Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

UNIFICATION IN LINEAR TIME AND SPACE: A STRUCTURED PRESENTATION

A. Martelli, U. Montanari

Nota Interna B76-16
Luglio 1976

ABSTRACT

Unification was introduced by J.A. Robinson in the context of automatic theorem proving and plays an important role in many areas of symbolic manipulation. We present a general framework where all known algorithms for unification can be described, and in this context we introduce the most general version of our algorithm. Two alternative refinements are then exposed, the first leading to a simpler algorithm of wide applicability which is linear with the total number of symbols and n log n with the number of distinct variables; the second to a more complicated linear algorithm. Full complexity analysis of both algorithms is included.

Key Words and Phrases: unification, resolution, automatic theorem proving, pattern matching, structured programming, analysis of algorithms.

CR Categories: 5.21, 5.25, 5.7, 4.9.

1 - INTRODUCTION

In its simplest form, the unification problem can be expressed as follows: Given two terms t_1 and t_2, with some variables, find, if it exists, the simplest substitution (i.e. an assignment of some term to every variable)as to make them equal. The resulting term is called the most general unifier (mgu) and is unique up to variable renaming.

An equivalent statement of the problem makes more clear its nature. Given a term t, let L(t) be the set of terms which can be obtained from t with any substitution. Given terms t_1 and t_2, find if it exists, a term $\bar{t}$ such that

$$L(\bar{t}) = L(t_1) \cap L(t_2)$$

For example, let

$$t_1 = f(x,f(z,z)) \text{ and } t_2 = f(g(y),y)$$

then the mgu is

$$\bar{t} = f(g(f(z,z)),f(z,z))$$

Unification was first introduced by J.A. Robinson [1,2] as the central step of the inference rule called resolution. This single, powerful rule can replace all the axioms and inference rules of first order predicate calculus, and thus was immediately recognized as expecially suited to mechanical theorem provers. In fact, a number of systems based on resolution were built, and tried on a variety of different applications [3]. Even if further research made apparent that resolution systems are difficult to direct during proof search and thus are often prone to combinatorial explosion [4], they are still likely to be the main symbol-cranching part of tomorrow's general purpose theorem provers. Recent research was directed towards a more accurate study of the data structures involved in this type of symbol manipulation [5] and towards the possibility of embedding in the resolution rule (and in the unification algorithm) such general properties as function commutativity and associativity [6].

However resolution theorem proving is not the only application of the unification algorithm. In fact its pattern matching nature can be exploited in other cases of symbolic manipulation (e.g. when deciding the applicability of a simplification rule); in procedure invocation through pattern matching (as in some special artificial intelligence programming languages [4]); in systems using a data base organized in terms of productions [7].

Obtaining efficient versions of the unification algorithm was immediately recognized as a main goal in symbolic manipulation [2] but only recently the techniques of concrete complexity theory have been applied to this subject [8]. Variations of the original algorithms were considered, which appeared to have quasilinear complexity [9,10]. Finally, Paterson [11] has presented a linear algorithm for unification. However he gave no details on the algorithm and no formal proof of its linearity.

The research described in this paper was carried on independently from Huet and Paterson work, even if the linear algorithm we present here is similar, in its main lines, to Paterson's. In Section 2 we represent the unification problem as the solution of a system of equations. A nondeterministic algorithm is then defined, and proved correct, which comprehends as special cases all known algorithms. To gain perspicuity, in Section 3 we group together all equations with some member in common, and thus we obtain a model for which the most general version of our algorithm can be exposed.

To help the reader to grasp the main tricks and to follow the complexity analysis, we give a structured presentation of the algorithm through a sequence of successive refinements [12]. The refined algorithms are expressed in the form of PASCAL programs, since this well-known language, through the concept of user defined data types, allows to develop step by step also the necessary data structures. The first refined program, even if very similar to the general version of the algorithm, allows full complexity analysis of the most involved part, where the actual matching of subterms takes place.

Two alternative refinements are then considered. The first resulting program, besides being linear on the total number of symbols, may require in the worst case a computing time which is n log n with the number of distinct variables in the problem. However this program uses substantially simpler data structures, is rather straightforward to understand, and will probably be faster on most problems of practical size than both the classical algorithms and the subsequent linear algorithm. Finally, in Section 6 we present the linear algorithm with its complete complexity analysis. Both programs are shown to be linear in space.

The Appendix contains the straightforward implementation of the last level data structures and procedures (mainly lists and standard operations on them) so that complete, running PASCAL programs for the two algorithms can be extracted from the paper.

2 - UNIFICATION AS THE SOLUTION OF A SET OF EQUATIONS: A NONDETERMINISTIC ALGORITHM

In this section we introduce the basic definitions and give a few theorems which will be useful in proving the correctness of the algorithms. Our way of stating the unification problem is slightly more general than the classical one due to Robinson [1] and directly suggests a number of possible solution methods.

Let

$$A = \bigcup_{i=0}^{m} A_i \quad (A_i \cap A_j = \emptyset,\ i \neq j)$$

be a ranked alphabet, where A_i contains the i-adic function symbols (the elements of A_0 are constant symbols). Furthermore, let V be the alphabet of the variables. The terms are defined recursively as follows:

a) Constant symbols and variables are terms.
b) If $t_1,\ldots,t_i$ $(i \geq 1)$ are terms and $f \in A_i$, then $f(t_1,\ldots,t_i)$ is a term.

A substitution is a set of ordered pairs $\vartheta = \{(t_1,x_1), (t_2,x_2), \ldots, (t_n,x_n)\}$ where t_i are terms and x_i are distinct variables, $i = 1,\ldots,n$. To apply a substitution ϑ to a term t, we simultaneously substitute all occurrences in t of every variable x_i in a pair of ϑ with the corresponding term t_i. The resulting term will be called t_ϑ.

For instance, given a term $t = f(x_1,g(x_2),a)$ and a substitution $\vartheta = \{(h(x_2),x_1), (b,x_2)\}$ we have $t_\vartheta = f(h(x_2),g(b),a)$.

The standard unification problem can be written as an equation

$$t' = t''$$

A solution of the equation, called a unifier, is any substitution ϑ, if it exists, which makes the two terms identical. For instance, two unifiers of the equation $f(x_1,h(x_1),x_2) = f(g(x_3),x_4,x_3)$ are $\vartheta_1 = \{(g(x_3),x_1),(x_3,x_2),(h(g(x_3)),x_4)\}$ and $\vartheta_2 = \{(g(a),x_1),(a,x_2),(a,x_3),(h(g(a)),x_4)\}$.

In what follows, it will be convenient to consider also sets of equations

$$t'_j = t''_j \qquad j = 1,\ldots,k \; .$$

Again, a unifier is any substitution which makes all pairs of terms t'_j, t''_j identical simultaneously. Now we are interested in finding transformations which produce equivalent sets of equations, namely transformations which preserve the sets of all unifiers. Let us introduce the following two transformations.

a) Term reduction

Let

$$f(t'_1,t'_2,\ldots,t'_i) = f(t''_1,t''_2,\ldots,t''_i) \qquad 0 \le i \le m \tag{2.1}$$

be an equation where both terms are not variables and where the two root function symbols are equal. The new set of equations is obtained by replacing such equation with the following ones

$$\begin{aligned} t'_1 &= t''_1 \\ t'_2 &= t''_2 \\ &\vdots \\ t'_i &= t''_i \end{aligned} \tag{2.2}$$

If i=0, f is a constant symbol and the equation is simply erased.

b) Variable elimination

Let

$$x = t$$

be an equation where x is a variable and t is any term (variable or not). The new set of equations is obtained by applying the substitution $\vartheta = \{(t,x)\}$ to both terms of all other equations in the set (without erasing x=t).

We can prove the following theorems.

Theorem 2.1 - Let S be a set of equations and let $f'(t'_1,\ldots,t'_i) = f''(t''_1,\ldots,t''_i)$ be an equation of S. If $f' \neq f''$ then S has no unifier. Otherwise the new set of equations S', obtained by applying term reduction to the given equation, is equivalent to S.

Proof - If $f' \neq f''$, then no substitution can make the two terms identical. If $f'=f''$, any substitution which satisfies (2.2) will also satisfy (2.1) and conversely for the recursive definition of term. □

Theorem 2.2 - Let S be a set of equations and let us apply variable elimination to some equation x=t, getting a new set of equations S'. If variable x occurs in t (but t is not x) then S has no unifier, otherwise S and S' are equivalent.

Proof - If variable x occurs in t (but t is not x), then no substitution ϑ can make the two members of the equation x=t identical, since the term which is substituted for x becomes a subterm of t_ϑ. Equation x=t belongs both to S and to S' and thus any solution of S or S' must unify x and t. Now let t_1 be any term in any other equation of S, and let t'_1 be the corresponding term in S'. Since t'_1 has been obtained by substituting t for every occurrence of x in t_1, any solution of S or S' must unify t_1 and t'_1. □

There is a special type of sets of equations for which the set of unifiers is evident. Such sets are called sets of equations in solved form and must satisfy the following conditions:

a) The equations are $x_j = t_j$, $j=1,\ldots,k$
b) Every variable which is the left member of some equation occurs only there.

A set of equations in solved form has an obvious unifier

$$\vartheta = \{(t_1,x_1),(t_2,x_2),\ldots,(t_k,x_k)\} .$$

Any other unifier (if any) can be obtained as

$$\sigma = \{((t_1)_\alpha,x_1),((t_2)_\alpha,x_2),\ldots,((t_k)_\alpha,x_k)\} \cup \alpha$$

where α is any substitution which does not rewrite variables $x_1,\ldots,x_k$. Thus ϑ is called a most general unifier (mgu).

The following nondeterministic algorithm shows how a set of equations can be transformed into an equivalent set of equations in solved form.

Algorithm A

Given a set of equations, repeatedly perform any of the following transformations. If no transformation applies, stop with success.

a) Select any equation of the form

$$t = x$$

where t is not a variable and x is a variable, and rewrite it as

$$x = t$$

b) Select any equation of the form

$$x = x$$

where x is a variable, and erase it.

c) Select any equation of the form

$$t' = t''$$

where t' and t" are not variables. If the two root function symbols are different, stop with failure; otherwise apply term reduction

d) Select any equation of the form

$$x = t$$

where x is a variable which occurs somewhere else in the set of equations, and $t \neq x$. If x occurs in t, then stop with failure; otherwise apply variable elimination.

As an example, let us consider the following set of equations

$$\left| \begin{array}{l} g(x_2) = x_1 \\ f(x_1, h(x_1), x_2) = f(g(x_3), x_4, x_3) \end{array} \right.$$

By applying transformation c) of Algorithm A to the second equation we get

$$\left| \begin{array}{l} g(x_2) = x_1 \\ x_1 = g(x_3) \\ h(x_1) = x_4 \\ x_2 = x_3 \end{array} \right.$$

By applying transformation d) to the second equation we get

$$\left| \begin{array}{l} g(x_2) = g(x_3) \\ x_1 = g(x_3) \\ h(g(x_3)) = x_4 \\ x_2 = x_3 \end{array} \right.$$

We now apply transformation c) to the first equation and transformation a) to the third equation

$$\left| \begin{array}{l} x_2 = x_3 \\ x_1 = g(x_3) \\ x_4 = h(g(x_3)) \\ x_2 = x_3 \end{array} \right.$$

Finally, by applying transformation d) to the first equation and transformation b) to the last equation, we get the set of equations in solved form

$$\left| \begin{array}{l} x_2 = x_3 \\ x_1 = g(x_3) \\ x_4 = h(g(x_3)) \end{array} \right.$$

Therefore, a mgu of the given system is

$$\vartheta = \{(g(x_3),x_1),(x_3,x_2),(h(g(x_3)),x_4)\}.$$

The following theorem proves the correctness of Algorithm A.

