamma = \sigma : \gamma_1$ and $\gamma' = \sigma' : \gamma_2$. The hypothesis is :

$$\exists\ \delta\ ,\ \delta' : \gamma\ ;\ \delta \equiv \gamma' : \delta'$$

Since $dom(\sigma) \subseteq nec_R(t)$ we have $\sigma \subseteq necred_R(\gamma\ ;\ \delta)$ and similarly $\sigma' \subseteq necred_R(\gamma' : \delta')$ thus $\sigma \uparrow \sigma'$ (since $necred_R(\gamma;\delta) = necred_R(\gamma';\delta')$)

Moreover $\sigma \uparrow \gamma':\delta'$ and $\sigma' \uparrow \gamma;\delta$ and

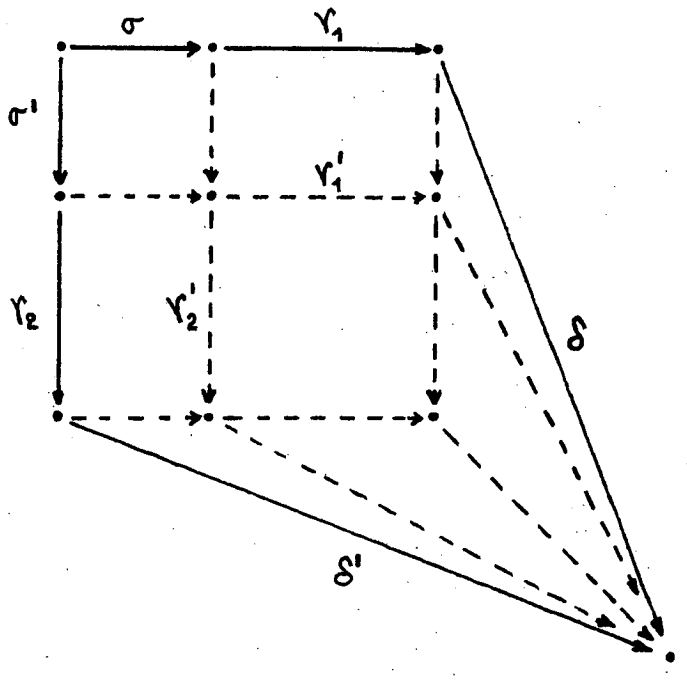
$$\sigma[\gamma';\delta'] = \emptyset = \sigma'[\gamma;\sigma].\ (\text{since } necred_R(\alpha) \subseteq extred_R(\alpha).)$$

Thus (permutation lemma)

$$\gamma;\delta \equiv \sigma'\ ;\ (\gamma;\delta)[\sigma'] = \sigma'\ ;\ \gamma[\sigma']\ ;\ \delta[\sigma'[\gamma]]$$

$$\gamma';\delta' \equiv \sigma\ ;\ (\gamma';\delta')[\sigma] = \sigma\ ;\ \gamma'[\sigma]\ ;\ \delta'[\sigma[\gamma']]$$

A picture may help :



By the simplification lemma $\gamma[\sigma'] : \delta[\sigma'[\gamma]] \equiv \gamma_2;\delta'$ thus, by induction hypothesis $\gamma[\sigma'] \uparrow \gamma_2$, and similarly $\gamma'[\sigma] \uparrow \gamma_1$

But since $\gamma[\sigma'] = \sigma[\sigma'] : \gamma_1[\sigma'[\sigma]]$ and $\gamma'[\sigma] = \sigma'[\sigma] : \gamma_2[\sigma[\sigma']]$ we get $\gamma \uparrow \gamma'$.

If we let $\gamma'_1 = \gamma_1[\sigma'[\sigma]]$, $\gamma'_2 = \gamma_2[\sigma[\sigma']]$ (which are call-by-need computations, by corollary 5-1) then

$$\begin{aligned}
\gamma;\delta &\equiv \sigma ; \sigma'[\sigma] : \gamma'_1 : \delta[\sigma'[\gamma]] \\
&\equiv \sigma';\sigma[\sigma'] : \gamma'_1 : \delta[\sigma'[\gamma]] \quad \text{(permutation lemma)} \\
&\equiv \gamma';\delta' \quad \text{(hypothesis)} \\
&\equiv \sigma';\sigma[\sigma'] ; \gamma'_2;\delta'[\sigma[\gamma']] \quad \text{(permutation lemma)}
\end{aligned}$$

Then by the simplification lemma

$$\gamma'_1 : \delta[\sigma'[\gamma]] \equiv \gamma'_2 ; \delta'[\sigma[\gamma']]$$

and thus, by induction hypothesis $\gamma'_1 ; \gamma'_2[\gamma'_1]$ is a lub of $\{\gamma'_1,\gamma'_2\}$

whence

$$\begin{aligned}
&\exists \delta'_1 : \gamma'_1 : \delta[\sigma'[\gamma]] \equiv \gamma'_1 ; \gamma'_2[\gamma'_1] : \delta'_1 \\
&\exists \delta'_2 : \gamma'_2 ; \delta[\sigma[\gamma']] \equiv \gamma'_1 : \gamma'_2[\gamma'_1] ; \delta'_2 \text{ (in fact } \delta'_1 \equiv \delta'_2)
\end{aligned}$$

Thus

$$\begin{aligned}
\gamma;\delta &\equiv \sigma : \sigma'[\sigma] : \gamma'_1 : \gamma'_2[\gamma'_1] : \delta'_1 \\
&\equiv \gamma : \gamma'[\gamma] ; \delta'_1 \quad \text{that is } \gamma ; \gamma'[\gamma] \leq \gamma;\delta \qquad \blacksquare
\end{aligned}$$

