Formal Description of Programming Concepts - II
D. Bjørner (ed.)
North-Holland Publishing Company
© IFIP, 1983

Recursive Decomposition Ordering

Jean-Pierre JOUANNAUD
Centre de Recherche en Informatique de Nancy
Campus Scientifique, BP 239
54050G Vandoeuvre-lès-Nancy,FRANCE

Pierre LESCANNE
Centre de Recherche en Informatique de Nancy
and
Laboratory for Computer Science, Massachusetts Institute of Technology,
545 Technology Square
Cambridge, Massachusetts, 02139, U.S.A.

Fernand REINIG
Centre de Recherche en Informatique de Nancy
Campus Scientifique, BP 239
54050G Vandoeuvre-lès-Nancy,FRANCE

Abstract: The Recursive Decomposition Ordering, a simplification ordering on terms, is useful to prove termination of term rewriting systems. In this paper we give the definition of the decomposition ordering and prove that it is a well-founded simplification ordering containing Dershowitz's Recursive Path Ordering. We also show that the Recursive Decomposition Ordering has a very interesting incremental property.

Résumé: L'ordre récursif de décomposition est un ordre de simplification utilisé pour prouver la terminaison des systèmes de réécriture de termes. Dans cette communication, nous donnons la définition de l'ordre récursif de décomposition, nous prouvons qu'il est bien fondé, qu'il est un ordre de simplification et qu'il contient l'ordre récursif sur les chemins de Dershowitz. Nous montrons aussi que l'ordre récursif de décomposition a une intéressante propriété d'incrémentalité.

1. Introduction

Term rewriting systems are an important model for non deterministic computations [21]. Therefore, methods for proving termination of term rewriting systems can provide a method for proving termination in other areas of programming. Term rewriting systems have also become a major tool in many fields related to programming, like abstract data type specifications (e.g., to establish their completeness by the Knuth-Bendix superposition procedure [13, 6]), program verification, theorem provers, and decision procedures for equational theories [1, 5, 24]. The Knuth-Bendix completion algorithm completes a non confluent set of term rewriting rules into a confluent (uniquely terminating) one and is used to prove the equivalence of abstract data type specifications via consistency of theories [3, 8, 18]. The Knuth-Bendix completion algorithm requires a universal method for proving finite termination, as in Huet's proof [7]. In other words, if the termination of the final set of rules is proved using a noetherian ordering, this ordering must be sufficient to prove the termination property of all the intermediate sets of term rewriting rules generated by the algorithm. Unfortunately, Huet and Lankford have shown that the finite termination of term rewriting systems is undecidable [9]. Thus, it is impossible to find a universal procedure to check for finite termination of any system, and people have been forced to look for specific techniques (see [10] for a survey).

In that vein Guttag, Kapur, and Musser [4] proposed a method based on superposition of terms which is similar to that used by Knuth and Bendix to prove confluence. Here we are mostly interested in simplification orderings. These are orderings compatible with the structure of terms and which have the subterm property. Dershowitz established that simplification orderings are powerful tools for proving termination and proposed recursive path ordering [2] after Plaisted's recursive path of subterms ordering [20]. These methods use an ordering on the set of function symbols. In [15] and [16] a new ordering was used to prove simply the well-foundedness of the recursive path ordering when the ordering on function symbols is total. In [22] and [23] it was shown that a similar ordering could also be defined when the ordering on function symbols is only partial, capturing easily the case of terms with variables. This ordering is a well-founded simplification ordering which has the additional useful properties. First, it contains strictly the path recursive ordering. Second, it is monotonic with respect to the ordering on the function symbols, i.e., if one increases the ordering on function symbols then one increases the ordering on terms. We call the third major property incrementality: it is easy to find an expansion of the ordering on the function symbols when a given pair of terms needs to be ordered. This idea might be used in the Knuth-Bendix completion algorithm to build the required universal ordering in an incremental and automatic way as the set of rules is completed.

In the second section of this paper we give classical definitions and notations about terms and orderings [10]. In the case of a total ordering on function symbols the decomposition ordering is based on a decomposition of terms into three parts which are compared lexicographically [16]. In the case of a partial ordering on function symbols these decompositions are quadruples, and, instead of one decomposition for a term, a set of decompositions is associated with each term and comparisons of sets of decompositions provide the decomposition ordering [22, 23]. The third chapter is devoted to extending these concepts to ground terms (i.e., terms without variables). In Section 5 and Section 6 the decomposition ordering is proved to be a simplification ordering and a well-founded ordering. In Section 7 we prove that the decomposition ordering is more powerful than the recursive path ordering. An extension of the decomposition ordering to non-ground terms is given in Section 8. The incrementality property is illustrated in the conclusion.

2. Orderings and Terms

2.1 Set and Multiset Ordering

An ordering $<$ on a set E can be extended to the set $\mathcal{P}(E)$ of sets on E by:

$S \ll T$ iff $S \neq T$ and $\forall x\ (x \in S$ and $x \notin T) \Rightarrow \exists y\ (y \in T$ and $y \notin S$ and $x < y)$.

Intuitively, a multiset on E is an unordered collection of elements of E, with possibly many occurrences of given elements. A multiset can be seen as a mapping $E \rightarrow \mathcal{N}$ where $\mathcal{N}$ is the set of natural numbers. Let $\mathcal{M}(E)$ be the set of all the finite multisets on E, i.e., the multisets M such that their support $\{x \in E \mid M(x) \neq 0\}$ is finite. The empty multiset { } is the multiset such that $\{\ \}(x) = 0$, for all x in E. A set is a particular case of a multiset such that S(x) is 0 or 1. Usually multisets are written as lists $\{x_1, \dots, x_m\}$ with a straightforward interpretation. If M is a multiset, $x \in M$ means $M(x) >_{\mathcal{N}} 0$. An ordering on E can be extended to multisets [11] by

$M \ll N$ iff $M \neq N$ and $\forall y\ ([N(y) <_{\mathcal{N}} M(y)] \Rightarrow [(\exists x \in E)\ y < x$ and $M(x) <_{\mathcal{N}} N(x)])$.

The extension to sets is a particular case of the extension to multisets. If the ordering is well-founded on E the extensions are well-founded on $\mathcal{M}(E)$ and $\mathcal{P}(E)$.

2.2 Terms and Occurrences

In this paper we will deal with terms with fixed arity function symbols. But all the results can easily be extended to varyadic terms. Suppose a set F of function symbols and a function $ar: F \rightarrow \mathcal{N}$ is given. T(F,X) is the set of terms on F with variables in X. $s \in T(F,X)$ is either a variable or of the form $f(s_1, \dots, s_m)$ with $f \in F$ such that $ar(f) = m$ and $s_1, \dots, s_m$ are in T(F,X). $T(F,X,\square)$ is the set of box terms. A box term is either the symbol $\square$ or has the form $f(s_1, \dots, s_m)$ for $f \in F$ such that $ar(f) = m$ and there exists $i \in [1..m]$ with $s_i \in T(F,X,\square)$ and, for $i \neq j$, $s_j \in T(F,X)$. Intuitively, $T(F, X, \square)$ is the set of terms with one terminal occurrence of $\square$. The symbol $\square$ may be viewed as the empty term. It is used to deal with function symbols having fixed arities. If X is empty, we will write T(F) and T(F,X) instead of $T(F,\square)$ and $T(F,X,\square)$ and we call these terms, *ground terms.*

We assume by convention that $ar(\square) = ar(x) = 0$ for all $x \in X$. Terms may be viewed as labeled trees in the following way. A term is a partial function of $\mathcal{N}_+^*$ (the monoid over $\mathcal{N}_+$ with ε as empty word) in $F \cup X$, such that the domain or *set of occurrences* $Occ(t) = \{u \in \mathcal{N}_+^* \mid t(u)$ is defined$\}$ verifies:

(1) $\varepsilon \in Occ(t)$

(2) $ui \in Occ(f(\dots t_i \dots))$ iff $\forall i\ 1 \leq i \leq ar(f) \Rightarrow u \in Occ(t_i)$.

If u and v belongs to $\mathcal{N}_+^*$ then u/v is a $w \in \mathcal{N}_+^*$ such that $vw = u$. In the following, $|t| = |\{u \in Occ(t) \mid t/u \neq \square\}|$ where t/u is the subterm of t at the occurrence u. $t[u \leftarrow t']$ is the term obtained by replacing t/u by t' in t. We define the set of *paths of s* as the set $Path(s) = \{p \in Occ(s) \mid ar(s(p)) = 0\}$. Given a path p, the set of *prefixes of p in s* is $Prefix(s, p) = \{p \in Occ(s) \mid u \leq p\}$ if $s/p \neq \square$ and $Prefix(s, p) = \{p \in Occ(s) \mid u < p\}$ if $s/p = \square$. Given a path p and a prefix u of p, we define $succ(u, p)$ as ui if $ui \in Prefix(s, p)$ and $succ(p, p) = \infty$, we will state $t/\infty = \square$. thus $t/succ(p, p) = \square$. A *substitution* is a mapping $\sigma: X \rightarrow T(F,X)$ such that $\sigma(x) = x$ except for a finite number of variables $x \in X$. It can be extended to a mapping $\sigma: T(F,X) \rightarrow T(F,X)$ by $\sigma(f(s_1, \dots, s_m)) = f(\sigma(s_1), \dots, \sigma(s_m))$.