Theorem 2.3 - Given a set of equations S

a) Algorithm A always terminates, no matter which choices are made.

b) If Algorithm A terminates with failure, S has no unifier. If Algorithm A terminates with success, the set S has been transformed in an equivalent set in solved form.

Proof - a) Let us define a function F mapping any set of equation S in a triple of natural numbers (n_1,n_2,n_3). The first number n_1 is the number of variables in S which do not occur only once as the left member of some equation. The second number n_2 is the total number of occurrences of function symbols in S. The third number n_3 is the sum of the numbers of equations in S of type x=x and t=x, where x is a variable and t is not. Let us define a total ordering on such triples as follows:

$$(n'_1,n'_2,n'_3) \supset (n''_1,n''_2,n''_3) \text{ if } n'_1 > n''_1$$
$$\text{or } n'_1=n''_1 \text{ and } n'_2 > n''_2$$
$$\text{or } n'_1=n''_1 \text{ and } n'_2=n''_2 \text{ and } n'_3 > n''_3$$

With the above ordering, N^3 becomes a well-founded set, i.e. a set where no infinite decreasing sequence exists. Thus, if we prove that any transformation of Algorithm A transforms a set S in a set S' such that $F(S') \subset F(S)$, we have proved the termination. In fact, transformations a) and b) always decrease n_3 and, possibly, n_1. Transformation c) can possibly increase n_3 and decrease n_1, but surely decreases (by two) n_2. Transformation d) can possibly change n_3 and increase n_2, but surely decreases n_1.

b) If A terminates with failure, the thesis immediately follows from theorems 2.1 and 2.2. If A terminates with success, the resulting set of equations S' is equivalent to the given set S. In fact, transformations a) and b) clearly do not change the set of unifiers, while for transformations c) and d) this fact is stated in theorems 2.1 and 2.2. Finally, S' is in solved form: In fact, if a), b) and c) cannot be applied, it means that the equations are all in the form x=t, with $t \neq x$. If d) cannot be applied, it means that every variable which is the left member of some equation occurs only there. □

In the frame settled by the above nondeterministic algorithm fit, to authors' knowledge, all known algorithms for unification [1,5,8,11,13,14]. For instance, Robinson's algorithm [1] (restricted to two terms) can be obtained by making Algorithm A deterministic as follows:

Algorithm R

(Consider the set of equations as consisting of two lists of equations)

Step 1 - Initialize the first list with the equation $t_1=t_2$ and set the second list to the empty list.

Step 2 - Repeat what follows until the first list is empty. Take the first equation of the first list; if it is of the form

- i) t=x, apply transformation a).
- ii) x=x, apply transformation b).
- iii) t'=t", apply transformation c) and put the resulting equations, in the order, on top of the first list.
- iv) x=t, apply transformation d), if possible, and move this equation to the second list.

Step 3 - Stop with success. (The second list is the final system in solved form).

For instance, let us compute with Algorithm R a mgu of the two terms $f(x_1,h(x_1),x_2)$ and $f(g(x_3),x_4,x_3)$. After initialization, we have the following two lists:

list1 : $(f(x_1,h(x_1),x_2) = f(g(x_3),x_4,x_3))$

list2 : ( )

By executing part iii) of Step 2 we get

list1 : $(x_1 = g(x_3);\ h(x_1) = x_4;\ x_2 = x_3)$

list2 : ( )

Now the first equation of list1 is of the form iv), and thus we can eliminate variable x_1:

list1 : $(h(g(x_3)) = x_4;\ x_2 = x_3)$

List2 : $(x_1 = g(x_3))$

By executing part i) of Step 2 we get

list1 : $(x_4 = h(g(x_3));\ x_2 = x_3)$

list2 : $(x_1 = g(x_3))$

Finally, the last two executions of Step 2 eliminate variables x_4 and x_2:

list1 : ( )

list2 : $(x_2 = x_3\ ;\ x_4 = h(g(x_3));\ x_1 = g(x_3))$

3 - A REFINED ALGORITHM WHICH EXPLOITS A PARTIAL ORDERING AMONG SETS OF VARIABLES

In this section we present an extension of the previous formalism to model more closely our algorithm. We first introduce the concept of multiequation. A multiequation groups together many equations with common members. It is of the form

$$S = M$$

where S is a set of variables and M is a multiset(*) of terms which are not variables. M, but not S, may be empty. Many different sets of equations may correspond to a multiequation. For instance, to

$$\{x_1, x_2, x_3\} = (t_1, t_2)$$

may correspond both

$$\{x_1 = x_2,\ x_3 = x_1,\ t_1 = x_1,\ x_2 = t_2,\ t_1 = t_2\}$$

and

$$\{x_1 = x_2,\ x_1 = x_3,\ x_1 = t_1,\ x_1 = t_2\}.$$

In general, a set of equations I

$$t'_j = t''_j \qquad j=1,\ldots,k$$

correspond to a multiequation S=M iff both t'_j and t''_j belong to $S \cup M$ $(j=1,\ldots,k)$ and for every t_r and $t_s \in S \cup M$ there exists a sequence

$$t_r = t_{j_1}, t_{j_2}, \ldots, t_{j_q} = t_s$$

(*) A multiset is a family of elements where no ordering exists, but where many identical elements may occur. The right member of a multiequation is a multiset, since we do not want to check for repetion of terms.

such that either $t_{j_{i-1}} = t_{j_i}$ or $t_{j_i} = t_{j_{i-1}}$ belongs to I, for i=2,...,q.

Obviously all sets of equations corresponding to a multiequation (or a set of multiequations) are equivalent, i.e. they have exactly the same solutions.

We now introduce a few transformations of sets of multiequations, which are generalizations of the transformations presented in the previous section.

To introduce the first transformation we define recursively the common part and the frontier of a multiset of terms (variables or not). The common part (if it exists) of a nonempty multiset is a term and the frontier is a set of multiequations.

Given a nonempty multiset M of terms, if some of the terms is a variable then

a) The common part of M is any of the variables and the frontier of M is a set containing a single multiequation whose left member is the set of all variables in M, and whose right member is the multiset of all terms in M which are not variables;

else

b) if all root function symbols in the terms of M are equal to the same symbol f, then

   b1) the common part of M is the term $f(t_1,t_2,\ldots,t_i)$ where t_j (j=1,...,i) is the common part of the multiset M_j obtained by taking the j-th argument of all terms in M, and the frontier of M is the union of the frontiers of all multisets M_j (*). If some multiset M_j has no common part and no frontier, then also M has no common part and no frontier;

   else

   b2) M has no common part and no frontier.

For instance, given the multiset of terms

$$(f(x_1,g(a,x_2)),f(h(a,x_3),g(a,b)),f(x_4,g(a,b)))$$

the common part is

$$f(x_1,g(a,x_2))$$

and the frontier is

$$\{\{x_1,x_4\} = (h(a,x_3)),$$
$$\{x_2\} = (b,b)\}$$

We can now define the transformation of multiequation reduction. Let S=M be a multiequation belonging to a set Z of multiequations. The transformation is defined only if M is nonempty and has a common

(*) If f is a zero-adic function symbol, then the common part of M is the constant f and the frontier is empty.

part. Let C be the common part and F the frontier of M. The new set of multiequations is obtained by replacing S=M with the union of the multiequation S=(C) and of all the multiequations of F.

Theorem 3.1 - Let S=M (M nonempty) be a multiequation of a set Z of multiequations. If M has no common part, or if some variable in S belongs to the left member of some multiequation in the frontier F of M, then Z has no unifier. Otherwise, by applying multiequation reduction to the multiequation S=M we get an equivalent set Z' of multiequations.

Proof - If the common part of M does not exist, then the multiequation S=M has no unifier, since two terms should be made equal having a different function symbol in the corresponding subterms. Moreover, if some variable x of S occurs in some left member of the frontier, then it also occurs in some term t of M, and thus the equation x=t, with x occurring in t, belongs to a set of equations equivalent to Z. But this set has no unifier according to theorem 2.2.

To prove that Z and Z' are equivalent, we show first that a unifier of Z is also a unifier of Z'. In fact, if a substitution ϑ makes all terms of M equal, it will also make equal all the corresponding subterms, in particular all terms and variables which belong to left and right members of the same multiequation in the frontier. The multiequation S=(C) is also satisfied by construction. Conversely, if ϑ satisfies Z', then the multiequation S=M is also satisfied. In fact all terms in S and M are made equal: in their upper part (the common part) due to the multiequation S=(C) and in their lower part (the subterms not included in the common part) due to the set of multiequations F.

□

We now introduce a second transformation. Given a set of multiequations Z, we can obtain an equivalent set Z' of multiequations with disjoint left members with the operation of compactification, defined as follows. The set Z is first partitioned in classes in such a way that every two multiequations in a class either have a nonempty intersection of the left members, or there exists a chain of multiequations in the class from the first to the second multiequation where every pair of successive multiequations has this property. Finally the multiequations in every class are merged, i.e. they are transformed in single multiequations by making the union of their left and right members. In other words, we repeatedly merge pairs of multiequations whose left members have a nonempty intersection, until all left members are disjoint. Clearly, Z and Z' are equivalent, since there exists a set of equations corresponding to both Z and Z'.

For convenience, in what follows, we want to give a structure to a set of multiequations. Thus we introduce the concept of system of multiequations. A system R is a pair (T,U) where T is a sequence and U is a set of multiequations (either possibly empty), such that:

a) The sets of variables which constitute the left members of all multiequations in both T and U contain all variables and are disjoint;

b) The right members of all multiequations in T consist of no more than one term;

c) All variables belonging to the left member of some multiequation in T can only occur in the right member of any preceding multiequation in T.

We present now an algorithm for solving a given system R of multiequations. When the computation starts, the T part is empty, and every step of the following Algorithm B consists of "transferring" a multiequation from the U part, i.e. the unsolved part, to the T part, i.e. the triangular or solved part of R. When the U part of R is empty, the system is essentially solved. In fact, to get a system which has an equivalent set of equations in solved form, it is sufficient to substitute backwards . Notice that by keeping a solved system in this triangular form, we can hope of finding efficient algorithms for unification even when the mgu has a size which is exponential with respect to the size of the initial system. For instance, the mgu of the set of multiequations

$$\{\{x_1\} = \emptyset$$
$$\{x_2\} = (h(x_1,x_1)),$$
$$\{x_3\} = (h(x_2,x_2)),$$
$$\{x_4\} = (h(x_3,x_3))\}$$

is

$$\{(h(x_1,x_1),x_2),(h(h(x_1,x_1),h(x_1,x_1)),x_3),(h(h(h(x_1,x_1),h(x_1,x_1)),$$
$$h(h(x_1,x_1),h(x_1,x_1))),x_4)\}.$$

However an equivalent solved system can be given with empty U part and whose T part is

$$(\{x_4\} = (h(x_3,x_3)),$$
$$\{x_3\} = (h(x_2,x_2)),$$
$$\{x_2\} = (h(x_1,x_1)),$$
$$\{x_1\} = \emptyset).$$

Furthermore, a term representation using factorized subtrees could use a solution directly in this form.

Given a system with an empty T part, an equivalent system with an empty U part can be computed with the following algorithm.

Algorithm B

Let R=(T,U) be the given system of multiequations.

Step 1 - Repeat Steps 2-7 until the U part of R contains only multiequations, if any, with empty right members.

Step 2 - Select a multiequation S=M of U, with $M \neq \emptyset$.

Step 3 - Compute the common part C and the frontier F of M. If M has no common part, stop with failure.

Step 4 - If the left members of the frontier of M contain some variable of S, stop with failure.

