

ASSOCIATIVE COMMUTATIVE PATTERN MATCHING

J.M. HULLOT
IRIA-LABORIA
Domaine de Voluceau
78150 - ROCQUENCOURT
FRANCE

Plotkin [10] has opened the way to the design of unification algorithm incorporating equality axioms and their use in complete proof systems. In particular he gives complete unification algorithms for terms containing associative function symbols and for an arithmetic theory. Huet gives in [18] a unification algorithm for λ -conversion and discusses in [19] general properties of complete set of unifiers. A complete and finite algorithm for unification with associative commutative function symbols is given in Stickel [14] and Lankford shows in [22] how to extend it to abelian group theory. Various other specific unification problems are discussed in [9,15,23]. In the case of equational theories which possess a complete set of reductions as defined in Knuth-Bendix [7], Fay has given in [20,21] a general unification algorithm.

We are interested in this paper in the special case of commutative associative function symbols. We first study the equivalence problem which is reduced to recognizing the equality of two multiset representations. In the same way, matching reduces to generating ordered partitions of a multiset. This generation is carried out by two processes operating on tree data structures spanning the lattice of partitions. Cardinality conditions inherited from the matching problem are used to cut the search drastically : our processes operate bidirectionnally in a coroutine manner, each one trimming sub-structures from both search spaces. The two processes are formally defined by a unique recursive definition, and their computations correspond to a call-by-need interpretation of this definition. The correctness of the method is proved formally. This basic organization is used to derive a new algorithm for associative commutative unification improving over Stickel [14]. We have programmed in LISP all these algorithms, using University of Vincennes's V-LISP system [3]. We have applied them to the Stickel-Peterson's generalization of the Knuth and Bendix extension algorithm [7,8,16]. Their algorithm relies on associative-commutative unification for the computation of critical pairs, but also most critically on associative-commutative match, in order to reduce terms to their normal forms. Stickel and Peterson proposed to use unification for matching, but this leads to unnecessary inefficiencies that preclude the practical use of their method. We give in the Appendix running times on a DEC KL10 using the VLISP system, corresponding to some of their examples.

Most proofs are omitted in this version, the full paper is available as a technical report [24].

I. FIRST ORDER TERMS AND EQUATIONAL THEORIES

Let V be a denumerable set of elements called variables, C a finite or denumerable set of elements called constants with $C \cap V = \emptyset$. Elements in C are graded by an arity function $\alpha: C \rightarrow \mathbb{N}$. The set of terms T is the smallest set containing V and closed by :

$t_1, \dots, t_{\alpha(f)} \in T \rightarrow f(t_1, \dots, t_{\alpha(f)}) \in T$ for every $f \in C$

A substitution is a mapping σ from V to T , with

$\sigma(x) = x$ almost everywhere. For any $t \in T$, $V(t)$ will denote the set of variables of t . An equational theory A is a theory in which every axiom is the universal closure of an equality. Two terms t_1 and t_2 are equivalent in A iff : $A \vdash t_1 = t_2$. In the remainder we will write $t_1 \equiv_A t_2$ or $t_1 \dot{=} t_2$ when A is implicit. Let $A \in C$ with $A = \{f_1 \dots f_p\}$ where $\forall i \ 1 \leq i \leq p \ f_i$ is binary and f_i is an associative and commutative function i.e. $\alpha \ f_i(x,y) = f_i(y,x)$

and $\beta) f_i(x, f_i(y, z)) = f_i(f_i(x, y), z)$. The equational theory of $f_1 \dots f_p$ is the theory with the $2p$ axioms α and β . In the remainder of this paper A will always denote the theory of $f_1, \dots, f_p$. All terms will be in right associative form. A term $t = f_i(t_1, f_i(t_2, \dots, f_i(t_{n-1}, t_n) \dots))$ where t_k doesn't begin with f_i is called NF-term. We shall abbreviate such a term by $f_i(t_1, \dots, t_n)$. If $t = f_i(t_1, \dots, t_n)$, $O(t)$ will denote n .

II. EQUIVALENCE

II.1 Equivalence lemma

Let T, T' be two NF-terms with head symbol f_i i.e. $T = f_i(t_1, \dots, t_n)$ and $T' = f_i(t'_1, \dots, t'_n)$. Then $T \equiv T'$ iff $n=m$ and there exists a permutation Π of m such that $t_{\Pi(i)} \equiv t'_i (1 \leq i \leq m)$.

