INFORMATIQUE THÉORIQUE ET APPLICATIONS

HABIB ABDULRAB

Solving word equations

Informatique théorique et applications, tome 24, n° 2 (1990), p. 109-130

<http://www.numdam.org/item?id=ITA_1990__24_2_109_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 24, n° 2, 1990, p. 109 à 130)

SOLVING WORD EQUATIONS (*)

by Habib ABDULRAB (1)

Abstract. – *Makanin's algorithm decides whether a word equation in words has a solution or not. The aim of this paper is to give an introduction to the algorithm and a general description of our work. This provides some simplifications and improvements to the algorithm thus allowing to achieve an effective implementation solving non trivial equations.*

Résumé. – *L'algorithme de Makanin permet de décider si une équation en mots admet une solution ou non. Cet article donne une introduction à cet algorithme ainsi qu'une description générale de nos travaux dans ce domaine. Ceux-ci introduisent quelques simplifications et améliorations à cet algorithme, et proposent une implémentation effective permettant de résoudre des équations non triviales.*

1. INTRODUCTION

The study of the properties and structure of the set of solutions of word equations was first initiated by Lentin and Schützenberger ([9], [10]) in the case of constant-free equations. Such equations always admit a solution. The study of word equations with constants has been tackled by Markov who gave an algorithm to decide whether a word equation in two variables has a solution or not. Hmelevskii [7] solved equations with constants in three variables. Makanin [11] showed that solving arbitrary equations is decidable. He gave an algorithm to decide whether a word equation with constants has a solution or not. His labour-consuming algorithm in one of the major results (and certainly one of the most difficult) in theoretical computer science. Pécuchet [12] gave a study unifying the two theories of equations with or without constants, and a new description of Makanin's algorithm. We have provided [1] some simplifications and improvements to this algorithm permitting an effective implementation.

(*) Received November 1987, final version in December 1988.
(1) Laboratoire d'Informatique de Rouen et L.I.T.P. Faculté des Sciences, B.P. n° 118, 76134 Mont-Saint-Aignan Cedex, France.

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754 90/02 109 22/$4.20/

Solving word equations arises in many areas of theoretical computer science, but especially in the area of combinatorics on words ([10], [13], [14]), and of unification in formal systems ([3], [4]).

Let X be a finite set (alphabet); we denote by X^* the free monoid on X. *i.e.* the set of all finite sequences (words) over X. The empty word is denoted by the symbol 1. The length of a word w (*i.e.* the number of letters composing it) is denoted by $|w|$. The number of occurrences of a letter x in $w (x \in X)$, is denoted by $|w|_x$. The alphabet of w is denoted by $\text{Alph}(w) = \{ x \in X / |w|_s > 0 \}$.

A morphism of a monoid M_1 into a monoid M_2 is a mapping α of M_1 into M_2 such that:

$$\alpha(m_1 m_2) = \alpha(m_1)\alpha(m_2), \quad \text{and} \quad \alpha(1) = 1$$
$$m_1, m_2 \in M_1.$$

Given two distinct alphabets **V** and **C**, a *word equation* e is an ordered pair (e_1, e_2); $e_1 = e_1(1) \ldots e_1(|e_1|)$, $e_2 = e_2(1) \ldots e_2(|e_1|)$; of elements of $\mathbf{L}^* = \{ \mathbf{V} \cup \mathbf{C} \}^*$.

The alphabet **V** is called the alphabet of variables (denoted by $x, y, z \ldots$). **C** is called the alphabet of constants (denoted by $A, B, C \ldots$). We will suppose that $\text{Alph}(e_1 e_2) = L$.

A *solution* of the equation e is a morphism $\alpha : \mathbf{L}^* \to \mathbf{L}^*$ such that $\alpha(e_1) = \alpha(e_2)$, and $\alpha(e) = c$ for each $c \in \mathbf{C}$.

Example: The equation $e = (Ay B, xxz)$ has the solution α:

$$\alpha(x) = AB, \quad \alpha(y) = BA, \quad \alpha(z) = 1, \quad \alpha(A) = A, \quad \alpha(B) = B.$$

Example: The equation $e = (Axxy, yyB)$ has no solution.

A constant-free equation ($C = \varnothing$) always has the trivial solution α given by: $\forall v \in \mathbf{V}$, $\alpha(v) = 1$. We will not consider (from now on) this simple case, and we will suppose that $\text{Card}(\mathbf{C}) > 0$.

A solution α is said to be *continuous* if $\alpha(v) \neq 1$ holds for all $v \in \mathbf{V}$.

The *projection* of the equation e into a subset Q of **V** is the equation defined by erasing all the occurrences of letters of $V \setminus Q$ in e. Consequently, an equation has $2^{\text{Card}(\mathbf{V})}$ projections. It is obvious that e admits a solution iff one of its projections admits a continuous solution.

The projections of the above-mentioned equation $e = (AyB, xxz)$ are given by:

$$(AB, 1), \quad (AyB, 1), \quad (AB, xx), \quad (AB, z),$$

$$(AyB, z), \quad (AB, xxz), \quad (AyB, xx), \quad (AyB, xxz).$$

An equation e is called *simple* if $\text{Card}(\mathbf{C}) = 1$. Consider a simple equation e with $\mathbf{V} = \{v_1 \ldots v_n\}$, $\mathbf{C} = \{c_1\}$. Let e' be the commutative image of e, *i.e.*

$$e' = (v_1^{p_1} \ldots v_n^{p_n} c_1^{p_{n+1}}, v_1^{q_1} \ldots v_n^{q_n} c_1^{q_{n+1}}).$$

Consider the linear diophantine equation e'', called the *length equation* associated with e:

$$e'': (p_1 - q_1) v_1' + \ldots + (p_n - q_n) v_n' = (q_{n+1} - p_{n+1}).$$

The isomorphism between $\{c_1\}^*$ and $(N, +)$ permits to show that e admits a solution iff e'' admits a non-negative integer solution. Finding a non-negative integer solution to linear diophantine equations is solved by several algorithms [8]. Thus, solving simple equations can be easily reduced to solving linear diophantine equations.