Step 5 - Transform U using multiequation reduction on the selected multiequation and compactification.

Step 6 - Let $S = \{x_1,\ldots,x_n\}$. Apply the substitution $\vartheta = \{(C,x_1),\ldots,(C,x_n)\}$ to all terms in the right member of the multiequations of U.

Step 7 - Transfer the multiequation S=(C) from U to the end of T.

Step 8 - Transfer all the multiequations of U to the end of T, and stop with success.

Of course, if we want to use this algorithm for unifying two terms t_1 and t_2, we have to construct an initial system with empty T part and with the following U part:

$$\{\{x\} = (t_1,t_2),\ \{x_1\} = \emptyset,\ \{x_2\} = \emptyset,\ldots,\ \{x_n\} = \emptyset\},$$

where $x_1,x_2,\ldots,x_n$ are all the variables in t_1 and t_2 and x is a new variable which does not occur in t_1 and t_2. For instance, let $t_1 = f(x_1,g(x_2,x_3),x_2,b)$ and $t_2 = f(g(h(a,x_5),x_2),x_1,h(a,x_4),x_4)$. The initial system is:

$$U : \{\{x\} = (f(x_1,g(x_2,x_3),x_2,b),f(g(h(a,x_5),x_2),x_1,h(a,x_4),x_4)),\ \{x_1\} = \emptyset;\ \{x_2\} = \emptyset,\ \{x_3\} = \emptyset,\ \{x_4\} = \emptyset,\ \{x_5\} = \emptyset\}$$

$$T : (\ )$$

After the first iteration of Algorithm B we get

$$U : \{\{x_1\} = (g(h(a,x_5),x_2),g(x_2,x_3)),\ \{x_2\} = (h(a,x_4)),\ \{x_3\} = \emptyset,\ \{x_4\} = (b),\ \{x_5\} = \emptyset\}$$

$$T : (\{x\} = (f(x_1,x_1,x_2,x_4)))$$

We now eliminate variable x_2, obtaining

$$U : \{\{x_1\} = (g(h(a,x_5),h(a,x_4)),g(h(a,x_4),x_3)),\ \{x_3\} = \emptyset,\ \{x_4\} = (b),\ \{x_5\} = \emptyset\}$$

$$T : (\{x\} = (f(x_1,x_1,x_2,x_4)),\ \{x_2\} = (h(a,x_4)))$$

By eliminating variable x_1, we get

$$U : \{\{x_3\} = (h(a,x_4)),\ \{x_4,x_5\} = (b)\}$$

$$T : (\{x\} = (f(x_1,x_1,x_2,x_4)),\ \{x_2\} = (h(a,x_4)),\ \{x_1\} = (g(h(a,x_4),x_3)))$$

Finally, by eliminating first the set $\{x_4, x_5\}$ and then x_3, we get the solved system

$$U : \emptyset$$

$$\begin{aligned} T : (\{x\} &= (f(x_1, x_1, x_2, x_4)), \\ \{x_2\} &= (h(a, x_4)), \\ \{x_1\} &= (g(h(a, x_4), x_3)), \\ \{x_4, x_5\} &= (b), \\ \{x_3\} &= (h(a, b)) \end{aligned}$$

We can now prove the correctness of Algorithm B.

Theorem 3.2 - Algorithm B always terminates. If it stops with failure, then the given system has no unifier. If it stops with success, the resulting system is equivalent to the given system and has an empty unsolved part.

Proof - All transformations obtain systems equivalent to the given one. In fact, in Step 5 multiequation reduction obtains an equivalent set of equations according to theorem 3.1 and compactification transforms it again in a system. Step 6 applies substitution only to the terms in U, and its feasibility can be proved as in theorem 2.2. Step 7 can be applied since the multiequation S=(C), introduced during multiequation reduction, has not been modified by compactification, due to the condition tested in Step 4. For the same condition, transferring multiequation S=(C) from U to T still leaves a system. Step 8 is clearly feasible.

If the algorithm stops with failure, the system presently denoted by R (equivalent to the given one) has no solution according to theorem 3.1. Otherwise the final system has clearly an empty U part. Finally, the algorithm always terminates since at every cycle some variable is eliminated from the U part.

□

It is easy to see that, for a given system, the size of the final system depends heavily on the order of elimination of the multiequations. For instance, given the same system we showed earlier

$$\begin{aligned} U : \{\{x_1\} &= \emptyset, \\ \{x_2\} &= (h(x_1, x_1)), \\ \{x_3\} &= (h(x_2, x_2)), \\ \{x_4\} &= (h(x_3, x_3))\} \end{aligned}$$

$$T : (\ )$$

By eliminating the variables in the order x_2, x_3, x_4, x_1 we get the final system

$U : \emptyset$

$$\begin{aligned} T : (\{x_2\} &= (h(x_1,x_1)), \\ \{x_3\} &= (h(h(x_1,x_1),h(x_1,x_1))), \\ \{x_4\} &= (h(h(h(x_1,x_1),h(x_1,x_1)),h(h(x_1,x_1),h(x_1,x_1)))), \\ \{x_1\} &= \emptyset) \end{aligned}$$

Instead, by eliminating the variables in the order x_4,x_3,x_2,x_1 we get

$U : \emptyset$

$$\begin{aligned} T : (\{x_4\} &= (h(x_3,x_3)), \\ \{x_3\} &= (h(x_2,x_2)), \\ \{x_2\} &= (h(x_1,x_1)), \\ \{x_1\} &= \emptyset) \end{aligned}$$

Looking at Algorithm B it is clear that the main source of complexity is Step 6, since it may make many copies of large terms. In the following (and this is the heart of our algorithm) we show that if the system has unifiers, then there always exists a multiequation in U (if not empty) such that by selecting it we do not need Step 6 of the algorithm, since the variables in its left member do not occur elsewhere in U. We need the following definition.

Given a system R, let us consider the subset V_u of variables obtained by making the union of all left members S_i of the multiequations in the U part of R. Since the sets S_i are disjoint, they determine a partition of V_u. Let us now define a relation on the classes S_i of this partition:

$S_i < S_j$ iff there exists a variable of S_i occurring in some term of M_j, where M_j is the right member of the multiequation whose left member is S_j. Let now $<^*$ be the transitive closure of $<$.

Now we can prove the following theorem and corollary.

<u>Theorem 3.3</u> - If a system R has a unifier, then the relation $<^*$ is a partial ordering.

<u>Proof</u> - If $S_i < S_j$, then in all unifiers of the system, the term substituted for every variable in S_i must be a subterm of the term substituted for every variable in S_j. Thus, if the system has a unifier, the graph of the relation $<$ cannot have cycles. Therefore its transitive closure must be a partial ordering. □

<u>Corollary</u> - If the system R has a unifier and its U part is nonempty, there exists a multiequation S=M such that the variables in S do not occur elsewhere in U.

Proof - Let S=M be a multiequation such that S is "on top" of the partial ordering $<^*$ (i.e., $\sim\exists\, S_i, S < S_i$). The variables in S so not occur neither in the other left members of U (since they are disjoint) not in any right member M_i of U, since otherwise $S < S_i$. □

We can now refine the nondeterministic Algorithm B giving the general version of our unification algorithm.

Algorithm UNIFY

Let R = (T,U) be the given system of multiequations.

Step 1 - Repeat Steps 2-6 until the U part of R is empty; then stop with success.

Step 2 - Select a multiequation S=M of U such that the variables in S do not occur elsewhere in U. If a multiequation with this property does not exist, stop with failure.

Step 3 - If M is empty, then transfer this multiequation from U to the end of T and go to Step 1.

Step 4 - Compute the common part C and the frontier F of M. If M has no common part, stop with failure.

Step 5 - Transform U using multiequation reduction on the selected multiequation, and compactification.

Step 6 - Transfer the multiequation S=(C) from U to the end of T.

A few comments are needed. Besides Step 6 of Algorithm B, we have erased also Step 4 for the same reason. Furthermore, in Algorithm B we were forced to wait to transfer multiequations with empty right members since substitution in that case would have required a special treatment.

By applying Algorithm UNIFY to the system which was previously solved with Algorithm B, we see that we must first eliminate variable x, then variable x_1, then variables x_2 and x_3 together and finally variables x_4 and x_5 together, getting the following final system

$$U : \emptyset$$

$$
\begin{aligned}
T : (\{x\} &= (f(x_1, x_1, x_2, x_4)),\\
\{x_1\} &= (g(x_2, x_3)),\\
\{x_2, x_3\} &= (h(a, x_4)),\\
\{x_4, x_5\} &= (b)\ )
\end{aligned}
$$

Note that the solution obtained using Algorithm UNIFY is more concise than the solution previously obtained using Algorithm B, for two reasons. First, variables x_2 and x_3 have been recognized as equivalent; second, the right member of x_1 is more factorized. This improvement is not casual, but is intrinsec in the ordering behaviour of Algorithm UNIFY.

```
type system = record
                T,U : ↑ListOfMulteq
              end;
     multiequation = record
                       S : ↑SetOfVariables;
                       M : ↑ListOfTerms
                     end;
     TempMultiequation = record
                           S,M : ↑ListOfTerms
                         end;
     term = record
              case isfun : boolean of
                true :(fsymb : funname;
                       args : ↑ListOfTerms);
                false :(v : ↑variable)
            end;
     Psystem = ↑system;
     Pterm = ↑term;
     PListOfTerms = ↑ListOfTerms;
     PListOfTempMulteq = ↑ListOfTempMulteq;
```

Fig. 1

---

4 - COMPLEXITY ANALYSIS OF MULTIEQUATION REDUCTION

We begin here the complexity analysis of our algorithm, by discussing the part performing multiequation reduction. To carry on this analysis we show in Fig. 1,2 and 3 a PASCAL version of the algorithm. This program is not complete and will be refined in the next sections. However, we emphasize that all the missing procedures, except for "SelectMultiequation" and "compact", have an obvious meaning and can be easily implemented with constant complexity. In the Appendix we give a possible implementation of these procedures. Similarly, the data types definitions in Fig. 1 will be refined in the next sections by adding new fields to the records. Furthermore, the unspecified data types "SetOfVariables" and "variable" will be defined, while the remaining unspecified data types are all straightforward and are implemented in the Appendix.

Note that the frontier is represented as a list of so called temporary multiequations, which are a simplified version of the multiequations. In a "TempMultiequation" the left member (S field) consists of a list of variable terms, whereas the left member of a "multiequation" has a more complex structure which will be described in the next sections.

The procedure "reduce" in Fig. 3 computes the common part and the frontier of a list of terms M, in a way which closely corresponds to the definition given in the previous section. The repeat statement

```
procedure unify (var R : Psystem);
var mult : ↑multiequation;C : ↑term;F : ↑ListOfTempMulteq;
begin
  repeat
    SelectMultiequation(R↑.U,mult);
    if not EmptyListOfTerms(mult↑.M) then
    begin
      reduce(mult↑.M,C,F);
      compact(F,R↑.U);
      mult↑.M:=AddToEndOfListOfTerms(C,CreateListOfTerms)
    end;
    R↑.T:=AddToEndOfListOfMulteq(mult,R↑.T)
  until EmptyListOfMulteq(R↑.U);
end; (*unify*)
```

Fig. 2

(lines 12-24) computes a list "argsofm" which contains all the arguments of the terms of M. More precisely, if each term of M has i arguments, then "argsofm" has i elements, and the j-th element of "argsofm" (j=1,...,i) is a multiset M_j of terms obtained by taking the j-th argument of all terms in M. For the efficiency of subsequent computations, we represent the elements M_j of "argsofm" as temporary multiequations, in such a way that we can separate variable and non variable terms by putting them in the left and right member of the temporary multiequations. This is achieved by procedure "AddTerm" which adds a term (the first argument) to the first temporary multiequation of a list of temporary multiequations (the second argument) and then moves this temporary multiequation to the end of the third argument.