Proof : see [24].

II.2 Equivalence algorithm-EQUIAC (t, t')

Let us now describe an equivalence algorithm between terms in A . We call unificand a finite set of pairs of terms. Unificands are denoted by $N, N_1, \dots$.

We describe this algorithm as the construction of a tree whose nodes are either a unificand consisting of pairs of NF-terms or $\emptyset$ for success or $\textcircled{F}$ for failure. We first define $\text{SIMPL}(N)$ which simplifies a unificand N

SIMPL(N)

- $\text{SIMPL}(\emptyset) = \emptyset$
- $\text{SIMPL}(\langle e, e' \rangle \cup N) =$
 if $e \in V$
 then if $e = e'$ then $\text{SIMPL}(N)$
 else $\textcircled{F}$
 else let $e = F(e_1 \dots e_n), e' = G(e'_1 \dots e'_n)$
 if $F \neq G$ then $\textcircled{F}$
 else if $F = G$ then $\langle e, e' \rangle \cup \text{SIMPL}(N)$
 else $\text{SIMPL}(\{\langle e_i, e'_i \rangle \mid 1 \leq i \leq n\} \cup N)$

Termination : by induction on $\langle e, e' \rangle \in N \quad \theta(e)$. $\square$

Let $e = f_i(t_1, \dots, t_n), e' = f_i(t'_1, \dots, t'_n)$, the function $\text{Permu}(e, e')$ given below returns a finite set of unificands :

$$\text{Permu}(e, e') = \{ \{ \langle t_1, t'_{\pi_1(1)} \rangle, \dots, \langle t_n, t'_{\pi_1(n)} \rangle \} \\ \dots \dots \dots \{ \langle t_1, t'_{\pi_{n!}(1)} \rangle, \dots, \langle t_n, t'_{\pi_{n!}(n)} \rangle \} \}$$

Where $\{\pi_1 \dots \pi_{n!}\} =$ set of all permutations of $\{1, \dots, n\}$.

The tree for $\langle t, t' \rangle$ is grown as follows :

- the root node is $N_0 = \text{SIMPL}(\langle t, t' \rangle)$
- node $\textcircled{F}$ is a leaf
- if a node is $\emptyset$, the construction is interrupted *exit True*
- to grow the successors of node N , select in N an arbitrary pair : $\langle e, e' \rangle$

If $\theta(e) \neq \theta(e')$ the only successor of N is $\textcircled{F}$, else $\text{Permu}(\langle e, e' \rangle)$ returns a finite set of unificands N_i . For each N_i grow an arc from N to N_i where :

- $N_i = \textcircled{F}$ if $\text{SIMPL}(N_i) = \textcircled{F}$
- $N_i = N - \{ \langle e, e' \rangle \} \cup \text{SIMPL}(N_i)$ otherwise

If the construction stops and all leaves are $\textcircled{F}$ *exit False*. The equivalence lemma proves the correctness of the method.

II.3 The heuristic part

In the implementation of the equivalence algorithm we do not generate all permutations for two given terms. Some structural remarks allow us to reduce the search space. The following example shows the behavior of our algorithm.

Let us assume

$$A = \{ f_1, f_2 \}, t = f_1(x_1, f_2(x_2, a), f_2(b, x_3)) \text{ and } t' = f_1(f_2(b, x_3), f_2(a, x_2), x_1)$$

where $a, b \in C$ and $x_1, x_2, x_3 \in V$.

We have $N_0 = \text{SIMPL}(\langle t, t' \rangle) = \{ \langle t, t' \rangle \}$.

Without loss of generality we generate now only

those permutations π such that if $\langle t_i, t_{\pi(i)} \rangle$ is part of a resulting unificand, then $\text{SIMPL}(\langle t_i, t_{\pi(i)} \rangle) \neq \textcircled{F}$. Thus, since $\text{SIMPL}(\langle x_1, f_2(b, x_3) \rangle) = \textcircled{F}$ and $\text{SIMPL}(\langle x_1, f_2(a, x_2) \rangle) = \textcircled{F}$ there are only two successors :

$N_1 = \{ \langle f_2(x_2, a), f_2(b, x_3) \rangle, \langle f_2(b, x_3), f_2(a, x_2) \rangle \}$ and $N_2 = \{ \langle f_2(x_2, a), f_2(a, x_2) \rangle, \langle f_2(b, x_3), f_2(b, x_3) \rangle \}$.