Now we can describe the preoder in $\mathcal{N}_R^\infty(t)$ as :

$$\gamma \lesssim^\infty \gamma' <=>_{def} \forall \delta \in \mathcal{N}_R(t) : \delta \sqsubseteq \gamma \Rightarrow \exists \delta' \in \mathcal{N}_R(t) : \delta' \sqsubseteq \gamma' \ \&\ \delta \lesssim \delta'$$

where, for finite computations :

$$\gamma \lesssim \gamma' <=>_{def} \exists \delta \in \mathcal{N}_R : \gamma' \equiv \gamma;\delta$$

for we have :

lemma 5-3

$$\text{for } \gamma,\gamma' \in \mathcal{N}_R^\infty : \gamma \leq^\infty \gamma' <=> \gamma \lesssim^\infty \gamma'$$

proof: obviously it suffices to prove $\gamma \leq \gamma' \Rightarrow \gamma \lesssim \gamma'$ for γ, γ' in $\mathcal{R}_R$. But $\gamma \leq \gamma'$ means that $\exists \delta : \gamma' \equiv \gamma : \delta$, and, by the above lemma, $\gamma \uparrow \gamma'$ and $\gamma : \gamma'[\gamma] \leq \gamma'$. Then (by permutation and simplification lemmas) $\gamma[\gamma'] \equiv \varepsilon$, $\gamma' \equiv \gamma : \gamma'[\gamma]$ and $\gamma'[\gamma] \in \mathcal{R}_R$ ∎

(in $c\mathcal{R}_R$ we find the usual definition of the preorder on computations : $\gamma \leq \gamma' \Leftrightarrow \gamma \uparrow \gamma' \ \&\ \gamma[\gamma'] \equiv \varepsilon$)

Let us denote for $\gamma \in \mathcal{R}_R^\infty$ and $\delta \in \mathcal{R}_R$ by $[\gamma]^\infty$ and $[\delta]$ resp. the equivalences classes of γ and δ in $\mathcal{R}_R^\infty$ and $\mathcal{R}_R$. Then we have :

theorem 5-1

(i) for all term t $< \mathcal{R}_R^\infty(t)/\equiv^\infty , \lesssim^\infty , [\varepsilon]^\infty >$ is a completion of $< \mathcal{R}_R(t)/\equiv , \lesssim , [\varepsilon] >$

(ii) for all $\gamma \in \mathcal{C}_R^\infty(t)$, $\hat{\gamma} = \{[\delta] / \delta \in \mathcal{R}_R(t) \ \&\ \delta \leq^\infty \gamma\}$ is an ideal of $\mathcal{R}_R(t)/\equiv$, $\gamma \equiv^\infty \gamma' \Rightarrow \hat{\gamma} = \hat{\gamma'}$ and $\ulcorner\gamma\urcorner^\infty \mapsto \hat{\gamma}$ is a strict continuous projection ($\hat{\gamma} \leq^\infty \ulcorner\gamma\urcorner^\infty$ & $\hat{\hat{\gamma}} = \hat{\gamma}$)

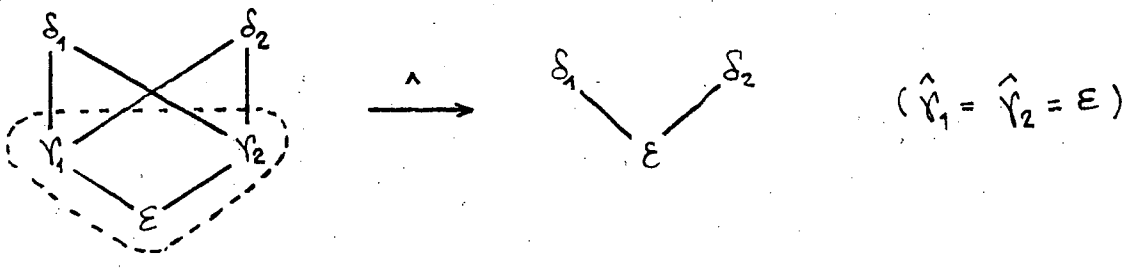
proof: From lemma 5-3, one can prove (i) exactly as we have proved the theorem 3-2. The second point is a trivial consequence of lemma 5-2 ∎

We shall ambiguously use the notation $\hat{\gamma}$, and for example call $\hat{\gamma}$ the (greatest) call-by-need subcomputation of γ. By the projection $\gamma \mapsto \hat{\gamma}$ some "bad" phenomenons disappear, as shown by :

Example 3-1 (continued) in R given by

h(a) -> b h(a) -> c
k(x,a) -> b k(x,a) -> c

for the computations of t = k(h(a), a) we get the picture :

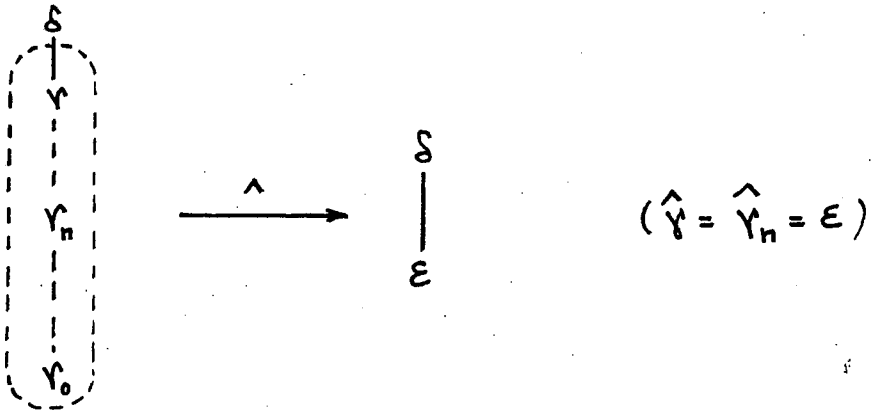


Example 3-2 (continued)

with R

a -> h(a)
k(x,b) -> c

for t = k(a,b) we have



In fact one can say more about the structures $\langle \mathcal{N}_R^\infty(t)/\equiv^\infty , \lesssim^\infty , [\varepsilon]^\infty \rangle$: they are coherent (any consistent subset-ie in which two elements are compatible - has a lub) algebraic (with denumerable base) partial order, thus they are "domaines de calcul" in the sense of G.Kahn and G.Plotkin [24]. We conjecture that these structures are event structures (more precisely, the ordered set of configurations of event structures) as defined by G.Winskel [46] and concrete domains ([24]) if R is non-ambiguous. We already have a concept of immediate incompatibility between application of redexes, and a notion of causability by means of creation of redexes. There are ambiguous TRS's for which $\mathcal{N}_R^\infty(t)/\equiv^\infty$ is not a concrete domain :

