J. Symbolic Computation (1989) **8**, 499–521

SOLVING WORD EQUATIONS

HABIB ABDULRAB AND JEAN-PIERRE PÉCUCHET
Laboratoire d'Informatique de Rouen et LITP.†
Faculté des Sciences, B.P. 118, 76134 Mont-Saint-Aignan Cedex
E.m.: abdulrab@geocub.greco-prog.fr pecuchet@geocub.greco-prog.fr

July 17, 1989

Pure associative unification is equivalent to the resolution of word equations. We give a short survey of known results and algorithms in this field and describe the central algorithm of Makanin.

Terminology and Notations

Pure associative unification is better known in literature as the resolution problem of word equations.

An algebra equipped with a single associative law is a **semigroup**. It is a **monoid** when it has a unit. The free monoid generated by the set A (also called **alphabet**) is denoted by A^*. Its elements are the **words** written on the alphabet A, the neutral element being the empty word denoted by 1. The operation is the concatenation denoted by juxtaposition of words. The **length** of a word w (the number of letters composing it) is denoted by $|w|$. For a word $w = w_1 \dots w_n$, with $|w| = n$, we denote by $w[i] = w_i$ the letter at the ith position. The number of occurrences of a given letter $a \in A$ in a word w, will be denoted by $|w|_a$.

In this terminology, the term algebra (in the sense of Fages & Huet (1986), Kirchner (1987)) built on a set of variables V, a set C of constants, and a set of operators constituted of an associative law, is nothing else than the free monoid $T = (V \bigcup C)^*$ over the alphabet of **letters** $L = V \bigcup C$.

A unifier of two terms $e_1, e_2 \in T$ is a monoid morphism $\alpha : T \longrightarrow T$ (i.e. a mapping satisfying $\alpha(mm') = \alpha(m)\alpha(m')$ and $\alpha(1) = 1$), leaving the constants invariant (i.e. satisfying $\alpha(c) = c$ for every $c \in C$) and satisfying the equality $\alpha(e_1) = \alpha(e_2)$.

The pair of words $e = (e_1, e_2)$ is called an **equation** and the unifier α is a **solution** of this equation. For technical reasons, we usually restrict the image to the set of words $T' = (V' \bigcup C)^*$ using the only variables appearing in the image αT (we also say that the morphism α is **total**).

---

† This work was also supported by the Greco de Programmation du CNRS and the PRC Programmation Avancée et Outils pour l'Intelligence Artificielle.

0747–7171/89/110499+23 $03.00/0

A solution $\alpha : T \longrightarrow T'$ **divides** a solution $\beta : T \longrightarrow T''$ if there exists a **continuous** morphism $\theta : T' \longrightarrow T''$ (i.e. satisfying $\theta(x) \neq 1$ for every x) such as $\beta = \alpha\theta$. We also say that α is **more general** than β. A solution α is said to be **principal** (or **minimal**) when it is divided by no other but itself (or by an **equivalent** solution, i.e. of the form $\alpha' = \alpha\theta$ with θ, an isomorphism).

The **rank** of a principal solution $\alpha : (V \bigcup C)^* \longrightarrow (V' \bigcup C)^*$ is the number of variables appearing in the solution (i.e., the cardinal of V', as the morphism is supposed to be total). This notion extends to any solution (cf. Pécuchet (1984)), and we call **rank** of an equation the maximum rank of its solutions.

Introduction

The three main problems concerning systems of equations are the existence of a solution, the computation of the set of minimal solutions (denoted by μCSU_A in Fages & Huet (1986)) and the computation of the rank. All these problems reduce to the case of a single equation, as by Albert & Lawrence (1985) every infinite system of equations is equivalent to one of its finite subsystems, and a finite system can be easily encoded in a single equation (cf. Hmelevskii (1971)).

The study of properties and structure of the set of solutions of a word equation was initiated by Lentin and Schützenberger (1967), Lentin (1972) in the case of constant-free equations ($C = \emptyset$).

In particular, Lentin shows that every solution is divided by a unique minimal one and gives a procedure (known as the **pig-pug**: paire initiale gauche, paire ultime gauche) allowing to enumerate the set of minimal solutions. This procedure extends without difficulty to the general case of an equation with constants (cf. Plotkin (1972), Pécuchet (1981)). The minimal solutions are obtained as labels of some paths of a graph. When this graph is finite, as in the case when no variable appears more than twice, we obtain a complete description of all solutions.

It can be shown that the rank of an equation is the maximum rank of its principal solutions, that is, the maximum number of variables V' appearing in its principal solutions. The computation of the rank of certain types of constant-free equations was solved by Lentin (1972). Then Makanin (1977) showed that one can compute the rank of an equation with constants in the case of four variables and in the general case (cf. Makanin (1978)). Pécuchet (1984) shows that one can compute the **rank** of any equation as soon as one knows how to decide the existence of a solution, showing thus that the two algorithms of Makanin (1977, 1978) given by Makanin are not independent.

The problem of the existence of a solution was first tackled by Hmelevskii (1971) who solved it in the case of three variables, then by Makanin (1977) who solved the general case. He gave an algorithm to decide whether a word equation with constants has a solution or not.

All the problems linked to word equations are very close to those linked to equations in the free group. Thus the computation of minimal solutions by the pig-pug method uses transformations close to those used by Nielsen (1918) in the study of automorphisms in free groups, then taken up again by Lyndon (1960). Finally, an adaptation of his first algorithm allowed Makanin (1983) to solve the problem of the existence of a solution to an equation in a free group. This paper also gives a bound on the length of a solution of minimal length in the case of free monoid, allowing to use the pig-pug method as an algorithm to solve the problem. However, no proof of this bound independent of Makanin's algorithm is known, and the value of this bound makes the pig-pug method almost inoperating in the general case. So the original algorithm of Makanin seems still to give the most reasonable implementation.