To grow the successors of node N_1 select the first pair of N_1 , say $\langle f_2(x_2, a), f_2(b, x_3) \rangle$. Since $\text{SIMPL}(\langle x_2, b \rangle) = \textcircled{F}$ and $\text{SIMPL}(\langle x_2, x_3 \rangle) = \textcircled{F}$, the only successor of N_1 is $\textcircled{F}$. To grow the successor of N_2 select $\langle f_2(x_2, a), f_2(a, x_2) \rangle$ since $\text{SIMPL}(x_2, a) = \textcircled{F}$ the only successor of N_2 is $N_3 = \langle f_2(b, x_3), f_2(b, x_3) \rangle$. In the same way $\emptyset$ is the only successor of N_3 and the construction terminates successfully.

III. MATCHING

III.1 Definition

Let M be a finite sequence of subsets of a set N (resp. a multiset N), $M = (A_1, \dots, A_n)$, $A_i \subset N$, $m \geq 1$, $A_i \neq \emptyset$. We say that M is an ordered partition of N iff $\bigcup_{1 \leq i \leq n} A_i = N$ and $A_i \cap A_j = \emptyset$ for $i \neq j$ (resp. $\bigcup_{1 \leq i \leq m} A_i = N$ where $\bigcup$ denotes union for multisets).

III.2 A complete matching algorithm

We can extend $\equiv$ to substitutions by defining $\sigma \equiv \sigma'$ iff $\forall t \in T \ \sigma t \equiv \sigma' t$. S is a complete set of matches of $\langle t, t' \rangle$ iff S is consistent ($\forall \sigma \in S, t' \equiv \sigma t$) and S is complete ($\forall \sigma$ such that $t' \equiv \sigma t$, $\exists \sigma' \in S, \exists \lambda$ substitution with $(\sigma \lambda \upharpoonright V(t)) \equiv \sigma'$) where $(\sigma \upharpoonright V)(x) = \sigma(x)$ if $x \in V$ and x otherwise). As for equivalence we shall construct a tree whose nodes are either a unificand or $\emptyset$ for success or $\textcircled{F}$ for failure. We first define $\text{SIMPL}_1(N)$ which simplifies a unificand N .

$\text{SIMPL}_1(N)$

- $\text{SIMPL}_1(\emptyset) = \emptyset$
 - $\text{SIMPL}_1(\langle e, e' \rangle \cup N) =$
 if $e \in V$ then $\langle e, e' \rangle \cup \text{SIMPL}_1(N)$
 else if $e' \in V$ then $\textcircled{F}$
 else let $e = F(e_1, \dots, e_n)$
 $e' = G(e'_1, \dots, e'_m)$
 if $F \neq G$ then $\textcircled{F}$
 else if $F = G \in A$ then $\langle e, e' \rangle \cup \text{SIMPL}_1(N)$
 else $\text{SIMPL}_1(\{ \langle e_i, e'_i \rangle \mid 1 \leq i \leq n \} \cup N)$

Termination : as for equivalence. $\square$

Let $e = f_i(t_1, \dots, t_n)$ and $e' = f_i(t'_1, \dots, t'_m)$ with $m \geq n$. The function $\text{PART}(e, e')$ given below returns a finite list of unificands.

Let $\Sigma_m = \{\text{ordered partitions of the set } \{t'_1, \dots, t'_m\} \text{ in } n \text{ non empty blocks}\} = \{P_1 \dots P_q\}$ where P_i are the partitions. Thus $P_i = \{\tau_i(1) \dots \tau_i(n)\}$ where $\tau_i(j)$ are the elements of the partition. Then we define $t_k^1 = \tau_k(1)$ if $|\tau_k(1)| = 1$ and $t_k^1 = f_i(T_1, \dots, T_r)$ if $\tau_k(1) = \{T_1 \dots T_r\}$ with $r > 1$. We define : $\text{PART}(e, e') = \{ \langle t_1, t'_1 \rangle, \dots, \langle t_n, t'_n \rangle \mid 1 \leq i \leq q \}$.