Example

Let R be given by

f(b,b) -> c
b -> a
f(a,a) -> d

then for t = f(b,b) $\mathcal{C}_R^{\infty}(t)/\equiv^{\infty} = \mathcal{N}_R(t)/\equiv$ and if

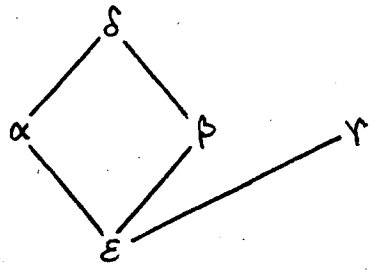
γ : f(b,b) -> c

α : f(b,b) -> f(a,b)

β : f(b,b) -> f(b,a)

δ : f(b,b) #> f(a,a) -> d

we get



We can describe the (representant of the) call-by-need subcomputation of finite computations :

lemma 5-4

for $\gamma \in \mathcal{C}_R(t)$ if $\gamma' = \gamma[\sigma]$ for some $\sigma \subseteq \text{necred}_R(\gamma)$

then $\hat{\gamma} \equiv \sigma ; \hat{\gamma'}$

proof: let $\delta \in \mathcal{N}_R(t)$ be such that $\delta \leq \gamma$. Then $\delta \leq \sigma ; \gamma'$ and by lemma 5-2 : $\delta \uparrow \sigma$ and $\sigma ; \delta[\sigma] \leq \sigma ; \gamma'$, thus $\delta[\sigma] \leq \gamma'$ we have shown $\hat{\gamma} \leq \sigma ; \hat{\gamma'}$. The converse inequality is rather trivial ■

As a corollary, we can define for $\gamma \in \mathcal{C}_R(t)$ the call-by-need subcomputation $\hat{\gamma}$ of γ by :

$\hat{\gamma} = \varepsilon$ if $\text{necred}_R(\gamma) = \emptyset$

$\hat{\gamma} = \text{necred}_R(\gamma) ; \hat{\gamma'}$ where $\gamma' = \gamma[\text{necred}_R(\gamma)]$ otherwise

This definition makes sense, since the relation

$$\gamma \rhd \gamma' <=>_{def} \exists (w,r) \in \text{necred}_R(\gamma) : \gamma' = \gamma[w,r]$$

and its transitive closure $\triangleright^+$ are notherian (by lemma 3-9).

To conclude this paragraph we add to the theorem more insight into maximal call-by-need computations :

$$\bar{\mathcal{N}}_R^\infty(t) = \{\gamma / \gamma \in \mathcal{N}_R^\infty(t) \;\&\; \forall \gamma' \in \mathcal{P}_R^\infty(t) : \gamma \lesssim^\infty \gamma' \Rightarrow \gamma' \equiv^\infty \gamma\}$$

Firstly we show that <u>full call-by-need computations are terminating (among call-by-need computations)</u>:

<u>lemma</u> 5-5

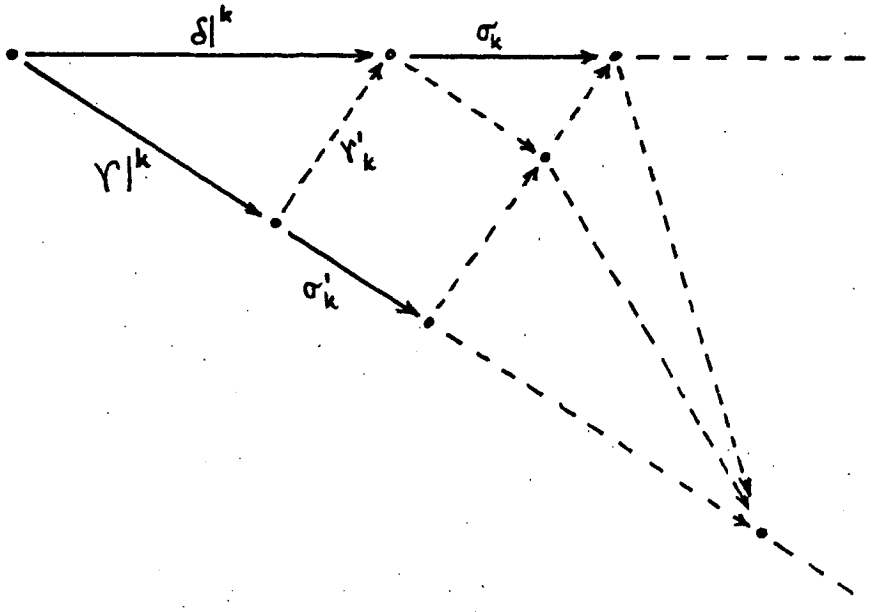
for all term t : $\mathcal{S}_R(t) \subseteq \bar{\mathcal{P}}_R^\infty(t)$

<u>proof</u> : let $\delta \in \mathcal{S}_R(t)$ and $\gamma \in \mathcal{P}_R^\infty(t)$ be such that $\delta \lesssim^\infty \gamma$.