2.3 Simplification Orderings

An ordering $\prec$ on T(F,X) is a *simplification ordering* if it has the properties:

Subterm Property: $t \prec f(\ldots,t,\ldots)$.

Compatibility Property: $t_1 \prec t_2 \Rightarrow f(\ldots,t_1,\ldots) \prec f(\ldots,t_2,\ldots)$.

Dershowitz's Theorem [2]: A term rewriting system $R = \{g_i \rightarrow d_i \mid i \in \mathcal{I}\}$ with a finite number of symbols is finitely terminating if there exist a simplification ordering $\prec$ such that for all i in $\mathcal{I}$ and for all substitution σ, $\sigma(g_i) \succ \sigma(d_i)$.

3. Decomposition Ordering for Ground Terms

We define first the concept of elementary decomposition of terms in T(F, $\square$).

Definition 1: Given a term t, a path $p \in Path(t)$ and an occurrence $u \in Prefix(t,p)$, the *elementary decomposition* $d_u^p(t)$ of t in u along the path p is the quadruple $\langle g, t', \mathcal{T}, t'' \rangle$ where

$g = t(u)$,

$t' = t/succ(u,p)$,

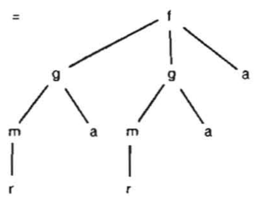
$\mathcal{T}$ is the multiset $\{t/uj \mid 1 \leq j \leq ar(t(u)), j \neq succ(u,p)\}$

t'' is the "box term" $t[u \leftarrow \square]$.

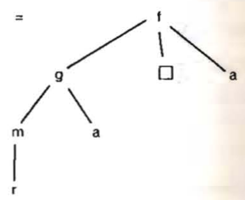
In the following, we never refer to the elementary decomposition in u where $t/u = \square$, thus we do not define such a decomposition.

<u>Example 1</u>: Let

t =



and s =



then $d_2^{211}(t) = \langle g; \begin{array}{c} m \\ | \\ r \end{array}; \{a\}; s \rangle$.

Assume now that a partial function Q: $T(F) \times \mathcal{N}_+^* \rightarrow \mathcal{P}(\mathcal{N}_+^*)$, called an *occurrence choice*, such that $Q(t,p)$ is defined if $p \in Path(t)$ and such that $Q(t,p) \subseteq Prefix(s, p)$ is given. We extend the previous definition to the set $Q(t,p)$ and we obtain the set $d_Q^p(t)$ (or more simply $d^p(t)$ if this is not ambiguous) that we call *path decomposition* of t along p:

$d_Q^p(t) = \{d_u^p(t) \mid u \in Q(t,p)\}$.

Note that for any Q and any p, $d_Q^p(\square)$ is the empty set.

In the same way, assume a partial function $P:T(F)\to\mathcal{P}(\mathbb{N}^*_+)$, called *path choice*, such that $P(t)\subseteq Path(t)$. We extend once more the definition and obtain a set of sets of decompositions, $d^P_Q(t)$ (or more simply $d(t)$ if no ambiguity on P and Q), that we call a *decomposition* of t:

$d^P_Q(t) = \{d^p_Q(t) \mid p\in P(t)\}$.

Example 2: If t is chosen as in Example 1 and if $P(t) = \{111,3\}$, $Q(t,111) = \{1,111\}$ and $Q(t,3) = \{\varepsilon\}$ then

$d(t) = \{ \{(\langle g; m(r) ; \{a\}; f(\Box, g(m(r), a), a)\rangle, s1)\ \langle r; \Box; \{\ \}; f(g(m(\Box), a), g(m(r), a), a) \rangle\}, \{f; a; \{g(m(r), a); g(m(r), a)\}; \Box\}\}$,

We are now able to define the decomposition ordering. Notice that in addition to the ordering $<_F$ on F this definition uses two other orderings $\overset{d}{<}_{QP}$ and $<_{QP}$. $\overset{d}{<}_{QP}$ is an ordering on T(F) and $<_{QP}$ is an ordering on decompositions which depends upon the choices P and Q. In order to simplify notations, the multiset extension of $<_{QP}$ and $\overset{d}{<}_{QP}$ will be written $\ll_{QP}$ and $\overset{dd}{\ll}_{QP}$ instead of $<_{QP}<_{QP}$ and $\overset{d}{<}_{QP}\overset{d}{<}_{QP}$.

Main Definition: Given a partial ordering $<_F$ on F, a path choice P and an occurrence choice Q, we define the *recursive decomposition ordering* (or more simply decomposition ordering) $\overset{d}{<}$ in the following way:

$s\overset{d}{<}_{QP}t$ iff $d^P_Q(s) \ll_{QP}\ll_{QP} d^P_Q(t)$

with

$d^p_u(s) = \langle f; s'; \mathcal{S}; s''\rangle \quad <_{QP} \quad d^q_v(t) = \langle g; t'; \mathcal{T}; t''\rangle$

iff in a lexicographical way:

(dec.1) $f<_F g$

(dec.2) $d_Q^{p/succ(u,p)}(s') \ll_{QP} d_Q^{q/succ(v,q)}(t')$

(dec.3) $\mathcal{S}\overset{dd}{\ll}_{QP}\mathcal{T}$

(dec.4) $d^u_Q(s'') \ll_{QP} d^v_Q(t'')$.

Remarks: 1) In general, we will have $p\in P(s)$, $u\in Q(s,p)$, $q\in P(t)$ and $q\in Q(t,q)$. Notice, however, that $d^p_u(s) <_{QP} d^q_v(t)$ has a meaning although $p\in Path(s)-P(s)$, $u\in Prefix(s,p) - Q(s,p)$, $q\in Path(t)-P(t)$, $v\in Prefix(t,q)-Q(t,q)$. We will use this fact in further proofs.

2) Cases (dec.2) and (dec.4) in the main definition do not use a path choice, because the path is fixed. Therefore it is not necessary to extend the concept of path choice for terms in $T(F,\Box)$.

Full examples are given in Appendix.

Theorem 1: $\overset{d}{<}_{QP}$ and $<_{QP}$ are strict orderings.

Proof: We prove the property for $<_{QP}$. It will be true for $\overset{d}{<}_{QP}$ which is a multiset extension of $<_{QP}$. The proof is easily done by induction, since the extensions of orderings as lexicographical ordering or set ordering preserve together irreflexivity and transitivity. ∎

4. Choices in Decomposition Ordering

The choice of P and Q in the previous definitions seems to be a main point. In this section, we study two possible choices: the first one consists of taking all the occurrences and paths, the second one consists of keeping only the maximum occurrences and paths (in a sense we will make precise later). The first one provides easier proofs and the second one leads to more efficient implementations, but they define the same decomposition ordering.

4.1 Entire Choice

Here we define two choices P_* and Q_*. They are called the entire choice because they correspond to choosing all the paths or occurrences in the term. P_* is defined by $P_*(t) = Path(t)$. Q_* is defined by $Q_*(t, p) = Prefix(t, p)$. We will write d^u_* and d^*_* for the associated sets and sets of sets, instead of $d^u_{Q_*}$ and $d^P_{Q_*}$. The associated ordering will be written $\overset{d}{<}_{**}$ and $<_{**}$.

In the following, we will use also the ordering $<_{*P}$ and $\overset{d}{<}_{*P}$ associated with the choices Q_* and P.

4.2 Maximal Choice

We define here the maximal choice which corresponds to selecting from among the paths and the occurrences the maximal ones.

Definition 3, *Maximal Paths*: The set $Mp(t)$ of the maximal paths of a term $t = g(t_1,\ldots,t_n)$ is defined by:

(1) $Mp(t) = \varepsilon$ if $n = 0$

(2) $Mp(t) = \bigcup_{i\in\mathcal{I}} i.Mp(t_i)$

such that $T = \{t_i \mid i\in\mathcal{I}\}$ is a minimal and complete set of maximal elements of $S = \{t_1,\ldots,t_n\}$, i.e.,

Minimality $(\forall t_i\in T)(\forall t_j\in T)\ \neg(t_i \overset{d}{\leq}_{**} t_j)$.

Completeness $(\forall t_j\in S)(\exists t_i\in T)\ t_j \overset{d}{\leq}_{**} t_i$.

Let now define P_+ and Q_+.

$P_+(t) = Mp(t)$

$Q_+(t, p) = \{v\in Prefix(t, p) \mid (\forall w)\ [w<v \Rightarrow \neg(t(v) \leq_F t(w))] \text{ and } [v<w\leq p \Rightarrow \neg(t(v) <_F t(w))]\}$

We will write $d^p_+(t)$ and $d^+_+(t)$ instead of $d^p_{Q_+}$ and $d^P_{Q^+_+}$. $\overset{d}{<}_{++}$ and $<_{++}$ will be the associated orderings.

4.3 Decomposition Ordering is Almost Independant of the Choices

Our aim now is to prove that $\overset{d}{<}_{++}$ and $\overset{d}{<}_{**}$ are the same ordering. More precisely, we want to prove that $\overset{d}{<}_{**}$ and $\overset{d}{<}_{QP}$ are the same ordering if the choices exhibit the following *minimality condition*:

$P(t) \supseteq P_+(t)$.

$Q(t, p) \supseteq Q_+(t, p)$ for all $p \in P_+(t)$.

Lemma 1: Let $p \in Path(s)$ and $u \in Prefix(s, p)$. Then there exists $u' \in Q_+(s, p)$ such that $d_u^p(s) \leq_{QP} d_{u'}^p(s)$, for any choice that satisfies the minimality conditions.

<u>Proof</u>: If $u \in Q_+(s, p)$ then $u' = u$.
If $u \notin Q_+(s, p)$ then there exits $u' \in Q_+(s, p)$ such that $s(u) \leq s(u')$.
If $s(u) < s(u')$ the result is true.
If $s(u) = s(u')$ then $u' < u$ and $s/succ(u, p)$ is a strict subterm of $s/succ(u', p)$. The result follows from (dec.2) and Lemma 2. ▮

Lemma 2: Let $p \in Path(s)$ and $k \leq p$. Then $d_Q^{p/k}(s/k) \leq\!\!\leq_{QP} d_Q^p(s)$, for any choices P and Q which satisfy the minimality condition. The inequality is strict if $k \neq \varepsilon$ that means s/k is a strict subterm of s.

<u>Proof</u>: By induction on $|p| - |k|$
Basic Case: If $|p| - |k| = 0$, then s/k is a term reduced either to $\Box$ then $d_Q^{s/k}(s/k) = \{\ \}$ or to the symbol $s(p)$ which occurs in s. The result will be true if a symbol greater or equal to $s(p)$ appears in the decomposition of s, that comes then from the minimality condition.

General Case: Let $u \in Q(s/k, p/k)$. For any decomposition of s/k in u along p/k, we want to find a greater decomposition of s in v along p. Two cases may happen:

- $ku \in Q(s,p)$. Then $v = ku$ works. The two decompositions are compared using case (dec.4) of the main definition applied to $d_Q^u(s/k[u \leftarrow \Box])$ and $d_Q^{ku}(s[ku \leftarrow \Box])$. The result is obtained from the induction hypothesis because the path ku in $s[ku \leftarrow \Box]$ has length $|ku| - 1 <_{\mathcal{N}} |p|$, the subterm occurrence being the same (that is k).

- $ku \notin Q(s, p)$. Thus there exists $v \in Q_+(s, p) = Q_+(t, p) \subseteq Q(s, p)$ such that $s(v) \geq s(ku)$. This case divides into two subcases:
  - $s(v) > s(ku)$, then the result is straightforward.
  - $s(v) = s(ku)$, then $v < ku$ and the result is obtained using case (dec.2) of the main definition applied to $d_Q^{p/k/succ(u,\, p/k)}(s/k/succ(u, p/k)$ and $d_Q^{p/succ(v,\, p)}(p/succ(v, p))$. On one hand, $p/k/succ(u, p/k)$ $= p/succ(ku, p)$ is a strict subterm of $p/succ(v, p)$ because $v < ku$. On the other hand,
  $$|p| - (|v| + 1) - [(|ku| + 1]) - (|v| + 1)] = |p| - |k| - |u| - 1 <_{\mathcal{N}} |p| - |k|.$$
  The wished result is thus true by using induction hypothesis. ▮

Proposition 1: $<_{QP}$ and $<_{*P}$ are the same ordering if the choices verify the minimality condition.

<u>Proof</u>: We prove $s <_{QP} t$ if and only if $s <_{*P} t$ by induction on $|s| + |t|$. Both $\Leftarrow$ and $\Rightarrow$ ways use Lemma 1 in order to delete the supplementary computation (for the $\Leftarrow$ way) or to add the missing ones (for the $\Rightarrow$ way). ▮

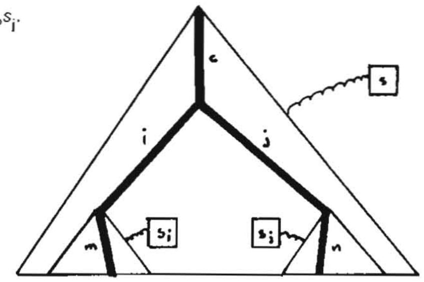
We now want to prove the equivalence of $<_{*P}$ and $<_{**}$. Once more the method is based upon a lemma which proves that the supplementary computation performed by $<_{**}$ is useless.

Lemma 3: Let s be a term in T(F) and p a path in s. If $p \notin P_+(s)$ then there exist $q \in P_+(s)$ such that:
$d_*^p(s) \ll_{*P} d_*^q(s)$.

Proof: By induction on $|s|$. If $p \in Path(s) - P_+(s)$, then there exists c, $m \in \mathcal{N}_+^*$ and $i \in \mathcal{N}^*$ such that:

- $p = c.i.m$.
- the subterm s/c belongs to a maximal class of the subterms at the depth c.
- $s_i = s/c.i$ does not belong to a maximal class of the subterm of s/c.

In addition there exists $j \in \mathcal{N}^*$ such that
$s_j = s/c.j$ and $s_i \not<_{*P} s_j$.



Therefore there exists $n \in P(s_j)$ such that $d_*^m(s_i) \ll_{*P} d_*^n(s_j)$. By induction, n can be supposed to belong to $P^+(s_j)$. Clearly $c.j.n$ belongs to $P_+(s)$. Let q be $c.j.n$, then the result
$d_*^{c.j.m}(s) \ll_{*P} d_*^{c.j.n}(s)$
is obtained by the following lemma. ∎

Lemma 4 Let s be a term in T(F) and c, n, m be in $\mathcal{N}_+^*$ and i,j be in $\mathcal{N}_+$ such that $c.i.n$ and $c.j.m$ are paths in s. If $d_*^m(s/c.i) \ll_{*P} d_*^n(s/c.j)$ then $d_*^{c.i.m}(s) \ll_{*P} d_*^{c.j.n}(s)$.

Proof:By induction on $|c| + |m|$. For all prefix u of $c.i.m$, one must find a prefix v of $c.j.n$ such that
$d_u^{c.i.m}(s) <_{*P} d_v^{c.j.n}(c)$.

Three cases can be distinguished:

1) $u<c$: then suppose $u = v$. The result is true by case (dec.2) by using the inequality
$d_*^{succ(u.c.i.m)}(s/succ(u,c.i.m)) \ll_{*P} d_*^{succ(u.c.j.m)}(s/succ(u,c.j.m))$
which is true by induction hypothesis applied to $s/succ(u, c)$ with $|c/succ(u, c)| + |m| <_{\mathcal{N}} |c| + |m|$.

2) $u = c$. Then $v = c$ and the result is true by case (dec.2) and the hypothesis $d_*^m(s/c.i) \ll_{*P} d_*^n(s/c.j)$.

3) $u<c$. Then there exists $h \in \mathcal{N}_+^*$ such that $u = c.i.h$ and $k \in \mathcal{N}_+^*$ such that: $d_h^m(s/c.i) \not<_{*P} d_k^n(s/c.j)$.

Let v be $c.j.k$ and let us prove $d_{c.i.h}^{c.i.m}(s) <_{*P} d_{c.j.k}^{c.j.n}(s)$. If (dec.1), (dec.2) or (dec.3) are used, the result is straightforward. If (dec.4) is used, that leads to prove $d_*^{c.i.h}(s[c.i.h \leftarrow \Box] \ll_{*P} d_*^{c.j.k}(s[c.j.k \leftarrow \Box]$ which results from $d_*^h(s/c.i[h \leftarrow \Box]) \ll_{*P} d_*^k(s/c.j[k \leftarrow \Box])$ by using the induction hvpothesis with $|c| + |h| <_{\mathcal{N}} |c| + |m|$. ∎

Proposition 2: $<_{*P}$ and $<_{**}$ are a same ordering if P verifies the minimality condition.

<u>Proof</u>: We prove $s \overset{d}{\lesssim}_{*P} t \Leftrightarrow s \overset{d}{\lesssim}_{**} t$ by induction $|s| + |t|$. Both $\Leftarrow$ and $\Rightarrow$ ways use lemma 3 in order to delete the supplementary computation (for the $\Leftarrow$ way) or add the missing ones (for the $\Rightarrow$ way). ∎

Theorem 2: If the choices confirm to the minimality condition, then the ordering $\overset{d}{\lesssim}_{QP}$ and $\overset{d}{\lesssim}_{**}$ are the same. In particular, the orderings $\overset{d}{\lesssim}_{++}$ and $\overset{d}{\lesssim}_{**}$ are the same.

<u>Proof</u>: We use successively Proposition 1 and Proposition 2. ∎

The definition of $\overset{d}{\lesssim}_{++}$ can now be made intrinsic, that means that instead of using $<_{**}$ in the definition of the maximal path and the maximal occurrence, we may use $<_{++}$ itself without changing the ordering as it is proved in [22].

In the following, we write $\overset{d}{\lesssim}$ for any decomposition ordering whose choices verify the minimality conditions. We will use $\overset{d}{\lesssim}_{**}$ for the most proofs.

5. Decomposition Ordering is a Simplification Ordering

Subterm Lemma: $t \overset{d}{\lesssim} f(\ldots,t,\ldots)$. <u>Proof</u>: By lemma 2. ∎

Compatibility Lemma: $t_1 \overset{d}{\lesssim} t_2 \Rightarrow f(\ldots,t_1,\ldots) \overset{d}{\lesssim} f(\ldots,t_2,\ldots)$.

<u>Proof</u>: By induction on $|f(\ldots,t_1,\ldots)| + |f(\ldots,t_2,\ldots)|$. Let $p \in Path(f(\ldots,t_1,\ldots))$ and $u \in Prefix(f(\ldots,t_1,\ldots),p)$. Two cases may happen.

Case 1: $p = kq$ and $f(\ldots,t_1,\ldots)/k \neq t_1$ or $u = \varepsilon$. We obtain easily the result $d_u^p(f(\ldots,t_1,\ldots)) < d_u^p(f(\ldots,t_2,\ldots))$ by using case (dec.4) of the main definition and the induction hypothesis.

Case 2: $p = kq$ and $f(\ldots,t_1,\ldots)/k = t_1$ and $u \neq \varepsilon$. Thus $q \in Path(t_1)$. As $t_1 \overset{d}{\lesssim} t_2$, there exists $q' \in Path(t_2)$ such that $(\forall\, v \leq q)\ (\exists\, v' \leq q')\ d_v^q(t_1) < d_{v'}^{q'}(t_2)$. If the proof of the last inequality is by case (dec.1), (dec.2) or (dec.3) of the main definition, the result is straightforward. If the proof is by case (dec.4) of definition, the result is achieved by using the induction hypothesis. ∎

Corollary: $\overset{d}{\lesssim}$ is a simplification ordering.

6. Decomposition Ordering improves over Recursive Path Ordering

Let us recall Dershowitz's definition of the Recursive Path Ordering [2].

Definition *Congruence of Permutation*: $f(s_1,\ldots,s_n) \overset{*}{=} g(t_1,\ldots,t_n)$ iff $f = g$ and there exists a permutation $\sigma \in S_n$ such that $s_i \overset{*}{=} t_{\sigma(i)}$.

Definition: The *recursive path ordering* over T(F) is recursively defined as follows:

$$s = f(s_1,\ldots,s_m) \overset{*}{\lesssim} g(t_1,\ldots,t_n) = t \qquad \text{iff}$$

(rpo.1) $f = g$ and $\{s_1,\ldots,s_m\} \overset{*}{\lesssim}\!\overset{*}{\lesssim} \{t_1,\ldots,t_n\}$

or (rpo.2) $f <_F g$ and for all s_i, $s_i \overset{*}{<} t$

or (rpo.3) $\neg f \leq_F g$ and for some t_j, $s \overset{*}{<} t_j$ or $s \overset{*}{=} t_j$

this definition can be made "less deterministic", by changing (rpo.3) to:

(rpo.3') for some t_j, $s \overset{*}{<} t_j$ or $s \overset{*}{=} t_j$

Theorem 3 (Dershowitz [2]): $\overset{*}{<}$ is a simplification ordering. If $<_F$ is is well founded on F, then $\overset{*}{<}$ is well-founded on T(F). If $<_F$ is total on F, then $\overset{*}{<}$ restricted to $T(F)/\overset{*}{=}$ is total.

We prove now that $\overset{d}{<}$ contains $\overset{*}{<}$. We first prove some technical lemmas, which prove actually that $\overset{d}{<}$ is a fixed point of the functional which defines $\overset{*}{<}$.

Lemma 5: $d^p(s_i) \ll d^q(g(t_1,\dots,t_n))$ and $f <_F g$ imply $d^{ip}(f(s_1,\dots,s_m)) \ll d^q(g(t_1,\dots,t_n))$.

Proof: By induction upon $|s_i|$.

Basic Case: $|s_i| = 0$, that is $s_i = \square$. The result is true because the only possible decomposition takes place in ε and because $f <_F g$.

General Case: Let $u \leq ip$. Two cases may be distinguished:

– $u = \varepsilon$ then $d_u^{ip}(f(\dots,s_i,\dots)) < d_\varepsilon^v(t)$ because $f <_F g$.

– $u = iv$, there exist $w \leq q$ such that $d_v^p(s_i) < d_w^q(t)$. If the proof is performed by case (dec.1), (dec.2) or (dec.3)of the main definition, then $d_{iv}^{ip}(s) < d_w^q(t)$ in the same way.

– If the proof is performed by case (dec.4), then we have : $d^v(s_i[v \leftarrow \square]) \ll d^w(t[w \leftarrow \square])$. By the induction hypothesis, we get $d^{iv}(s[iv \leftarrow \square]) \ll d^w(t[w \leftarrow \square])$, which proves the desired result by case (dec.4) of the main definition. ∎

Lemma 6: For all i, $s_i \overset{d}{<} t$ and $f <_F g$ implies $f(s_1,\dots,s_m) \overset{d}{<} t$.

Proof: Straightforward from Lemma 5. ∎

Lemma 7: $d^u(s_i) \ll d^v(t_j)$ implies $d^{iu}(f(\dots,s_i,\dots)) \ll d^{jv}(f(\dots,t_j,\dots))$.

Proof: By induction on $|s_i|$. We have to prove that for any $p \leq iu$ there exist $q \leq jv$ such that $d_p^{iu}(f(\dots,s_i,\dots)) < d_q^{iv}(f(\dots,t_j,\dots))$. Two case must be distinguished.

– $p = \varepsilon$, then $q = \varepsilon$. The result follows from the hypothesis using (dec.2).

– $p = ip'$. Then there exists q' such that $d_{p'}^u(s_i) < d_{q'}^v(t_j)$. If it is proved by (dec.1), (dec.2) or (dec.3) of definition, then the desired result is proved in the same way. If it is proved by case (dec.4), then we obtain $d^{p'}(s_i[p' \leftarrow \square]) \ll d^{q'}(t_j[q' \leftarrow \square])$ which proves the desired result by (dec.4). ∎

Lemma 8: $\{\dots,s_i,\dots\} \overset{d}{<}\overset{d}{<} \{\dots,t_j,\dots\} \Rightarrow f(\dots,t_i,\dots) \overset{d}{<} f(\dots,t_j,\dots)$.

Proof: By applying Lemma 7. ∎

Lemma 9: If $<_F$ is total on F, $\overset{d}{<}$ is total on $T(F)/\overset{*}{=}$.

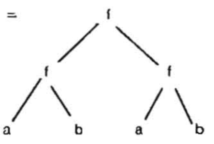
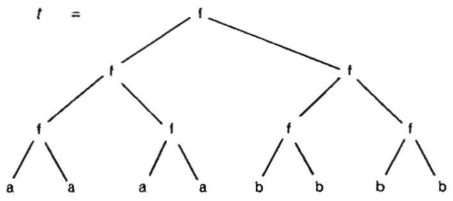
Proof: By induction on max(|s|, |t|). Suppose $<_F$ is total and neither $s \overset{d}{<} t$ nor $t \overset{d}{<} s$, let us prove that $s \overset{*}{=} t$. By the induction hypothesis, there exist in s a path p and an occurrence u such that $d^{u'}_{p'}(s) \overset{d}{\leq} d^{u}_{p}(s)$, for all other paths p' and occurrences u'. The same thing happens for q and v in t. Let d^{u}_{p} be ⟨f, s', $\mathcal{S}$, s''⟩ and $d^{v}_{q}(t)$ be ⟨g, t', $\mathcal{T}$, t''⟩. Because neither $s \overset{d}{<} t$ nor $t \overset{d}{<} s$ and by the induction hypothesis, f = g, $s' \overset{*}{=} t'$, $\mathcal{S} \overset{*}{=}\overset{*}{=} \mathcal{T}$ (where $\overset{*}{=}\overset{*}{=}$ is the congruence on multisets deduced from $\overset{*}{=}$) and $s'' \overset{*}{=} t''$. Then it is easy to see that $s \overset{*}{=} t$. ∎

Theorem 4: Given a partial ordering $<_F$ on F, we have $\overset{*}{<} \subseteq \overset{d}{<}$. If $<_F$ is total then $\overset{*}{<} = \overset{d}{<}$. Otherwise the inclusion is strict (whenever there exists a function symbol f∈F such that $ar(f) \geq 2$).

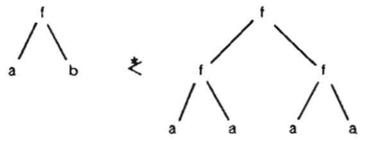
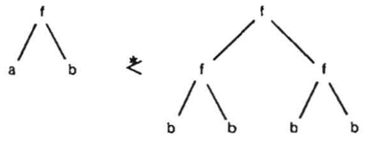
Proof: We can replace "iff" by "if" and $\overset{*}{<}$ by $\overset{d}{<}$ in the definition of recursive path ordering. By lemma 8 $\overset{d}{<}$ verifies (rpo.1), by lemma 6 $\overset{d}{<}$ verifies (rpo.2), by subterm lemma and transitivity $\overset{d}{<}$ verifies (rpo.3). Then $\overset{*}{<} \subseteq \overset{d}{<}$ is a consequence of the least fixed point property of $\overset{*}{<}$. That ends the first part of the proof.

To prove that $\overset{*}{<} = \overset{d}{<}$ if $<_F$ is total, we remark that both $\overset{*}{<}$ and $\overset{d}{<}$ are total ordering on $T(F)/\overset{*}{=}$ and do not compare terms s and t such that $s \overset{*}{=} t$. As $\overset{*}{<} \subseteq \overset{d}{<}$, we necessarily have $\overset{*}{<} = \overset{d}{<}$ in this case.

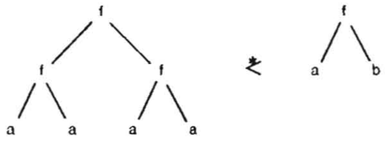
To prove that the inclusion is strict if $<_F$ is not total, we give a counter example. To build this counter example, we only need a binary function symbol f. Assume now a and b are incomparable and let

s =  and t = 

We assume without loss of generality that a and b are symbols of arity 0. If it is not the case we replace a by a(...,c,...) and b by b(...,c,...). It is not possible to compare s and t using $\overset{*}{<}$ because

 or 

is false. On the other hand,



is also false. However, we have $s \overset{d}{<} t$ because $d^{11}(s) < d^{111}(t)$ and $d^{12}(s) < d^{211}(t)$. ▮

7. Well foundednes of $\overset{d}{<}$

The well-foundedness is based on the following lemma.

Monotonicity Lemma: $\prec_F \subseteq <_F \Rightarrow \overset{d}{\prec} \subseteq \overset{d}{<}$.

Proof: Easy, see [22]. ▮

Theorem 5: $\overset{d}{<}$ is well-founded if and only if $<_F$ is well-founded.

Proof: Assume $\overset{d}{<}$ is not well-founded. Thus, there exists an infinité decreasing sequence $s_1 \overset{d}{>} s_2 \overset{d}{>} \dots \overset{d}{>} s_n \overset{d}{>} \dots$ Let now $\prec_F$ be a total well-founded ordering (i.e., a well-ordering) on F which contains $<_F$ (such an ordering exists by a variant of Zermelo's Theorem which can be seen as a transfinite topological sort). Using the monotonicity lemma, we obtain $s_1 \overset{d}{\succ} s_2 \overset{d}{\succ} \dots \overset{d}{\succ} s_n \overset{d}{\succ} \dots$ But $\overset{d}{\prec} = \overset{*}{<}$ by Theorem 3, which contradicts the well-foundedness of the recursive path ordering [2, 15]. ▮.

8. Extension of the decomposition orderings to non-ground terms

We will now define two formally different extensions of the decomposition ordering to non-ground terms. These two extensions are proved to be equivalent . The first one is more tractable for proofs, the second one leads to more efficient implementations. Moreover, these extensions are coherent with the definition of the decomposition ordering on ground terms, i.e.,

$$s \overset{d}{<} t \Rightarrow \sigma(s) \overset{d}{<} \sigma(t) \text{ for any substitution } \sigma.$$

Definition *by extension of the basic ordering*: Let $<_F$ a partial ordering on F. The decomposition ordering $\overset{d}{<}$ on T(F) is extended to T(F, X) by simply extending $<_F$ to F∪X in the following way:

$$a <_{F \cup X} b \text{ iff } a \in F,\ b \in F \text{ and } a <_F b.$$

In other words, the $<_{F \cup X}$ ordering is the same as $<_F$ for functions symbols. Variable symbols can be compared with no other symbols using $<_{F \cup X}$. The orderings $\overset{d}{<}$ and $<$ deduced from this definition will be written $\overset{d}{<}_1$ and $<_1$ for the time being. This definition of the decomposition ordering leads to inefficient computations. For instance, let us suppose that $s/p \in X$ and $t/q \in X$ and $s/p \neq t/q$. It is quite obvious in this case that the two decomposition sets $d^p(s)$ and $d^q(t)$ cannot be compared using the new $\overset{d}{<}$. However they will be recognized to be incomparable after a lot of useless computations. We give now a new definition of $\overset{d}{<}$ on T(F, X) which avoids this drawback. The basic idea is to modify the definition of the multiset extension $\ll$ in order to compare sets of decompositions $d^p(s)$ and $d^q(t)$ only when it is necessary.

Definition *by extension of the decomposition definition*: $d^p(s) \ll_X d^q(t)$ iff

(1) $s/p \notin X$ and $d^p(s) \ll d^q(t)$

(2) $s/p \in X$ and $t/q = s/p$ and $d^p(s) \ll d^q(t)$.

In the following, we write $\ll$ instead of $\ll_X$. Using this definition of $\ll$, it is now possible to decrease the size of the set $Q(s, p)$ of given occurrences in s along the path p, by ruling out the occurrences p if s/p is a variable.

Definition: $Q(s, p) \subseteq Prefix(s, p)$, if $s/p \notin X$

$Q(s, p) \subseteq \{u \in Occ(s) \mid u<p\}$, if $s/p \in X$

$Q_+(s, p) = \{v \in Prefix(s, p) \mid (\forall\, w<v), \neg(s(v) \leq s(w))\ \&\ (\forall v<w\leq p)\ \neg(s(v)<s(w))\}$, if $s/p \notin X$

$Q_+(s, p) = \{v<p \mid (\forall\, w<v)\ \neg s(v) \leq s(w)\ \&\ (\forall w>v)\ \neg s(v)<s(w)\}$, if $s/p \in X$.

Notice the analogy between the definitions of $Q(s, p)$ and $Q_+(s, p)$ when $s/p = \Box$ and when $s/p \in X$. The orderings $\overset{d}{<}$ and $<$ deduced from this definition will be written $\overset{d}{<}_2$ and $<_2$. It is clear that the ordering $\overset{d}{<}_1$ does not depend upon the choices P and Q, as stated by Theorem 2. But it is not so obvious for the ordering $\overset{d}{<}_2$. So we will prove that $\overset{d}{<}_1$ and $\overset{d}{<}_2$ are the same ordering, which will prove the property for $\overset{d}{<}_2$.

Theorem 6: $\overset{d}{<}_1 = \overset{d}{<}_2$.

Proof: Let us use the same choice for both orderings. In fact, the choice Q_1 of the ordering $\overset{d}{<}_1$ is not exactly the choice Q_2 of the ordering $\overset{d}{<}_2$, because if $s/p \in X$ then $p \in Q_1(s, p)$ and $p \notin Q_2(s, p)$. Both choices are the same in all the other cases.

Let us now prove that $s \overset{d}{<}_1 t \Rightarrow s \overset{d}{<}_2 t$, by induction on $|s| + |t|$. Let $p \in P_1(s)$ and $q \in P_1(t)$ such that $d^p(s) \ll_1 d^q(t)$. Two cases can occur.

Case 1: If $s/p = x \in X$ then $t/q = x$ because $p \in Q_1(s, p)$ and x is incomparable with any other symbol and therefore x must appear in a decomposition along the path q in t. Thus it is possible to compare $d^p(s)$ and $d^q(t)$ with $\ll_2$. Let $u \in Q_2(s, p)$. Then $u \in Q_1(s, p)$ and there exists $v \neq q$ and $v \in Q_1(t, q)$ such that $d^p_u(s) <_1 d^q_v(t)$. Using now the four cases of the definition and the induction hypothesis, we obtain $d^p_u(s) <_2 d^q_v(t)$ and thus $d^p(s) \ll_2 d^q(t)$.

Case 2: If $s/p \notin X$, the $Q_1(s, p) = Q_2(s, p)$. If $t/q \notin X$, there is no problem because $Q_1(t, q) = Q_2(t, q)$. If $t/q \in X$ then $q \in Q_1(t, q)$ and $q \notin Q_2(t, q)$. However, $d^p_u(s) <_1 d^q_v(t)$ implies that $v \neq q$ because $t(q)$ is not comparable with $s(u) \notin X$. The result is easily obtained by induction as in Case 1.

Let us prove now that $s \overset{d}{<}_2 t$ implies $s \overset{d}{<}_1 t$ by induction on $|s| + |t|$. In the same way as before, $d^p_u(s) <_1 d^q_v(t)$ follows from $d^p_u(s) <_2 d^q_v(t)$ if $s/p \notin X$. If $s/p \in X$ and $t/q \in X$, we have to prove the inequality:

$$\langle s(p); \Box; \{\,\}; s[p \leftarrow \Box] \rangle <_1 \langle t(q); \Box; \{\,\}; t[q \leftarrow \Box] \rangle$$

and $d^p(s[p \leftarrow \Box]) \ll_1 d^q(t[q \leftarrow \Box])$ follows in the same way as before from $d^p(s) \ll_2 d^q(t)$. ∎

Notice that all theorems proved in previous sections remain valid because of the definition $\overset{d}{<}_1$.

Theorem 7: $\overset{d}{<}$ is closed under instantiation, i.e., $s \overset{d}{<} t \Rightarrow \sigma(s) \overset{d}{<} \sigma(t)$, for any substitution σ.

Proof: Straightforward using definition by extension of the decomposition definition. ∎

9. Conclusion

A major advantage of the decomposition ordering is its utility in easily building, from a set of rewrite rules to be oriented, an ordering on the set F of function symbols. We will illustrate this property with an example from Dershowitz [2, 20], a system which provides normal disjunctive forms of propositional expressions:

(i) $\neg\neg x \rightarrow x$

(ii) $\neg(x \vee y) \rightarrow \neg x \wedge \neg y$

(iii) $\neg(x \wedge y) \rightarrow \neg x \vee \neg y$

(iv) $x \wedge (y \vee z) \rightarrow (x \wedge y) \vee (x \wedge z)$

(v) $(y \vee z) \wedge x \rightarrow (y \wedge x) \vee (z \wedge x)$.

The termination of the rule (i) is immediate by the subterm property. Let us prove the termination of the rule (ii). That means

$$s = \neg x \wedge \neg y \overset{d}{<} \neg(x \vee y) = t.$$

The decompositions are:

$$d_*^{11}(t) = \{\langle\neg; x \vee y; \{\ \}; \square\rangle, \langle\vee; x; \{y\}; \neg\,\square\rangle\}$$
$$d_*^{12}(t) = \{\langle\neg; x \vee y; \{\ \}; \square\rangle, \langle\vee; y; \{x\}; \neg\,\square\rangle\}$$
$$d_*^{11}(s) = \{\langle\wedge; \neg x; \{y\}; \square\rangle, \langle\neg; x; \{\ \}; \square \wedge \neg\, y\rangle\}$$
$$d_*^{21}(s) = \{\langle\wedge; \neg x; \{y\}; \square\rangle, \langle\neg; y; \{\ \}; \neg x \wedge \square\rangle\}$$

Then we will get $d_*^{11}(s) \overset{dd}{\ll} d_*^{12}(t)$, and $d_*^{21}(s) \overset{dd}{\ll} d_*^{12}(t)$, only if $\wedge <_F \neg$ or $\wedge <_F \vee$. By exchanging the symbols $\wedge$ and $\vee$ we will get the condition "$\vee <_F \neg$ or $\vee <_F \wedge$" from (iii). Let us now orient the rule (iv).

$$s = (x \wedge y) \vee (x \wedge z) \overset{d}{<} x \wedge (y \vee z) = t$$
$$d_*^{11}(s) = \{\langle\vee; x \wedge y; \{x \wedge z\}; \square\rangle, \langle\wedge; x; \{y\}; (\square \wedge y) \vee (x \wedge z)\rangle\}.$$
$$d_*^{12}(s) = \{\langle\vee; x \wedge y; \{x \wedge z\}; \square\rangle, \langle\wedge; y; \{x\}; \ldots\rangle\}$$
$$d_*^{21}(s) = \{\langle\vee; x \wedge z; \{x \wedge y\}; \square\rangle, \langle\wedge; x; \{z\}; \ldots\rangle\}$$
$$d_*^{22}(s) = \{\langle\vee; x \wedge z; \{x \wedge y\}; \square\rangle, \langle\wedge; z; \{x\}; \ldots\rangle\}$$

and

$$d_*^{1}(t) = \{\langle\wedge; x; \{y \vee z\}; \square\rangle\}$$
$$d_*^{21}(t) = \{\langle\wedge; y \vee z; \{x\}; \square\rangle; \langle\vee; y; \{z\}; \ldots\rangle\}$$
$$d_*^{21}(t) = \{\langle\wedge; y \vee z; \{x\}; \square\rangle; \langle\vee; z; \{y\}; \ldots\rangle\}.$$

In order to get $d_*^{11}(s) \overset{dd}{\ll} d_*^{1}(t)$ and $d_*^{21}(s) \overset{dd}{\ll} d_*^{1}(t)$ we need $\vee <_F \wedge$. This condition provides successfully the comparison of s and t. The rule (v) can be oriented by the same condition. From those conditions we get easily the following ordering on F: $\vee <_F \wedge <_F \neg$. Such a process can obviously be performed starting from a given partial ordering on F.

This property of the recursive decomposition ordering which leads to the automatic construction of the right ordering $<_F$ on the function symbols is a consequence of our definition when two symbols f and g are incomparable. In that case the two decompositions $\langle f; s'; \mathcal{S}; s''\rangle$ and $\langle g; t'; \mathcal{T}; t''\rangle$ are incomparable. Thus the comparison process stops whenever two such decompositions are required to be comparable. The idea is then to add at this step the pair $\langle f,g\rangle$ to the ordering $<_F$ in order to get comparable decompositions. Such a technique does not work with the recursive path ordering because the comparison fails when it exhausts one of the two terms. Because of this essential feature, our ordering is more suitable than Dershowitz's to any application which requires automatic proofs of termination. Our ordering is thus useful in implementing the Knuth-Bendix completion algorithm. A non-incremental version of the decomposition ordering is now implemented and we are currently implementing the incremental one.

Acknowledgment: We would like to thank Nachum Dershowitz and Jean-Luc Remy for their helpful suggestions and John Guttag for reading the manuscript.

10. References

1. Boyer R. S., Moore J S., *A Computational Logic*, Academic Press (1979).
2. Dershowitz N., *Ordering for Term Rewriting Systems*, Proc. 20th Symposium on Foundations of Computer Science (1979), 123-131.
3. Goguen J.A., *How to prove Algebraic Inductive Hypothesis Without Induction*, 5th conf. on Automated Deduction, Lecture Notes in Computer Science, 87 (1980), 356-373.
4. Guttag J.V., Kapur D., Musser D.R., *On Proving Uniform Termination and Restricted Termination of Rewriting Systems*, 9th Int. Coll. on Automata, Languages and Programming, Aarhus, Denmark (1982).
5. Hsiang J., *Refutational Theorem Proving using Term Rewriting Systems*, Dept of Computer Science, University of Illinois at Urbana-Champaign (1981).
6. Huet G., *Confluent Reduction: Abstract Properties and Applications to Term Rewriting Systems*, J. ACM, 27 (1980), 797-821.
7. Huet G., *A complete proof of correctness of the Knuth-Bendix Completion algorithm*, J. Comp. Sys. Sc., 23 (1981), 11-21.
8. Huet G., Hullot J., *Proof by Induction in Equational Theories with Constructors*, Proc. 21th Symposium on Foundations of Computer Science (1980).
9. Huet G., Lankford D.S., *On the Uniform Halting Problem for Term Rewriting Systems*, Rapport Laboria 283, IRIA, Mars 1978, INRIA Rocquencourt, France.
10. Huet G., Oppen D.C., *Equations and Rewrite Rules: a Survey*, in *Formal Languages perspectives and Open Problems*, Ed. Book R., Academic Press (1980).
11. Jouannaud J.P., Lescanne P., *On Multiset Orderings*, to appear in Inform. Proc. Ltrs.
12. Kamin S., Lévy J.J., *Attempts for generalizing the Recursive Path Ordering* to appear.
13. Knuth D.E., Bendix P., *Simple Word Problems in Universal Algebra*, in *Computational Problems in Abstract Algebra*, Ed. Leech J., Pergamon Press (1970), 263-297.
14. Lescanne P., *Two Implementations of the Recursive Path Ordering on Monadic Terms*, 19th Annual Allerton Conf. on Communication, Control, and Computing, Allerton House, Monticello, Illinois (1981), 634-643.
15. Lescanne P., *Some properties of Decomposition Ordering*, Symposium AFCET "The Mathematics for Computer Science", Paris, (march 16-18, 1982).
16. Lescanne P., *Decomposition Ordering as a Tool to prove the Termination of Rewriting Systems*, 7th IJCAI, Vancouver, Canada (1981), 548-550.

17. Lescanne P., Reinig F., *A Well-Founded Recursively Defined Ordering on First Order Terms*, Centre de Recherche en Informatique de Nancy, France, CRIN 80-R-005 (1980).

18. Musser D.L., *On Proving Inductive Properties of Abstract Data Types*, Proc. 7th ACM Symposium on Principles of Programming Languages (1980), 154-162.

19. Plaisted D., *Well-Founded Orderings for Proving Termination of Systems of Rewrite Rules*, Dept of Computer Science Report 78-932, University of Illinois at Urbana-Champaign, July 1978.

20. Plaisted D., *A Recursively Defined Ordering for Proving Termination of Term Rewriting Systems*, Dept of Computer Science Report 78-943, University of Illinois at Urbana-Champaign, Sept. 1978.

21. Raoult J-C., Vuillemin J., *Operational and Semantic Equivalence Between Recursive Programs*, J. ACM., **27** (1980), 772-796.

22. Reinig F., *Les Ordres de Décomposition: un outil incrémental pour prouver la terminaison finie de systèmes de réécriture de termes*, Thèse Université de Nancy, Octobre 1981.

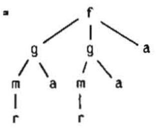
23. Reinig F., Jouannaud J.P., *Decomposition Orderings: a new family of decompostion orderings*, Centre de Recherche en Informatique de Nancy, France, CRIN 81-R-040.

24. Thompson D. H., ed, *AFFIRM Reference Manual*, USC Information Science Institute (1979).

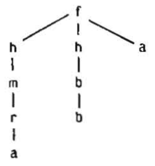
APPENDIX: Examples of decompositions

Let $f < g < h$ and $a < b$.

$s =$



$t =$

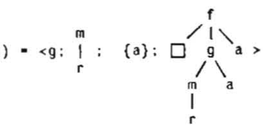


$P(s) = \{111, 12\}$
$Q(s, 111) = \{1, 11, 111\}$
$Q(s, 12) = \{1, 12\}$

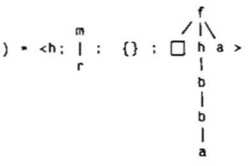
$P(t) = \{111, 2111\}$
$Q(t, 111) = \{1, 11, 111\}$
$Q(t, 2111) = \{2, 21\}$

$d^{111}_{Q}(s)$ $\qquad$ $d^{111}_{Q}(t)$

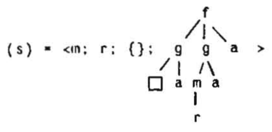
$d^{111}_{1}(s) =$



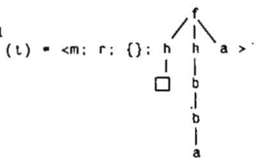
$d^{111}_{1}(t) =$



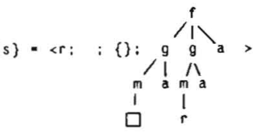
$d^{111}_{11}(s) =$



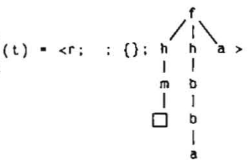
$d^{111}_{11}(t) =$



$d^{111}_{111}(s) =$



$d^{111}_{111}(t) =$



```
             12                                    2111
            d  (s)                                d     (t)
             Q                                     Q


                                     f                   b            f
   12                     m        / | \     2111        |          / | \
  d  (s) = <g; a; {|};  □    g    a  >      d     (t) = <h; b; {}; h   □   a >
   1                      r        / \        2              |       |
                                  m   a                      a       m
                                  |                                  |
                                  r                                  r


                              f                                        f
   12                       / | \            2111                    / | \
  d  (s) = <a; ; {}; g     g    a >         d     (t) = <b; b; {}; h   h   a >
   12               / \   | \                21            |       |   |
                   m   □  m  a                             a       m   □
                   |      |                                        |
                   r      r                                        r
```

SESSION DISCUSSION

Responder: G.Cousineau, Paris

Cousineau: I wish to address the problem of implementation. My reason for that is that I am involved in a project which implements a system which manipulates mathematical theories and programs. So we should normally benefit from the kind of work reported here. °°° The last speaker claimed he had produced an ordering which was much more powerfull than recursive path ordering, and if true this will have very great importance in connection with Knuth-Bendix completion. °°° So we must state precisely the problem in order to discuss this problem of implementation of the recursive decomposition ordering and see what we can do with it. °°° Our system deals, among other things with equational theories, and among things it can do, it can do Knuth-Bendix completion, Peterson-Stickel completion, and also prove equalities in the initial algebra. So the first two are certainly related to the last two papers, because in doing Knuth-Bendix competion, we do ordering on terms, this is crucial, and the third point is certainly related to the talk of Padawitz. °°° So let me recall: what is Knuth-Bendix *(KB)* completion *(KBC)*. KBC is going from an equational axiom system *S* to a rewriting system *R* which is canonical in the sense that we can check equality in the system *S* only by rewriting rules, and see whether we come to the same normal form. To do KBC: -- we recall the method: Let us just look at the completion method. We start with a set of equations. We have first to orient the equations in such a way that the resulting system is Noetherian. So we have the problem of proving termination. Then we compute critical pairs of the system. If there are none, or if they are just such that their left and right members have the same normal form then the obtained system is canonical, and we are finished. And for those whose members have different normal forms, we orient them using again the recursive path ordering. And then add them to the system and iterate the process. Now, to orient the rule and prove that the rewriting system obtained is terminating you need some ordering, that was the subject of the last talk, and the ordering that we have implemented is recursive path ordering. So here (Cousineau points: ¢) I just recall the definition of recursive path ordering. You start with an order on symbols, and you define recursively the order on terms using the definition that Jouannaud has presented. °°° It happens that we have applied KBC to a lot of classical mathematical structures including groups, rings, modules over rings, etc. And that to be able to do ordering of terms we had to use not only the recursive path ordering of Derschowitz, but an extension that was defined by Kamin & Lévy. And this is crucial. In the Kamin and Lévy extension of recursive path ordering each function symbol can be of either type multiset or lexicographic. In the definition we have more choice. The definition is the following: if the head symbols of the two terms are ordered by the order on symbols, then the definition is the same as for recursive path ordering. But if the head symbols of two terms are the same, then we can compare sub-terms either in the multiset way or in the lexicographic way. This is crucial if, for example, you want to deal with associativity as a rewrite rule. So, now we would like to understand what we could gain by using recursive decomposition ordering instead of recursive path ordering. So my first question to Jouannaud would be: is the recursive decomposition ordering also an extension of this Kamin and Lèvi ordering?

Jouannaud: I guess that you mean: Am I able to define a lexicographic extension? OK. Up to now I did not succeed, but I only worked half a day on that problem. But it seems to me that it is not so easy. Because, using paths maybe cannot fit with lexicographic ordering. But I am not sure.

Cousineau: OK. So for the moment we must be careful. We know that recursive path ordering with is (¢) definition works for a lot of structures, but we are still not sure whether recursive decomposition ordering will work.

Jouannaud: I know that it can be used, not with this definition, but with another one which is much simpler. With this simple definition, I can have a lexicographic extension, but it is not as powerful as the recursive decomposition ordering.

Cunningham: Could we have a source?

Cousineau: You should write to Jean-Jacques Levy at INRIA. ᵒᵒᵒ My second point is that we do not want to do only KBC, but also Peterson-Stickel completion. That means: we want to be able to deal with associative and commutative theories. The problem here is that if you have commutativity then you can no more have a terminating rewriting system. So you have to deal in a different way with commutativity. And, moreover, when symbols are commutative you cannot deal with associativuty the way you did before. So you have to deal with associative and commutative symbols in a special way. And that is done by an extension to the KB algorithm which is due to Peterson and Stickel. Now we do not manipulate terms, but terms modulo commutativity and associativity of certain symbols. One good point is that we have been able to extend the recursive path ordering to deal with associativity and commutativity. And the way to do this is very simple. When we have associative and commutative symbols, for example, let us take binary + (plus), instead of representing it as a binary (dyadic) symbol, we represent it as a vari-adic symbol, with a list of arguments, and we apply the recursive path ordering defined here (¢). The crucial point for doing that is that the recursive path ordering is compatible with the associativity and commutativity property. That is: if two terms are equivalent modulo associativity and commutativity, they are equivalent in the equivalence associated with the recursive path ordering. So I come to my second question. Do you think we can also extend the recursive decomposition ordering to associative and commutative theories?

Jouannaud: My answer in this case is: Yes, don't ask me for the paper. Because it has'nt been written. But there is no problem in building any equational theory into the recursive decomposition ordering. It is valid not only for associative and commutative theories, but for any.

Cousineau: Maybe I have alast point about this paper. And that is the problem of implementing this order. My experience with recursive path ordering is very good. It took me one day and one night to put the recursive path ordering in our system including the user interface. It is an order which is very simple to program. What is your opinion about the implementation of recursive decomposition ordering?

Jouannaud: It is very easy, because in LISP you can use functionals as arguments. But in that case, you get some kind of "a brute force" implementation. If you want to get a really good implementation, then you have to do much more.

Wagner: I am actually interested in the connection between term-rewriting systems and the work on abstract data types, because of the question of implementing (abstract) data types. Most of the systems that I am aware of have problems because they could'nt handle such things as commutativity. But I gather that that has now been resolved. But does it actually provide an effective, reasonable system.

Cousineau: Yes, very reasonable. And if you are interested I have here the listing of sessions of the system dealing with associativuty and commutativity. You can have a look.

Wagner: Can you actually, in effect, run programs in this system? How far up, how I am also interested in the speakers comment on this, by the way — how they feel.

Padawitz: We have no system. But I think the point with commutativity and associativity is that it is not enough for data types. If you remember my example with arrays, there is an axiom which looks like commutativity, but it is not real commutativity. For such examples the methods which incorporate commutativity and associativity don't work. They do not work for equations which are a little bit different from "classical" commutativity axiom. This was one reason for me to introduce the notion of relative confluence with respect to some rules which may be, for example, associativity and commutativity axioms, but it may be also more than these. There is no requirement to the basic rules except relative confluence. Especially the basic rules do not need to be normalizing, or normalizing modulo something.

Jouannaud: I can also answer this question. In my "talk" I mentioned two systems. System REVE which is implemented by Pierre Lescanne, and the system FORMEL - and the system FORMEL is exactly the system of Guy Cousineau and Gerard Huet. So I practiced on this system. I want to say that it is a very powerful system. You can perform proofs with this system. I have used it in a quite different way, which was: we had to prove that an axiomatization was not minimal. So, in fact we suspected one axiom to be a logical consequence of the others. And we effectively proved it using this system, by removing the axiom from the set of axioms, arranging the others as a set of rules and computing with several hundred new rules. And after maybe half an hour MULTICS system computation, we had our axiom generated by the system. So it is a very powerful system which can be used for purposes of program proof, but also for many other purposes. I think it is very important to have such systems.

Wagner: Do you think you can debug specifications?

Jouannaud: Yes, sure. You know the work of Musser. You can prove (in)consistencies with these systems. Using KBC you can prove (in)consistencies. So it is debugging specifications.

Wagner: That is a very sophisticated form of debugging.

Jouannaud: Yes, it was a special case, but it works with this case.

Cunningham: With some trepidation I ask the responder: is he aware of any work that extends these techniques to partial algebras.

Cousineau: No, in fact, that was also one of my questions to the second author. The only thing we know how to do presently in our system for initial algebras is using the method of Huet and Hullot which was published in FOCS, I think in 1980. This works in the case where you can partition your sets of symbols between what is called constructor defined symbols and the other which are called differencing rules. In the set of axioms which are satisfying some definition principles. The definition principle is, in a few words: any ground term is equivalent only to some ground term using only constructors — as one point — and the second point: two ground terms using only constructors are equal in the theory if they are identical. So in that case we can proved that an equation $t=e$ is satisfied in the initial algebra by doing the completion method. And that is a question I would like to ask to the second speaker: have you been interested in implementing such methods, and do you think that this kind of method could be adapted to your notion of partial algebra? Of course, if you introduce an undefined symbol, i.e. it can certainly not be a constructor, but maybe we could replace the condition on constructors - with another condition which is less strict. I mean: instead of requiring that two terms are equivalent if they are identical, we could ask just for a confluent, terminating rewriting system for this ground term with constructors.

Padawitz: Two points: first to the partial functions. The aim of introducing a discrete specification which has a least element is to allow correctness proofs of the recursive, partial functions with respect to a discrete domain. If you have proved this correctness, then you can forget the discrete specifications. The discrete specification is only there to get the proof: to see that the discrete domain is really specifiable. The other point is the following: The conditions I give for discrete specifications are also conditions which are sufficient for what is called completeness and consistency of data types with respect to some basic data type. So one has two specifications: one is included in the other, and one wants to prove that the semantics of the enriched specification includes the semantics of the basic specification. To get this property I have similar sufficient conditions like confluence and normalization conditions which guarantee completeness and consistency. Referring to the definition principle mentioned by the responder, the constructors are just the operations of the basic specification. The definition principle then says that every term can be reduced to constructors. And if constructor terms are equivalent, then they are identical. This condition, I think, is too restrictive for the following reason: one reason is completeness: this is just the property that you can reduce every term to a constructor term. The other reason is consistency: this means that terms of constructors which are congruent in the extended specification are already congruent in the basic specification. The definition principle would require that they are already equal. But this is sometimes a too restrictive.

Culik: I have a question to Bergstra: Different equivalences (coming, for instance, from Manna's book, you mentioned unfolding, and he has also isomorphism equivalence) can be just treated somehow. In your point of view there are special cases. I would like to know something more. While function equivalence is undecidable, the isomorphism equivalence is decidable, and therefore it is much more suitable, for instance, in proving the correctness of a compiler. Can you be more specific about the relationships among them, e.g. the process equivalence, is (it) really decidable, and therefore also execution sequences? And, because *we* are constructing the compiler we can always restrict ourselves to this decidable case. But, however decidable, it still can be unfeasible. Can you comment on that?

Bergstra: That is not so easy for me. I have not said anything about feasibility. And this is, maybe, the key point (of course). So I assume this kind of isomorphism equivalence is within the scope of these methods. What gets lost is the modularity of the issue. And this is already a problem in pencil-and-paper work. So mathematically it is within the scope of our method.

Culik: Well, I assumed you could tell us immediately how to formulate isomorphism in the Manna sense (PS: there is an incaccuracy in his definition) in your framework, because it would help me to understand that.

Bergstra: Well, that is not my interest. I want a general theory, not nice transformations.

Cousineau: (to Bergstra:) How should we take your paper. Should we take it as a theoretical result relating two different proof systems, or proof methods for programs, or should we take it as something that could help, in some sense, in doing the proofs?

Bergstra: So you are asking me: what the applications of this kind of work would be? Now, of course, I prepared myself for that question, because you said you would ask it. (Laughter) In the time I worked together with John Tucker we had long discussions on the applicability of our work. And we came to the conclusion that among the many kinds of applications that one can imagine one application is the application to teaching — which is big business too. The clear mathematical analysis that we aim at could have an application to teaching. Just explaining these notions and their relations. If it has a spin-off in terms of providing efficient proof systems, then it is not up to me to predict this in any way. I would recall the fact that combinatory logic was invented to get around set theory, and is now applied to get around sequential programming. Schönfinckel would not have predicted that presumeably. So — I am not ranking our work to these kind of issues — but nevertheless

I am making the point that on principle I have not had the idea that in whatsoever way this will lead to applicable systems. I think that these mathematical facts are such that their main application (is) in the development of the field and in teaching. Because in teaching itself it is not so essential that everything has an application. Is that an answer?

Cousineau: Yes, Oh yes.

Culik: I was teaching a program correctness course to graduate students 2 years ago using Manna's book. I could lecture on Greatest Common Divisor all-right, and prove it correct. But when challenged, by my students, to produce other examples, I had to say: Sorry, but I did not prepare myself — constructing proper assertions, and so on. ooo We are in this business faced, ultimately, with proving correctness of real programs. And I do not think we have succeeded in doing so. This is the real difficulty. Even for teaching, the practicability of proof was, and is, so disappointing. I have very deep doubts that it can be overcome. We are here (w.r.t. Bergstra's talk) on the second floor: in the meta-theory. Trying to formalize proofs. We are forgetting (mathematically) about the difficulties, the real difficulties in problem-solving. It just disappeared somehow. But if you are thinking about real problems, this difficulty will be there. And my opinion is that there is no way to believe that all these formalized proofs will succeed. It failed in Mathematics 50 years ago. And therefore we should look for something else.

Reynolds: In response to the question that was just raised. I have been teaching a course in programming for graduate students for a decade now in which I have always presented considerable material on program proving in the style of Hoare. And my answer to your objections, which I have also heard from my students as well, is that we do not reasonable expect a professional programmer to formally prove every program that he writes, but (we) expect him to understand formal proofs as ground work for the intuition needed for producing actual, informal proofs. When a mathematician is presenting a proof he will write down some formulae. He will certainly not proceed to give the details of the formal proof at the level that: here we use commutativity, and we here we use associativity, and there distributivity, and so on. But he understands how to do that. Everyone in his audience understands how to do that. And there is a common agreement that given a little bit of good will, he won't try to pull the wool over anybody's eayes. It is an entirely different thing when one starts to fiddling such equations, without knowing such things as distributivity or associativity or commutativity, to an audience with the same characteristics. Then one can very easily get informal proofs of quite incorrect programs. Anytime one programmer walks into another and says: why does this program work, the response he gets is a proof. And the problem is to train programmers in such a way that they do not convince each other that incorrect programs are correct. And formal proofs help in that. Not because people are going to do formal proofs of every program, they write, but because it gives them the necessary ground work to build up intuitions about how much they have to think, and in what areas they have to think, about whether their programs are right or not.

Bergstra: May I add one more comment (to my previous remarks): Of course, the prime difficulty in many cases is to find intermediate assertions, invariants, and so on. But in our paper we provide a calculus where these things are just treated as variables. So you just have *R1* which is a variable which stands for an assertion, *R2* and so on. And then one gets a calculus about assertions. And the nice thing is that one gets to conclusions even without ever actually finding such an assertion. So the point of our paper is that you need not immediately turn to finding an assertion: as soon as you spot a point where an assertion could be written down it need not be the actual next step to find such an assertion. You can also introduce a variable which stands for it, and then you start the formal calculations with that variable — and come to conclusions which are relevant to the problem. So that would be my technical response to Culik's question.