The while statement at the end of procedure "reduce" (lines 25-37) computes, at each iteration, the common part and the frontier of the j-th element M_j of "argsofm". According to the definition given in the previous section, two cases may arise: either some term of M_j is a variable or not. In the former case, i.e. when the S field of the temporary multiequation representing M_j is not empty, the common part and the frontier are obtained according to part a) of the definition; otherwise "reduce" is called recursively on M_j.

We first analyze a single call to procedure "reduce". Let c_a be a constant denoting the complexity in time of executing lines 1-11, 25[*], and 38-39; c_b of 12-18, and 23-24; c_c of 18-22; c_d of 25[*]-29, and 35-37; c_e of 30-33 and c_f of 34. Furthermore let n_m, n_c and n_f be the number of (function and variable) symbols respectively in the multiset M of terms (which is the datum of procedure "reduce"), in the common part and in the frontier of M (if they exist); and let n_{tm} and n_{tf} be the number of terms respectively in M and in all the multiequations of the frontier of M.

(*) In the complexity analysis, the first line of a while statement is considered twice to take into account the first time the test is executed.

```
procedure reduce(M : PListOfTerms;var commonpart : Pterm;
                 var frontier : PListOfTempMulteq);
var fs : funname;argsofm,argsofm1,newfrontier : ↑ListOfTempMulteq;
    t,newcommonpart : ↑term;argsoft,argsofcp : ↑ListOfTerms;
    temp : ↑TempMultiequation;
begin
  frontier:=CreateListOfTempMulteq;
  argsofcp:=CreateListOfTerms;
  argsofm:=CreateListOfTempMulteq;
  t:=HeadOfListOfTerms(M);
  fs:=t↑.fsymb;
  repeat
    argsofm1:=CreateListOfTempMulteq;
    t:=HeadOfListOfTerms(M);
    M:=TailOfListOfTerms(M);
    if diffsymb(t↑.fsymb,fs) then fail;
    argsoft:=t↑.args;
    while not EmptyListOfTerms(argsoft) do
    begin
      AddTerm(HeadOfListOfTerms(argsoft),argsofm,argsofm1);
      argsoft:=TailOfListOfTerms(argsoft)
    end;
    argsofm:=argsofm1
  until EmptyListOfTerms(M);
  while not EmptyListOfTempMulteq(argsofm) do
  begin
    temp:=HeadOfListOfTempMulteq(argsofm);
    argsofm:=TailOfListOfTempMulteq(argsofm);
    if not EmptyListOfTerms(temp↑.S) then
    begin
      newcommonpart:=HeadOfListOfTerms(temp↑.S);
      newfrontier:=AddToEndOfListOfTempMulteq(temp,CreateListOfTempMulteq)
    end;
    else reduce(temp↑.M,newcommonpart,newfrontier);
    argsofcp:=AddToEndOfListOfTerms(newcommonpart,argsofcp);
    frontier:=AppendListsOfTempMulteq(frontier,newfrontier)
  end;
  commonpart:=BuildFunctionTerm(fs,argsofcp)
end; (*reduce*)
```

Fig. 3

Note that n_{tm} is the number of terms in the datum of every recursive call of "reduce", and is also the number of terms (variable and not) in every multiequation in the frontier. Thus, the value n_{tf}/n_{tm} is the number of multi-equations in the frontier. Furthermore, among n_m, n_c, n_f, n_{tm} and n_{tf} the following relation holds

$$n_c \cdot n_{tm} = n_m - n_f + n_{tf}$$

In fact, every symbol in the common part which is not a variable stands for n_{tm} function symbols in the part of M not included in the frontier, while every symbol of the common part which is a variable stands for a multiequation in the frontier.

We can now prove the following theorem.

Theorem 4.1 - Let us consider a call to procedure "reduce" with a multiset M of terms.

a) If the procedure terminates with success, then the complexity in time is

(4.1) $$C_m = (n_m - n_f)\left(c_b + c_c + \frac{c_a + c_d + c_f}{n_{tm}}\right) + n_{tf}\left(c_c + \frac{c_d + c_e}{n_{tm}}\right) - (c_c n_{tm} + c_d + c_f)$$

b) If the procedure terminates with success or if the procedure fails, then teh complexity in time is bounded by

(4.2) $$C_m \leq n_m\left(c_b + c_c + \frac{c_a + c_d + c_e + c_f}{n_{tm}}\right) - (c_c n_{tm} + c_d + c_f)$$

Proof -

a) We prove (4.1) inductively on the recursive calling structure of "reduce". Thus the basis consists of analyzing the complexity of a call to "reduce" which does not call itself. There are two possibilities: i) the root function symbol of all terms in M is a constant; ii) for every j, there exists a term in M such that its j-th argument is a variable. We will prove the basis together with the inductive step. During a generic call to "reduce", let S_1 (S_2) be the set of argument positions j, for which there exists (does not exist) a term in M such that its j-th argument is a variable. Furthermore, let k_1 and k_2 be the cardinalities of S_1 and S_2. Thus $k_1+k_2=k$, where k is the number of arguments of all terms in M. Therefore the basis i) corresponds to the case $k_1=k_2=k=0$, while the basis ii) corresponds to the case $k_2=0$. Now let M_j be the multiset of terms obtained by taking the j-th argument of all terms in M, and, if $j \in S_2$, let n^j_m, n^j_c, n^j_f and n^j_{tf} be the above defined quantities for M_j. Note that $n^j_{tm}=n_{tm}$. We have the following relations:

(4.3) $$\begin{cases} n_{tf} = \sum_{j \in S_2'} n^j_{tf} + k_1 \cdot n_{tm} \\ n_m - n_f = n_{tm} + \sum_{j \in S_2} (n^j_m - n^j_f) \end{cases}$$

Finally let C^j_m be the complexity of applying the procedure "reduce" to M_j ($j \in S_2$).

By symbolically executing the procedure "reduce" on the multiset of terms M, we have:

(4.4) $$C_m' = c_a + n_{tm}c_b + kn_{tm}c_c + kc_d + k_1c_e + \sum_{j \in S_2} C^j_m + k_2c_f$$

Using the inductive hypothesis

$$c_m^j = (n_m^j - n_f^j)\left(c_b + c_c + \frac{c_a+c_d+c_f}{n_{tm}}\right) +$$

$$+ n_{tf}^j \left(c_c + \frac{c_d+c_e}{n_{tm}}\right) - (c_c n_{tm} + c_d + c_f),$$

from (4.4) and (4.3) we get (4.1).

b) We prove (4.2) inductively. Here the basis is the same as for part a) (which is proved simply by checking that the right member of (4.2) is an upper bound for the right member of (4.1)) plus the case in which the procedure fails directly. Here the complexity is at most

$$c_a + n'_{tm} c_b + (n_m - n_{tm})\ c_c$$

and thus (4.2) holds, since $n_m \geq n_{tm}$.
During the induction step we will use the following relation

(4.5) $$\sum_{j \in S_2} n_m^j \leq n_m - n_{tm} - k_1 n_{tm}$$

By symbolically executing the procedure "reduce", we have the same right member as in (4.4), but here the $\leq$ sign holds, since some of the work may be skipped if one of the internal recursive calls to "reduce" fails. Using the inductive hypothesis

$$c_m^j \leq n_m^j \left(c_b + c_c + \frac{c_a+c_d+c_e+c_f}{n_{tm}}\right) - (c_c n_{tm} + c_d + c_f)$$

from (4.5) we get (4.2).

□

We can now give an upper bound to the complexity of all the calls to "reduce".

Theorem 4.2 - Let t_s be the total number of symbols in the initial system of multiequations, and let t_{tfr} be the total number of terms (variable and not) which appear in all the frontiers returned by "reduce" in all iterations of "unify" before its termination (with success or failure). Then the complexity in time of all the calls to "reduce" is bounded by

$$C_r \leq t_s(c_a + c_b + c_c + c_d + c_e + c_f) + t_{tfr}(c_c + c_d + c_e)$$

Proof - The expression $(n_m - n_f)$ in (4.1) is a lower bound to the total number of symbols eliminated from the U part of the system in an iteration of "unify". In fact, the call to "reduce" eliminates $(n_m - n_f)$ symbols, but creates a new term which is the common part. However this term (together with some variables) is put in the T part by line 12 of "unify". Furthermore, other variable symbols may be eliminated by "compact". The total complexity of the calls to "reduce" in a number of iterations of "unify" is thus bounded by

$$(4.7) \qquad C'_r \le (t_s - t'_s)(c_a + c_b + c_c + c_d + c_f) + t'_{tfr}(c_c + c_d + c_e)$$

where t'_s is the total number of symbols in the U part of the system after these iterations and t'_{tfr} is the total number of terms which appeared in all the frontiers obtained up to now. Note that (4.7) can be obtained from (4.1) since $n_{tm} \ge 1$, because otherwise "reduce" would not be called.

If "unify" terminates with success, then (4.6) is obtained from (4.7) since at the end $t'_s = 0$ and $t_{tfr} = t'_{tfr}$. If "unify" fails because "Select-Multiequation" fails, then $t_{tfr} = t'_{tfr}$ and (4.6) still holds. If "unify" fails because "reduce" fails, then the complexity of the last call to "reduce" is bounded by (4.2). Since $t'_s \ge n_m$ and $t_{tfr} = t'_{tfr}$, we can thus derive (4.6).

□

5 - A FINAL REFINEMENT: A SIMPLE ALGORITHM FOR UNIFICATION

In this section we present an algorithm which associates to every multiequation a counter which contains the number of other occurrences in U of the variables in its left member. This counter is initialized by scanning the whole U part at the beginning; is decremented whenever occurrences of some of its variables appear in the left members of the multiequations of some frontier after multiequation reduction; is tested for zero to select the multiequation to be transferred. More specifically, to avoid scanning the whole U part, whenever the counter of a multiequation is decreased to zero, the multiequation is put on a stack of multiequations ready to be transferred. When two or more multiequations in U are merged in the compactification phase, the counter associated with the new multiequation is obviously set to a value which is the sum of the contents of the old counters.

In Fig. 4-6 we add the data type definitions and the procedures necessary to completely specify the program in Fig. 2 and 3 (a few straightforward parts are still missing and will be reported in the Appendix). We want to add a few comments. According to the definition of system of multiequations, every variable occurs in the left member of a single multiequation. In the compactification phase, this multiequation must be accessed from other occurrences of the variable. Thus all variable occurrences (represented by "terms" with the tag field equal to false) have a field "v" pointing to a single "variable", and the "variable" has a field "m" pointing to the multiequation. When two multiequations are merged by "compact", one of them is erased and thus all the pointers to it must be moved to the other. Therefore, to minimize the computing cost, we add to every multiequation S=M a counter "varnumb" containing the number of variables in S, and we choose to erase the multiequation with the smallest number of variables.