The matching tree MT for $\langle t, t' \rangle$ is grown as follows :

- the root node is $N_0 = \text{SIMPL}_1(\langle t, t' \rangle)$. We set $\sigma_{N_0} = \emptyset$
 - node $\textcircled{F}$ is a leaf
 - to grow the successors of node N , select in N an arbitrary pair $\langle e, e' \rangle$.
 if $e \in V$ then if $\sigma_N e$ is undefined
 then grow an arc labelled with $\langle e, e' \rangle$ from N to $N' = N - \{ \langle e, e' \rangle \}$. Set $\sigma_{N'} = \langle e, e' \rangle \cup \sigma_N$
 else if $\text{EQUIAC}(e', \sigma_N e)$ then grow an arc from N to $N - \{ \langle e, e' \rangle \}$ labelled by the empty substitution $\epsilon(N \rightarrow N - \{ \langle e, e' \rangle \})$
 else grow an arc $N \xrightarrow{\textcircled{F}}$
 else if $\theta(e') < \theta(e)$ then grow are $N \xrightarrow{\textcircled{F}}$

else the function $PART(e, e')$ returns a finite set of unificands $\{N_1 \dots N_q\}$. For each N_i grow an arc labelled with e from N to N_i' where :
 $N_i' = \textcircled{F}$ if $SIMPL_1(N_i) = \textcircled{F}$
 $N_i' = (N - \{<e, e'>\}) \cup SIMPL_1(N_i)$ otherwise

The set of matches of t, t' is $S = \{\sigma_N \mid N = \emptyset \text{ in } MT\}$

The correctness of the algorithm is given in [24].

Remark1: As for equivalence, some structural conditions allow us to reduce the search space (using $SIMPL_1$ instead of $SIMPL$). Thus we use the function $PART$ only when the left term has only variables as arguments.

Remark2 : One may define the function $PART$ in using the concept of ordered partitions of a multiset. Such an improvement can be found in [24].

Let us now give an efficient way to generate ordered partitions of a set.

IV. THE UNDERLYING DATA STRUCTURE

We give in this part a basic algorithm which can be used to :

- generate ordered partitions of a set (detailed in example 1) or a multiset (see [24]),
- generate ordered partitions of sets or multisets with cardinality constraints on the blocks (see [24]),
- improve over Stickel's unification algorithm (see example 2 and [24] for more details).

IV.1 The Basic problem

Let $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$ be two binary vectors of size n . We define the partial ordering relation P_1 on binary vectors by $x P_1 y$ iff $\forall i \geq x_i \geq y_i$ and $\exists j$ such that $x_j \neq y_j$. We define P_0 by $x P_0 y$ iff $y P_1 x$. A graph of the partial or-

der P_0 or P_1 for $n=3$ is shown in Fig.1.

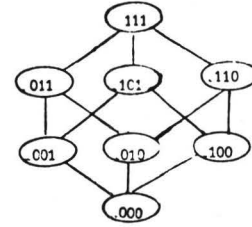


Fig.1

Throughout the remainder of this paper P_0 and P_1 will denote two predicates on binary vectors such that :

$$\begin{aligned} A_0. P_0(y) = \text{True and } x P_0 y &\Rightarrow P_0(x) = \text{True} \\ A_1. P_1(y) = \text{True and } x P_1 y &\Rightarrow P_1(x) = \text{True.} \end{aligned}$$

Our aim is to design an algorithm for generating this subset of all binary vectors of size n such that $p_0(x) = \text{True}$ and $p_1(x) = \text{True}$.

Example 1 : Let M be a binary vector of size $m \times n$, $M = h_1 \dots h_n$ with $h_i = h_{i,1} \dots h_{i,m}$ where $h_{i,j}$ is 0 or 1. Let v_j be the binary vector of size n , $v_j = h_{1,j} \dots h_{n,j}$. We define predicates q_1, q_2, q_3 by :

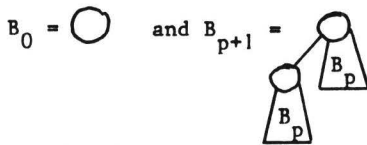
$$\begin{aligned} q_1(M) = \text{True} &\Leftrightarrow \forall i \ 1 \leq i \leq n \ h_i \neq 0 \dots 0 \\ q_2(M) = \text{True} &\Leftrightarrow \forall j \ 1 \leq j \leq m \ v_j \neq 0 \dots 0 \\ q_3(M) = \text{True} &\Leftrightarrow \forall j \ 1 \leq j \leq m \text{ at most one element} \\ &\text{in } v_j \text{ is 1,} \end{aligned}$$

and predicates P_0 and P_1 by $P_1(M) = q_1(M) \wedge q_2(M)$, $P_0(M) = q_3(M)$. The set of all ordered partitions of m elements into n blocks is represented by $\{M \mid P_0(M) = \text{True and } P_1(M) = \text{True}\}$ in the following interpretation : the j^{th} element is in the i^{th} block iff $h_{i,j} = 1$. Furthermore p_0 (resp. p_1) has property A_0 (resp. A_1).

Example 2 : Let $M = z_1 \dots z_n$ a binary vector of size n , $M_1 \dots M_p$ some fixed such vectors and $M_{i_1} \dots M_{i_q}$ some of the M_i ($q \leq p$). We define

$L(M)$ = number of 1 in M , and predicates P_0 and P_1 by $P_0(M) = \text{True} \Leftrightarrow \forall i, 1 \leq i \leq q, L(MAMi_j) \leq 1$ and $P_1(M) = \text{True} \Leftrightarrow \forall i, 1 \leq i \leq p, MAMi \neq 0$, P_0 (resp. P_1) has property A_0 (resp. A_1). A reader familiar with Stickel's unification algorithm [14] will recognize the basic situation of the second part of this algorithm (see [24] for more details).

IV.2. The algorithm is first conceived as a sequence of two passes that happen one after the other. It is then elaborated by stepwise refinement in order to produce the final version. In the first pass we search only those binary vectors such that $p_1(x) = \text{True}$. We may use the search method of Balas [1]. This method is equivalent to scan the combinatorial data structure called binomial tree (Vuillemin [17]) considered as a spanning tree of the graph for P_1 defined above. Binomial trees are defined by :



The B_3 binomial tree T_1 associated with P_1 is pictured in Fig.2.

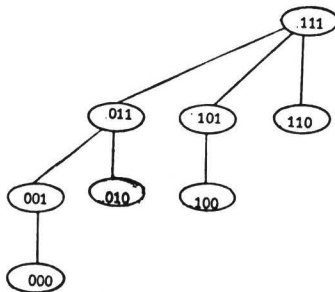


Fig.2

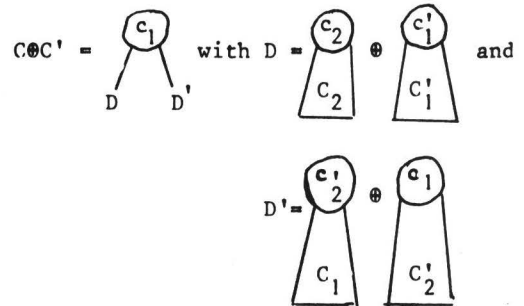
If for some $x, p_1(x) = \text{False}$ we do not have to scan the subtree rooted at x since $p_1(y) = \text{False}$ for all y in this subtree (property A_1). Similarly, we generate those binary vectors such that $P_0(x) = \text{True}$, in using the binomial tree T_0 associated with P_0 (T_0 is the same as T_1 except that if $x = x_1 \dots x_n$ is the label of node h in T_1 ,

$y = (1-x_1) \dots (1-x_n)$ is the label of node h in T_0). The final set of solutions S will be the intersection of these two sets. The most obvious source of inefficiency of this algorithm may be viewed on the following example : assume $n=3$, $P_0(100) = \text{False}$ and begin to scan T_1 . Since, in this first pass, we do not know that the subtree of T_0 rooted at 100 must be eliminated we have to test P_1 for the 4 corresponding nodes.

IV.3. Our refinement will consist in generating S by an algorithm which does not begin to test P_0 or P_1 for a node x until all fathers of x in T_0 (resp. T_1) have been tested for P_0 (resp. P_1). To achieve this aim, we shall represent the two binomial trees as one binary tree.

Let C and C' be two labeled B_p binomial trees. We define $C \oplus C'$ as follows :

Case 1 : $p=0$. Then $C \oplus C' = C$
Case 2 : $p>0$. Let $C =$ and C' similarly defined, then



$T_1 \oplus T_0$ for $p=3$ is pictured in Fig.3. $\boxed{h}$ (resp. $\odot h$) means that n comes from T_1 (resp. T_0).

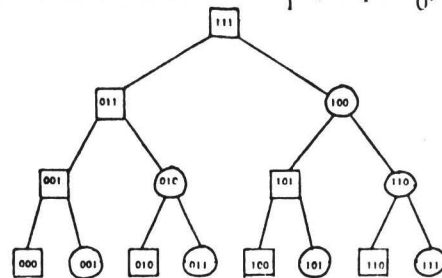


Fig.3

For any binary tree B , we define its set of occurrences $O(B)$ as a finite subset of $\{0,1\}^*$ and the subtree of B at u denoted by B/u by :

- if $B=0$ (only the root) then $O(B)=\{\wedge\}$ and $B/\wedge=B$
- if $B = \begin{array}{c} \text{O} \\ \swarrow \quad \searrow \\ B_0 \quad B_1 \end{array}$ then $O(B)=\{\wedge\} \cup \{0 \cdot u \mid u \in O(B_0)\} \cup \{1 \cdot u \mid u \in O(B_1)\}$ and $B/\wedge=B$,
 $B_0/u=B/0 \cdot u, B/1 \cdot u=B_1/u$.

Thus, any occurrence u is either $\wedge$ or $u = \text{father}(u) \cdot \text{last}(u)$ where $\text{father}(u)$ is an occurrence and $\text{last}(u)$ is 0 or 1 identified respectively with False and True. Conventionally we define $\text{last}(\wedge)=0$.

Lemma : There is a bijective mapping from the occurrences of length n onto the labels of the non terminal nodes of the binary tree of depth $n+1$ $T_{1,0}T_0$. Its expression is given by : $\text{label}(u)=u \cdot b^h$ where $b = \text{last}(u)$ and $h=n-|u|$.

Proof : in [24].

We define $R_n = \{w \mid w \in \{0,1\}^*, |w|=n\}$. The definition of label is extended to R_n by : $\forall w \in R_n \text{ label}(w)=w$. Furthermore, we define :

$$\text{Set}_n = \{w \in R_n \mid \text{tree}(w)\}$$

where $\forall u$ such that $|u| \leq n$:

$\text{tree}(u)$ if $u=\wedge$ then $p(0^n, \text{False})$
 else if $\text{tree}(\text{father}(u))$
 then $p(\text{label}(u), \text{last}(u))$
 else False.

where $p(w, \text{True})=p_1(w)$ and $p(w, \text{False})=p_0(w)$.

Theorem : $\forall w \in R_n \text{ tree}(w) \iff p_0(w) \text{ and } p_1(w)$

Proof : see [24].

Our algorithm is conceived as a call by need interpretation of the recursive definition of tree . The theorem proves the correctness of this algorithm.

V. GENERALIZATION OF THE KNUTH-BENDIX ALGORITHM

It is not our purpose to present the

basic Knuth-Bendix's algorithm (see [7] and [6] and its generalizations ([8],[16])). Our aim is to apply our algorithms to the Stickel-Peterson's generalization of the Knuth-Bendix's extension algorithm. Their algorithm relies on associative-commutative unification for the computation of the critical pairs, but also most critically on associative-commutative match in order to reduce terms to their normal forms. Stickel and Peterson proposed to use unification for matching, but this leads to unnecessary inefficiencies that preclude the practical use of their method. With our match algorithm a reasonable implementation is possible as shown in the Appendix where we give detailed running times on a DEC KL-10, using the V-LISP system ([3],[4]), corresponding to some of their examples.

VI. CONCLUSION

We have presented in this paper a new matching algorithm in term structures containing associative-commutative function symbols. Our experiments carried out on Stickel-Peterson's generalization of the Knuth and Bendix algorithm have shown that this algorithm together with our improvements over Stickel's algorithm provide an efficient way to implement large systems that deal with associativity and commutativity. For example we believe that the multiset matching algorithm obtained is efficient enough for the practical implementation of pattern directed invocation of procedures with multiset variables (Q-LISP [11,12,15]).

Furthermore we hope that the techniques used here for the description of our partition generator can be extended to other graph algorithms.

Acknowledgements : This work is part of a thesis conducted at LRI, University Paris Sud. I would especially like to thank my supervisor G. Huet who has contributed to many ideas presented here. I would also like to thank J. Chailloux and

P. Greussay (University of Vincennes).

Appendix

We wrote a commutative associative extension algorithm as a 1500 line LISP program. It runs interpreted under the V-LISP system [3] on a DEC KL10 computer, with 1500 storage cells. We give the same examples as in Stickel-Peterson [16]. We have two versions for simplification : inside-out and outside-in. The former is more efficient, but uses more memory and therefore the latter may be preferable for large examples. The version used is indicated in Fig.4, where U_1 (resp. U_2) denotes Stickel's unification algorithm with (resp. without) our improvement and M_1 (resp. M_2) denotes matching where our algorithm (resp. unification) is used.

		superposition		simplification		other parts	Total			
Abelian groups	inside	U_1	U_2	M_1	M_2		$U_1 + M_1$	$U_2 + M_1$	$U_1 + M_2$	$U_2 + M_2$
		8"	12"	12"	36"	1"	21"	25"	45"	49"
Abelian rings with unit	out	28"	45"	50"	4'	2"	1'20"	1'37"	4'30"	4'47"
Distributive lattices	outside in	1'30"	3'	3'30"	13'	5"	5'5"	6'35"	14'35"	16'5"

References :

- [1] E. BALAS : "An additive algorithm for solving linear programs with zero one variables". Op. Res. 13 (1965).
- [2] V.J. BOWMAN & J.H. STARR : "Partial orderings in implicit enumeration" An. of discr. math. 1(1977).
- [3] J. CHAILLOUX : VLISP 10.3 manuel de référence, Uni. Paris 8, Vincennes (1978).
- [4] P. GREUSSAY : "Contribution à la définition interprétative et à l'implémentation des lambda-langages". Thèse n°78-2, Paris 7(1977).
- [5] G. HUET : "An algorithm to generate the basis of solutions to homogeneous linear diophantine equations". Inf. Proc. Let.7, 3 April 78.
- [6] G. HUET : "Confluent reductions : Abstract properties and Applications to term rewriting systems" In Proc. of the 18th Symposium on FOCS (1977).
- [7] D.E. KNUTH & P.B. BENDIX : "Simple word problems in universal Algebras". In Leech, J.(Ed.), Comp. problems in Abstract Algebras. Pergamon Press, 1970.
- [8] D.S. LANKFORD & A.M. BALLANTYNE : "Decision procedures for simple equational theories with commutative-associative axioms : complete sets of commutative associative reductions". Tech. rep. Math. dept. U. of Texas at Austin, ATP 39 (1977).
- [9] M. LIVESEY & J. SIEKMANN : "Unification of A+C-terms (bags) and A+C+I-terms (sets)". Int. Ber. Nr 3176, Inst. für Inf. I, Universität Karlsruhe.
- [10] G. PLOTKIN : "Building in equational theories" In Meltzer & Michie(Ed) Machine Intelligence 7, Edinburgh University Press 1972.
- [11] R. REBOH & E. SACERDOTI : "A preliminary Q-LISP Manual". Tech. note 81, SRI, Menlo Park, 1973.
- [12] J.F. RULIFSON, J.A. DERKSEN & R.J. WALDINGER : "QA4 : a procedural calculus for intuitive reasoning". Tech. Note 71, SRI, Menlo Park, 1972.
- [13] J.R. SLAGIE : "Automated Theorem proving for theories with simplifiers commutativity and associativity", J. ACM 21,4 (1974).
- [14] M.E. STICKEL : "A complete unification algorithm for associative-commutative functions". Proc. of the 4th IJCAI Tbilisi (1975).
- [15] M.E. STICKEL : "Unification algorithms for artificial intelligence languages" Ph.D. Carnegie Mellon, Pittsburgh, Penn (1976).
- [16] M.E. STICKEL & G.E. PETERSON : "Complete sets of reductions for equational theories" U. of Arizona, Techn. Report (1978).
- [17] J. VUILLEMIN : "A data structure for manipulating priority queues" C. ACM Vol 21 (4)(1978).
- [18] G. HUET : "A unification Algorithm for typed λ -calculus" Theo. Com. Science 1 (1975).
- [19] G. HUET : "Résolution d'équations dans les langages d'ordre 1,2,..., ω " Thèse d'Etat U. Paris VII (1976).
- [20] M.J. FAY : "First order unification in equational theories" Tech. Report N°78-5-002 U. of California Santa-Cruz (1978).
- [21] M.J. FAY : "First order unification in equational theories" Fourth Workshop on automated deduction Austin Texas(1979)
- [22] D. LANKFORD : "A unification algorithm for abelian group theory" Report MRP1. Dep. of Math. Louisiana Tech. Uni.
- [23] M. LIVESEY, J. SIECKMANN, P. SZABO, F. UNVEECHT : "Unification problems for combinations of associativity, commutativity, distributivity and idempotence axioms. Fourth Workshop on automated deduction. Austin Texas Feb.(1979)
- [24] J.M. HULLOT : "Associative-commutative pattern matching" Report IRIA-LABORIA to appear (1979).