We sketch the proof by induction of the fact :

$\forall k \in \mathbb{N} \quad \gamma|^k \lesssim \delta|^k$ (which is obvious for k=o)

Assuming that $\gamma|^k \lesssim \delta|^k$, there exists $\gamma'_k \in \mathcal{N}_R$ s.t. $\delta|^k \equiv \gamma|^k ; \gamma'_k$. If $\delta|^{k+1} = \delta|^k ; \sigma_k$ and $\gamma|^{k+1} = \gamma|^k ; \sigma'_k$ we have $\gamma'_k ; \sigma_k \uparrow \sigma'_k$ (by lemma 5-2, and the permutation and simplification lemma), since $\delta \lesssim^\infty \gamma$. The following picture may help understanding :



Thus $\sigma'_k[\gamma'_k] \uparrow \sigma_k$ but (corollary 5-1)

$dom(\sigma'_k[\gamma'_k]) \subseteq dom(\sigma_k)$ since δ is full

whence $\sigma'_k[\gamma'_k] \subseteq \sigma_k$ and

$$\delta|^{k+1} \equiv \gamma|^{k+1} \; ; \; \gamma'_k[\sigma'_k] \; ; \; \sigma_k[\sigma'_k[\gamma'_k]] \quad \blacksquare$$

The second property about termination is that in non-ambiguous systems, maximal call-by-need computations are, up to $\equiv^\infty$, full call-by-need computations, and the continuous projection preserves (relative) termination :

lemma 5-6

Let R be a non-ambiguous TRS. Then, for all term t :

$$\gamma \in \bar{\mathcal{N}}^\infty_R(t) \iff \exists\, \delta \in \mathcal{T}_R(t) : \gamma \equiv^\infty \delta$$
$$\iff \exists\, \delta \in \bar{\mathcal{E}}^\infty_R(t) : \gamma \equiv^\infty \delta$$

proof: for this proof, we extend $necred_R$ to infinite computations :

$$necred_R(\gamma) = \bigcup \{necred_R(\delta) / \delta \in \mathcal{C}_R \;\&\; \delta \subseteq \gamma\}$$

in such a way that for all γ there exists a finite prefix δ of γ such that $necred_R(\gamma) = necred_R(\delta)$. If, for this δ, $\gamma = \delta \; ; \; \gamma'$, we denote by $\gamma[necred_R(\gamma)]$ the computation $\delta[necred_R(\gamma)];\gamma'$, we have $\gamma \equiv^\infty necred_R(\gamma) \; ; \; \gamma[necred_R(\gamma)]$.

Our first claim is :

(i) $\gamma \in \bar{\mathcal{E}}^\infty_R(t) \Rightarrow dom(necred_R(\gamma)) = necocc_R(t)$

For, suppose that there exists $w \in nec_R(t)$ and a rule $r \in R$ such that $(w,r) \in red_R(t)$ & $w \notin dom(necred_R(\gamma))$. Since R is non-ambiguous, we have $(w,r) \uparrow necred_R(\gamma)$. By an obvious induction we get $\forall k \in \mathbb{N}\ (w,r) \uparrow \gamma|^k$ and $(w,r)[\gamma|^k] = (w,r)$.

This is trivial for k=o. Assuming that $(w,r) \uparrow \gamma|^k$ and $(w,r)[\gamma|^k] = (w,r)$ and $\gamma|^{k+1} = \gamma|^k \; ; \; \sigma_k$ we have

$(w,r) \uparrow \sigma_k$ since $(w,r) \not\uparrow \sigma_k$ would mean $w \in dom(\sigma_k)$ (for R is non-ambiguous) and thus $w \in dom(necred_R(\gamma)$. By corollary 5-1 if $\gamma|^k : t \xrightarrow[R]{*} t'$ we have $w \in nec_R(t')$, thus w cannot be covered by a redex of σ_k, thus $(w,r)[\sigma_k] = (w,r)$.

Now if γ' is the lub of the $\subseteq$-increasing sequence $((w,r);(\gamma|^k)[w,r])_{k\in\mathbb{N}}$ we have $(w,r) \in necred_R(\gamma')$, and $\gamma \leq^\infty \gamma'$. But since γ is maximal $\gamma' \equiv^\infty \gamma$ thus $(w,r) \in necred_R(\gamma)$, a contradiction ▪

For $\gamma \in \mathcal{C}^\infty_R(t)$, let us define

$\delta_o = \varepsilon \quad \gamma_o = \gamma$

$\delta_{n+1} = \delta_n : necred_R(\gamma_n) \quad \gamma_{n+1} = \gamma_n[necred_R(\gamma_n)]$

and let δ be the lub of the $\subseteq$-increasing sequence $(\delta_n)_{n\in\mathbb{N}}$

Our second claim is :

(ii) $\gamma \in \bar{\mathcal{C}}^\infty_R(t) \Rightarrow \delta \in \mathcal{S}_R(t)$ & $\delta \equiv^\infty \hat{\gamma}$

To show this point, let us remark that an obvious consequence of the simplification lemma is that any suffix of terminating computation is a terminating one :

$$\alpha : t \xrightarrow[R]{*} t' \;\&\; \alpha;\beta \in \bar{\mathcal{C}}^\infty_R(t) \Rightarrow \quad \beta \in \bar{\mathcal{C}}^\infty_R(t')$$

Here we have, for all n : $\gamma \equiv^\infty \delta_n ; \gamma_n$. Thus if γ is maximal, then γ_n is maximal and by the first claim δ is full.