Consider now an equation f with $\text{Card}(\mathbf{C}) \geqq 2$, and let f' be the simple equation associated with f by replacing all the constants of f by c_1. The length equation f'' associated with f is, by definition, that associated with f'. It is easy to see that if f admits a solution then f'' admits a non-negative integer solution. The converse of this implication is false as shown in the following counter-example: the length equation associated with the equation $e = (Axxy, yyB)$ of the previous example has the non-negative integer solution: $x' = 1$, $y' = 2$.

2. MAKANIN'S ALGORITHM

2.1. Informal presentation of two examples

We describe here the basic notions of Makanin's algorithm and its general behavior via two examples.

Consider the previously seen equation $e = (AyB, xxz)$. This equation is not simple, and its length equation has a non-negative integer solution. So, it cannot be solved by the results given in the previous section.

The first step of the algorithm consists in the computation of all the projections of e in order to find a continuous solution to one of these projections.

The second step consists in associating for each projection $p = (p_1, p_2)$ of e, all the possible ways of choosing the position of the symbols of p_1 according

to those of p_2. The following diagram illustrates one possibility for the projection (AyB, xx) (see *fig.* 1).

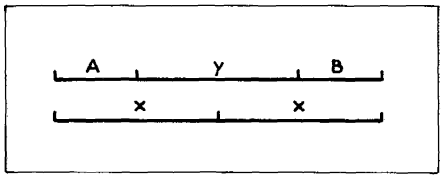

Figure 1

Now, this *scheme applicable* to p will be transformed into a so called *position equation.* This new object inherit the seven boundaries of the scheme and of all occurrences of constants (these occurrences are called *constant bases*), but variables will be treated in a special manner.

Single occurrence variables, such as y, will disappear.

The n occurrences $(n>1)$ of other variables are replaced by $2n-2$ new variables associated via a symmetrical binary relation (called duality relation). These new variables are called *variable bases.*

The position equation E_0 computed from the previous scheme applicable to e is: (see *fig.* 2).

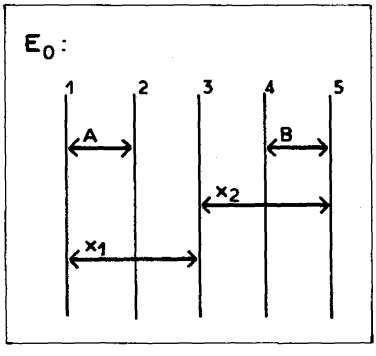

Figure 2

Here, x_1 is called the *dual* of x_2 and conversely.

Note that, after the second step of the algorithm (*i.e.* the computation of all the schemes applicable to all the projections of e), the algorithm develops a tree level by level. The tree is denoted by $\mathscr{A}$ and its levels by $L_i (i \geqq 0)$. This tree is called the tree of *admissible and normalized position equations.*

The first level of $\mathscr{A}$ contains the position equations computed from the schemes applicable to the projections. In our example, L_0 contains the previous position equation E_0. It is important to observe that it is the only

step where new variable bases will be generated. Their number will thereafter remain bounded.

The step from level $L_i (i >= 0)$ to level L_{i+1} is based on the *transformation* and *normalization* of position equations. There are five distinct types of position equations. According to its type, each position equation E existing in $\mathscr{A}$ is transformed into a set $T(E)$ of position equations.

We will describe here how to transform the position equation E_0 of L_0 in order to generate L_1.

Note first that the largest leftmost variable of E_0 (*i.e.* x_1) is called the *carrier*. The first occurrence of A with a left boundary equal to 1 is called *leading base*. Having a carrier and another leading base characterizes one of the five types of position equations. This type is called Type 5. The transformation of a position equation of this type consists in transferring the leading base A, in all the possible ways, under the dual of the carrier. There are two distinct ways to do the transfer. Either A takes all the space between the boundaries 3 and 4, or a part of this space. So $T(E_0)$ has the following two position equations (denoted respectively by E_1, E_2): (*see* fig. 3).

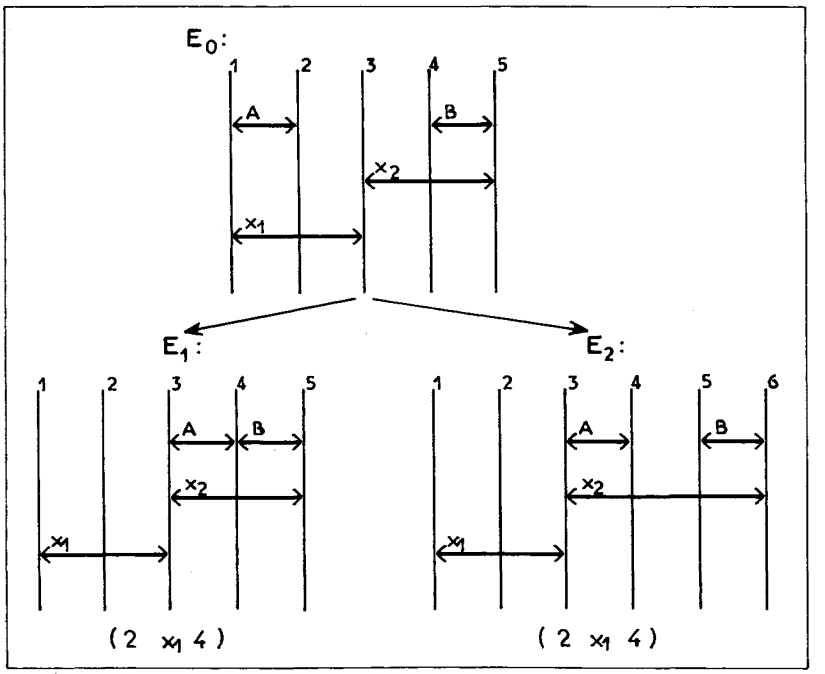

Figure 3

The list (2 x1 4) of the last two position equations is called *connection*. Such an object is created in order to avoid any loss of information during this move. It plays the role of a link between old and new positions of A. This connection (2 x1 4) indicates that the prefix of x1 ending at boundary 2 is equal to the prefix of its dual (*i.e.* x2) ending at boundary 4.