Finally, we remark that, to avoid using a doubly-linked list, we do not actually remove erased multiequations, but simply mark them using the "erased" field. Furthermore we use as a stack the same list representing the U part by moving to the top the multiequations ready to be transferred.

```
type system = record
                T,U : ↑ListOfMulteq
              end;
     SetOfVariables = record
                        counter,varnumb : integer;
                        vars : ↑ListOfVariables
                      end;
     multiequation = record
                       erased : boolean;
                       S : ↑SetOfVariables;
                       M : ↑ListOfTerms
                     end;
     TempMultiequation = record
                           S,M : ↑ListOfTerms
                         end;
     variable = record
                  name : varname;
                  m : ↑multiequation
                end;
     term = record
              case isfun : boolean of
                true :(fsymb : funname;
                       args : ↑ListOfTerms);
                false :(v : ↑variable)
            end;
     Psystem = ↑system;
     Pterm = ↑term;
     PListOfTerms = ↑ListOfTerms;
     PListOfMulteq = ↑ListOfMulteq;
     PListOfTempMulteq = ↑ListOfTempMulteq;
     Pmultiequation = ↑multiequation;
```

Fig. 4

To complete the complexity analysis of procedure "unify", let

c_g be the complexity in time of executing lines 1-3 and 14 of "unify";

c_h of lines 4-13 of "unify", 1-8 and 14 of "SelectMultiequation", 1-5 and 42 of "compact";

c_i of lines 9-13 of "SelectMultiequation";

c_j of lines 5-13 and 39-41 of "compact";

c_k of lines 13-28 and 35-38 of "compact";

c_l of lines 29-34 of "compact".

Furthermore let

t_s be the total number of symbols in the initial system of multiequations;

t_{tfr} be the total number of terms (variable and not) which appear in all the frontiers;

t_{mei} be the total number of multiequations in the initial system;

```
 1  procedure SelectMultiequation (var U : PListOfMulteq;
 2                         var mult : Pmultiequation);
 3  var NotErasedHeadOfU : boolean;m : ↑multiequation;
 4  begin
 5    mult:=HeadOfListOfMulteq(U);
 6    if mult↑.erased or not(mult↑.S↑.counter = 0) then fail;
 7    mult↑.erased:=true;
 8    NotErasedHeadOfU:=false;
 9    repeat
10      m:=HeadOfListOfMulteq(U);
11      if m↑.erased then U:=TailOfListOfMulteq(U)
12      else NotErasedHeadOfU:=true
13    until EmptyListOfMulteq(U) or NotErasedHeadOfU;
14  end; (*SelectMultiequation*)
```

Fig. 5

---

t_{mef} be the total number of multiequations in the final system;

t_{mefr} be the total number of temporary multiequations which appear in all the frontiers;

t_{vfr} be the total number of variable occurrences in the left members of the temporary multiequations in all the frontiers;

t_v be the total number of distinct variables in the system;

t_{pf} be the total number of pointers from variables to multiequations which are moved by "compact" in the merging phase.

The following theorem can be easily proved.

Theorem 5.1 - The complexity in time of "unify", with the versions of "SelectMultiequation" and "compact" given in this section, is bounded by

$$C \le (c_a + c_b + c_c + c_d + c_e + c_f)t_s + (c_c + c_d + c_e)t_{tfr} + c_g + c_h t_{mef} + c_i(t_{mei} + 2t_{mef}) + c_j t_{mefr} + c_k t_{vfr} + c_l t_{pf} \tag{5.1}$$

Proof - Most of the terms in (5.1) can be derived by the definitions of "unify", "SelectMultiequation" and "compact" by simple inspection or by utilizing theorem 4.2. We only want to comment on the fifth term. The total number of elements deleted from the list U in line 11 of "SelectMultiequation" is equal to the number of initial multiequations plus the number of "ready" multiequations added on top of the list U by line 40 of "compact". The latter is exactly t_{mef}. Furthermore the body of the repeat statement of "SelectMultiequation" is executed once for each call of "SelectMultiequation" (i.e. t_{mef} times) without deleting any element. Finally, note that if "unify" fails, (5.1) still holds if the values t_{tfr}, t_{mef}, t_{mefr}, t_{vfr} and t_{pf} are referred to the state of the system when failure occurs. □

```
procedure compact(F : PListOfTempMulteq;var U : PListOfMulteq);
var temp : ↑TempMultiequation;mult,mult1,multt : ↑multiequation;
    v : ↑variable;varterm : ↑term;
begin
  while not EmptyListOfTempMulteq(F) do
  begin
    temp:=HeadOfListOfTempMulteq(F);
    F:=TailOfListOfTempMulteq(F);
    varterm:=HeadOfListOfTerms(temp↑.S);
    mult:=varterm↑.v↑.m;
    temp↑.S:=TailOfListOfTerms(temp↑.S);
    mult↑.S↑.counter:=mult↑.S↑.counter - 1;
    while not EmptyListOfTerms(temp↑.S) do
    begin
      varterm:=HeadOfListOfTerms(temp↑.S);
      mult1:=varterm↑.v↑.m;
      temp↑.S:=TailOfListOfTerms(temp↑.S);
      mult1↑.S↑.counter:=mult1↑.S↑.counter - 1;
      if not(mult = mult1) then
      begin
        if mult↑.S↑.varnumb < mult1↑.S↑.varnumb then
        begin
          multt:=mult1;
          mult1:=mult;
          mult:=multt
        end;
        mult↑.S↑.counter:=mult↑.S↑.counter + mult1↑.S↑.counter;
        mult↑.S↑.varnumb:=mult↑.S↑.varnumb + mult1↑.S↑.varnumb;
        repeat
          v:=HeadOfListOfVariables(mult1↑.S↑.vars);
          mult1↑.S↑.vars:=TailOfListOfVariables(mult1↑.S↑.vars);
          v↑.m:=mult;
          mult↑.S↑.vars:=AddToEndOfListOfVariables(v,mult↑.S↑.vars)
        until EmptyListOfVariables(mult1↑.S↑.vars);
        mult↑.M:=AppendListsOfTerms(mult↑.M,mult1↑.M);
        mult1↑.erased:=true
      end
    end;
    mult↑.M:=AppendListsOfTerms(mult↑.M,temp↑.M);
    if mult↑.S↑.counter = 0 then U:=AddToFrontOfListOfMulteq(mult,U)
  end
end; (*compact*)
```

Fig. 6

We can prove the following theorem.

Theorem 5.2 - Let t_v be the total number of distinct variables in the system. Then an upper bound(*) to t_{pf} is given by

$$t_{pf} \leq t_v \lfloor \log t_v \rfloor$$

(*) The better bound $t_{pf} \leq \left\lfloor \frac{t_v}{2} \right\rfloor \cdot \lceil \log t_v \rceil$ can be obtained with a more accurate analysis.

Proof - Let us compute the maximum number of times the procedure "compact" can move a pointer from a variable V_i to a multiequation. A pointer from V_i to a multiequation S=M is moved when this multiequation is merged with a multiequation S'=M' such that $|S| \leq |S'|$. Thus, after the merging, V_i will point to a new multiequation S"=M" with $|S"| \geq 2|S|$. Therefore, the total number of times the pointer from V_i can be moved is bounded by $\lfloor \log t_V \rfloor$. Since the total number of variables is t_V, then t_{pf} must be bounded by $t_V \lfloor \log t_V \rfloor$.

□

We can now determine the complexity of algorithm "unify".

Theorem 5.3 - The computational complexity in time of algorithm "unify", with the versions of "SelectMultiequation" and "compact" given in this section, is bounded by

$$C \leq c_A + c_B t_s + c_C t_V \lfloor \log t_V \rfloor$$

where t_s is the total number of symbols and t_V the total number of distinct variables in the initial system of multiequations, and c_A, c_B, c_C are suitable constants.

Proof - This result derives from theorem 5.1 by noting that we trivially have $t_{mef} \leq t_V$, $t_{mei} \leq t_V$, $t_{mefr} \leq t_{tfr}$, $t_{vfr} \leq t_{tfr}$ and $t_{pf} \leq t_V \lfloor \log t_V \rfloor$ for theorem 5.2. Furthermore, $t_{tfr} \leq t_s$ since all the terms in the frontier appear in only one frontier and their roots are different symbols of the initial system. □

Theorem 5.4 - The algorithm "unify" with the versions of "SelectMultiequation" and "compact" given in this section, is linear in space with the total number of symbols in the initial system of multiequations.

Proof - In our program, memory is allocated only through declaration of unstructured variables and through the execution of the single record-allocation procedure "new". Thus any part of the program which is linear in time must also be linear in space. But the only part which is not linear in time is lines 29-34 of "compact" where no variable declaration is present, and where the number of calls to the allocating procedure " new" is equal to the number of calls to the deallocating procedure "dispose". Thus the total number of allocated cells is not modified in this part. □

In many practical cases the number of distinct variables is small with respect to the total number of symbols and thus the dominant term in the complexity is the linear one. However, there are pathological cases where the logarithmic term is dominant. For instance, let us consider the class of unification problems exemplified by the following problem with two terms and 8 distinct variables:

$f(x_1, x_3, x_5, x_7, x_1, x_5, x_1)$ and

$f(x_2, x_4, x_6, x_8, x_3, x_7, x_5)$.

It is easy to see that the total number of times pointers are moved is proportional to $t_v \log t_v$, whereas the total number of symbols t_s is linear with t_v.

We point out that we could improve the worst case behaviour of our algorithm with a different implementation of the operation of multiequation merging. In fact, we could represent sets of variables as trees instead of as lists, and we could use the well-known UNION-FIND algorithms [15]. The complexity in time of this implementation is known to be almost linear with the number of FIND operations, i.e. of variable occurrences in our case. However such an implementation would be more complicated that the one presented in this section, and, in some cases, it might be less efficient. Furthermore, even with the above modification, the complexity of the algorithm would not be linear, whereas an algorithm of linear complexity is presented in the next section.

6 - AN ALTERNATIVE FINAL REFINEMENT: THE LINEAR ALGORITHM

In the previous section we have seen a technique for determining a multiequation S=M in the U part of the system such that the variables in S do not occur elsewhere in U. This technique used a counter added to every multiequation. In this section we introduce a depth-first search technique which is better in the worst case.

We remind that we have defined in section 3 a relation $<$ between the sets of variables S_i which constitute the left members of the multiequations in the U part of a given system. If $S_i < S_j$, then a term can be found in M_j where a variable of S_i does occur. Our technique consists of choosing a multiequation $S_i = M_i$ whatsoever, and of constructing a sequence

$$S_{i_0} = S_i,\ S_{i_1},\ S_{i_2}, \ldots,\ S_{i_n}$$

with

$$S_{i_k} < S_{i_{k+1}} \qquad k=0,1,\ldots,n-1$$

until a set S_{i_n} is found such that either no S_j exists with $S_{i_n} < S_j$ or $S_{i_n} = S_{i_k}$ for some $0 \leq k \leq n-1$. In the first case we have found a suitable multiequation to transfer, while in the second case theorem 3.3 assures that the system has no unifier.

Interpreting the above construction in terms of the graph G of the relation $<$, we follow a simple path in the graph starting from a node S_i whatsoever and marking the nodes on the path. We stop with a node S_{i_n} when either no next node exists or when we find a marked next node. In the latter case we stop with failure, while in the former case we perform a step of algorithm "unify" and we obtain a new system R'. It is easy to see that the graph G' of the relation $<$ defined in R' is obtained from G with the following operations

i) delete S_{i_n} and all its incoming arcs;

ii) coalesce some sets of nodes of G;

iii) add some arcs.

```
type system = record
                T,U : ↑ListOfMulteq
              end;
     SetOfVariables = record
                        vars : ↑ListOfVariables;
                        varocc : ↑ListOfTerms;
                        eqvar : ↑ListOfMulteq;
                        marked : boolean;
                        mergedmult : ↑multiequation
                      end;
     multiequation = record
                       erased : boolean;
                       S : ↑SetOfVariables;
                       M : ↑ListOfTerms
                     end;
     TempMultiequation = record
                           S,M : ↑ListOfTerms
                         end;
     variable = record
                  name : varname;
                  M : ↑multiequation
                end;
     term = record
              marked : boolean;
              case isfun : boolean of
                true :(fsymb : funname;
                       args : ↑ListOfTerms;
                       case top : boolean of
                         true :(mult : ↑multiequation);
                         false :(ffather : ↑term));
                false :(v : ↑variable;
                        vfather : ↑term;
                        deleted : boolean)
            end;
     Psystem = ↑system;
     Pterm = ↑term;
     PListOfTerms = ↑ListOfTerms;
     PListOfMulteq = ↑ListOfMulteq;
     PListOfTempMulteq = ↑ListOfTempMulteq;
     Pmultiequation = ↑multiequation;
```

Fig. 7

By marking a coalesced node iff any of its components is marked, the path of marked nodes in G is still a path of marked nodes in G'. If this path is nonsimple, we can stop with failure. Otherwise we can continue the marking operation from its last node. If the deleted node S_{i_n} coincide with the initial node S_{i_0}, namely if it was the only marked node, we can restart in G' from any new node.