Obviously $\delta \leq^\infty \gamma$ thus $\hat{\delta} \lesssim^\infty \hat{\gamma}$ but $\delta = \hat{\delta}$ and by lemma 5-5 $\delta \equiv^\infty \hat{\gamma}$ ▪

Now we get the lemma (by means of lemma 5-5)

$\gamma \in \bar{\mathcal{N}}^\infty_R(t) \Rightarrow \exists\, \gamma' \in \bar{\mathcal{C}}^\infty_R(t) : \gamma \leq^\infty \gamma' \Rightarrow \hat{\gamma} \leq^\infty \hat{\gamma}'$ thus $\hat{\gamma}' \equiv^\infty \hat{\gamma}$

and we just have seen that $\exists\, \delta \in \mathcal{S}_R(t) : \hat{\gamma}' \equiv^\infty \delta$ ▪

There are ambiguous TRS's for which these properties are false :

Example

If R is given by

f(a,b) -> c
b -> a

then for t = f(b,b) we get

$\mathcal{I}_R(t)$ = {t #> f(a,a)} whereas $\overline{\mathcal{N}}^{\infty}_R(t)$ contains t -> f(a,b) -> c

(and any computation is a call-by-need one).

Remark: we could have take ext_R instead of nec_R without affecting any of the results of this paragraph (except lemma 5-1 !) But in general, $ext_R(t)$ is not computable.

Example

Let R be given by

k(x,a) -> c
$\not\phi$ -> ...

then $k_1 \notin ext_R(k(x,\not\phi))$ <=> $\not\phi \overset{*}{\underset{R}{\rightarrow}} a$ and this is undecidable (even in our TRS's : in the monadic case, one get the Post's correspondance problem).

5-2 Sequential systems : the correctness theorem.

As we have seen (the variant of Berry's example), we may have, even in non-ambiguous systems $necocc_R(t) = \emptyset$ and $occ_R(t) \neq \emptyset$, thus the call-by-need computation rule is incorrect in general. Some other difficulties (of the same nature, however) come from the fact that we do not have restrict ourselves to finite termination :

Example :

If we have in R (deterministic)

$g(b,c,x) \to k$
$g(x,b,c) \to k$
$h(x) \to h(h(x))$
$\psi \to a$

then in the term $t = g(\psi,h(x),\psi)$ the only strongly needed occurrence of a rule is g_2 since in $\overline{R}$ we have :

$g(\bot,\bot,x) \to \bot$
$g(x,\bot,\bot) \to \bot$

And the only full call-by-need computation of t is :

$$t \xrightarrow[R]{} \ldots \xrightarrow[R]{} g(\psi,h^{n+1}(x),\psi) \xrightarrow[R]{} \ldots$$

and we cannot get the result $g(a,\bot,a)$ of

$$t \xrightarrow[R]{\#} g(a,h(x),a) \xrightarrow[R]{} \ldots \xrightarrow[R]{} g(a,h^{n+1}(x),a) \xrightarrow[R]{} \ldots \text{ (terminating).}$$

In some sense, strongly needed redexes have to be "uniform", at leat along unnecessary rewritings. This means that rewriting a non-necessary redex, we cannot create the necessity of another one (which is the case in the above example, for $t \to g(a,h(x),\psi)$ and $g_3 \in nec_R(g(a,h(x),\psi))$.)or cannot create a needed redex, as in the ambiguous TRS :

$f(g(x,b)) \to c$
$g(a,b) \to c$
$h \to a$

in which for $t = f(g(h,b))$ we have $f_1 g_1 \notin nec_R(t)$ and $f_1 \notin occ_R(t)$ but $t \xrightarrow[R]{} f(g(a,b))$ and $f_1 \in necocc_R(f(g(a,b)))$.

In fact we have already seen in the preceeding paragraph that we cannot actually treat the ambiguous case. Thus a suitable restriction on TRS's should be something like the following definition :

a TRS R is sequential iff

(i) R is non-ambiguous

(ii) $\forall t \in \overline{M}_F(X) : w \notin nec_R(t) \;\&\; t \xrightarrow[R]{(w,r)} t' \Rightarrow$
$necocc_R(t) = necocc_R(t')$

The second point is not exactly a kind of restriction allowed, as we have said above, since it is expressed by means of the rules of R, not only there left-hand sides. Nevertheless : can we "localize" the property, and is sequentiality decidable? Obviously a stronger condition is

$$(ii)' \forall t \in M_F(X)\ w \notin nec_R(t) \Rightarrow \forall t'\ nec_R([t'/w]t) = nec_R(t)$$

(which, in non-ambiguous systems, is equivalent to :

$$\forall t\ \forall w\ w \in occ_R(t) \;\&\; w \notin nec_R(t) \Rightarrow \forall t'\ necocc_R(t) = necocc_R([t'/w]t)\ .)$$

and with this last hypothesis (together with non-ambiguity) we get systems which are "strongly sequential" as defined by G.Huet and J.J.Levy [23]. Thus in this case, one has a nice efficient algorithm to compute $necocc_R$. These definitions does not coincide however since for example "strict sequentiality" ((i)+(ii)') and "simple sequentiality" of [23] are incomparable properties :

```
e (h(a,b,x)) -> ...
f (h(b,x,a)) -> ...
g (h(x,a,b)) -> ...
```

is "strictly sequential" but not "simple sequential", whereas

```
g (b,c,x) -> ...
g (x,b,c) -> ...
```

is "simple sequential" but not "strictly sequential".

Here we left open numerous interesting questions which actually

concern implementation (such as : is there an efficient way to build the matching dag of [23] for a "strictly sequential" system ?). Can we, at least in the deterministic case, support the definition of needed occurrences by some semantical considerations ? Let us only point out that the non-deterministic recursive definitions ([2,33]) are trivially ("strictly") sequential, and for these TRS's, the call-by-need is exactly the "parallel outermost" computation rule ($necocc_R(t)$is allways a set of mutually disjoint occurrences, if R is non ambiguous).

Our (last) aim is now to prove the correctness of call-by-need for sequential systems, for which the continuous projection $y \mapsto \hat{y}$ preserves the result of computations.