This paper is divided into two parts. The first one will be devoted to a brief presentation of the pig-pug method which gives, for simple cases, the most efficient unification algorithm. The rest of this paper will be devoted to Makanin's Algorithm (1977) as it is implemented by Abdulrab (1987). In order to keep a reasonable size to this paper, most of the proofs will be omitted.

1. The pig-pug

In the remaining part of this paper, we assume without loss of generality, that the alphabets of variables $V = \{v_1 \dots v_n\}$ and of constants $C = \{c_1 \dots c_m\}$ are finite and disjoint. We make the convention to represent the variables by lower-case letters, as $x, y, z \dots$, and the constants by upper-case letters as $A, B, C \dots$ We call **length** of an equation $e = (e_1, e_2)$ the integer $d = |e_1 e_2|$.

The **projection** of an equation e over a subset Q of V is the equation obtained by "erasing" all the occurrences of $V \setminus Q$. Consequently, an equation has 2^n projections $(\Pi_Q e_1, \Pi_Q e_2)$ where $\Pi_Q : (V \bigcup C)^* \longrightarrow (Q \bigcup C)^*$ is the projection morphism.

One easily proves the following proposition which reduces the research of a solution to that of a continuous one.

PROPOSITION 1.1 *An equation e has a solution iff one of its projections has a continuous solution.* ▮

The pig-pug method consists in searching for a continuous solution α in the following manner: it visits the lists $e_1[1], \dots, e_1[|e_1|]$ and $e_2[1], \dots, e_2[|e_2|]$ of symbols of e from left to right and at the same time, one tries to guess how their images can overlap. At each step, one makes a non deterministic choice for the relative lengths of the images of the first two symbols $e_1[1]$ et $e_2[1]$. According to the choice made :

$$|\alpha(e_1[1])| < |\alpha(e_2[1])|, \quad |\alpha(e_1[1])| = |\alpha(e_2[1])|, \quad |\alpha(e_1[1])| > |\alpha(e_2[1])|$$

one applies to the equation one of the three substitutions to variables :

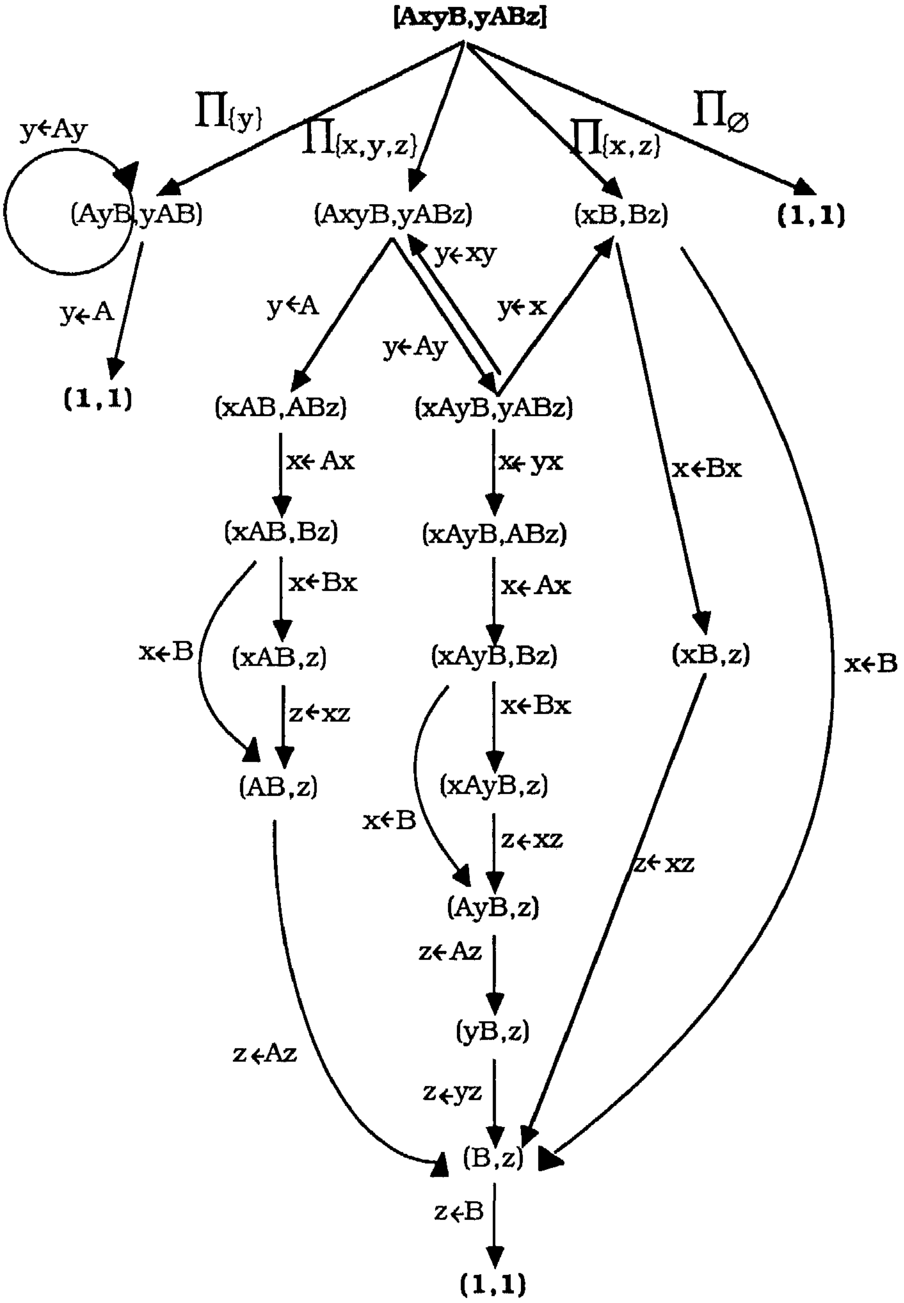
$$e_2[1] \leftarrow e_1[1]e_2[1], \quad e_2[1] \leftarrow e_1[1], \quad e_1[1] \leftarrow e_2[1]e_1[1].$$

The process is repeated until the **trivial** equation (1,1) is obtained.

EXAMPLE :

Consider the equation $(AxyB, yABz)$.

The application of the pig-pug method to all the projections of this equation is illustrated by the graph of Figure 1, in which, for readability, the equation (1,1) is duplicated and the unsuccessful paths are removed.



Here is a sequence of possible choices for the equation $(AxyB, yABz)$:

equation	substitution of variables
$(AxyB, yABz)$	$y \leftarrow Ay$
$(xAyB, yABz)$	$y \leftarrow x$
(xB, Bz)	$x \leftarrow Bx$
(xB, z)	$z \leftarrow xz$
(B, z)	$z \leftarrow B$
$(1, 1)$	

We thus obtain the solution $\alpha : \{x, y, z, A, B\}^* \longrightarrow \{x, A, B\}^*$ defined by: $\alpha(x) = Bx$, $\alpha(y) = ABx$, $\alpha(z) = xB$.

The following result, a proof of which can be found in Pécuchet (1981), shows that this graph enumerates all the minimal solutions.

THEOREM 1.2 (cf. Lentin (1972)) The set of minimal solutions of a word equation is given by the labels of the paths linking the root to the trivial equation in the pig-pug graph. ▮

One can consider the graph associated with an equation e, as a deterministic automaton M, in which :

1) the alphabet is given by the projections and the variable substitution.
2) the states of the automaton are the vertices of the graph.
3) the final state is $(1, 1)$.

The language accepted by this automaton, designated $L(M)$, is precisely the set of all the minimal solutions of e, that is, a complete set of minimal unifiers.

When the graph is finite, $L(M)$ is a regular language. But $L(M)$ can be a regular language accepted by an infinite automaton, (for example, this is the case of the automaton associated with $e = (Axx, xAx)$).

One can observe that $L(M)$ is not necessarily a regular language, (for example, this can be produced in the automaton associated with $e = (xy, Axx)$).

More generally, $L(M)$ is not necessarily a context-free language, (for example, this is the case of the automaton M associated with the equation $e = (xAAyx, AxxBz)$).

When the graph associated with an equation is finite, the pig-pug method provides a particularly simple unification algorithm. That is the case when no variable appears more than twice. In fact, it is easy to prove the following proposition:

PROPOSITION 1.3 When each variable appearing in the equation e has at most two occurrences, the length of all the equations derived by the pig-pug method is bounded by the length of e.

EXAMPLE :

The previous equation $(AxyB, yABz)$ is of this type. Let us note that the presence of cycles in the graph implies that the set of minimal solutions is infinite.

In the general case, the pig-pug's graph will be infinite. However one can always decide the existence of a solution by:

THEOREM 1.4 (cf. Makanin (1983)) One can construct a recursive function F such that, if an equation of length d has a solution, then there exists one in which the lengths of the components are bounded by $F(d)$. ▮

Note that the only known function F is that derived from Makanin's algorithm (1977) that we will describe in the next section. Another reason for the study of this algorithm is that it leads to a better pruning of the graph, and is more efficient than the pig-pug method in some cases.

Note also that this bound concerns the solutions of minimal length, and allows to use the pig-pug as a decision procedure. When the set of minimal solutions is finite, the pig-pug's graph is not necessarily finite, and no bound concerning the length of minimal solutions is known. In this case the pig-pug cannot be used as a unification procedure, because it is not known how to stop the graph construction. The only known unification procedure is that of Jaffar (1989), based on Makanin's algorithm that we will describe now.

2. Length equations

Before describing Makanin's algorithm, we introduce the notion of length equations which is related to integer programming.

First note that if $Card(C) = 0$, the equation e has necessarily the trivial solution $\alpha(v) = 1$, for every $v \in V$. Consequently, we can assume that $Card(C) > 0$. An equation is called **simple** if $card(C) = 1$. Such equations are related to integer equations in the following way:

Let e be a simple equation, consider the commutative image e' of e:

$$(v_1^{p_1} \dots v_n^{p_n} c_1^{p_{n+1}}, v_1^{q_1} \dots v_n^{q_n} c_1^{q_{n+1}})$$

The linear diophantine equation:

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

is called the **length equation** associated with e.

The isomorphism between $c_1{}^*$ and $(N, +)$ gives the following correspondence:

PROPOSITION 2.1 There is a bijective correspondence between the solutions of a simple equation and the non-negative integer solutions of its length equation. ▮

With every equation e we associate the simple equation e' obtained by the substitution of all the constants of e by c_1. The length equation associated with e' is by definition that associated with e.

The following necessary condition is easily shown:

PROPOSITION 2.2 If an equation e admits a solution, then its length equation admits a non-negative integer solution. ▮

The converse is false as shown by the following counter-example:

EXAMPLE :

The equation $e = (Ax, xB)$ has no solution but its length equation admits the non-negative integer solution $x' = 1$.

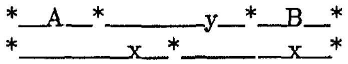
Thus solving simple equations reduces to integer programming. Next we shall see how Makanin's algorithm can solve non-simple equations.

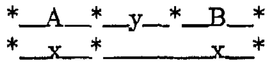
3. Equation with scheme and position equation

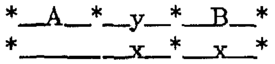
In this section, we describe two basic notions appearing in Makanin's algorithm: the notion of an equation with scheme, and that of a position equation. We show how to compute a position equation from an equation with scheme.

3.1. Equation with scheme

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. For example, the following diagrams illustrate such possibilities for the equation $e = (AyB, xx)$.







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.

Formally, a **scheme** is a 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 $e = (e_1, e_2)$ if the following conditions are satisfied:

1) $|s|_< + |s|_= = |e_1| + 1$.
2) $|s|_> + |s|_= = |e_2| + 1$.

where $|s|_\phi$, 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 $>$.

EXAMPLE :

In the first scheme applicable to $e = (AyB, xx)$ given above, $lb(e_2[1])$ is equal to 1 and $rb(e_2[1])$ is equal to 3.

An **equation with scheme** is a 6-tuple (V, C, e, s, lb, rb), where e is an equation over the alphabet of variables V and the alphabet of constants C, s is a scheme applicable to e, lb and rb are left and right boundary maps.

We say that a solution α of e **satisfies** a scheme applicable to e, if there exist words

$$T, L(1)\ldots L(|s|), R(1), \ldots, R(|s|) \text{ of } C^*$$

satisfying the following four conditions:

1- $L(1) = 1, L(|s|) = T$.

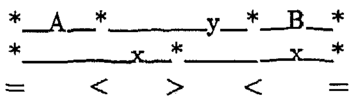
2- $|L(1)| < |L(2)| < \ldots < |L(|s|)|$.

3- $\forall i = 1, \ldots, |s|, \quad L(i)R(i) = T$.

4- For each symbol t of e, $\quad L(lb(t))\alpha(t)R(rb(t)) = T$.

EXAMPLE :

The equation with scheme $(\{x,y\}, \{A,B\}, (AyB, xx), =<><=, lb, rb)$ is illustrated by the following diagram:



For example, $lb(e_2[1])$ is equal to 1 and $rb(e_2[1])$ is equal to 3.

The solution $\alpha(x) = AB, \alpha(y) = BA$ of e satisfies the scheme, with the following choice of words $T, L(i), R(i)$:

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

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

$T = ABAB.$

3.2. *Informal presentation of an example*

An equation with scheme will now be transformed into a new object over which further processes will be applied. Before we give the formal definitions in the next sections, we introduce here the different notions and notations via an example.

Consider the equation with scheme

$$(\{A,B\},\{x,y,z\},AxyBz = zzx, =<><=<=, lb, rb)$$

corresponding to the following diagram:

```
*__A__*________x__*__y__*__B__*__z__*
*________z__*________z__*________x__*
=     <     >     <     =     <     =
```

This equation with scheme will be transformed into a so called position equation. This object inherits the seven boundaries of the equation with scheme and of all occurrences of constants, but variables will be treated in a special manner.

Variables with single occurrence, like y, will disappear.

Other occurrences of variables will be renamed in order to avoid the growth of the equations appearing in the pig-pug method. The renamed occurrences of a similar variable will be associated via a symmetrical binary relation on variables (called duality relation) or a positional equivalence, depending on the number of these occurrences, as shown below:

For a two occurrence variable, like x, the two occurrences will be renamed x_1 and x_2 and associated in the duality relation. That is, $x_1 = dual(x_2)$ and $x_2 = dual(x_1)$.

For a m occurrence variable u, with m greater than two, like z, $2m-2$ renamed variables will be generated as follows :

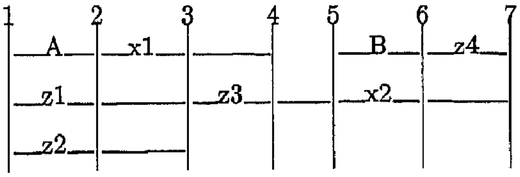
1) $m-1$ renamed variables (here z_1 and z_2) will receive the place of the first occurrence of u, (here the first occurrence of $z = e_2[1]$).

2) $m-1$ other renamed variables (here z_3 and z_4) will receive the place of the $m-1$ other occurrences of u, (here, $e_2[2]$ and $e_1[5]$).

3) each variable of the first set is in duality with one of the second set, (here, we have $dual(z_1) = z_3, dual(z_3) = z_1, dual(z_2) = z_4, dual(z_4) = z_2$).

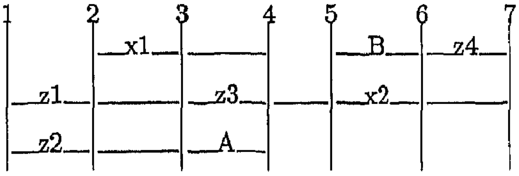
So, the duality relation gives a sort of "link" between the first $m-1$ occurrences associated with u, and the last $m-1$ occurrences. Since $z_1 = z_2$ because they have the same position, the equality of all the renamed variables $z_1 = z_2 = z_3 = z_4$ is obtained.

We then obtain an object which can be illustrated by the following diagram:

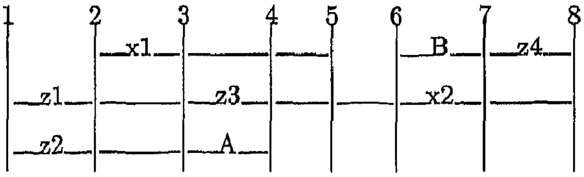


As in the pig-pug method, this position equation E will be transformed by pointing out the leftmost element (here the occurrence of A) for substituting it into the largest leftmost element (here the first largest leftmost variable z_1). The difference is that we put A at the beginning of the dual of z_1 (i.e. z_3), rather than substitute A in every variable associated with z_1. Note that, there are two ways to put A as a prefix of z_3. Either A takes all the segment between 3 and 4, or a part of this segment.

In order to avoid any loss of information during this move, a link is created between old and new positions of A in the form of a list called connection. This transformation gives rise to the two new position equations E' and E'' represented below and corresponding to the two possible positions of A relatively to the boundary 4. The link or connection (2 z_1 4) means that the prefix of z_1 ending at boundary 2 is equal to the prefix of its dual (i.e. z_3) ending at boundary 4.



(2 zl 4)



(2 zl 4)

The concept of a **solution** of a position equation will be given in the next section. Intuitively, if a position equation has a solution, then one can replace its variables by some words such that the positional constraints and the duality relation are satisfied. There is an intimate relationship between the existence of a solution of a position equation and the existence of solutions of the position equations resulting from its transformation. In the previous equation E, A is a prefix of z_1 and $z_1 = dual(z_1) = z_3$. So, A is also a prefix of z_3, and one of the two position equations E' and E'' has a solution. Conversely, if E' or E'' has a solution, so has E.

The moves used in the transformations of position equations will ensure that the number of variable occurrences remains bounded. But the growth of equation length appearing in the pig-pug method will now reappear in the growth of connection length. So, what did we win? The fact that one can prove that position equations with "long" connections can be eliminated, even while in the pig-pug method we have no direct proof that "long" equations can be deleted.

3.3. *Position equation*

Let us now give the formal definitions.

The notion of position equation given here is a constraint version of the notion of generalized equation introduced by Makanin (1977).

Formally, a position equation E is given by the following data:

DEFINITION 3.3.1 :

a- An alphabet of **constants** $C = \{c(1)\dots c(r)\}, (r \geq 0)$.

b- A list of **variables** $X = x(1)\dots x(2n), (n \geq 0)$.

c- An involution

$$\mathbf{dual}: X \longrightarrow X: \quad dual(x_i) = x_{i+n}, 1 \leq i \leq n.$$

exchanging each variable base with its dual; i.e. $x = dual(dual(x))$, for all $x \in X$.

d- A list called

Bases

consisting of $2n + m$ elements, $(m \geq r)$. The $2n$ first elements are the variables of X. The other bases are the occurrences of the constants of C (each constant having at least one occurrence in E).

e- A nonempty strictly ascending integer list

$$\{1, \dots, \$\}$$

called **Boundaries**.

f- Two mappings
left_boundary, right_boundary : *Bases* $\longrightarrow$ *Boundaries*.
satisfying the following two conditions:

$left\_boundary(x) < right\_boundary(x)$, for all $x \in X$.

$right\_boundary(w) = successor(left\_boundary(w))$ for each constant base w.

The boundaries of E are divided into two parts: the **essential** and the **inessential** boundaries. The set of essential boundaries contains all the left and right boundaries of the bases of E.

g- A finite (possibly empty) set of **connections**. Each connection is a list of the form

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

where p and q are two boundaries and $y(1)\dots y(k)(k \geq 1)$ are variable bases.

Each connection satisfies the following conditions:

g.1- $left\_boundary(y(1)) < p < right\_boundary(y(1))$.

g.2- $left\_boundary(y(i+1)) \leq left\_boundary(dual(y(i))).(1 \leq i \leq k-1)$.

g.3- $left\_boundary(y(k)) < q < right\_boundary(y(k))$.

g.4- q is an essential boundary.

g.5- Each inessential boundary is the original boundary of a connection.

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.

The bases with left boundary equal to 1 are called the **leading bases**. The first variable of X with left boundary equal to 1 and with greatest right boundary is called the **carrier**. Two dual variables with the same left and right boundaries are said to be **matched**.

DEFINITION 3.3.2 :

A **solution** of a position equation is defined by a a vector of words:

$$S = ((X(1),..,X(2n)),(L(1),..,L(\$)),(R(1),..,R(\$)),T) \in C^{+^{2n}} \times C^{*^{\$}} \times C^{*^{\$}} \times C^{+}$$

satisfying the following conditions:

a- $X(i) = DUAL(X(i))$ $(i = 1 \dots n)$. where $DUAL$ is the function leading to the following commuting diagram:

$$\begin{tikzcd}
\{\ X(1).\ \dots\ X(2n)\ \} \arrow[r, "DUAL"] & \{\ X(1).\ \dots\ X(2n) \\
X \arrow[u] \arrow[r, "dual"] & X \arrow[u]
\end{tikzcd}$$

}.

b- $L(1) = 1$, $L(\$) = T$.

c- $|L(1)| < \dots < |L(\$)|$.

d- $L(i)R(i) = T$ $(i = 1 \dots \$)$.

e- $L(left\_boundary(w))wR(right\_boundary(w)) = T$ if $w \in C$.
$L(left\_boundary(w))X(i)R(right\_boundary(w)) = T$ if $w = x(i)$.

f- For each connection $c = (p, x_1, \dots, x_k, q)$ there exist $B(1),..,B(k)$ in C^+ and $C(1),..,C(k)$ in C^+ such that:

$X_i = B(i)C(i)$ $(i = 1 \dots k)$

$L(p) = L(left\_boundary(x_1))B(1)$

$L(left\_boundary(dual(x_i)))B(i) = L(left\_boundary(x_{i+1}))B(i+1)$ with $i = 1,..,k-1$.

$L(q) = L(left\_boundary(dual(x_k)))B(k)$.

The position equation computed from an equation with scheme as described in 3.2 satisfies the following:

PROPOSITION 3.3.1 (cf. Abdulrab (1987)): A solution α of an equation e satisfies a scheme s iff the position equation computed from the equation e with scheme s has a solution. ▮

3.4. Admissible position equations

When a position equation E is simple (i.e. when $Card(C) = 1$), the system of equations given by the definition [3.3.2] can be transformed, via the isomorphism between C^* and $(N, +)$, into a system of linear diophantine equations called the **system of length equations of E**. In this case, E has a solution iff its system of length equations has a non-negative integer solution.

Consider now a non simple position equation E and let E' be the position equation obtained from E by replacing all the constants by a new and unique constant. The existence of a non-negative integer solution of the system of length equations of E' is only a necessary condition of the existence of a solution for E. A position equation is called **admissible** when its system of length equations has a non-negative integer solution.

The non-admissible position equations will be systematically eliminated. It is shown in Abdulrab (1987) how to construct efficiently the system of length equations, and how to adapt the cutting plan method of Gomory (1963) to solve such a system.

4. Transformation and normalization of position equations

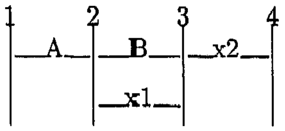
We describe in this section the two main operations on position equations.

4.1. Transformation of position equations

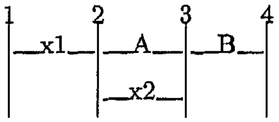
As seen in the example of 3.2, every position equation will be submitted to a transformation, which in the general case, consists in selecting the longest variable x with left boundary equal to 1 (i.e. the carrier), and in transferring all the other bases with left boundary equal to 1 under $dual(x)$.

More precisely, five different cases will occur corresponding to the five types of position equations listed below: (we assume that the boundary list $\{1, \ldots, \$\}$ is the interval $[1\ \$]$)

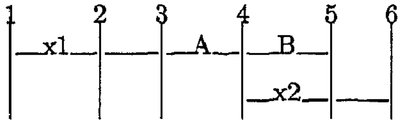
Type 1 : The position equation E has no carrier.



Type 2 : E has a carrier v, there is no other leading base, and the right boundary of v is equal to 2.

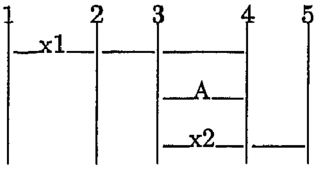


Type 3 : E has a carrier v, there is no other leading base, the right boundary of v is greater than 2 and equal to the second minimal essential boundary.



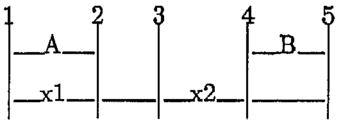
(2 x1 5)

Type 4 : E has a carrier v, there is no other leading base, the right boundary of v is greater than 2 and greater than the second minimal essential boundary.



(2 x1 4)

Type 5 : E has a carrier and other leading bases.



The transformation of a position equation E leads to a set of transformed position equations denoted by $T(E)$. Let us now describe more precisely what is this transformation in three of these cases.

The transformation of a position equation E of Type 1 consists in deleting the first boundary and the first constant base (if there is one).

The transformation of a position equation E of Type 3 consists in the following steps:

a- All the boundaries between the left and right boundaries of the carrier v are transferred between the left and right boundaries of $dual(v)$ in all the possible ways, but in the same order. We shall denote by p' the image of p in the transformed equation.

b- Each connection (p v q) is deleted.

c- Each connection (p v ... q) is transformed into (p' ... q).

d- The first boundary of E is deleted.

e- v and $dual(v)$ are deleted.

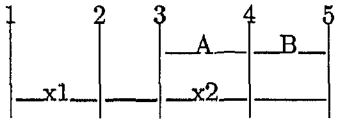
The transformation of a position equation E of Type 5 consists in transferring every leading base w under $dual(v)$, where v is the carrier. Each possibility gives a new position equation. If w is a variable base the list of the connections of a transformed position equation $E' \in T(E)$ is obtained in the following way:

a- Each occurrence of w in each connection of E is replaced by v, w and each occurrence of $dual(w)$ is replaced by $dual(w), dual(v)$:
$(\ldots, w, \ldots, dual(w), \ldots) \longrightarrow (\ldots, v, w, \ldots, dual(w), dual(v), \ldots)$

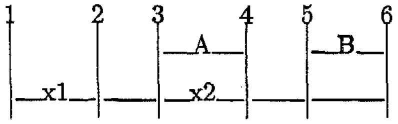
b- Let p be the right boundary of w in E, and l' its right boundary in E'. The new connection (p, w, l') is added to E'. In addition, each connection with concluding boundary equal to p is transformed into:
$(\ldots.p) \longrightarrow (\ldots., v, l')$.

EXAMPLE :

The set $T(E)$ computed from the position equation of Type 5 given in this section consists in the following two position equations:



(2 x1 4)



(2 x1 4)

4.2. Normalization of position equations

In order to eliminate position equations containing redundant information, we introduce some further conditions to be satisfied by the position equations computed by the algorithm. This will be crucial to ensure the finiteness of the set of equations to be developed by the algorithm.

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

N1- (cf. Makanin (1977)) No connection of E contains a segment of the form $(x_i, dual(x_i))$.

N2- (cf. Makanin (1977)) If the connection $(p, x_1, \ldots, x_k, q)$ contains two variable bases x_i, x_j with $(x_i = x_j \textit{ and } i < j)$ then there exists an integer l $(i \leq l < j)$ satisfying: $left\_boundary(x_{l+1}) < left\_boundary(dual(x_l))$.

N3- (cf. Pécuchet (1981)) The position equation E has no matched variables.

Whenever a non normalized position equation E is created by the transformation process, it will be replaced by a set $N(E)$ of normalized equations derived from E. In fact, the following result shows that this can only appear during the transformation of a Type 5 position equation.

PROPOSITION 4.2.1 (cf. Abdulrab (1987)): If an admissible and normalized position equation is of Type j $(1 \leq j \leq 4)$, then all the transformed position equations of $T(E)$ are normalized. ∎

5. Makanin's algorithm

The aim of this section is to describe Makanin's algorithm and to illustrate it with an example. Let e be an equation in L^*. We start by a general description of the algorithm:

If the length equation associated with e has no non-negative integer solution then, by proposition 2.1, it is clear that e has no solution. Otherwise if the equation e is simple then, by proposition 2.2, e has a solution. Otherwise the algorithm develops a tree level by level. The tree is denoted by $\mathcal{A}$ and its levels by $L_i (i \geq 0)$.

The first level L_0 contains the position equations computed from the schemes applicable to the projections of e. It is important to observe that it is the only step where new variable bases will be generated. Their number will then remain bounded.

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

Each position equation E of L_i is transformed as seen in 4.1 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 of boundaries, and we delete every position equation already occurring at 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. This is the central point which ensures the finiteness of the developed tree $\mathcal{A}$ and makes the algorithm to halt.

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 simple position equation, in which case e has a solution.

This leads to the following algorithm. The relations between two successive levels proving the correctness of the algorithm, and the combinatorical lemma justifying the use of $K(d)$ will be seen in the next section.

Makanin's algorithm:

```
 Input : an equation $e$ of $L^*$
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- $$L_i \leftarrow \bigcup_{j=1}^{2^{Card(V)}} e(Pj)$$
       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 Makanin (1977) by a strengthening of normalization conditions introduced in Pécuchet (1981) and the elimination of the already developed position equations introduced in Abdulrab (1987).

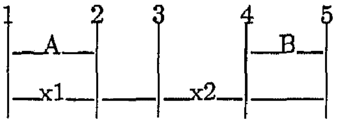
EXAMPLE :

We describe here an example solving the previously seen equation $e = (AyB, xx)$. This equation is not simple and its length equation has a solution. So we have to consider all the schemes applicable to the four projections of e: $(AB, 1), (AB, xx), (AyB, 1)$ and (AyB, xx). We will consider here only the following scheme applicable to the projection (AyB, xx):

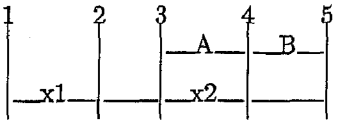
```
*__A__*_________y__*__B__*
*__________x__*_________x__*
```

In fact, all the other applicable schemes can be directly eliminated as shown in Abdulrab (1987).

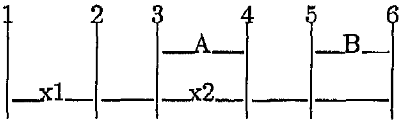
The position equation E computed from the previous equation with scheme is the following:



As seen in 4.1 this position equation is of Type 5, and its transformation leads to the following two position equations which are verified to be admissible and normalized.



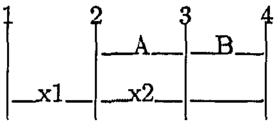
(2 xl 4)



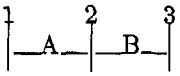
(2 xl 4)

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

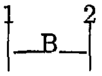
The transformation of this position equation of Type 3 consists in transferring all inessential boundaries, existing between the left and right boundaries of the carrier $x1$, to its dual. There is only one way to transfer the boundary 2 between the boundaries 3 and 5 because the connection (2 x1 4) identifies the part of $x1$ between 1 and 2, and the part of $x2$ between 3 and 4. Thus we have to transfer boundary 2 to 4 and to delete the connection, obtaining the following admissible and normalized position equation:



The transformation of this position equation of Type 2 consists in deleting the carrier and its dual leading to the following admissible and normalized position equation:



This position equation of Type 1 is then transformed into:



This last position equation is simple, and so the initial equation has a solution.

6. Proof of the algorithm

We state in this section the main results used for the proof of Makanin's algorithm. A complete proof is given in Pécuchet (1981).

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 x of X) such that:

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

The following theorem is fundamental:

THEOREM 6.1 (cf. Makanin (1977)) The exponent of periodicity s_0 of any minimal continuous solution of an equation e (with $|e| = d$) satisfies:

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

The proof of this result is given in Makanin (1977) 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 following result justifies the answers given by the algorithm. It is a reformulation of the theorem given in Makanin (1977) p. 168. A proof of this theorem, long and technical, can be found in Pécuchet (1981).

THEOREM 6.2 (cf. Makanin (1977)) If any level L_i ($i \geq 0$) of $\mathcal{A}$ contains a position equation admitting a solution with exponent of periodicity equal to s, and if there is no simple position equation in L_i then L_{i+1} contains a position equation with exponent $s' \leq s$. Conversely, if L_{i+1} contains a position equation admitting a solution then L_i contains a position equation which has a solution. ∎

The two previous results show that we can delete from $\mathcal{A}$ every position equation whose smaller solution has a coefficient of periodicity greater than

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

The next result shows that, this is the case of position equations whose connections exceed a certain length, and justifies the elimination of such position equations.

THEOREM 6.3 (cf. Makanin (1977)) Let (Y_i), $i = 1, \ldots, k$ $(k > 0)$, be an arbitrary sequence of the set $\{X_1 \ldots X_n\}$ of nonempty words. Let B_i, C_i $(i = 1, \ldots, k)$ be nonempty words, and S_i, R_i, T_i $(i = 1, \ldots, k-1)$ be arbitrary words, such that:

1. $Y_i = B_iC_i$ $(i = 1, \ldots, k)$.
2. $B_{i+1} = S_iB_i$ $(i = 1, \ldots, k-1)$.
3. $C_iR_i = C_{i+1}T_i$ $(i = 1, \ldots, k-1)$.
4. $j - i \geq K \Longrightarrow |S_iS_{i+1} \ldots S_j| > 0$.

Then some word Y_t has the form $Y_t = P^sQ$, where P is a nonempty word, and

$$s \geq \frac{k}{K.n.(n+1)} - 2.$$

The normalization and the elimination of the non admissible position equations and of those with long connections ensure the halt of the algorithm as shown by the following theorem:

THEOREM 6.4 (cf. Abdulrab (1987)) The cardinal of $\mathcal{A}$ is bounded by the number

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

with:

$l =$ *number of variable bases* $\leq 2d$, $m = Card(C)$, $d = |e|$,
$t =$ *the maximal length of a connection* $\leq 2d(2d+1)^2(s_0+2)$,
$k =$ *the maximal number of boundaries* $\leq (2l+1)^t d + d$ ▮

The previous bound improves the one given in Makanin (1977) and reduces the complexity of the algorithm by one exponential. We then prove the:

PROPOSITION 6.5 The initial equation e has a solution iff there exists a level in $\mathcal{A}$ which contains a simple position equation.

Proof : Suppose that L_i contains a simple position equation E. This equation is necessarily admissible and so it has a solution. By theorem 6.2 L_0 contains a position equation E_0 which has a solution. The proposition 3.3.1. shows that e has a solution.

Conversely, suppose that e has a solution. If there is no level in $\mathcal{A}$ which contains a simple position equation, then theorem 6.2 implies that there exists an infinite number of levels in $\mathcal{A}$, but this implies a contradiction with theorem 6.4. ▮

7. Implementation of the algorithm

The first implementation of the version of Makanin's algorithm presented in this paper, is described in Abdulrab (1987a,b).

This implementation is an interactive system written in LISP and running on VAX780 under UNIX, and on LISP Machine. This system visualizes the position equations, computes effectively a solution of the initial equation whenever there exists one, and provides a tool permitting to understand, experiment and study introdthe algorithm. Our implementation has also been coded in CAML by Rouaix (1987).

We show in Abdulrab (1987b) that the present implementation is not efficient enough to be used as a unification module in a programming system. We also describe the characteristics of a new version which could realize this goal.

The purpose of the algorithm being to decide whether an equation admits a solution or not, we provide an algorithm (cf. Abdulrab (1989)) which, by taking advantage of the tree $\mathcal{A}$, effectively computes a solution to the initial equation. The idea is to compute a solution to the equation e, from the equation with scheme which generates the root of the subtree containing a simple position equation.

Here is in milliseconds some execution 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).

Conclusion

In this paper, we gave a complete description of Makanin's algorithm which allows to decide whether a word equation has a solution or not.

New concepts, such as equations with schemes and position equations are introduced in our description. A new presentation of Makanin's algorithm permitting to all its steps to be illustrated graphically, is described in this paper. Several examples, illustrating all the procedures and the notions of the algorithm, are presented. The fundamental point is that our version has been effectively implemented.

When a word equation has a solution, one can use the pig-pug method to compute the set of all the minimal solutions of e. This set is presented as a language accepted by a deterministic automaton (not necessarily finite).

The main results of the proof of Makanin's algorithm are set out in this paper. Some improvements concerning the reduction of the number of types of position equations, and the reduction of the complexity of the algorithm by one exponential, are given here. Other technical results concerning our work in this area can be found in Pécuchet (1981) and Abdulrab (1987). We prove, for example, that one of the three conditions of the normalization is always satisfied. The size of the tree $\mathcal{A}$ is greatly reduced in our version. Our construction of $\mathcal{A}$ is based on the notion of a solution of an equation with scheme, which allows the elimination of some types of equations with schemes which have no solution.

Acknowledgements: We thank the referees for their helpful comments which allowed us to improve the presentation of some sections of this paper.

References

Abdulrab H. (1987). Résolution d'équations sur les mots: étude et implémentation LISP de l'algorithme de Makanin. (Thèse). University of Rouen. And Rapport LITP 87-25, University of Paris-7.

Abdulrab H. (1987). Implementation of Makanin's algorithm. Rapport LITP 87-72, University of Paris-7.

Abdulrab H. (1989). Solving equations in words. to appear in RAIRO d'Info. th..

Albert M.H, Lawrence J. (1985). A proof of Ehrenfeucht's conjecture. Theor. Com. Sci. 41 p. 121-123.

Fages F., Huet G. (1986). Complete sets of unifiers and matchers in equational theories. Theor. Com. Sci., 43, p. 189-200.

Gomory R.E. (1963). An algorithm for integer solutions to linear programs. Recent advances in mathematical programming, Eds R.L Graves et p. Wolfe. p. 269-302.

Hmelevskii Yu. I. (1971). Equations in free Semigroups. Trudy Mat. Inst. Steklov, 107.

Jaffar J. (1989). Minimal and complete word unification. To appear in JACM.

Kirchner C. (1987). From unification in combination of equational theories to a new AC-Unification algorithm. Proc. of the CREAS, Austin, Texas.

Lentin A. (1972). Équations dans le monoïde libre, Gauthier – Villars, Paris.

Lentin A., Schutzenberger M.P. (1967). A combinatorial problem in the theory of free monoids. Proc. of the University of northCarolina, p. 128-144.

Lyndon R.C. (1960). Equations in free groups, Trans. Amer. Math. Soc. 96, p. 445-457.

Makanin G.S. (1977). On the rank of equations in four unknowns in a free semigroup. English transl. in Math. USSR Sb. 29, p. 257-280.

Makanin G.S. (1977). The problem of solvability of equations in a free semigroup. Mat. Sb. 103(145) (1977) p. 147-236 English transl. in Math. USSR Sb. 32.

Makanin G.S. (1978). Algorithmic decidability of the rank of constant free equations in a free semigroup. Dokl. Akad. Nauk. SSSR 243.

Makanin G.S. (1983). Equations in a free group. Math. USSR Izvestiya Vol. 21, No 3.

Nielsen J. (1918). Die Isomorphismen der allgemeinen, unendlichen Gruppe mit zwei Erzeugenden, Math. Ann. 78, p. 385-397.

Pécuchet J.P. (1981). Équations avec constantes et algorithme de Makanin. (Thèse). University of Rouen.

Pécuchet J.P. (1984). Solutions principales et rang d'un système d'équations avec constantes dans le monoïde libre. Discrete Mathematics, 48, p. 253-274.

G. Plotkin. (1972). Building-in equational theories. Machine Intelligence 7, p. 73-90.

Rouaix F. (1987). Une implémentation de l'algorithme de Makanin en CAML. Mémoire de DEA d'Informatique, University of Paris7.