Here, we transform only the first position equation E_1. The transformation of the second one is realized in the same way.

E_1 has a carrier with left boundary greater than 2, it has no leading base, and all the boundaries between its left and right boundaries are *non-essential.* (*i.e.* it is neither a left or right boundary of a base, nor a last boundary of a connection). This situation characterizes another type (called Type 3) of position equations. The transformation of such a type consists in the transfer of all the boundaries, existing between the left and right boundaries of the carrier, into the dual of the carrier. In our example, we transfer the boundary 2, in all the possible ways, between the boundaries 3 and 5. This move can be realized in three ways:

1. The boundary 2 will be located between the boundaries 3 and 4.
2. The boundary 2 will be located between the boundaries 4 and 5.
3. The boundary 2 will be identified with the boundary 4.

Note here that the first two possibilities contradict the information, given by the connection-*i.e.*: the segment between the boundaries 1 and 2 is equal to the segment between the boundaries 3 and 4. These two possibilities are not *admissible*, and must be eliminated.

The transformation of E_1 gives rise to the following position equation E_3: (see *fig.* 4).

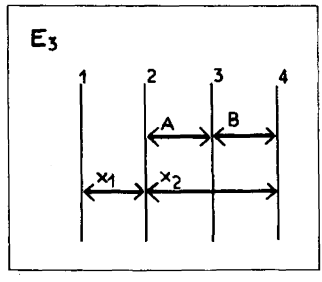

Figure 4

Note that the connection is deleted.

The type of this position equation features each position equation having a carrier with a right boundary equal to 2, and no other leading base. Such a type will be called Type 2.

The transformation of such a position equation consists in deleting the carrier and its dual, leading to the following position equation: (see *fig.* 5).

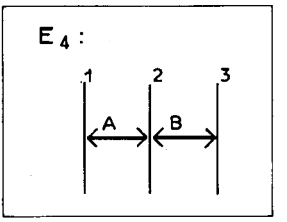

Figure 5

This last position equation has no carrier. The transformation of each position equation of this type (called Type 1) consists in deleting the first boundary, and the leading base (if one exists). So we obtain: (see *fig.* 6).

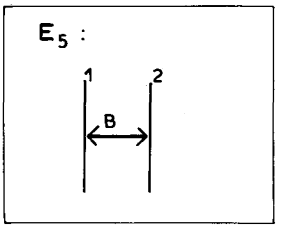

Figure 6

This last position equation is *simple* (that is, it has only one constant letter), and so the initial equation e has a solution.

The equality of two position equations is based on the following equivalence relation among position equations: two position equations E_1 and E_2 are called *equivalent* when they differ only by renaming of variables or constants. Note that the correspondence of the names of the bases of E_1 and E_2 must conserve the definition of the relation between two dual variables. More precisely, if x_1 and x_2 are two dual bases of E_1, and x'_1 and x'_2 are the bases of E_2 corresponding to x_1 and x_2, then x'_1 and x'_2 must be dual in E_2.

Note that, the algorithm develops a tree level by level until we obtain an empty level or a level *equivalent* to a previous level, in which case the initial

equation e has no solution, or a level containing a simple poisition equation, in which case e has a solution.

We give now another example solving an equation which has no solution.

Consider the equation $e=(Ax, xB)$. The following tree describes the resolution of e (see *fig.* 7).

The position equation E_0 is of Type 5, its transformation gives rise to E_1 and E_2 as seen in the previous example.

E_1 has the type of all position equations having a carrier, no other leading base and an essential boundary between the left and right boundaries of the carrier. This is called Type 4. Its transformation consists in transferring the essential boundary 2, in all the possible ways, between the left and right boundaries of its dual. According to the connection of E_1 there is one way to do the transfer. It consits in identifying the boundaries 2 and 3. Next, the prefix of the carrier ending at the boundary 2, and the identical prefix of its dual, are deleted. The transformation of E_2 is realized in the same way of E_1 leading to the previously developed position equation E_0. This resulting position equation is deleted. The transformation of E_3 (Type 5) consists in transferring A in the same segment of B, leading to a position equation directly eliminated. The fourth level of the tree is empty, and so e has no solution.

Let us now describe two basic notions appearing in Makanin's algorithm: The notion of *schemes applicable* to an equation and that of *position equation.*

2.2. Scheme applicable to an equation

Essentially, we introduce [1] the formal notion of a scheme applicable to an equation e to formalize the concept, used by Makanin's algorithm, of "mixing" the positions of the symbols of an equation, in all the possible ways.

Obviously, there are many possible ways of choosing the positions of the symbols of e_1 according to those of the symbols of e_2.

Example: the following diagrams illustrate some possibilities for the equation $e=(xAz, AzB)$: (see *fig.* 8).

Informally, a scheme applicable to an equation $e=(e_1, e_2)$ indicates how to locate the positions of the symbols of e_1 according to those of e_2 in a possible solution of e.

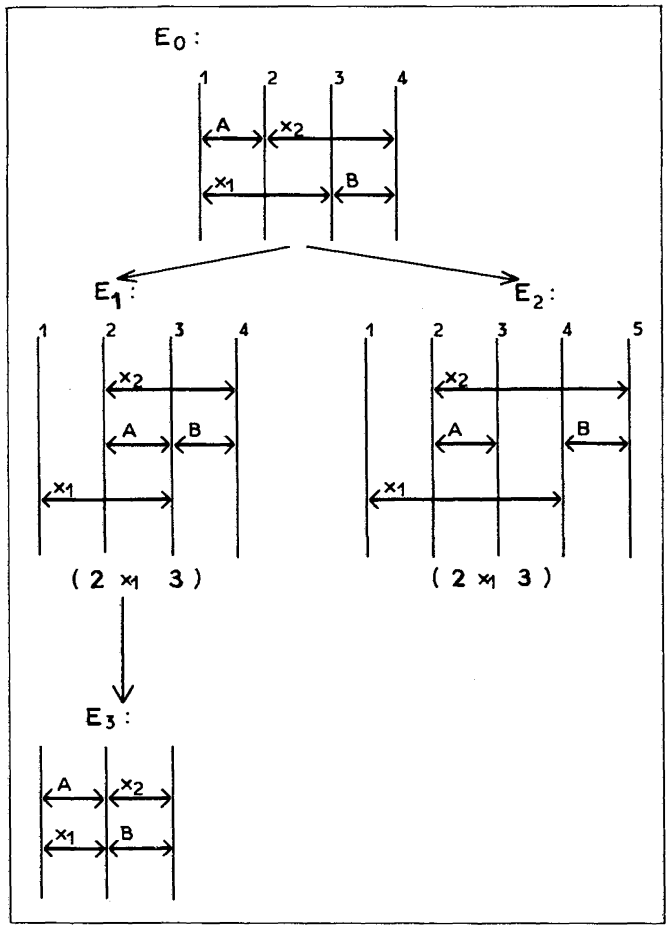

Figure 7

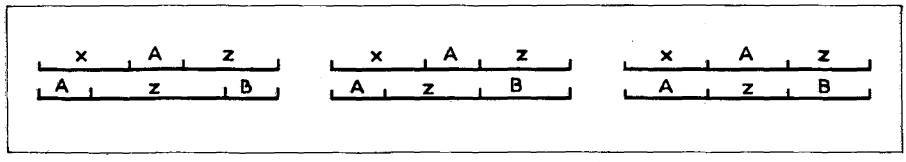

Figure 8

We denote by the symbol = each common boundary of both sides of e, by $<$ each lone boundary of e_1 and by $>$ each lone boundary of e_2. The second scheme of the previous example can be represented by $(= > < = =)$.

Formally, a *scheme* is any word $s \in \{=, <, >\}^* =$, that is a word over the alphabet $\{=, <, >\}$ beginning and ending with the letter $=$.

A scheme s is called *applicable* to an equation $e=(e_1, e_2)$, $|e_1| \neq 0$, and $|e_1| \neq 0$, if the following conditions are satisfied [1]:

1. $S_< + S_= = |e_1| + 1$.
2. $S_> + S_= = |e_2| + 1$.

where S_φ, $\varphi \in \{=, <, >\}$ is the number of occurrences of φ in s.

The left and right boundaries of a symbol t [denoted by $lb(t)$ and $rb(t)$] in a scheme s applicable to e are the integers of the interval $[1 | s |]$, defined in the following way:

If $t = e_1(n)$ then $lb(t)$ is the length of the prefix of s whose length is equal to n over the alphabet $\{=, <\}$, and $rb(t)$ is the length of the prefix of s whose length is equal to $n+1$ over $\{=, >\}$. The definition in the case $t = e_2(n)$ is obtained from the previous one by exchanging $<$ and $>$.

Note that the set T_e of all the schemes applicable to an equation e is recognized by the following automaton [1]: (see *fig.* 9).

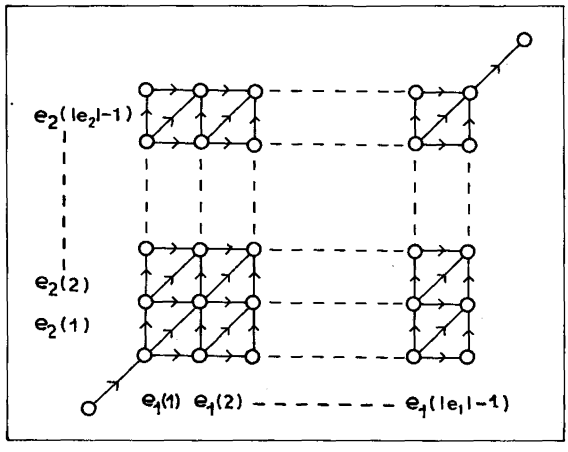

Figure 9

Each horizontal edge is labeled with $<$, each vertical edge is labeled with $>$, and each diagonal edge is labeled with $=$.

We prove [1] that the size of T_e grows exponentially with the length of e. More precisely:

$$\text{Card}(T_e) = \sum_{i=0}^{n_1} C_{n_1}^{i} C_{n_1+n_2-i}^{n_1}$$

where $n_1 = |e_1| - 1$, and $n_2 = |e_2| - 1$.

In addition, T_e contains generally a very important number of schemes that can be eliminated because they imply some contradictions on the lengths or the values of the letters of e.

Example:

– The length of z in the second scheme of the previous example is both greater than and equal to the length of a constant symbol.

– The value of z in the third scheme of the previous example is equal both to A and B.

These observations lead to a definition of the concept of a *solution* of a scheme applicable to an equation.

Definition: A *solution* of a scheme $s \in T_e$ is given by a multi-word S:

$$[2.2.1]: S = ((V_1, \ldots, V_{\text{Card}(\mathbf{V})}), (L(1), \ldots, L(|s|)), (R(1), \ldots, R(|s|)), T)$$

element of $\mathbf{C}^{+\text{Card}(\mathbf{V})} \times \mathbf{C}^{*|s|} \times \mathbf{C}^{*|s|} \times \mathbf{C}^{+}$, such that:

(*a*) $L(1) = 1, L(|s|) = T$.

(*b*) $|L(1)| < \ldots < |L(|s|)|$.

(*c*) $L(i)\,R(i) = T$, $(i = 1, \ldots, |s|)$.

(*d*) for each symbol y of the equation e we have:

$$L(lb(y))\,y\,R(rb(y)) = T \text{ if } y \text{ belongs to } \mathbf{C}$$

and

$$L(lb(y))\,V_i\,R(rb((y))) = T \text{ if } y = v_i.$$

Example: A solution of the following scheme applicable to $e = (AyB, xx)$ (see *fig.* 1).
is given by:

$$L(1) = 1, \quad L(2) = A, \quad L(3) = AB, \quad L(4) = ABA, \quad L(5) = ABAB.$$

$$R(1) = ABAB, \quad R(2) = BAB, \quad R(3) = AB, \quad R(4) = B, R(5) = 1.$$

$$T = ABAB.$$

$V_1 = AB, \quad V_2 = BA.$ (Remark: $\mathbf{V} = \{v_1, v_2\} = \{x, y\}$).

PROPOSITION (2.2.1): *An equation e has a continuous solution iff one of its applicable schemes has a solution.*

Proof:

– If a scheme s applicable to e has a solution then:

$$\alpha(t) \equiv \text{ if } t \text{ is a variable } v_i \text{ then } V_i \text{ else } t.$$

is a continuous solution of e.

– Consider the set A:

$$A = \{\alpha(e_i(1) \ldots e_i(j))/i = 1,2 \text{ and } 1 \leqq j \leqq |e_i|\}.$$

where α is a continuous solution of e. Let $K(1) \ldots K(\text{Card}(A))$ be the ordered sequence of the elements of A, such that $|K(i)| < |K(i+1)|$, $(1 \leqq i \leqq \text{Card}(A) - 1)$, and consider the following scheme s:

1. $s(1)$ is equal to $=$.

2. For each $i (1 \leqq i \leqq \text{Card}(A) + 1)$, $s(i)$ is equal to:

$=$, if $K(i-1) = \alpha(P_1) = \alpha(P_2)$, with P_1 and P_2 are two prefixes of e_1 and e_2.

$<$, if $K(i-1) = \alpha(P_1)$.

$>$, if $K(i-1) = \alpha(P_2)$.

It is not hard to see that s is a scheme applicable to e, and s has a solution. ■

The concept of a solution of a scheme applicable to an equation e enables us to state some necessary conditions [1] satisfied by each scheme $\in T_e$ which has a solution, and so, the application of the algorithm can be reduced to the only subset S_e of applicable schemes which satisfy these conditions.

Example: Consider the equation $e = (xyxyA, yxyByy)$. T_e contains 3,653 schemes applicable to e, whereas S_e contains only 2 schemes applicable to e.

We show [1] how S_e can be constructed effectively. Of course, we do not proceed by computing at first T_e and then removing all the schemes that do not satisfy the necessary conditions, but we compute S_e directly.

Note that the computation of S_e arises in several steps of the algorithm: the position equations of the first level of A are directly computed from S_e. The transformation of every position equation uses this computation to realize all the possibilities of the transfer. Finally, the computation of S_e

arises in our algorithm which computes effectively a solution to the equation e whenever one exists.

2.3. Position Equation

A position equation E can be represented by a diagram as the one shown below:

Example (see *fig.* 10).

The occurrences of the letters of E are called the *bases* of E. The bases y_1, $y_2, \ldots, y_4, z_5, z_6$ are called the *variable* bases. Each variable x_i is called the *dual* of x_{i+1} $(i=1,3\ldots)$ and conversely.

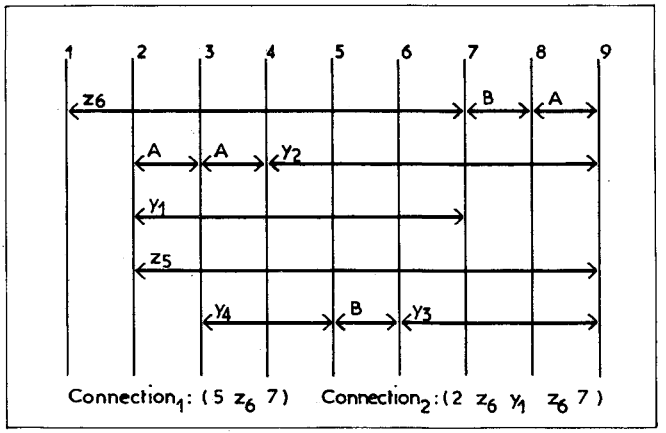

Figure 10

Each base of E has a *left boundary* and a *right boundary*. The left boundary of z_6 is, for example, equal to 1 and its right boundary is equal to 7.

The lists $connection_1$, $connection_2$ are called the *connections* of E.

Two dual variable bases having the same left and right boundary are called *matched.*

More generally, a position equation is defined by its variable and constant bases, the two mappings left and right boundary, the duality relation, and a finite (possibly emtpy) set of connections. Each connection is a list of the form

$$(p, y(1), \ldots, y(k), q).$$

where p and q are two boundaries and $y(1) \ldots y(k)$ $(k \geqq 1)$ are variable bases. The formal definition can be found in ([1], [12]).

This concept [12] is a constraint version of the notion of generalized equation introduced by Makanin [11]. The unique difference between this notion and that of generalized equation of Makanin is that the boundaries are *totally* ordered and that the right boundary of a constant is the successor of its left boundary. This concept provides a "geometrical" interpretation to the concept of generalized equation used by Makanin, and permits to the processes of transformation and normalization to be described graphically.

The transformation of position equations in the previous example differs from that of the original algorithm [11] by the reduction, from 7 to 5, of the number of types of position equations [12].

Let us now describe the notions of *admissible* and *normalized* position equations.

A system of linear diophantine equations (called the system of *length equations*) $\mathbf{AX}=\mathbf{B}$; $\mathbf{A}$ and $\mathbf{B}$ with integer entries; is associated with each position equation. A position equation E is called *admissible* when this system has a non-negative integer solution. Fundamentally, this system has a non-negative integer solution whenever the lengths of the segments and the bases of E are consistent.

A position equation is called *normalized* when the three following conditions over its variables and connections are satisfied. Such conditions are introduced both to reduce the number of equation types (for the sake of the proof) and to ensure termination of the algorithm by generating only a finite number of position equations.

A position equation E is called *normalized* when it satisfies the three following conditions:

N1. [11] No connection of E contains a segment of the form $(x_i, dual(x_i))$.

N2. [11] If the connection $(p, x_1, \ldots, x_k, q)$ contains the two variable bases x_i, x_j with $(x_i = x_j$ and $i<j)$ then there exists an integer l $(i \leqq l<j)$ satisfying: *left boundary* $(x_{l+1}) <$ *left boundary* $(dual(x_l))$.

N3. [12] The position equation E has no matched variables.

2.4. Makanin's algorithm

Remember that if the length equation associated with e has no non-negative integer solution then e has no solution. Otherwise, if the equation e is simple then e has a solution. Otherwise, the algorithm develops a tree level by level.

The first level L_0 contains the position equations computed from the schemes applicable to the projections of e.

The step from level L_i to level L_{i+1} in Makanin's algorithm is based on the two fundamental operations of *transformation* and *normalization.*

Each position equation E of $L_i (i \geqq 0)$ is transformed into a set denoted by $T(E)$ of position equations from which the non admissible ones are deleted.

If any position equation E' of this set is not normalized, it is replaced by a set $N(E')$ of normalized position equations computed from E. The result of these transformation and normalization process to every position equation of L_i leads to a set $N(T(L_i))$ of admissible and normalized position equations.

The next level L_{i+1} will be deduced from $N(T(L_i))$ by deleting a certain number of position equations:

We first identify position equations which differ only by a renaming of bases or boundaries and delete every position equation already occurring in a previous level.

We then also delete all position equations whose maximum length of connections is greater than a number $K(d)$ depending only on the length $d = |e|$ of the original equation e.

The development of levels is repeated until we obtain an empty level, in which case the equation e has no solution, or a level containing a position equation with only one constant (E' is then called *simple*), in which case e has a solution.

Makanin's algorithm:

Input: an equation e of $\mathbf{L}^*$

Output: YES if e admits a solution,

NO otherwise.

1. If the length equation associated with e has no non-negative integer solution then END: NO.

2. If the equation is simple then END: YES.

3. $i \leftarrow 0$.

4.

$$L_i \leftarrow \bigcup_{j=1}^{2^{\mathrm{Card}(\mathbf{V})}} e(Pj)$$

where P_j is the j-th projection of e, and $e(Pj)$ is the set of all the admissible and normalized position equations, computed from the schemes applicable to Pj.

5. Loop:

5. *a*. If $L_i = ()$ then END: NO.

5. *b*. If L_i contains a simple equation then END: YES.

5. *c*. $i \leftarrow i+1$.

5. *d*. $L_i \leftarrow$ the set of all admissible and normalized position equations resulting from the transformation and the normalization of the elements of L_{i-1}, the elimination of all the already developed position equations and of the position equations containing any connection whose length is greater than $K(|e|)$.

Remark: This version of the algorithm differs from the original one of [11] by the elimination of the already developed position equations introduced in [1].

We state in this section one of the main results used in the proof of Makanin's algorithm.

The notion of exponent of periodicity of a solution plays an essential role in the proof. This notion is given both for a word equation e, and for a position equation E.

The *exponent of periodicity* of a solution α of an equation e (respectively of a position equation E), is defined as the greatest integer s satisfying the following property:

There exists a word $p\,(p \neq 1)$ and there exists a variable v of **V** (respectively a variable base x) such that:

$$v = up^s w \text{ (respectively } x = up^s w) \text{ (} u \text{ and } w \text{ are two words of } \mathbf{C}^*\text{)}.$$

The following theorem is fundamental:

THEOREM (6.1) [11]: *The exponent of periodicity s_0 of a minimal continuous solution of an equation e (with $|e| = d$) satisfies:*

$$s_0 \leqq (6d)^{2^{2^{4}}} + 2.$$

The proof of this result is given in [11], p. 132-134. It consists in isolating the primitive factors of each word of a solution, using the fact that primitive words cannot overlap with their square, and showing that products P^s of

such primitive words with great s can be simplified to obtain a smaller solution. The value of s_0 depends on the bound h of the heigh of minimal solutions of linear diophantine equations, given in [11]. Other bounds [5], improving the value of h exists since Makanin's paper. These new bounds permit to reduce the value of h.

3. COMPUTATION OF A SOLUTION

The purpose of the algorithm being to decide whether an equation has a solution or not, we provide [1] an algorithm which, by taking advantage of the tree $\mathscr{A}$, computes effectively a solution to the initial equation. The idea is to compute a solution of the scheme which generates the root of the subtree containing a simple position equation.

In fact, we prove [1] that if a subtree of $\mathscr{A}$ contains a simple position equation E_0, then the scheme s_0, which generates the root of this subtree, has a solution.

We show here how the computation of a solution of e is obtained from s_0 and give an example illustrating this computation.

We call first the *pair* (P_1, P_2) *of prefixes* of e, associated with a boundary b, the two projections of b on the axes of e_1 and e_2, in the automaton given in (2.2). Informally, P_i is the greatest prefix of e_i, lying to the left of the boundary b.

The computation is described by a tree U. Each node of U is labeled by (f, s, α), where f is an equation, s is a scheme applicable to f, and α is a morphism $\alpha : \mathbf{L}^* \to \mathbf{L}^*$, connected with an equation already visited. We define α by a set of all pairs $(l, \alpha(l))$ such that $\alpha(l) \neq l$, $l \in L$. Initially, the root of U is labeled by $(e, s_0, (\ ))$, where $(\ )$ is the identical mapping.

Suppose that we visit the node $n = (f, s, \alpha)$ with $f = (f_1, f_2)$ and α is connected with the pair (g_1, g_2) already visited.

Let (P_1, P_2) be the smallest pair of prefixes of (f_1, f_2), associated with a boundary b of s, such that P_1 and P_2 are two non empty words. Let $(S_1, S_2) = (P_1^{-1} f_1, P_2^{-1} f_2)$ be the pair of suffixes associated with the boundary b, and let s' be the suffix of s beginning in the boundary b. Note that $b \in \{ <, =, > \}$.

Consider the following relation:

$$\alpha(P_1)\, b\, \alpha(P_2). \qquad (*)$$

If neither $\alpha(P_1)$ nor $\alpha(P_2)$ is a word with one letter, or if $|\alpha(P_1)|=1$, $|\alpha(P_2)|>1$ and b is equal to $<$, then the successors of n in U are labeled by

$$((\alpha(P_1)S_1, \alpha(P_2)S_2), s''s', \alpha).$$

with $s'' \in S_{\alpha(P_1)b\alpha(P_2)}$. This set is the sufficient set of applicable schemes described in (2.2).

Otherwise, suppose that $|\alpha(P_1)|=1$, and b is not equal to $<$, then (*) has the following trivial solution α':

– if b is equal to $=$, then α' is given by: $\alpha' \equiv \alpha'(\alpha(P_1))=\alpha(P_2)$, and the successor of n is labeled by

$$((S_1, S_2), s', \alpha' o\alpha).$$

– Otherwise, suppose that b is equal to $>$, then α' is given by: $\alpha' \equiv \alpha'(\alpha(P_1))=\alpha(P_2)?$, with ? is a new variable. The successor of n is labeled by:

$$((?S_1, S_2), =s', \alpha' o\alpha).$$

Here is an example illustrating the construction of U: (see *fig.* 11).

We give here the values of α_i and $\alpha_i(P_1)b\alpha_i(P_2)$, associated with the nodes of the levels $\mathbf{L}(0)$, $\mathbf{L}(1)\ldots$ of U.

$\mathbf{L}(0)$: ( ), $Ax=z$.

$\mathbf{L}(1)$: $\{(z, Ax)\}$, $y>Ax$.

$\mathbf{L}(2)$: $\{(y, Axy') \cup \alpha_1\}$, $y'x<Axy'$.

$\mathbf{L}(3)$ [This level is constituted by four nodes. We develop here the first one only, the others can be developed in the same way.]

$$:\alpha_2, y'=A.$$

. . .

$\mathbf{L}(8)$: $\{(x, B) \cup \alpha_5\}$, ( )=( ). ■

One can observe that

$$\alpha_8(z)=AB, \alpha_8(y)=ABA, \alpha_8(x)=B.$$

is a solution of $e=(Axyxz, zzyB)$.

PROPOSITION (3.1): *Let s be a scheme applicable to an equation e, and suppose that s has a solution. The previous algorithm computes a solution of e.*

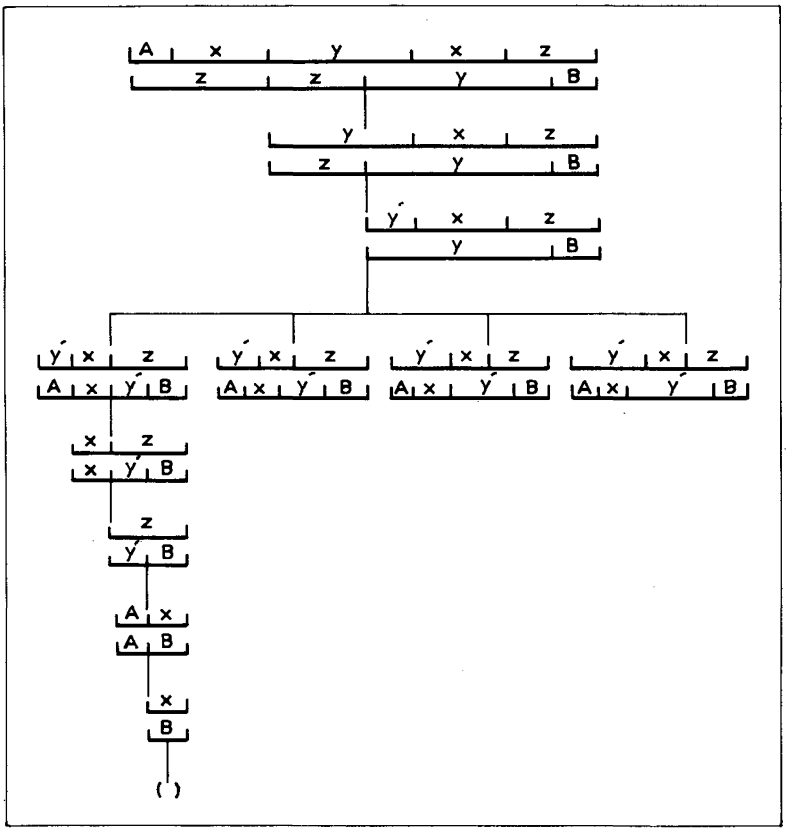

Figure 11

Proof: We say first that a node $n=(f,s,\alpha)$ has a solution when f has the solution

$$\alpha(t)\equiv \text{ if } t \text{ is a variable } v_i \text{ then } V_i \text{ else } t.$$

where V_i is given by the solution of s (*cf.* [2.2.1]).

The proof of this algorithm is based on the following two observations:

1. Suppose that n has a solution. Let l_1 be the length of a minimal solution of n. Then *successor* (*successor* (n)) contains a node having a minimal solution, whose length l_2 satisfy $l_2<l_1$.

2. If *successor* (n) contains a node having a solution, then n has a solution.

These two observations permit to show that there exists an integer k such that $(0\leqq k\leqq 2^{*}l_e)$, with l_e is the length of a minimal solution of the root of

U, and $\mathbf{L}(\mathbf{k})$ contains a node labeled by $(((\ ),(\ ))$, $(=), \beta)$. A solution of e can be deduced easily from β. ■

4. OTHER RESULTS

We now give a summary of other results, given in [1], and used in our implementation.

Concerning the transformation and the normalization of a position equation (these operations are the core of the algorithm), we show that the position equations resulting from any of four types of transformations (among five) are always normalized. We show also that one of the three conditions of normalization is always satisfied. These results play an interesting role, reducing the cost of testing the normalization.

Another important operation on position equations is to test whether an equation is admissible. We show how to construct efficiently the linear diophantine system associated with every position equation, and how to adapt the "cutting plane" method [6] for solving such a system.

For what concerns the tree $\mathscr{A}$, we provide from one side some improvements for the bound of its size. More precisely.

THEOREM (5.1)[1]: *The cardinal of $\mathscr{A}$ is bounded by the number*

$$(n+1) * (t+1) * d * m^{m+1} * k^{2^{2n+m}} * 2^{2^{(2n+1)^t (m+d)+m+d}}$$

with:

$$n = \mathrm{Card}(\mathbf{V}), \qquad m = \mathrm{Card}(\mathbf{C}), \qquad d = |e|,$$

$$t = \textit{the maximal length of a connection} = 2d(2d+1)(4d+1)s_0,$$

$$k = \textit{the maximal number of boundaries} = d(2n+1)^t + d. \quad ■$$

The previous bound improves the one given in [11] and reduces the complexity of the algorithm by one exponential. Essentially, the gain is based on the elimination of the equivalence test among the levels of $\mathscr{A}$.

We provide, from the other side, a strategy for the construction of A allowing a faster halting of the algorithm when the initial equation has a solution.

The representation of a position equation, in our implementation, is designed to achieve conveniently the basic operations of the algorithm. One of

these operations is to test whether, given a position equation, there exists an equivalent equation in $\mathcal{A}$.

We give an efficient algorithm to test the equivalence between position equations.

An interesting family of equations is the one which generates some position equations whose connection length grows continually. We provide some examples of this family illustrating the growth of connection length. The concept of generalized equivalence between position equations is then introduced. Informally, it identifies two position equations which differ by a periodical factor. We conjecture that the elimination of every position equation having an equivalent equation (in the sense of the generalized equivalence) permits to remove from the algorithm the test which eliminates every position equation containing a connection whose length is greater than the bound given by Makanin.

5. ACTUAL COMPUTER IMPLEMENTATION

Our implementation ([1], [2]) presents an interactive system written in LISP and running on VAX780 under UNIX, and on LISP Machine. This system visualizes the position equations, computes a solution whenever there exists one, and provides a tool permitting to understand, experiment with and study the algorithm.

The complete text of the LISP program is available in [1].

Here are in milliseconds some running results on a LMI LISP Machine.

1. Equation (*zxzyCBzxzx*, *yAByzxB*) has no solution: (433 ms).

2. Equation (*xAByCBzxtzux*, *yABytzuxB*) has no solution: (757 ms).

3. Equation (*xxAyBy*, *CAyvABD*) has solution $x = CAABD$, $v = CAABDAABDB$, $y = ABD$: (19 ms).

4. Equation (*BlABlABlA*, *rouDouDou*) has solution $l = ADABADAB$, $o = ABA$, $r = BADABADABABADABADAB$, $u = 1$: (43 ms).

ACKNOWLEDGMENTS

I wish to express special thanks to J. P. Pécuchet and B. V. Rozenblat for their helpful comments and criticisms. I would like to express my gratitude to the referees who helped me to improve the presentation of this paper.

REFERENCES

1. H. ABDULRAB, Résolution d'équations sur les mots: Étude et Implémentation LISP de l'Algorithme de Makanin, *Thèse*, Université of Rouen, 1987. And Rapport L.I.T.P. 87-25, University of Paris-VII, 1987.
2. H. ABDULRAB, *Implementation of Makanin's Algorithm*, Rapport L.I.T.P. 87-72, octobre 1987.
3. F. FAGES and G. HUET, Complete Sets of Unifiers and Matchers in Equational Theories, *Theoretical Computer Science*, Vol. 43, 1986, pp. 189-200.
4. W. M. FARMER, *A Unification Algorithm For Second-Order Monadic Terms*. The Mitre Corporation, Bedford, Massachusettes, 10, 1986.
5. J. GATHEN and M. SIEVEKING, A Bound on Solutions of Linear Integer Equalities and Inequalities, *Proc. of Amer. Math. Soc.*, 1978, pp. 155-158.
6. R. E. GOMORY, *An Algorithm For Integer Solutions to Linear Programs*, Recent advances in mathematical programming, R. L. GRAVES et P. WOLFE Ed., p. 269-302.
7. Yu. I. HMELEVSKII, Equations in Free Semigroups, *Trudy Mat. In st. Steklov*, Vol. 107, 1971.
8. G. HUET, An Algorithm to Generate the Basis of Solutions to Homogeneous Linear Diophantine Equations, *Inf. Proc. Let.*, Vol. 7, (3), 1978, pp. 144-145.
9. A. LENTIN, *Équations dans les monoïdes libres*, Gauthier-Villars, Paris, 1972.
10. A. LENTIN et M. P. SCHUTZENBERGER, A Combinatorial Problem in the Theory of Free Monoids, *Proc. of the University of North-Carolina*, 1967, pp. 128-144.
10. M. LOTHAIRE, *Combinatorics on Words*, Addison-Wesley Publishing Company, 1983.
11. G. S. MAKANIN, The Problem of Solvability of Equations in a Free Semigroup, *Mat. Sb.*, Vol. 103, (145), 1977, pp. 147-236. English transl. in *Math. U.S.S.R. Sb.*, Vol. 32, 1977.
12. J. P. PÉCUCHET, Équations avec constantes et Algorithme de Makanin, *Thèse*, University of Rouen, 1981.
13. J. P. PÉCUCHET, Solutions principales et rang d'un système d'équations avec constantes dans le monoïde libre, *Discrete Mathematics*, Vol. 48, 1984, pp. 253-274.
14. D. PERRIN, Equations in Words: a Survey, *Proceeding of the Colloquium on Resolution of Equations in Algebraic Structures* (*CREAS*), Austin, 1987 (to appear).