Since a mark can be eliminated only by erasing a node, whereas trying to mark a marked node causes failure, no node can be marked twice. Thus

```
procedure SelectMultiequation (var U : PListOfMulteq;var mult :
                                Pmultiequation);
var me,me1 : ↑multiequation;ontop,Me1EquatedToMe,ErasedHeadOfU :
    boolean;vterm : ↑term;
begin
  me:=HeadOfListOfMulteq(U);
  me↑.S↑.marked:=true;
  ontop:=false;
  repeat
    while not EmptyListOfTerms(me↑.S↑.varocc) do
    begin
      vterm:=HeadOfListOfTerms(me↑.S↑.varocc);
      if vterm↑.deleted then
        me↑.S↑.varocc:=TailOfListOfTerms(me↑.S↑.varocc)
      else me:=DominatingMulteq(vterm)
    end;
    Me1EquatedToMe:=true;
    while (not EmptyListOfMulteq(me↑.S↑.eqvar)) and Me1EquatedToMe do
    begin
      me1:=HeadOfListOfMulteq(me↑.S↑.eqvar);
      if (me1 = me) or (me1↑.S↑.mergedmult = me) then
        me↑.S↑.eqvar:=TailOfListOfMulteq(me↑.S↑.eqvar)
      else Me1EquatedToMe:=false
    end;
    if EmptyListOfMulteq(me↑.S↑.eqvar) then ontop:=true
    else
    begin
      me↑.S↑.eqvar:=TailOfListOfMulteq(me↑.S↑.eqvar);
      merge(me,me1)
    end
  until ontop;
  mult:=me;
  me↑.erased:=true;
  ErasedHeadOfU:=true;
  while (not EmptyListOfMulteq(U)) and ErasedHeadOfU do
  begin
    me1:=HeadOfListOfMulteq(U);
    if me1↑.erased then U:=TailOfListOfMulteq(U)
    else ErasedHeadOfU:=false
  end;
end; (*SelectMultiequation*)
```

Fig. 8

---

the cost of the above construction, during the entire execution of algorithm "unify", is linear with the number of nodes in the initial graph, i.e. with the number of distinct variables in the system.

The nonlinear behaviour of the program given in the previous section was due to the fact that in the compactification phase two multiequations were possibly merged which were both already the result of previous mergings. Since every merging implies rewriting some pointers, it could happen that some pointer had to be rewritten a number of times which, in the worst case, was logarithmic with the number of distinct variables. To

```
function DominatingMulteq(vterm : Pterm) : Pmultiequation;
var me : ↑multiequation;fterm : ↑term;
begin
  vterm↑.marked:=true;
  fterm:=vterm↑.vfather;
  if fterm↑.marked then fail;
  fterm↑.marked:=true;
  while not fterm↑.top do
  begin
    fterm:=fterm↑.ffather;
    if fterm↑.marked then fail;
    fterm↑.marked:=true
  end;
  me:=fterm↑.mult;
  if me↑.S↑.marked then fail;
  me↑.S↑.marked:=true;
  DominatingMulteq:=me
end; (*DominatingMulteq*)
```

Fig. 9

avoid this fact, instead of actually merging two multiequations, a two-way link could simply be added between every pair of multiequations to be merged. The actual merging should then take place only when a "head" can be determined, for each set of multiequations to be merged, which constitutes a unique propagation starting point. Of course one must be sure never to have to merge two heads.

In the program described in the previous section such a delayed merging technique was not useful, since at any time an updated global counter had to be maintained for every set of "equivalent" variables. The pointers to such global counters might then again have to be moved a logarithmic number of times.

Here one may delay mergings until the above mentioned marked node path meets a node with no successors. Then this node is considered a suitable propagation head and the mergings with it take place until either a node to be merged is marked or a merged node happens to have successors. In the former case the program stops with failure, while in the latter case the marking phase is continued. If all the required mergings take place without finding successors, then the resulting node corresponds to a multiequation suitable to be transferred. In conclusion, in any intermediate step of the algorithm, there is more than one propagation head. However, all heads are marked nodes, and thus any attempt to merge two propagation heads causes failure.

In Fig. 7-11 we give the data type definitions and the procedures necessary to complete the program in Fig. 2-3 (straightforward parts are given in the Appendix). During the marking phase, the path of marked nodes is obtained by visiting the terms from the leaves (variable occurrences) to the roots. Thus a field pointing to the father (selected by "ffather" in function terms and by "vfather" in variable terms) has been added to each term. When a function term is in the right member of a multiequation it has no father and thus an alternative field "mult" points to such a multiequation.

```
procedure merge(me,me1 : Pmultiequation);
var fterm : ↑term;v : ↑variable;
begin
  if me1↑.S↑.marked then fail;
  me1↑.S↑.marked:=true;
  repeat
    v:=HeadOfListOfVariables(me1↑.S↑.vars);
    me1↑.S↑.vars:=TailOfListOfVariables(me1↑.S↑.vars);
    v↑.m:=me;
    me↑.S↑.vars:=AddToEndOfListOfVariables(v,me↑.S↑.vars)
  until EmptyListOfVariables(me1↑.S↑.vars);
  me↑.S↑.varocc:=me1↑.S↑.varocc;
  me↑.S↑.eqvar:=AppedListsOfMulteq(me↑.S↑.eqvar,me1↑.S↑.eqvar);
  me1↑.S↑.mergedmult:=me;
  while not EmptyListOfTerms(me1↑.M) do
  begin
    fterm:=HeadOfListOfTerms(me1↑.M);
    me1↑.M:=TailOfListOfTerms(me1↑.M);
    fterm↑.mult:=me;
    me↑.M:=AddToEndOfListOfTerms(fterm,me↑.M)
  end;
  me1↑.erased:=true
end; (*merge*)
```

Fig. 10

In a "SetOfVariables", besides the field "vars" containing a list of the variables of the set, there are the fields "varocc" and "eqvar". The first field contains the list of all the occurrences of the variables in the set, and the second field contains the list of all the multiequations containing variables which have been equated to some variables in the set. In other words "eqvar" contains the list of all multiequations which ought to be merged with the given multiequation, but whose merging has been delayed. After merging, the field "mergedmult" of the eliminated multiequation points to the resulting multiequation. This field is needed to redirect the pointers from the "eqvar" field of other multiequations. Furthermore, instead of actually deleting variable occurrences from the "varocc" lists, we mark their "deleted" field.

The multiequation to be transferred is detected by the <u>repeat</u> statement (lines 9-31) of "SelectMultiequation". The body of this statement consists of the sequence of two phases: the marking phase and the merging phase. The marking phase (<u>while</u> statement, lines 10-16) goes on until a multiequation is found such that the variables in its left member do not occur elsewhere in the U part (i.e. its "varocc" field is empty). Then this multiequation becomes a propagation head in the merging phase, and it is merged with the first multiequation different from itself (if any) found in its "eqvar" list.

To perform the complexity analysis of procedure "unify", let

c_g be the complexity in time of executing of lines 1-3 and 14 of "unify";

c_m of lines 4-13 of "unify", of lines 1-8, 32-35 and 41 of "SelectMultiequation", and of lines 1-5 and 41 of "compact";

```
procedure compact(F : PListOfTempMulteq;var U : PListOfMulteq);
var temp : ↑TempMultiequation;vterm,vterm1,t : ↑term;
    mult,mult1 : ↑multiequation;
begin
  while not EmptyListOfTempMulteq(F) do
  begin
    temp:=HeadOfListOfTempMulteq(F);
    F:=TailOfListOfTempMulteq(F);
    vterm:=HeadOfListOfTerms(temp↑.S);
    temp↑.S:=TailOfListOfTerms(temp↑.S);
    mult:=vterm↑.v↑.m;
    vterm↑.deleted:=true;
    if vterm↑.marked then U:=AddToFrontOfListOfMulteq(mult,U);
    while not EmptyListOfTerms(temp↑.S) do
    begin
      vterm1:=HeadOfListOfTerms(temp↑.S);
      temp↑.S:=TailOfListOfTerms(temp↑.S);
      mult1:=vterm1↑.v↑.m;
      vterm1↑.deleted:=true;
      if vterm1↑.marked then U:=AddToFrontOfListOfMulteq(mult1,U);
      if not(mult = mult1)then
      begin
        mult↑.S↑.eqvar:=AddToEndOfListOfMulteq(mult1,mult↑.S↑.eqvar);
        mult1↑.S↑.eqvar:=AddToEndOfListOfMulteq(mult,mult1↑.S↑.eqvar)
      end
    end;
    while not EmptyListOfTerms(temp↑.M) do
    begin
      t:=HeadOfListOfTerms(temp↑.M);
      temp↑.M:=TailOfListOfTerms(temp↑.M);
      t↑.top:=true;
      t↑.mult:=mult;
      mult↑.M:=AddToEndOfListOfTerms(t,mult↑.M);
      if t↑.marked then
      begin
        if mult↑.S↑.marked then fail
        else U:=AddToFrontOfListOfMulteq(mult,U)
      end
    end
  end
end; (*compact*)
```

Fig. 11

---

c_n of lines 9-10, 17-18 and 25-31 of "SelectMultiequation", and of lines 1-5, 12-15 and 22-23 of "merge";

c_p of lines 10-16 of "SelectMultiequation", and of lines 1-8 and 14-18 of "DominatingMulteq";

c_q of lines 8-13 of "DominatingMulteq";

c_r of lines 18-24 of "SelectMultiequation";

c_s of lines 6-11 of "merge";

c_t of lines 15-21 of "merge";

c_u of lines 35-40 of "SelectMultiequation";
c_v of lines 5-14, 27 and 40 of "compact";
c_w of lines 14-26 of "compact";
c_x of lines 27-39 of "compact".

In addition to the quantities t_s, $t_{tfr}, \ldots$ defined in the previous section, let t_{vo} be the total number of variable occurrences in the initial system.

We can now prove the following theorem.

Theorem 6.1 - The complexity in time of "unify", with the versions of "SelectMultiequation" and "compact" given in this section, is bounded by

$$(6.1) \quad C \leq (c_a + c_b + c_c + c_d + c_e + c_f) t_s + (c_c + c_d + c_e) t_{tfr} + c_g + c_m t_{mef} + c_n t_{mei} + 2c_p t_{vo} + c_q t_s + 2c_r t_{vfr} + c_s t_v + c_t t_s + c_u (t_{mei} + 2t_{mef}) + c_v t_{mefr} + c_w t_{vfr} + c_x (t_{tfr} - t_{vfr})$$

Proof - Most of the terms in (6.1) can be derived from the procedure definitions by simple inspection or by analogy with the corresponding terms in (5.1). Among the new terms, we comment on terms $c_q t_s$, $c_s t_v$ and $c_t t_s$. The first term derives from the fact that every term can be marked at most once, since if we try to mark it twice we cause failure. The second and third term measure the cost of multiequation merging and rely on the fact that every multiequation is merged at most once with a propagation head. Finally the term $2.c_r.t_{vfr}$ measures the complexity of examining the lists of two-way links between multiequations generated in delaying merging.