We have (lemma 5-1) localize the cause of non-necessity to the property of being covered by an instance of an element of E_R. In non-ambiguous systems, left-hand sides of the rules cannot "non-trivially" overlap the elements of E_R :

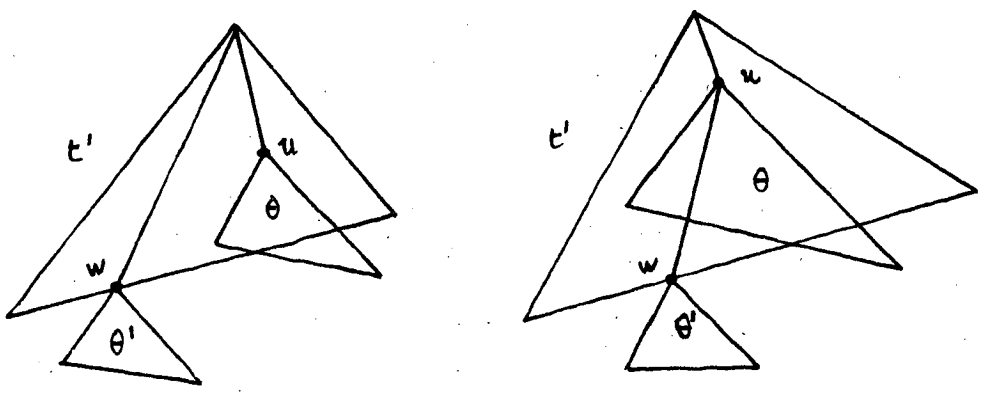
Lemma 5-7

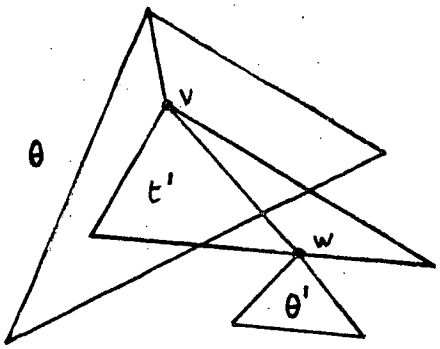
if R is non-ambiguous then for all $\theta \in dom(R)$ and $t \in E_R$:
$\theta \wedge t \Rightarrow \exists\, w \in occ(t) : (t/w)$ is an instance of θ

proof: by induction on n such that $t \in E_n$.

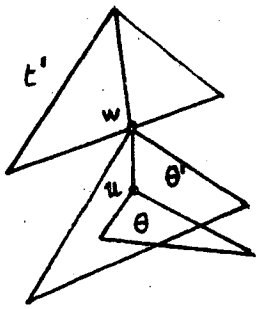
(i) if n=o then $\exists\, \theta' \in dom(R)\ \exists$ s permutation of X : $t \preceq s^{\infty}(\theta')$ and non-ambiguity insures in this case $\theta=\theta'$ (and $t = s^{\infty}(\theta)$)

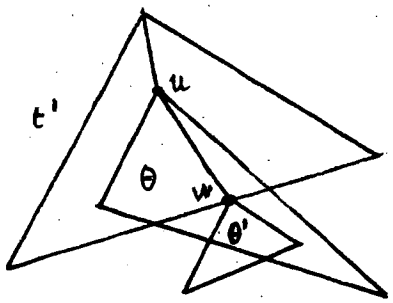
(ii) $t = [\theta'/w]t'$ for some $t' \in E_n$, $\theta' \in dom(\bar{R})$ and $(t'/w)=\perp$. We left the reader find the suitable argument in each case :

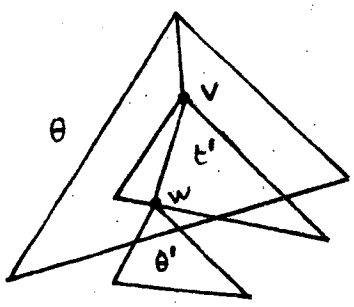




in these three cases $\theta \gg t' \;\&\; t' \in E_n$







in these three cases $\theta \gg \theta' \;\&\; \theta' \in E_o$ ∎

Corollary 5-2

Let R be sequential system, $t \xrightarrow[R]{(w,r)} t'$ and $\gamma \in \mathcal{C}_R(t')$ Then :
$necred_R((w,r);\gamma) = \emptyset \Rightarrow necred_R(\gamma) = \emptyset$

proof: let us assume that $necred_R((w,r);\gamma) = \emptyset$ and $(u,r') \in necred_R(\gamma)$
Obviously $w \notin nec_R(t)$

(i) if $w \leq_{pref} u$, then $u \notin nec_R(t')$ (for $u=wv$ and $[t''/u]t = [[t''/v]rhs(r)/w]t$ but $w \notin nec_R(t)$) and this is a contradiction.
(ii) if $u \# w$ then, since P is non-ambiguous, $(u,r') \in red_R(t)$ and, since R is sequential, $u \in necocc_R(t)$, thus $(u,r') \in necred_R((w,r);\gamma)$, a contradiction.

(iii) if $u <_{pref} w$, one has two cases :

(iv) $(u,r') \in red_R(t)$ thus, since R is non-ambiguous, $(u,r') \uparrow (w,r)$ and $(u,r')[w,r] = (u,r')$.
Here again : $(u,r') \in necred_R((w,r);\gamma)$ a contradiction.
(v) $(u,r') \notin red_R(t)$ ((u,r') is created by (w,r).)
Since $w \notin nec_R(t')$, by lemma 5-1 $\exists\, t'' \in E_R\ \exists\, w'$ $\exists\, v \in occ(X,t'')$: (t'/w') is an instance of t'' and $w'v \leq_{pref} w$
Thus $u \leq_{pref} w'$ or $w' \leq_{pref} u$ but we cannot have $w'v \leq_{pref} u$ since $u \in nec_R(t')$. If $w = uu'$, we must have $u' \in int(lhs(r'))$ for $(u,r') \notin red_R(t)$. Thus $lhs(r')$ must overlap t''.