□

We can now give our final results.

Theorem 6.2 - The computational complexity in time of algorithm "unify", with the versions of "SelectMultiequation" and "compact" given in this section, is linear with the total number t_s of symbols in the initial system of multiequations.

Proof - Since all quantities t_{tfr}, $t_{mei}, \ldots$, t_{vo} in (6.1) are bounded by t_s, the result follows from theorem 6.1.

□

Theorem 6.3 - The algorithm "unify" with the versions of "SelectMultiequation" and "compact" given in this section, is linear in space with the total number t_s of symbols in the initial system of multiequations.

Proof - Linearity in space descends from linearity in time using the same argument of theorem 5.4.

□

REFERENCES

1. Robinson, J.A. A machine-oriented logic based on the resolution principle. J. ACM 12, 1 (Jan. 1965), 23-41.

2. Robinson, J.A. Computational logic: the unification computation. Machine Intelligence 6, B. Meltzer and D. Michie (eds.), Edinburgh University Press, 1971, 63-72.

3. Chang, C.L., and Lee, R.C. Symbolic Logic and Mechanical Theorem Proving. Academic Press, 1973.

4. Hewitt, C. Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot. Ph. D. Thesis, Dept. of Math., MIT, Cambridge MA, 1972.

5. Boyer, R.S., and Moore, J.S. The sharing of structure in theorem--proving programs. Machine Intelligence 7, B. Meltzer and D. Michie (eds.), Edinburgh University Press, 1972, 101-116.

6. Stickel, M.E. A complete unification algorithm for associative--commutative functions. Proc. Fourth Int. Joint Conf. on Artificial Intelligence, Tbilisi, USSR, 1975, 71-76.

7. Shortliffe, E.H. Computer-Based Medical Consulation: MYCIN. Elsevier, New York, 1976.

8. Venturini Zilli, M. Complexity of the unification algorithm for first-order expressions. Calcolo XII, Fasc. IV, (Oct.-Dec. 1975), 361-372.

9. Huet, G. Private communication.

10. Robinson, J.A. Fast unification. Theorem Proving Workshop, Oberwolfach, West Germany, Jan. 1976.

11. Paterson, M.S., and Wegman, M.N. Linear unification. Proc. Eighth Annual ACM Symposium on Theory of Computing, Hershey, Penn., May 1976, 181-186.

12. Gries, D. Describing an algorithm by Hopcroft. Acta Informatica 2, 2 (1973), 97-109.

13. Waldinger, R.J., and Levitt, K.N. Reasoning about programs. Artificial Intelligence 5, (1974), 235-316.

14. V. Henke, F.W., and Luckham, D.C. Automatic program verification III: a methodology for verifying programs. Stanford Artificial Intelligence Laboratory Memo AIM-256, Stanford University, Dec. 1974.

15. Aho, A.V., Hopcroft, J.E., and Ullman, J.D. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass., 1974.

APPENDIX

In Fig. 12 and 13 we show a few straightforward data type definitions and procedures necessary to complete both programs described in sections 4, 5 and 6. We give the representation and the operation definition only for the lists of type "ListOfTerms". The lists of type "ListOfMulteq", "ListOfTempMulteq" and "ListOfVariables" are defined exactly in the same way.

To help the reader in preparing a running program, in Fig. 14 we give two procedures for reading and writing a system of multiequations, while in Fig. 15 and 16 we show two initialization procedures for the two programs of sections 5 and 6. The reading convenctions are as follows. Variables and function symbols are represented by "V" and "F" immediately followed by a two-figure integer. Terms must be written in the usual functional notation, while a multiequation is represented as a sequence of the variable symbols in the left member and of the terms in the right member, all separated by equal signs. If the right member is empty, then "=E" must follow the last variable symbol. Finally, multiequations are separated by semicolons and the whole system is enclosed in a pair of brackets, while blanks are neglected everywhere. We show below an example of acceptable input:

(VO1 = F10(F15(VO2),FO1) = F10(VO3,VO4); VO2 = E; VO3 = E; VO4 = E)

Data type and procedure definitions must be assembled into a main PASCAL program according to the following paradigm.

```
program unification(input,output);
label 1;
  {type definitions}
var s : ↑system;
  {procedure definitions}
begin
  s:=readsys;
  initialize(s);
  unify(s);
  writesys(s);
1:end
```

To run the simpler program described in section 5, the part {type definitions} must consist of Fig. 12 and 4 and the part {procedure definitions} of Fig. 13, 15, 14, 6, 5, 3 and 2 in the order. Conversely, for obtaining a running version of the linear program described in section 6 {type definitions} must be substituted with Fig. 12 and 7 and {procedure definitions} with Fig. 13, 16, 14, 11, 10, 9, 8, 3 and 2 in the order.

```
type funname = array [1..12] of char;
     varname = array [1..12] of char;
     AuxListOfTerms = record
                        head : ↑term;
                        tail : ↑AuxListOfTerms
                      end;
     ListOfTerms = record
                     first,last : ↑AuxListOfTerms
                   end;
```

Fig. 12

```
procedure AddTerm(t1 : Pterm;var argsofm,argsofm1 : PListOfTempMulteq);
var temp : ↑TempMultiequation;
begin
  if EmptyListOfTempMulteq(argsofm) then
  begin
    new(temp);
    temp↑.S:=CreateListOfTerms;
    temp↑.M:=CreateListOfTerms
  end
  else begin
    temp:=HeadOfListOfTempMulteq(argsofm);
    argsofm:=TailOfListOfTempMulteq(argsofm)
  end;
  if t1↑.isfun then
    temp↑.M:=AddToEndOfListOfTerms(t1,temp↑.M)
  else temp↑.S:=AddToEndOfListOfTerms(t1,temp↑.S);
  argsofm1:=AddToEndOfListOfTempMulteq(temp,argsofm1)
end; (*AddTerm*)
function BuildFunctionTerm(fs : funname;args : PListOfTerms): Pterm;
var t : ↑term;
begin
  new(t,true);
  t↑.isfun:=true;
  t↑.fsymb:=fs;
  t↑.args:=args;
  BuildFunctionTerm:=t
end; (*BuildFunctionTerm*)
function diffsymb(fs1,fs2 : funname): boolean;
begin
  diffsymb:=not(fs1 = fs2)
end; (*diffsymb*)
procedure fail;
begin
  writeln('no unification');
  goto 1
end; (*fail*)
```

Fig. 13a

```
function CreateListOfTerms : PListOfTerms;
var s : ↑ListOfTerms; l : ↑AuxListOfTerms;
begin
  new(s);new(l);
  s↑.first:=l;s↑.last:=l;
  l↑.head:=nil;l↑.tail:=nil;
  CreateListOfTerms:=s
end; (*CreateListOfTerms*)
function AddToEndOfListOfTerms(t : Pterm;s : PListOfTerms): PListOfTerms;
var l : ↑AuxListOfTerms;
begin
  new(l);l↑.head:=nil;l↑.tail:=nil;
  s↑.last↑.head:=t;s↑.last↑.tail:=l;
  s↑.last:=l;AddToEndOfListOfTerms:=s
end; (*AddToEndOfListOfTerms*)
function HeadOfListOfTerms(s : PListOfTerms): Pterm;
begin
  HeadOfListOfTerms:=s↑.first↑.head
end; (*HeadOfListOfTerms*)
Function TailOfListOfTerms(s : PListOfTerms): PListOfTerms;
var l : AuxListOfTerms;
begin
  l:=s↑.first;
  s↑.first:=l↑.tail;
  dispose(l);
  TailOfListOfTerms:=s
end; (*TailOfListOfTerms*)
function EmptyListOfTerms(s : PListOfTerms): boolean;
begin
  if s↑.first = s↑.last then EmptyListOfTerms:=true
  else EmptyListOfTerms:=false
end;(*EmptyListOfTerms*)
function AddToFrontOfListOfTerms(t : Pterm ;s : PListOfTerms): PListOfTerms;
var l : ↑AuxListOfTerms;
begin
  new(l);l↑.head:=t;l↑.tail:=s↑.first;
  s↑.first:=l;AddToFrontOfListOfTerms:=s
end; (*AddToFrontOfListOfTerms*)
function AppendListsOfTerms(t1,t2 : PListOfTerms):PListOfTerms;
begin
  if not(t2↑.first = t2↑.last) then
  begin
    t1↑.last↑.head:=t2↑.first↑.head;
    t1↑.last↑.tail:=t2↑.first↑.tail;
    t1↑.last:=t2↑.last
  end;
  dispose(t2↑.first);
  dispose(t2);
  AppendListsOfTerms:=t1
end; (*AppendListsOfTerms*)
```

Fig. 13b

```
FUNCTION READSYS : PSYSTEM;
TYPE TWODIGIN = 0..99;
VAR CH : CHAR;SYS : !SYSTEM;AVAR : ARRAY&TWODIGIN? OF !VARIABLE;
    MULT : !MULTIEQUATION;SETV : !SETOFVARIABLES;IND : TWODIGIN;V : !VARIABLE;
    T : !TERM;
PROCEDURE NEXTCHAR;
BEGIN
  REPEAT
    READ(CH)
  UNTIL CH <> ' '
END; (*NEXTCHAR*)
FUNCTION DECODE (NAME : VARNAME) : TWODIGIN;
BEGIN
  DECODE:=(ORD(NAME&2?)-ORD('0'))+(ORD(NAME&1?)-ORD('0'))*10
END; (*DECODE*)
FUNCTION GETVAR : PVARIABLE;
VAR INDAR : VARNAME;IND : TWODIGIN;V : !VARIABLE;
BEGIN
  READ(CH);INDAR&1?:=CH;
  READ(CH);INDAR&2?:=CH;
  IND:=DECODE(INDAR);
  IF AVAR&IND? = NIL THEN
  BEGIN
    NEW(V);
    AVAR&IND?:=V;GETVAR:=V;
    V!.NAME:=INDAR
  END
  ELSE GETVAR:=AVAR&IND?
END; (*GETVAR*)
FUNCTION RDVTERM : PTERM;
VAR T : !TERM;
BEGIN
  NEW(T,FALSE);T!.ISFUN:=FALSE;
  T!.V:=GETVAR;
  NEXTCHAR;
  RDVTERM:=T
END; (*RDVTERM*)
FUNCTION RDFTERM : PTERM;
VAR T,ARG : !TERM;RIGHTPAR : BOOLEAN;
BEGIN
  NEW(T,TRUE);T!.ISFUN:=TRUE;
  READ(CH);T!.FSYMB&1?:=CH;
  READ(CH);T!.FSYMB&2?:=CH;
  T!.ARGS:=CREATELISTOFTERMS;
  NEXTCHAR;
  CASE CH OF
    ',',')','=',';' : ;
    '(' : BEGIN
          REPEAT
            NEXTCHAR;
            CASE CH OF
              'F' : ARG:=RDFTERM;
              'V' : ARG:=RDVTERM
            END;
            T!.ARGS:=ADDTOENDOFLISTOFTERMS(ARG,T!.ARGS);
```

Fig. 14a

```
              CASE CH OF
               ',' : RIGHTPAR:=FALSE;
               ')' : RIGHTPAR:=TRUE
              END
            UNTIL RIGHTPAR;
            NEXTCHAR
          END
  END;
  RDFTERM:=T
END; (*RDFTERM*)
BEGIN
  FOR IND:=0 TO 99 DO AVAR&IND?:=NIL;
  NEW(SYS);
  SYS!.T:=CREATELISTOFMULTEQ;
  SYS!.U:=CREATELISTOFMULTEQ;
  NEXTCHAR;
  REPEAT
    NEW(MULT);MULT!.M:=CREATELISTOFTERMS;
    NEW(SETV);MULT!.S:=SETV;
    SETV!.VARS:=CREATELISTOFVARIABLES;
    SYS!.U:=ADDTOENDOFLISTOFMULTEQ(MULT,SYS!.U);
    REPEAT
      NEXTCHAR;
      CASE CH OF
        'V' : BEGIN
                V:=GETVAR;
                V!.M:=MULT;
                SETV!.VARS:=ADDTOENDOFLISTOFVARIABLES(V,SETV!.VARS);
                NEXTCHAR
              END;
        'F' : BEGIN
                T:=RDFTERM;
                MULT!.M:=ADDTOENDOFLISTOFTERMS(T,MULT!.M)
              END;
        'E' : NEXTCHAR
      END
    UNTIL CH <> '='
  UNTIL CH <> ';' ;
  READSYS:=SYS
END; (*READSYS*)
PROCEDURE WRITESYS (SYS : PSYSTEM);
CONST MAXLINE = 70;
VAR LINELEN : INTEGER;MULTLIST : !LISTOFMULTEQ;MULT : !MULTIEQUATION;
    VARLIST : !LISTOFVARIABLES;V : !VARIABLE;
PROCEDURE OUT (CH : CHAR);
BEGIN
  IF LINELEN > MAXLINE THEN
  BEGIN
    WRITELN(OUTPUT);LINELEN:=0
  END;
  WRITE(CH);LINELEN:=LINELEN+1
END; (*OUT*)
PROCEDURE WRVAR (V : PVARIABLE);
BEGIN
  OUT('V');
```

Fig: 14b

```
  OUT(V!.NAME&1?);
  OUT(V!.NAME&2?)
END; (*WRVAR*)
PROCEDURE WRFTERM (T : PTERM);
VAR ARGLIST : !LISTOFTERMS;ARG : !TERM;
BEGIN
  OUT('F');
  OUT(T!.FSYMB&1?);
  OUT(T!.FSYMB&2?);
  IF NOT EMPTYLISTOFTERMS(T!.ARGS) THEN
  BEGIN
    OUT('(');
    ARGLIST:=T!.ARGS;
    T!.ARGS:=CREATELISTOFTERMS;
    REPEAT
      ARG:=HEADOFLISTOFTERMS(ARGLIST);
      ARGLIST:=TAILOFLISTOFTERMS(ARGLIST);
      T!.ARGS:=ADDTOENDOFLISTOFTERMS(ARG,T!.ARGS);
      IF ARG!.ISFUN THEN WRFTERM(ARG)
        ELSE WRVAR(ARG!.V);
      IF NOT EMPTYLISTOFTERMS(ARGLIST) THEN OUT(',')
    UNTIL EMPTYLISTOFTERMS(ARGLIST);
    OUT(')')
  END
END; (*WRFTERM*)
BEGIN
  LINELEN:=0;
  WRITELN(OUTPUT);OUT('(');
  MULTLIST:=SYS!.T;
  SYS!.T:=CREATELISTOFMULTEQ;
  WHILE NOT EMPTYLISTOFMULTEQ(MULTLIST) DO
  BEGIN
    MULT:=HEADOFLISTOFMULTEQ(MULTLIST);
    MULTLIST:=TAILOFLISTOFMULTEQ(MULTLIST);
    SYS!.T:=ADDTOENDOFLISTOFMULTEQ(MULT,SYS!.T);
    VARLIST:=MULT!.S!.VARS;
    MULT!.S!.VARS:=CREATELISTOFVARIABLES;
    REPEAT
      V:=HEADOFLISTOFVARIABLES(VARLIST);
      VARLIST:=TAILOFLISTOFVARIABLES(VARLIST);
      MULT!.S!.VARS:=ADDTOENDOFLISTOFVARIABLES(V,MULT!.S!.VARS);
      WRVAR(V);
      OUT('=')
    UNTIL EMPTYLISTOFVARIABLES(VARLIST);
    IF EMPTYLISTOFTERMS(MULT!.M) THEN OUT('E')
      ELSE WRFTERM(HEADOFLISTOFTERMS(MULT!.M));
    IF NOT EMPTYLISTOFMULTEQ(MULTLIST) THEN
    BEGIN
      OUT(';');WRITELN(OUTPUT);LINELEN:=0
    END
  END;
  OUT(')');WRITELN(OUTPUT)
END; (*WRITESYS*)
```

Fig. 14c

```
PROCEDURE INITIALIZE (VAR SYS : PSYSTEM);
VAR MULT : !MULTIEQUATION;MULTLIST : !LISTOFMULTEQ;
    VARLIST : !LISTOFVARIABLES;TERMLIST : !LISTOFTERMS;
PROCEDURE COUNTOCC (T : PTERM);
VAR ARGLIST : !LISTOFTERMS;
BEGIN
  IF T!.ISFUN THEN
  BEGIN
    ARGLIST:=T!.ARGS;
    T!.ARGS:=CREATELISTOFTERMS;
    WHILE NOT EMPTYLISTOFTERMS(ARGLIST) DO
    BEGIN
      COUNTOCC(HEADOFLISTOFTERMS(ARGLIST));
      T!.ARGS:=ADDTOENDOFLISTOFTERMS(HEADOFLISTOFTERMS(ARGLIST),T!.ARGS);
      ARGLIST:=TAILOFLISTOFTERMS(ARGLIST)
    END
  END
  ELSE T!.V!.M!.S!.COUNTER:=T!.V!.M!.S!.COUNTER + 1
END; (*COUNTOCC*)
BEGIN
  MULTLIST:=SYS!.U;
  SYS!.U:=CREATELISTOFMULTEQ;
  REPEAT
    MULT:=HEADOFLISTOFMULTEQ(MULTLIST);
    MULTLIST:=TAILOFLISTOFMULTEQ(MULTLIST);
    MULT!.ERASED:=FALSE;
    MULT!.S!.COUNTER:=0;
    VARLIST:=MULT!.S!.VARS;
    MULT!.S!.VARS:=CREATELISTOFVARIABLES;
    MULT!.S!.VARNUMB:=0;
    REPEAT
      MULT!.S!.VARNUMB:=MULT!.S!.VARNUMB + 1;
      MULT!.S!.VARS:=ADDTOENDOFLISTOFVARIABLES(HEADOFLISTOFVARIABLES(VARLIST),
                                               MULT!.S!.VARS);
      VARLIST:=TAILOFLISTOFVARIABLES(VARLIST)
    UNTIL EMPTYLISTOFVARIABLES(VARLIST);
    SYS!.U:=ADDTOENDOFLISTOFMULTEQ(MULT,SYS!.U)
  UNTIL EMPTYLISTOFMULTEQ(MULTLIST);
  MULTLIST:=SYS!.U;
  SYS!.U:=CREATELISTOFMULTEQ;
  REPEAT
    MULT:=HEADOFLISTOFMULTEQ(MULTLIST);
    MULTLIST:=TAILOFLISTOFMULTEQ(MULTLIST);
    TERMLIST:=MULT!.M;
    MULT!.M:=CREATELISTOFTERMS;
    WHILE NOT EMPTYLISTOFTERMS(TERMLIST) DO
    BEGIN
      COUNTOCC(HEADOFLISTOFTERMS(TERMLIST));
      MULT!.M:=ADDTOENDOFLISTOFTERMS(HEADOFLISTOFTERMS(TERMLIST),MULT!.M);
      TERMLIST:=TAILOFLISTOFTERMS(TERMLIST)
    END;
    SYS!.U:=ADDTOENDOFLISTOFMULTEQ(MULT,SYS!.U)
  UNTIL EMPTYLISTOFMULTEQ(MULTLIST);
  MULTLIST:=SYS!.U;
  SYS!.U:=CREATELISTOFMULTEQ;
```

Fig. 15a

```
  REPEAT
    MULT:=HEADOFLISTOFMULTEQ(MULTLIST);
    MULTLIST:=TAILOFLISTOFMULTEQ(MULTLIST);
    IF MULT!.S!.COUNTER = 0 THEN
      SYS!.U:=ADDTOFRONTOFLISTOFMULTEQ(MULT,SYS!.U)
    ELSE SYS!.U:=ADDTOENDOFLISTOFMULTEQ(MULT,SYS!.U)
  UNTIL EMPTYLISTOFMULTEQ(MULTLIST)
END; (*INITIALIZE*)
```

Fig. 15b

```
PROCEDURE INITIALIZE (VAR SYS : PSYSTEM);
VAR MULT : !MULTIEQUATION;MULTLIST : !LISTOFMULTEQ;T : !TERM;
    TLIST : !LISTOFTERMS;
PROCEDURE INITTERM (T : PTERM);
VAR ARG : !TERM;ARGLIST : !LISTOFTERMS;
BEGIN
  T!.MARKED:=FALSE;
  IF T!.ISFUN THEN
  BEGIN
    ARGLIST:=T!.ARGS;
    T!.ARGS:=CREATELISTOFTERMS;
    WHILE NOT EMPTYLISTOFTERMS(ARGLIST) DO
    BEGIN
      ARG:=HEADOFLISTOFTERMS(ARGLIST);
      ARGLIST:=TAILOFLISTOFTERMS(ARGLIST);
      INITTERM(ARG);
      IF ARG!.ISFUN THEN
      BEGIN
        ARG!.TOP:=FALSE;
        ARG!.FFATHER:=T
      END
      ELSE ARG!.VFATHER:=T;
      T!.ARGS:=ADDTOENDOFLISTOFTERMS(ARG,T!.ARGS)
    END
  END
  ELSE
  BEGIN
    T!.DELETED:=FALSE;
    T!.V!.M!.S!.VAROCC:=ADDTOENDOFLISTOFTERMS(T,T!.V!.M!.S!.VAROCC)
  END
END; (*INITTERM*)
```

Fig. 16a

```
BEGIN
  MULTLIST:=SYS!.U;
  SYS!.U:=CREATELISTOFMULTEQ;
  REPEAT
    MULT:=HEADOFLISTOFMULTEQ(MULTLIST);
    MULTLIST:=TAILOFLISTOFMULTEQ(MULTLIST);
    MULT!.ERASED:=FALSE;
    MULT!.S!.VAROCC:=CREATELISTOFTERMS;
    MULT!.S!.EQVAR:=CREATELISTOFMULTEQ;
    MULT!.S!.MARKED:=FALSE;
    MULT!.S!.MERGEDMULT:=NIL;
    SYS!.U:=ADDTOENDOFLISTOFMULTEQ(MULT,SYS!.U)
  UNTIL EMPTYLISTOFMULTEQ(MULTLIST);
  MULTLIST:=SYS!.U;
  SYS!.U:=CREATELISTOFMULTEQ;
  REPEAT
    MULT:=HEADOFLISTOFMULTEQ(MULTLIST);
    MULTLIST:=TAILOFLISTOFMULTEQ(MULTLIST);
    TLIST:=MULT!.M;
    MULT!.M:=CREATELISTOFTERMS;
    WHILE NOT EMPTYLISTOFTERMS(TLIST) DO
    BEGIN
      T:=HEADOFLISTOFTERMS(TLIST);
      TLIST:=TAILOFLISTOFTERMS(TLIST);
      INITTERM(T);
      T!.TOP:=TRUE;
      T!.MULT:=MULT;
      MULT!.M:=ADDTOENDOFLISTOFTERMS(T,MULT!.M)
    END;
    SYS!.U:=ADDTOENDOFLISTOFMULTEQ(MULT,SYS!.U)
  UNTIL EMPTYLISTOFMULTEQ(MULTLIST)
END; (*INITIALIZE*)
```

Fig. 16b