But by lemma 5-7 there exists a subtree of t'' which is an instance of $lhs(r')$ and this contradicts $u' \in int(lhs(r'))$ or $w'v \leq_{pref} w$ for some $v \in occ(X,t'')$ ■

This property allows to prove that in sequential systems, the continuous projection from $\mathcal{G}^{\infty}_R$ to $\mathcal{N}^{\infty}_R$ preserves results :

lemma 5-8

if R is a sequential system then for all term t and $\gamma \in \mathcal{G}^{\infty}_R(t)$: $res_R(\hat{\gamma}) = res_R(\gamma)$

proof: we only have to establish the lemma for finite computations, in which case we proceed by $\rhd^+$-noetherian induction (see lemma 5-4) : for $\gamma \in \mathcal{G}_R(t)$:

(i) if $necred_R(\gamma) = \emptyset$ then $\hat{\gamma} = \varepsilon$ thus $res_R(\hat{\gamma}) = \pi_R(t)$.

By induction on the size $\|\gamma\|$ we prove $res_R(\gamma) = \pi_R(t)$. This is trivial if $\|\gamma\| = 0$. Otherwise, cancelling the empty steps and applying the

permutation lemma, we may assume that :

$$\gamma = (w,r) ; \gamma' \quad \text{where } w \notin nec_R(t)$$

If $t \xrightarrow[R]{(w,r)} t'$ then $w \notin nec_R(t) \Rightarrow \pi_R(t') = \pi_R(t)$

since (by corollary 5-2) $necred_R(\gamma') = \emptyset$ we have by induction hypothesis $res_R(\gamma') = \pi_R(t') = \pi_R(t)$ and by definition $res_R(\gamma) = res_R(\gamma')$

(ii) If $necred_R(\gamma) \neq \emptyset$ then $\gamma \equiv necred_R(\gamma) ; \gamma'$ where $\gamma' = \gamma[necred_R(\gamma)]$ and $\gamma \rhd^+ \gamma'$. But $res_R(\gamma) = res_R(\gamma') = res_R(\hat{\gamma'})$ (induction hypothesis) and (lemma 5-4) $\hat{\gamma} = necred_R(\gamma) ; \hat{\gamma'}$ thus $res_R(\hat{\gamma'}) = res_R(\hat{\gamma})$ ∎

Putting alltogether the results of this section, we get :

Correctness theorem 5-2

If R is a sequential system then the call-by-need computation rule is universally correct, that is

$$\forall t \in \overline{M}_F(X) \ \forall A \in \mathfrak{I}_R : Comp_{<R,A>}(t) = \{res_A(\gamma)/\gamma \in \mathcal{S}_R(t)\}$$

proof: one has only to prove this for the symbolic interpretation. But we have seen (lemma 5-6) that, since R is non-ambiguous, $\gamma \mapsto \hat{\gamma}$ is a surjection from $\mathcal{E}^{\infty}_R(t)$ to $\mathcal{S}_R(t)$ which, since R is sequential, preserves the (symbolic) results ∎

6. Conclusion

To conclude this work, let us take again and try to extend the historical picture. At the present time, we know that for our semantics of non-deterministic recursive definitions, to find an adequate

denotational approach is far from being trivial. There are various constructions of powerdomains [35,43,18,19] each suited to some formal or informal operational point of view, but w.r.t. our computational (and algebraic) approach, they are not adequate, see [7,8]. For example with the recursive definitions :

$$\varphi(x) \to \underline{or}\ (x,\ \varphi(a(x)))$$
$$\psi(x) \to \underline{or}\ (\xi,\ \varphi(x))$$
$$\xi \to a(\xi)$$

we have to distinguish $\varphi(x)$ and $\psi(x)$

The status of the full abstraction problem is somewhat analogous : we do not have a clear understanding of what is allowed to be finitely observable. For example, if the choice operator $\underline{or}$ is actually an explicit non-deterministic function, may we allow hardly discriminating contexts such as operators selecting "left choice" and "right choice" given by :

$$\gamma\ (\underline{or}(x,y)) \to x \quad \& \quad \delta\ (\underline{or}(x,y)) \to y\ ?$$

(with this non-deterministic context, we can finitely distinguish $\varphi(x)$ and $\psi(x)$ of the above example).

At least we may say that our computational approach gives some framework to understand these problems.

The future takes the form of a wide range of gaps (the reader might have yield some open questions along the lines of the paper). Here we just briefly look out some of them.

Certainly our notions of call-by-need and sequentiality are too restrictive. Technically it seems possible to include as sequential some overlaping TRS's (a good closure property w.r.t glb in the subsumption preorder have to be found). From a semantical point of view it remains to investigate the well-foundedness of these concepts. In the confluent case, do we define only stable functions [3] or even sequential algorithm [6], and again is there any proper extension to model non-determinism ? About implementation, all has to be done, with the

underlying question of optimality, as in [44,4].

Finally the most interesting prespective opened to the computational approach is perhaps to try to extend what have been done here to conditional rewriting systems : these are some kind of Horn clauses, which may be found when preconditions are required (e.g. in abstract data types) but above all they seem especially well-suited to specify operational behavious, as shown by the nice lecture of G.Plotkin [36].

Acknowledgement

I am greatly indebted to M.Nivat whose accurate interest in this matter gave to me opportunity and feeling up to write this paper.

References

[1] ADJ : "Initial Algebra Semantics and Continuous Semantics" JACM 24 (1977) 68-95

[2] A.Arnold, M.Nivat : "Formal Computations of Non-deterministic Recursive Schemes", MST 13 (1980) 219-236

[3] G.Berry : "Stable Models of Typed Lambda Calculi" , 5th ICALP, LNCS 62 (1978) 72-89

[4] G.Berry, J.J. Levy : "Minimal and Optimal Computations of Recursive Programs", JACM 26 (1979) 148-175

[5] G.Berry, J.J.Levy : Letter to the Editor, Sigact News 11 (1979)

[6] G.Berry, P.L.Curien : "Sequential Algorithms on Concrete Data Structures" TCS 20 (1982) 265-321

[7] G.Boudol : "Sémantique opérationnelle et algébrique des programmes récursifs non-déterministes", Thèse, Université Paris 7 (1980)

[8] G.Boudol : "Une sémantique pour les arbres non-déterministes", 6th CAAP, LNCS 112 (1981) 147-161.

[9] R.Burstall : "Design Considerations for a Functional Programming Language", Infotech State of the Art Conf. (1977).

[10] J.M.Cadiou : "Recursive Definitions of Partial Functions and their Computations", Ph.D.Thesis, Stanford (1972)

[11] P.Chew : "An Improved Algorithm for Computing with Equations", 21st FOCS (1980) 108-117

[12] A.Church : "The Calculi of Lambda Conversion", Princeton University Press (1941)

[13] B.Courcelle, M.Nivat : "Algebraic Families of Interpretations", 17th FOCS (1976) 137-146

[14] B.Courcelle, M.Nivat : "The Algebraic Semantics of Recursive Program Schemes", 7th MFCS, LNCS 64 (1978) 16-30

[15] B.Courcelle : "Infinite Trees in Normal Form and Recursive Equations Having a Unique Solution", MST 13 (1979) 131-180

[16] P.J.Downey, R.Sethi : "Correct Computation Rules for Recursive Languages", SIAM J. on Computing 5 (1976)

[17] I.Guessarian : "Algebraic Semantics", LNCS 99 (1981)

[18] M.C.B.Hennessy, G.Plotkin : "Full Abstraction for a Simple Parallel Language", MFCS 79, LNCS 74 (1979) 108-120

[19] M.C.B.Hennessy : "Powerdomains and Non-deterministic Recursive Definitions", 5th Int.Symp. on Programming, LNCS 137 (1982)

[20] M.Hoffman, M.O'Donnell : "Interpreter Generation Using Tree Pattern Matching", 6th POPL (1979)

[21] G.Huet : "Confluent Reductions : Abstract Properties and Applications to Term Rewriting Systems", JACM 27 (1980) 797-821

[22] G.Huet, D.Oppen : "Equations and Rewrite Rules : a Survey", SRI Report (1979)

[23] G.Huet, J.J.Lévy : "Call-by-need Computations in Non-ambiguous Linear Term Rewriting Systems", Rapport INRIA n.359 (1979)

[24] G.Kahn, G.Plotkin : "Domaines Concrets", Rapport INRIA n.336 (1978)

[25] R.M.Keller : "Denotational Models for Parallel Programs with Indeterminate Operators", in "Formal Description of Programming Concepts", F.J.Neuhold, Ed., North-Holland (1977) 337-366

[26] S.C.Kleene : "Introduction to Metamathematics", North-Holland (1952)

[27] J.J.Lévy : "Le problème du partage dans l'évaluation des lambda-expressions", 1er Coll. AFCET-SMF de Maths Appliquées, Palaiseau (1978) 139-154

[28] J.Mc Carthy : "A Basis for a Mathematical Theory of Computation", in "Computer Programming and Formal Systems", P.Braffort & D.Hirschberg, Eds, North-Holland (1963) 33-70

[29] R. MILNER : "Implementation and Application of Scott's Logic For Computable Functions", ACM Conf. on Proving Assertions about Programs, SIGPLAN Notice 7 (1972)

[30] M.Nielsen, G.Plotkin, G.Winskel : "Petri Nets, Event Structures and Domains", TCS 13 (1981)

[31] M.Nivat : "On the Interpretation of Recursive Polyadic Program Schemes", Symposia Matematica Vol XV (1975) 255-281

[32] M.Nivat : "Interprétation Universelle d'un Schéma de Programmes Récursifs", Informatica 7 (1977) 9-16

[33] M.Nivat : "Non-Deterministic Programs : an Algebraic Overview", Proc. IFIP Congress 80, North-Holland (1980)

[34] M.J.O'Donnell : "Computing in Systems Described by Equations"' LNCS 58 (1977)

[35] G.Plotkin : "A Powerdomain Construction", SIAM J. on Computing 5 (1976) 452-486

[36] G.Plotkin : "A Structural Approach to Operational Semantics", Daimi FN-19 Rep. Aarhus Univ. (1981)

[37] M.Polian : " Différents types de dérivations infinies dans les grammaires algébriques d'arbres", 6th CAAP, LNCS 112 (1981) 340-349

[38] J.C.Raoult, J.Vuillemin : "Operational and Semantic Equivalence between Recursive Programs", JACM 27 (1980) 772-796

[39] B.K.Rosen : "Tree Manipulating Systems and Church-Rosser Theorems", JACM 20 (1973) 160-188

[40] D.Scott : "Outline of a Mathematical Theory of Computation", Technical Mono PRG 2, Oxford (1970)

[41] D.Scott, C.Strachey : "Toward a Mathematical Semantics for Computer Languages", Technical Mono PRG 6, Oxford (1971)

[42] D.Scott : "The Lattice of Flow Diagrams", Symp. on Semantics of Algorithmic Languages, Lecture Notes in Mathematics 182 (1971) 311-366

[43] M.Smyth : "Powerdomains", JCSS 16 (1978) 23-26

[44] J.Vuillemin : "Correct and Optimal Implementation of Recursion in a Simple Programming Language", JCSS 9 (1974) 332-354

[45] J.Vuillemin : "Syntaxe, sémantique et axiomatique d'un langage de programmation", Thèse, Université Paris 7 (1974)

[46] G.Winskel : "Events in Computations", Thesis, Edinburgh (1980)

[47] G.Winskel : "Event Structure Semantics for CCS and Related Languages", Proc. 9th ICALP, LNCS 140 (1982) 561-576

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique