Associative-commutative unification

Francois Fages

INRIA
CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France

Tél. 954 90 20

Rapports de Recherche

N° 287

ASSOCIATIVE-COMMUTATIVE UNIFICATION

François FAGES

Avril 1984

Résumé

L'unification dans les théories équationnelles, c'est-à-dire la résolution d'équations dans les variétés, est particulièrement important en démonstration automatique. Les résultats récents sur les systèmes de réécriture, comme dans [Peterson et Stickel 81] et [Hsiang 82], reposent sur l'unification en présence d'opérateurs associatifs et commutatifs. Stickel [75,81] a donné un algorithme d'unification associative-commutative, mais sa terminaison dans le cas général était toujours questionnée. Ici nous donnons un cadre abstrait pour présenter les problèmes d'unification, et nous prouvons la correction totale de l'algorithme de Stickel.

ASSOCIATIVE-COMMUTATIVE UNIFICATION

François Fages

CNRS, LITP 4 place Jussieu 75221 Paris Cedex 05,
INRIA Domaine de Voluceau Rocquencourt, 78153 Le Chesnay.

ABSTRACT

Unification in equational theories, that is solving equations in varieties, is of special relevance to automated deduction. Recent results in term rewriting systems, as in [Peterson and Stickel 81] and [Hsiang 82], depend on unification in presence of associative-commutative functions. Stickel [75,81] gave an associative-commutative unification algorithm, but its termination in the general case was still questioned. Here we give an abstract framework to present unification problems, and we prove the total correctness of Stickel's algorithm.

The first part of this paper is an introduction to unification theory. The second part is devoted to the associative-commutative case. The algorithm of Stickel is defined in ML [Gordon, Milner and Wadsworth 79] since in addition to being an effective programming language, ML is a precise and concise formalism close to the standard mathematical notations. The proof of termination and completeness is based on a relatively simple measure of complexity for associative-commutative unification problems.

1. Unification in equational theories

1.1. Equational theories

We assume well known the concept of an algebra $\mathbf{A} = \langle A, F \rangle$ with A a set of elements (the carrier of $\mathbf{A}$) and F a family of operators, given with their arities. More generally, we may consider heterogeneous algebras over some set of sorts, but all the notions considered here carry over to sorted algebras without difficulty, and so we will forget sorts and even arities for simplicity of notation. With this provision, all our definitions are consistent with [Huet and Oppen 80].

We denote by $T(F)$ the set of (ground) terms over F. We assume that there is at least one constant (operator of arity 0) in F so that this set is not empty. We also assume the existence of a denumerable set of variables V, disjoint from F, and denote by $T(F, V)$ the set of terms with variables over F and V. When F and V are clear from the context, we abbreviate $T(F, V)$ as T and $T(F)$ as G (for ground). We denote terms by M,N,..., and write $V(\mathrm{M})$ for the set of variables appearing in M.

We denote by $\mathbf{T}$ (resp. $\mathbf{G}$) the algebra with carrier T (resp. G) and with operators the term constructors corresponding to each operator of F.

The *substitutions* are all mappings from V to T, extended to T, as endomorphisms of $\mathbf{T}$. We denote by S the set of all substitutions. If $\sigma \in S$ and $\mathrm{M} \in T$, we denote by $\sigma\mathrm{M}$ the application of σ to M. Since we are only interested in

substitutions for their effect on terms, we shall generally assume that $\sigma x=x$ except on a finite set of variables $D(\sigma)$ which we call the *domain* of σ by abuse of notation. Such substitutions can then be represented by the finite set of pairs $\{<x,\sigma x> \mid x\in D(\sigma)\}$. We define the *range* $R(\sigma)$ of σ as :

$$R(\sigma) = \bigcup_{x\in D(\sigma)} V(\sigma x).$$

We say that σ is *ground* iff $R(\sigma)=\phi$. The composition of substitutions is the usual composition of mappings: $(\sigma\circ\rho)x=\sigma(\rho x)$. And we say that σ *is more general than* ρ : $\sigma\leq\rho$ iff $\exists\eta\ \eta\circ\sigma=\rho$.

An equation is a pair of terms M=N. Let E be a set of equations (axioms), we define the *equational theory presented by* E as the finest congruence over T containing all pairs $\sigma M=\sigma N$ for M=N in E and σ in S. It is denoted by $=_E$. An equational theory presented by E is *axiomatic* iff E is finite or recursive.

An algebra **A** is a *model* of an equation M=N if and only if $\nu M=\nu N$ as elements of A. for every assignment ν (i.e. mapping from V to A extended as a morphism from **T** to **A**). We write $\mathbf{A} \models M=N$. **A** is a model of an equational theory E iff $\mathbf{A} \models E$ for every E in E. We denote by $M(E)$ the class of models of E, which we call the *variety* defined by E.

E-equality in T is extended to substitutions by extensionality:

$$\sigma =_E \rho \text{ iff } \forall x\in V\ \sigma x =_E \rho x.$$

We write for any set of variables V:

$$\sigma \overset{V}{=}_E \rho \text{ iff } \forall x\in V\ \sigma x =_E \rho x.$$

In the same way, σ *is more general than* ρ *in* E *over* V,

$$\sigma \overset{V}{\leq}_E \rho \text{ iff } \exists\eta\ \eta\circ\sigma \overset{V}{=}_E \rho.$$

The corresponding equivalence relation on substitutions is denoted by $\overset{V}{\equiv}_E$; i.e. $\sigma \overset{V}{\equiv}_E \rho$ iff $\sigma \overset{V}{\leq}_E \rho$ and $\rho \overset{V}{\leq}_E \sigma$. We will omit V when $V=V$, and E when $E=\phi$.

1.2. *E*-unification

1.2.1. Historical

Let E be an equational theory. A substitution σ is a *E-Unifier* of terms M and N if and only if $\sigma M =_E \sigma N$.

Hilbert's tenth problem (solving of polynomial equations over integers, called Diophantine equations) is the unification problem in arithmetic. Livesey, Siekmann, Szabo and Unvericht [79] have proved that Associative-Distributive unification is undecidable, and thus that the undecidability of Hilbert's tenth problem [Matiyasevich 70, Davis 73] does not rely on a specific property of integers.

We denote by U_E the set of all E-unifiers of M and N:

$$U_E(M,N) = \{\sigma\in S \mid \sigma M =_E \sigma N\}.$$

Axiomatic equational theories are semi-decidable, and U_E is always recursively enumerable, but of course we are mostly interested in a generating set of

the E-unifiers (called Complete Set of E-Unifiers by Plotkin [72], and denoted by CSU_E), from which we can generate U_E by instantiations. Or better by a basis of U_E (called Complete Set of Minimal Unifiers and denoted by μCSU_E) satisfying the minimality condition $\sigma \neq \sigma' \Rightarrow \sigma \not\leq_E \sigma'$.

So we shall make the difference between *unification procedures* which enumerate a CSU_E (the exhaustive enumeration procedure in semi-decidable theories enumerates U_E entirely), *unification algorithms*, which always terminate with a finite CSU_E, empty if terms are not unifiable, and *minimal unification procedures or algorithms* which compute a μCSU_E.

Unification has been for the first time studied in first order languages (the case $E=\phi$) by Herbrand [30]. In his thesis he gave an explicit algorithm to compute a most general unifier. However the notion of unification really grew out of the work of the researchers in automatic theorem-proving, since the unification algorithm is the basic mechanism needed to explain the mutual interaction of inference rules. Robinson [65] gave the algorithm in connection with the resolution rule, and proved that it indeed computes a most general unifier. Independently, Guard [64] presented unification in various systems of logic. Unification is also central in the treatment of equality [Robinson and Wos 69, Knuth and Bendix 70]. Implementation and complexity analysis of unification is discussed in [Robinson 71], [Venturini-Zilli 75], [Huet 76], [Baxter 77], [Paterson and Wegman 78] and [Martelli and Montanari 82]. Paterson and Wegman give a linear algorithm to compute the most general unifier.

First order unification was extended to infinite (regular) trees by Huet [76], who showed that a single most general unifier exists for this class, computable by an almost linear algorithm. This problem is relevant to the implementation of PROLOG like programming languages [Colmerauer 72,82, Fages 83].

In the context of higher order logic, the problem of unification was studied by Gould [66], who defined "general matching sets" of terms, a weaker notion than that of CSU. The existence of a unifier is shown to be undecidable in third order languages in [Huet 73], and at second order by Goldfarb [81]. The general theory of CSU's and μCSU's in the context of higher order logic is studied in [Huet 76, Jensen and Pietrzykowski 77].

Unification in equational theories has been first studied by Plotkin [72] in the context of resolution theorem provers to build-up the underlying equational theory into the rules of inference. In this paper Plotkin conjectured that there existed an equational theory E where a μCSU_E did not always exist. Theorem 1 in the next chapter proves this conjecture.

Further interest in unification in equational theories arose from the problem of implementing programming languages with "call by patterns", such as QA4 [Rulifson 72]. Associative unification (finding solutions to word equations) is a particularly hard problem. Plotkin [72] gives a procedure to enumerate a μCSU_A (eventually infinite), and Makanin [77] shows that the word equation problem is decidable. Stickel [75,81] and independently Livesey and Siekmann [76,79], give an algorithm for unification in presence of associative-commutative operators. However its termination in the general case was still questionned since some recursive calls are made on terms having a bigger size than the initial terms. Theorems 3 and 4 in this paper prove the termination and the completeness in the general case. Siekmann [78] studied the general problem in his thesis, especially the extension of the AC-unification algorithm to idempotence and identity. Lankford [79,83] gave the extension to a unification procedure in Abelian group theory.

In the class of equational theories for which there exists a canonical term rewriting system (see [Huet and Oppen 80]), Fay [79] gives a universal procedure to enumerate a CSU_E. It is based on the notion of "narrowing", as defined in [Slagle 74]. Hullot [80] gives a similar procedure and a sufficient termination criterion, further generalized in [Jouannaud and Kirchner 83]. Siekmann and Szabo [82] investigate the domain of regular canonical term rewriting systems in order to find general minimal unification procedures, but we show in [Fages and Huet 83] that even in this framework μCSU_E may not exist.

Termination or minimality of unification procedures is much harder to obtain than completeness. However the main applications of unification in equational theories to the generalizations of the Knuth and Bendix algorithm, such as in [Peterson, Stickel 81 and Hsiang 82], are covered by the associative-commutative unification algorithm.

1.2.2. Definitions

Let $M,N \in T$, $V = \mathcal{V}(M) \cup \mathcal{V}(N)$ and W be a finite set of "protected variables". S is a *Complete Set of E-Unifiers of* M *and* N *away from* W if and only if :

a) $\forall \sigma \in S \;\; D(\sigma) \subseteq V$ and $R(\sigma) \cap W = \phi$ (purity)

b) $S \subseteq U_E(M,N)$ (correctness)

c) $\forall \rho \subseteq U_E(M,N) \; \exists \sigma \subseteq S \;\; \sigma \overset{V}{\leq_E} \rho$ (completeness)

Furthermore S is a *complete Set of Minimal E-Unifiers of* M *and* N *away from* W if additionally :

d) $\forall \sigma,\sigma' \in S \;\; \sigma \neq \sigma' \Rightarrow \sigma \overset{V}{\not\leq_E} \sigma'$ (minimality)

Remark that the most general unifiers in first order languages are μCSU_ϕ reduced to one element. The reason to consider W is that in many algorithms, unification must be performed on subterms, and *it is necessary to separate the variables introduced by unification from the variables of the context.* It is the case for instance for resolution in equational theories [Plotkin 72], and for the generalization of the Knuth and Bendix completion procedure in congruence classes of terms [Peterson and Stickel 81]. It is easy to show that there always exists a CSU_E away from W, by taking all E-unifiers satisfying a).

We may add to the definition of CSU_E :

d') $\forall \sigma,\sigma' \in S \;\; \sigma \neq \sigma' \;\Rightarrow\; \sigma \overset{V}{\not\leq_E} \sigma'$ (non-congruency)

Such CSU_E still always exist but we loose the property that if U_E is recursively enumerable then there exists a recursively enumerable one. For example, in undecidable axiomatic equational theories U_E is recursively enumerable but in general the CSU_E satisfying d') are not.

1.2.3. Existence of basis of the E-unifiers

It is well known that *there may not exist a finite* CSU_E. For instance a*x=x*a in the theory where * is associative [Plotkin 72]. When there exists a finite CSU_E, there always exists a minimal one, by filtering out redundant elements. But it is not true in general :

Theorem 1 (non-existence of basis) : In some first order equational theory E there exist E-unifiable terms for which there is no μCSU_E.

Proof : The proof is in [Fages and Huet 83], where it is shown that there may be infinite strings of E-unifiers more and more general. And thus that minimality d) may be incompatible with completeness c). The example is E={f(0,x)=x , g(f(x,y))=g(y)}, for terms g(x) and g(a). ■

However when a μCSU_E exists, it is unique up to $\overset{V}{\equiv}_E$.

Theorem 2 (unicity of basis) : Let M and N be two terms, U_1 and U_2 be two μCSU_E of M and N. There exists a bijection $\varphi : U_1 \to U_2$ such that $\forall \sigma \in U_1 \ \ \sigma \overset{V}{\equiv}_E \varphi(\sigma)$.

Proof : $\forall \sigma \in U_1 \ \exists \rho \in U_2 \ \ \rho \overset{V}{\leq}_E \sigma$ since U_2 is complete. We pick-up one such ρ as $\varphi(\sigma)$.

$\forall \sigma' \in U_2 \ \exists \rho' \in U_1 \ \ \rho' \overset{V}{\leq}_E \sigma'$. We pick-up one such ρ' as $\psi(\sigma')$.

Thus $\forall \sigma \in U_1 \ \ \psi(\varphi\sigma) \overset{V}{\leq}_E \sigma$ so $\psi(\varphi\sigma)) = \sigma$ by minimality,

$\varphi(\sigma)) \overset{V}{\leq}_E \sigma \overset{V}{\leq}_E \varphi(\sigma)$ i.e. $\sigma \overset{V}{\equiv}_E \varphi(\sigma)$.

■

2. ML as a programming language for the term structure

ML is an applicative language with *exceptions* in the line of ISWIM [Landin 66], POP2 [Burstall, Collins and Popplestone 71], GEDANKEN [Reynolds 70]. Functions are "first class citizens", the type inference mechanism allows *functionals* at any order, and *polymorphic* operators. *Primitive types* are : void, bool, int and string, and *type operators* are : cartesian product ×, sum + and function →. Type variables are denoted by *, **, ... The declaration of an *abstract type* consists in the definition of constructors and external functions. For instance, the polymorphic list type, ** list = void + (* × * list)*, is predefined with the constructors **nil** and **cons** (noted . in infix), and the destructors **hd** : ** list → ** and **tl** : ** list → * list*. Variables can be structured in lists and pairs, bindings are then made by *matching*. We refer to the LCF manual [Gordon, Milner and Wadsworth 79] for the syntax and semantics of ML.

The composition of functions is an infix operator **o** : *(** → ***) × (* → **) → * → ****.
It could be defined by :

```
mlinfix 'o';;
let (f o g) x = f (g x) ;;
```

The curried function **map** : *(* → **) → * list → ** list* returns the list of the function applications to a list of arguments. It is equivalent to :

```
letrec map f l = if l = nil then nil else f (hd l) . (map f (tl l));;
```

Particularly important is **itlist** : *(* → ** → **) → * list → ** → *** which iterates the application of a function to a list of arguments, by composing the results :
itlist f $[l_1;\ldots;l_n]$ x = $(f\ l_1\ (f\ l_2\ \cdots\ (f\ l_n\ x)\ \cdots\ )) = ((f\ l_1) o (f\ l_2) o \ldots o (f\ l_n))\ x$. It is defined by :

```
letrec itlist f l x = if l=nil then x else f (hd l) (itlist f (tl l) x);;
```

For example **flatten** : ** list list → * list* which eliminates the first level parentheses in a list can be defined by :

```
let flatten l = itlist append l nil;;
```

We generalize itlist to the iterative application of a curried function on two lists of arguments, with :

```
letrec itlist2 f k l x = if k=nil then x else f (hd k) (hd l) (itlist2 f (tl k) (tl l) x);;
```

We will not detail the abstract types related to the term structure, we keep previous notations related to them, and define the function **op** : $T \to F$ which

returns the head symbol of a term, and **largs** : $T \rightarrow T\ list$ which returns the list of the arguments.

```
abstype F = ...;;
abstype V = ...;;
absrectype T = F × T list + V
  with ... and op M = ... and largs M = ... and isvar M = ... and equal(M,N) = ...;;
lettype S = T→T';; mlinfix '←';; letrec x←M = ...;;
```

Since substitutions are functions from terms to terms, the composition of substitutions is exactly the composition of functions **o**, and the identity function **I** serves as the identity substitution.

If we suppose defined the function **occurs** : $T \times T \rightarrow bool$ which recognizes if a variable occurs in a term, the unification algorithm of Robinson [65] may be defined by the function **uni** : $T \times T \rightarrow S$ with :

```
letrec uni(M,N) =
    if isvar M then if equal(M,N) then I
        else if occurs(M,N) then fail else M←N
    else if isvar N then if occurs(N,M) then fail else N←M
    else if op M≠op N then fail
        else (itlist2 unicompound (largs M) (largs N) I)
and unicompound A B σ = uni (σ A,σ B) o σ;;
```

3. AC-unification

3.1. Connection with the solving of linear homogeneous diophantine equations

A binary function + is associative and commutative iff it satisfies (in infix notation) :

$$\begin{cases} x+y = y+x \\ (x+y)+z = x+(y+z) \end{cases}$$

The set of those function symbols is denoted by F_{AC}. We will not consider them as symbols of variable arity in order to stay in the term algebras formalism, but we define the function **largAC** : $T \rightarrow T\ list$ which returns the list of the AC-arguments, that is the list of the arguments after the elimination of parentheses on the head symbol if it is AC. For example with M = (x+(x+y))+(f(a+(a+a))+(b+c)) where $+ \in F_{AC}$, we have op M = +, largs M = [x+(x+y) ; f(a+(a+a))+(b+c)] and largAC M= [x ; x ; y ; f(a+(a+a)) ;b ; c].

Let two terms M and N beginning with the same AC head symbol to be unified. Stickel [81] proved that the elimination of common arguments, pair by pair, does not change $U_{AC}(M,N)$. For example the unification of M = (x+(x+y))+(f(a+(a+a))+(b+c)) and N = ((b+b)+(b+z))+c, where $+ \in F_{AC}$, is equivalent to the unification of the arguments lists [x;x;y;f(a+(a+a))] and [b;b;z], obtained by eliminating the common arguments b and c. If a variable is eliminated in this operation, it must be added to the set of context variables W.

Unification of (non-ordered) lists of arguments is the problem of solving equations in the free abelian semigroup, which is isomorphic to the solving of homogeneous linear diophantine equations $\sum a_i x_i = \sum b_j y_j$ over $\mathbb{N}-\{0\}$. Thus for any AC-unification problem, we associate such an equation by associating integer variables to distinct arguments, with their multiplicity as coefficient. For instance $2x_1+x_2+x_3 = 2y_1+y_2$ in the example. In return the solutions to the diophantine equation induce the unifiers of the lists elements, that are still to unify with the effective arguments.

Stickel [76] and Huet [78] give an algorithm to solve homogeneous linear diophantine equations, which enumerates a basis of solutions, by backtracking

with a certain bound on the value of the variables, and elimination of the redundant solutions. The best bound, is double :

1) $\forall i\ x_i \leq \max_k b_k\ ,\ \forall j\ y_j \leq \max_l a_l$

2) $\forall i\ \forall j\quad x_i \leq \dfrac{lcm(a_i,b_j)}{a_i}$ or $y_j \leq \dfrac{lcm(a_i,b_j)}{b_j}$

The basis of solutions can be represented as a matrix with as many columns as variables in the equation, and as many lines as solutions in the basis. For example the equation $2x_1+x_2+x_3 = 2y_1+y_2$ admits as solutions basis the matrix :

	x_1	x_2	x_3	y_1	y_2
s_1	0	1	1	1	0
s_2	0	0	2	1	0
s_3	0	2	0	1	0
s_4	1	0	0	1	0
s_5	0	0	1	0	1
s_6	0	1	0	0	1
s_7	1	0	0	0	2

Any linear combination of the seven elements of the basis is a solution to the equation. However, because the absence of a zero in the unification problem, we must consider all subsets of the solutions basis, with the constraints that the sum of the coefficients in a column must be non null, and equal to 1 if the corresponding term is not a variable.

Hullot [80] gives a method for a *constrained enumeration of partitions*, in order to reduce the high complexity of this computation. We refer to his thesis for details of this technique which is crucial in an implementation. In the example, among the 2^7=128 solutions, only 6 are to be considered. Let us examine $\{s_4,s_5,s_6,s_7\}$: $x_1 = s_4+s_7$, $x_2 = s_6$, $x_3 = s_5$, $y_1 = s_4$, $y_2 = s_5+s_6+s_7$. We deduce a unifier from it, by associating to each solution s_i a new variable u_i :

$\sigma = \{x \leftarrow b+u_7\ ,\ z \leftarrow f(a+(a+a))+(y+(u_7+u_7))\ ,\ u_6 \leftarrow y\ ,\ u_5 \leftarrow f(a+(a+a))\ ,\ u_4 \leftarrow b\}$.

3.2. The algorithm in ML

The unification of non-ordered lists of arguments, described in the previous chapter, may be defined with the function **unilist** : *T list*×*T list*×*F*→*T list*×*(T list list)* as :

```
let unilist (lM,lN,f) = let lA,matrix,n = dio (elimcom (lM,lN))
                        in lA , trad (f , lA , partit (matrix,n));;
```

where **elimcom** : *T list*×*T list*→*T list*×*T list* eliminates common terms to the two lists of arguments; **dio** : *T list*×*T list*→*T list*×*(int list list)*×*int* solves the diophantine equation associated to the two lists of arguments, and returns the simplified list of arguments with variable arguments first, the matrix of integer solutions that forms a basis (with variable arguments columns first) together with the number of variable arguments; **partit** : *(int list list)*×*int*→*(int list list list)* enumerates all the partitions of the basis satisfying the constraint that the sum of the coefficients in a column must be non null, and greater than 1 for the non-variable arguments (last columns); and where **trad** : *F*×*T list*×*(int list list list)*→*(T list list)* expresses each solution attached to a partition as a list of terms built on f and new variables, that are still to unify with the effective arguments in lA.

We give the AC-unification algorithm as the ML definition of the function **uniAC** : $T \times T \rightarrow S$ *list* which returns as a list a finite CSU_{AC} of its arguments, empty if they are not unifiable :

```
letrec uniAC(M,N) =
    if isvar M then if equal(M,N) then [I]                          {case 1}
                    else if occurs(M,N) then nil                    {case 2}
                         else [M←N]                                 {case 3}
    else if isvar N then if occurs(N,M) then nil                    {case 4}
                         else [N←M]                                 {case 5}
    else if op M≠op N then nil                                      {case 6}
    else if isAC (op M) then                                        {case 7}
        let lA,llS = unilist (largAC M,largAC N,op M)
        in (flatten (map (λ lS . itlist2 unicompound lA lS [I]) llS))
    else (itlist2 unicompound (largs M) (largs N) [I])              {case 8}
and unicompound A B lσ = flatten (map continue lσ)
    where (continue σ = map compose (uniAC (σ A,σ B))
          where compose ρ = ρ o σ);;
```

3.3. A measure of complexity for AC-unification problems

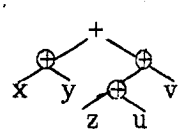
We give here a measure of complexity for AC-unification problems, which is at the basis of our proof of termination and completeness of uniAC. In order to define precisely this measure of complexity, we see terms as labeled trees, and we define *the set of occurrences of a term* M, $O(M)$, as the set of nodes of the tree, designed by integer lists [Huet and Oppen 80]. ε is the empty list, occurrences are denoted by u, v, w, ...

$$\begin{cases} O(M) = \{\varepsilon\} \text{ if M is a variable or a constant} \\ O(f(M_1,\ldots,M_n)) = \{\varepsilon\} \cup \{i.u \mid 1 \le i \le n \ \&\ u \in O(M_i)\} \end{cases}$$

We denote by M/u *the subterm of* M *at occurrence* u :

$$\begin{cases} M/\varepsilon = M \\ f(M_1,\ldots,M_n)/i.u = M_i/u \end{cases}$$

We are not interested in occurrences of a AC symbol inside a small homogeneous tree such as :



Accordingly, we say that an occurrence $u \in O(M)$ is *admissible* if and only if $u=\varepsilon$ or $op(M/u) \notin F_{AC}$ or $u=v.i \ i \in \mathbb{N}$ & $op(M/u) \neq op(M/v)$. In the above example, the circled occurrences of + are not admissible.

An occurrence is *strict* if it is not the occurrence of a variable. The set of strict and admissible occurrences of a term M is denoted by $\hat{O}(M)$. For example, let $M = (x+a)+(f(x)*b)$, $F_{AC} = \{+,*\}$ and $x \in V$. The occurrence 1 is not admissible, 1.1 and 2.1.1 are occurrences of variables, therefore $\hat{O}(M) = \{\varepsilon, 1.2, 2, 2.1, 2.2\}$. We denote by SUBT(M) *the set of the non-variable admissible subterms of* M : $SUBT(M) = \{M/u \mid u \in \hat{O}(M)\}$.

We want also to distinguish the variables having occurrences immediately under at least two different function symbols. *The set of immediate operators of a term* N *in a term* M, is the set $Op(N,M) = \{op\ M/u \mid u \in O(M) \ \&\ \exists i \in \mathbb{N}\ \ M/u.i = N\}$.

The complexity of AC*-unification of two terms* M *and* N, is a pair (ν,τ), denoted by $C_{AC}(M,N)$, where ν is the number of distinct variables in M and N,

having occurrences immediately under at least two different function symbols : ν = card{x∈V| card(Op(x,M)∪Op(x,N))>1}, and τ is the number of distinct non-variable admissible subterms in both M and N : τ = card(SUBT(M)∪SUBT(N)). That is τ is the number of non-variable nodes in the minimal graph of representation of M and N where AC symbols are represented as operators of variable arity.

For example, consider $+\in F_{AC}$, $f\notin F_{AC}$, M = (x+x)+f(y+(x+x)) and N = f(y). Only y has occurrences under + and f, and we count four admissible non-variable subterms. C_{AC}(M,N) = (1,4). If + was not AC then C_{AC}(M,N) would be equal to (1,5).

We shall use the lexicographic ordering on complexities :

$$(\nu,\tau) < (\nu',\tau') \text{ iff } \nu < \nu' \text{ or } \nu = \nu' \,\&\, \tau < \tau'$$

Lemma 1 < is a nœtherian ordering.

Proof : < is the lexicographic extension of two nœtherian orderings on terms. ■

Now we express that the complexity of two admissible subterms in two terms is strictly smaller.

Lemma 2 Let M,N∈T-V, u∈$\hat{O}$(M)-{ε}, v∈$\hat{O}$(N)-{ε}. We have C_{AC}(M/u,N/v)<C_{AC}(M,N).

Proof : Let $(\nu,\tau)=C_{AC}$(M/u,N/v) and $(\nu',\tau')=C_{AC}$(M,N). Obviously $\nu\leq\nu'$. Since u is in $\hat{O}$(M), SUBT(M/u)⊆SUBT(M). Similarly SUBT(N/v)⊆SUBT(N). Since u≠ε, M is not a subterm of M/u. Similarly N is not a subterm of N/v. Either M is not a subterm of N/v, in which case M∉SUBT(N/v), or N is not a subterm of M/u, in which case N∉SUBT(M/u). Therefore since M∈SUBT(M) and N∈SUBT(N) (they are not variables), we get $\tau<\tau'$. ■

Remark that in this lemma, the hypotheses on u and v are necessary. For example for M=N=(a+b)+c where $+\in F_{AC}$, C_{AC}(M,N)=(0,1), M/1=a+b, N/2=c and C_{AC}(M/1,N/2)=(0,2). But 1 is not an admissible occurrence in M.

The next lemma exhibits the elementary substitutions that are composed by uniAC as it will be shown in theorem 3, and proves that they do not increase the complexity of the initial terms.

Lemma 3 : Assume M and N are two terms, W is a finite set of variables, and σ is an elementary substitution of one of the four forms :
1) M←N (resp. N←M) if M (resp. N) is a variable not appearing in N (resp. M),
2) x←T where (Op(x,M)∪Op(x,N))∩(Op(T,M)∪Op(T,N)))≠ϕ and x∉V(T).
3) x←T where x∉V(M)∪V(N)∪W,
4) x←x_1+...+x_m+T_1+...+T_n where m+n≥1, $+\in F_{AC}$, +∈Op(x,M)∪Op(x,N), +∈Op(T_i,M)∪Op(T_i,N) , $x_i\notin V$(M)∪V(N)∪W and x∉V(T_i).
We have $C_{AC}(\sigma M,\sigma N)\leq C_{AC}$(M,N).

Proof :
Let $(\nu,\tau)=C_{AC}$(M,N) and $(\nu',\tau')=C_{AC}(\sigma M,\sigma N)$. In case 1 $\nu'=\nu$ and $\tau'=\tau$. In case 2 $\nu'\leq\nu$. Suppose $\nu'=\nu$ and T∉V. Since T is a subterm of M or N, the set of non-variable subterms of M and N is the same as the set of non-variable subterms of σM and σN. Furthermore, if some occurrence of T was admissible in M or N, then it will stay so after substitution; otherwise let +=op(T), all occurrences of T in M and N are immediately under +, the same for x because $\nu'=\nu$, thus all new occurrences of T in σM and σN are still not admissible. In all cases SUBT(σM)∪SUBT(σN) = SUBT(M)∪SUBT(N) and $\tau'=\tau$. Case 3 is trivial, $\nu'=\nu$, $\tau'=\tau$. In case 4 the first subcase is m=1, n=0, where trivially $C_{AC}(\sigma M,\sigma N)=C_{AC}$(M,N). Otherwise since $x_i\notin V$(M)∪V(N), we have Op(x_i,σM)∪Op(x_i,σN)={+} and since

$+\in Op(T_i,M)\cup Op(T_i,N)$, we have $\nu'\leq\nu$. Furthermore if $\nu'=\nu$, since $x\notin V(T_i)$, we must have $card(Op(x,M)\cup Op(x,N))=1$, thus $Op(x,M)\cup Op(x,N)=\{+\}$, and similarly to case 2 we conclude that $\tau'=\tau$.

■

Corollary : Assume W is a finite set of variables. For any terms M and N if $\sigma=\sigma_n \circ \cdots \circ \sigma_1$ is the composition of elementary substitutions satisfying the hypotheses of lemma 3 on M, N and W for σ_1, on $\sigma_1 M$, $\sigma_1 N$ and W for σ_2, ... , on $\sigma_{n-1} \cdots \sigma_1 M$, $\sigma_{n-1} \cdots \sigma_1 N$ and W for σ_n, then $C_{AC}(\sigma M,\sigma N)\leq C_{AC}(M,N)$.

Proof : By n applications of lemma 3.

■

The last lemma expresses some kind of stability by context of these properties. It is needed in the induction steps for the proof of termination.

Lemma 4 : Assume P and Q are two terms, M and N are two subterms such that $Op(M,P)\cap Op(N,Q)\neq\phi$, and W is a finite set of variables such that $V(P)\cup V(Q)\subseteq W\cup V(M)\cup V(N)$. If σ is a substitution satisfying the hypotheses of corollary of lemma 3 on M, N and W, then $C_{AC}(\sigma P,\sigma Q)\leq C_{AC}(P,Q)$.

Proof : We just have to prove that the hypotheses of lemma 3 on σ, M, N and W are equivalent to the hypotheses of lemma 3 on σ, P, Q and W. Case 1 with M and N is identical to case 2 with P and Q. Case 2 with M and N remains the same with P and Q since M and N are subterms of P and Q. Case 3 and case 4 are also unchanged because $V(P)\cup V(Q)\subseteq W\cup V(M)\cup V(N)$.

■

3.4. An example

Let M=x+(y+(z+f(x,y,z))) to be unified with N=u+(v+(w+f(u,v,w))) where $+\in F_{AC}$ and $f\notin F_{AC}$. $C_{AC}(M,N)=(6,4)$.

This example corresponds to the case 7 of the algorithm. M and N do not share arguments, lM = [x;y;z;f(x,y,z)], lN = [u;v;w;f(u,v,w)]. The function dio solves the associated diophantine equation, $u_1+u_2+u_3+u_4=v_1+v_2+v_3+v_4$, and returns the list lA of arguments with variables first :

x y z u v w f(x,y,z) f(u,v,w)

together with the matrix of solutions (with variable arguments columns first) :

1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0
1	0	0	0	0	0	0	1
0	1	0	1	0	0	0	0
0	1	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	0	0	1
0	0	1	1	0	0	0	0
0	0	1	0	1	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	0	0	1	0	1	0
0	0	0	0	0	1	1	0
0	0	0	0	0	0	1	1

and the number of variable arguments, here 6. The function partit returns the list of all the partitions of the 16 lines of the matrix that satisfy the constraint that only variable arguments are affected by a sum of coefficients greater than 1, and no column has a null sum. Let us consider such a particular partition by

keeping only lines 1,2,3,5,6,7,9,10,11 and 16. The traduction of this solution is the list
$[x_1+(x_2+x_3); x_4+(x_5+x_6); x_7+(x_8+x_9); x_1+(x_4+x_7); x_2+(x_5+x_8); x_3+(x_6+x_9); x_{10}; x_{10}]$, where $\{x_1,...,x_{10}\}$ are new variables attached to the 10 solutions of the partition. The terms of this list are then recursively unified with the terms in lA. First recursive calls are simple instanciations of x,y,z,u,v,w and x_{10}, but *the last recursive call relates to two terms of greater size than M and N, and containing simultaneously more variables and more distinct subterms*, that is : $f(x_1+(x_2+x_3), x_4+(x_5+x_6), x_7+(x_8+x_9))$ with $f(x_1+(x_4+x_7), x_2+(x_5+x_8), x_3+(x_6+x_9))$.

However the complexity is $(0,8) < C_{AC}(M,N)$. Unification goes on with $x_1+(x_2+x_3)$ and $x_1+(x_4+x_7)$ of complexity (0,2), and so on ...

3.5. Proof of total correctness

There is a notational difficulty in proofs of correctness, connected to the set W of protected variables. In the implementation W is implicit for a gensym like mechanism; this is what we expressed by "new variables" above. In the proofs below we must be more precise and give explicitly W as a third argument of uniAC.

Theorem 3 : Assume W is a finite set of variables. For any terms M and N uniAC(M,N,W) terminates.

Proof : The proof consists in showing that all recursive calls uniAC(M',N',W') in cases 7 and 8 of uniAC(M,N,W) are such that either $M' \in V$, or $N' \in V$, or $C_{AC}(M',N') < C_{AC}(M,N)$. In the first two cases, they terminate immediatly. For the third case, we have to show also that any σ in uniAC(M',N',W') satisfies the hypotheses of the corollary of lemma 3 on M, N and W, this by nœtherian induction on $C_{AC}(M,N)$.
In cases 3 and 5, σ is of the form 1 in lemma 3.
In case 8, let k be the arity of op(M), $\sigma_0=I$, $W_0=W$ and for $1 \le i \le k$, let $M_i=\sigma_{i-1} \cdots \sigma_1 M/i$, $N_i=\sigma_{i-1} \cdots \sigma_1 N/i$, $W_i=(W_{i-1} \cup V(\sigma_{i-1} \cdots \sigma_1 M) \cup V(\sigma_{i-1} \cdots \sigma_1 N)) - (V(M_i) \cup V(N_i))$ and σ_i be any unifier in uniAC(M_i,N_i,W_i). We prove by induction on i that $C_{AC}(\sigma_i \cdots \sigma_1 M, \sigma_i \cdots \sigma_1 N) \le C_{AC}(M,N)$.
By induction on i, $C_{AC}(\sigma_{i-1} \cdots \sigma_1 M, \sigma_{i-1} \cdots \sigma_1 N) \le C_{AC}(M,N)$. Now either $M_i \in V$, or $N_i \in V$, or by lemma 2 we get $C_{AC}(M_i,N_i) < C_{AC}(M,N)$.
Furthermore, by nœtherian induction σ_i satisfies the hypotheses of corollary of lemma 3 on M_i, N_i and W_i, thus also on $\sigma_{i-1} \cdots \sigma_1 M$, $\sigma_{i-1} \cdots \sigma_1 N$ and W_i by lemma 4, $C_{AC}(\sigma_i \cdots \sigma_1 M, \sigma_i \cdots \sigma_1 N) \le C_{AC}(M,N)$.
In case 7, let +=op(M), and lA,lIS=unilist (largAC M, largAC N, W). Assume $lA=[M_1;...;M_m;M_{m+1};...;M_n]$ and $lS=[X_1;...;X_n]$ is an element of lIS. $X_1,...,X_n$ are terms built on + and some new variables (not appearing in $V(M) \cup V(N) \cup W$). $M_1,...,M_n$ are variables and $M_{m+1},...,M_n$ are arguments not headed by +. The first n recursive calls are simple instanciations of the form 4 in lemma 3, so the complexity of M and N does not increase. The last recursive calls are performed on either one new variable and a term in lA, leading to a substitution of form 3 or 2, or either on two non-variable arguments of lA, namely $\sigma_{i-1} \cdots \sigma_1 M/j$ and $\sigma_{i-1} \cdots \sigma_1 N/k$, and the proof is the same as in case 8, or on one non-variable argument of lA and a term headed by +, thus leading to a failure.

■

Remark that if variable arguments were not instanciated first, next recursive calls could be done on two terms headed by +, and the proof of termination would be a little more complicated. For this, uniAC is nearer Livesey and Siekmann's algorithm [76,79] than Stickel's one [75,81]. They differ themselves only on this point.

Let U(M,N,W) be the finite set of substitutions returned by uniAC(M,N,W) and restricted to $V=\mathcal{V}(M)\cup\mathcal{V}(N)$,

Theorem 4 : Assume M and N are two terms, and W is a finite set of variables disjoint from $\mathcal{V}(M)\cup\mathcal{V}(N)$. U(M,N,W) is a CSU_{AC} of M and N away from W.

Proof : We show by nœtherian induction on $C_{AC}(M,N)$ that :
1) $\forall\sigma\in U(M,N,W)\ D(\sigma)\subseteq V$ and $R(\sigma)\cap W=\phi$
2) $U(M,N,W)\subseteq U_{AC}(M,N)$
3) $\forall\rho\in U_{AC}(M,N)\ \exists\sigma\in U(M,N,W)\ \sigma \underset{AC}{\overset{V}{\leq}} \rho$.

Cases 1,2,4,6 are trivial.
Cases 3,5 : If M (resp. N) is a variable having no occurrence in N (resp. M), then the substitution M←N (resp. N←M) satisfies the three properties.
Case 8 : Let k be the arity of op(M). We show that the set

$$S=\{\sigma_k o\ldots o\sigma_1{}_{|V} \mid \sigma_1\in U(M/1, N/1, W),\ \sigma_2\in U(\sigma_1 M/2, \sigma_1 N/2, W_2),\ \cdots,$$
$$\sigma_k\in U(\sigma_{k-1}\cdots\sigma_1 M/k, \sigma_{k-1}\cdots\sigma_1 N/k, W_k)\}$$

is a CSU_{AC} of M and N away from W, with $W_0=W$ and for $1\leq i\leq k$ $W_i=(W_{i-1}\cup\mathcal{V}(\sigma_{i-1}\cdots\sigma_1 M)\cup\mathcal{V}(\sigma_{i-1}\cdots\sigma_1 N))-(\mathcal{V}(\sigma_{i-1}\cdots\sigma_1 M/i)\cup\mathcal{V}(\sigma_{i-1}\cdots\sigma_1 N/i))$.
Notice that that for any i $W\subseteq W_i$.

1) $D(\sigma_k o\ldots o\sigma_1{}_{|V})\subseteq V$, $R(\sigma_k o\ldots o\sigma_1{}_{|V})\subseteq\bigcup_{i=1}^{k} R(\sigma_i)$, and by nœtherian induction $R(\sigma_i)\cap W_i=\phi$. Thus $R(\sigma_k o\ldots o\sigma_1)\cap W=\phi$, since $W\subseteq W_i$.
2) By nœtherian induction $\forall i\ 1\leq i\leq n\ \sigma_i\cdots\sigma_1 M/i \underset{AC}{=} \sigma_i\cdots\sigma_1 N/i$,
thus $\sigma_k\cdots\sigma_1 M/i \underset{AC}{=} \sigma_k\cdots\sigma_1 N/i$, i.e. $\sigma_k o\ \cdots\ o\ \sigma_1{}_{|V}$ is a AC-unifier of M and N.
3) Let $\rho\in U_{AC}(M,N)$, M and N being AC-unifiable. We show there exists $\sigma\in U(M,N,W)$ such that $\sigma\underset{AC}{\leq}\rho$, by nœtherian induction on k.
k=0 trivial since $U(M,N,W)=\{I\}$.
k>0 by nœtherian induction $\exists\lambda_{i-1}\in S\ \lambda_{i-1}o\sigma_{i-1}o\ldots o\sigma_1 \underset{AC}{\overset{V}{=}} \rho$.
But $\lambda_{i-1}\sigma_{i-1}\cdots\sigma_1 M/i \underset{AC}{=} \lambda_{i-1}\sigma_{i-1}\cdots\sigma_1 N/i$,
so by induction $\exists\sigma_i\in U(\sigma_{i-1}\cdots\sigma_1 M/i, \sigma_{i-1}\cdots\sigma_1 N/i, W_i)\ \sigma_i \underset{AC}{\overset{V_i}{\leq}} \lambda_{i-1}$,
with $V_i=\mathcal{V}(\sigma_{i-1}\cdots\sigma_1 M/i)\cup\mathcal{V}(\sigma_{i-1}\cdots\sigma_1 N/i)$.
$D(\sigma_i)\subseteq V_i$ so $\sigma_i \underset{AC}{\overset{V}{\leq}} \lambda_{i-1}$ and $\sigma_i o\ldots o\sigma_1 \underset{AC}{\leq} \rho$.
Therefore $\sigma_k o\ldots o\sigma_1 \underset{AC}{\leq} \rho$.

Case 7 : Let $lA=[M_1,\ldots,M_n]$ and $llS=[lS_1,\ldots,lS_p]$ be the solutions returned by unilist (largAC M, largAC N, W). By isomorphism with the solving of the equation $\sum_{i=1}^{m'} a_i x_i = \sum_{j=1}^{n'} b_j y_j$ over $\mathbb{N}-\{0\}$, we prove with the same inductions than in case 8, that the AC-unifiers of the effective arguments in lA with the terms in the solutions lS in llS form a $CSU_{AC}(M,N,W)$.

■

In general U(M,N,W) is not a μCSU_{AC} of M and N, but because it is finite, it suffices to eliminate the redundant unifiers (by AC-matching) in a final pass to get one.

3.6. Extension to identity and idempotence

In [Fages 83] we describe the extension to a unification algorithm in presence of operators that may be associative (A), commutative (C), idempotent (I) with unit (U), with the only restriction that an associative operator must be commutative.

Unification of terms built over only one function symbol ACU or ACUI, is studied in [Livesey and Siekmann 76,79]. These authors show that the case 7 in the algorithm is simplified since the presence of a unit permits to consider not all subsets of the solutions base, but only the *sum*, and idempotence allows to solve the associated equation in *0,1* rather than over integers.

However in the general case of unification in presence of operators U or I, case 6 is no longer a fail case. For example if 1 is the unit of f, f(x,y) and g(a,b) are unifiable with the CSU_U $\{\{x \leftarrow 1, y \leftarrow g(a,b)\}, \{y \leftarrow 1, x \leftarrow g(a,b)\}\}$. If f is also idempotent, $\{x \leftarrow g(a,b), y \leftarrow g(a,b)\}$ is another UI-unifier, and so on ... The cases 6 and 8 must be changed in this way.

Termination can be proved as for AC-unification, since the only introduced symbols are unit constants and new variables under the same function symbol. The combinatory explosion in the case 7 on the computation of partitions is eliminated when the AC function has a unit. The possibility of having more than one single most general U-unifier comes from the cases 8 and 6, which conversely introduces a new combinatory on the arguments, especially if both head functions are idempotent.

Conclusion

We have shown that varieties defined by a set of associative-commutative function symbols are finitary for the unification problem, by proving the termination and completeness of Stickel's algorithm. In the "formel" system, under development at INRIA, we use an implementation in Maclisp originally written by J.M. Hullot [79,80]. In this program the constrained generation of partitions, coded by the binary representation of bignums, allows to treat efficiently very large AC-unification problems.

In [Fages 83] we show that the sharing of common subterms is possible by representing terms by ordered graphs, and we extend the algorithm to unify finite and infinite rational terms modulo associativity-commutativity, identity and idempotence. In presence of Abelian group operators, the existence of a unification procedure extending the one of Lankford [83] which would terminate in the general case is an open problem.

Acknowledgements

I wish to thank Gérard Huet for his decisive help in the presentation, Guy Cousineau and Thierry Coquand for their comments.

References

Baxter L.D. The Complexity of Unification. *Ph.D. Thesis, University of Waterloo.* (1977)

Baxter L.D. The Undecidability of the Third Order Dyadic Unification Problem. *Information and Control 38, p170-178.* (1978)

Burstall R.M., Collins J.S. and Popplestone R.J. Programming in POP-2 *Edinburgh University Press.* (1971)

Colmerauer A. Prolog II, manuel de référence et modèle théorique. *Rapport interne, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II.* (Mars 1982)

Colmerauer A., Kanoui H. and Van Caneghem M. Etude et Réalisation d'un

système PROLOG. *Rapport Interne, GIA, Univ. d'Aix-Marseille Luminy.* (1972)
Davis M. Hilbert's Tenth Problem is Unsolvable. *Amer. Math. Monthly 80,3, pp. 233-269.* (1973)
Fages F. Formes canoniques dans les algèbres booléennes, et application à la démonstration automatique en logique de premier ordre. *Thèse, Université de Paris VI.* (June 83)
Fages F. Note sur l'unification des termes de premier ordre finis et infinis. *Rapport LITP 83-29.* (May 1983)
Fages F. and Huet G. Unification and Matching in Equational Theories. *CAAP 83, l'Aquila, Italy. Lecture Notes in Computer Science 159.* (March 1983)
Fay M. First-order Unification in an Equational Theory. *4th Workshop on Automated Deduction, Austin, Texas, pp. 161-167.* (Feb. 1979)
Goldfarb W.D. The Undecidability of the Second-order Unification Problem. *Theoritical Computer Science, vol. 13, pp. 225-230. North Holland Publishing Company.* (1981)
Gordon M.J., Milner A.J. and Wadsworth C.P. Edinburgh LCF. *Springer-Verlag LNCS 78.* (1979)
Gould W.E. A Matching Procedure for Omega Order Logic. *Scientific Report 1, AFCRL 66-781, contract AF19 (628)-3250.* (1966)
Guard J.R. Automated Logic for Semi-Automated Mathematics. *Scientific Report 1, AFCRL 64,411, Contract AF19 (628)-3250.*(1964)
Herbrand J. Recherches sur la théorie de la démonstration. *Thèse, U. de Paris, In: Ecrits logiques de Jacques Herbrand, PUF Paris 1968.* (1930)
Hsiang J. Topics in Automated Theorem Proving and Program Generation. *Ph.D. Thesis, Univ. of Illinois at Urbana-Champaign.* (Nov. 1982)
Huet G. The Undecidability of Unification in Third Order Logic. *Information and Control 22, pp. 257-267.* (1973)
Huet G. A Unification Algorithm for Typed λ-Calculus. *Theoretical Computer Science 1,1, pp. 27-57.* (1975)
Huet G. Résolution d'équations dans des langages d'ordre 1,2, ... omega. *Thèse d'Etat, Université de Paris VII.* (1976)
Huet G. An Algorithm to Generate the Basis of Solutions to Homogenous Linear Diophantine Equations. *Information Processing Letters 7,3, pp. 144-147.)* (1978)
Huet G. and Oppen D. Equations and Rewrite Rules: a Survey. *In Formal Languages: Perspectives and Open Problems, Ed. Book R., Academic Press.* (1980)
Hullot J.M. Associative-Commutative Pattern Matching. *Fifth International Joint Conference on Artificial Intelligence, Tokyo.* (1979)
Hullot J.M. Compilation de Formes Canoniques dans les Théories Equationnelles. *Thèse de 3ème cycle, U. de Paris Sud.* (Nov. 80)
Jensen D. and Pietrzykowski T. Mecanizing ω-Order Type Theory through Unification. *Theoretical Computer Science 3, pp. 123-171.* (1977)
Jouannaud J.P. and Kirchner C. Incremental Construction of Unification Algorithms in Equational Theories. *Proc. 10th ICALP.* (1983)
Kirchner H. and Kirchner C. Contribution à la résolution d'équations dans les algèbres libres et les variétés équationnelles d'algèbres. *Thèse de 3ème cycle, Université de Nancy.* (Mars 1982)
Knuth D. and Bendix P. Simple Word Problems in Universal Algebras. *In Computational Problems in Abstract Algebra, Pergamon Press, pp. 263-297.* (1970)
Landin P.J. The Next 700 Programming Languages. *CACM 9,3.* (March 1966)
Lankford D.S. A Unification Algorithm for Abelian Group Theory. *Report MTP-1, Math. Dept., Louisiana Tech. U.* (Jan. 1979)
Lankford D.S., Butler G. and Brady B. Abelian Group Unification Algorithms for

Elementary Terms. *Math. Dept., Louisiana Tech. U., Ruston Louisiana 71272* (1983)

Livesey M. and Siekmann J. Unification of Bags and Sets. *Internal Report 3/76, Institut fur Informatik I, U. Karlsruhe.* (1976)

Livesey M., Siekmann J., Szabo P. and Unvericht E. Unification Problems for Combinations of Associativity, Commutativity, Distributivity and Idempotence Axioms. *4th Workshop on Automated Deduction, Austin, Texas, pp. 161-167.* (Feb. 1979)

Makanin G.S. The Problem of Solvability of Equations in a Free Semigroup. *Akad. Nauk. SSSR, TOM pp. 233,2.* (1977)

Martelli A. and Montanari U. An Efficient Unification Algorithm. *ACM T.O.P.L.A.S., Vol. 4, No. 2, pp 258-282.* (April 1982)

Matiyasevich Y. Diophantine Representation of Recursively Enumerable Predicates. *Proceedings of the Second Scandinavian Logic Symposium, North-Holland.* (1970)

Paterson M.S. and Wegman M.N. Linear Unification. *J. of Computer and Systems Sciences 16, pp. 158-167.* (1978)

Peterson G.E. and Stickel M.E. Complete Sets of Reduction for Equational Theories with Complete Unification Algorithms. *JACM 28,2 pp 233-264.* (1981)

Plotkin G. Building-in Equational Theories. *Machine Intelligence 7, pp. 73-90.* (1972)

Raulefs P., Siekmann J., Szabo P., Unvericht E. A Short Survey on the State of the Art in Matching and Unification Problems. *Sigsam Bulletin 1979, Vol. 13.* (1979)

Reynolds John C. GEDANKEN - A Simple Typeless Language Based on the Principle of Completeness and the Reference Concept. *CACM 13,5, pp. 308-319.* (May 1970)

Robinson J.A. A Machine Oriented Logic Based on the Resolution Principle. *JACM 12, pp. 32-41.* (1965)

Robinson G.A. and Wos L.T. Paramodulation and Theorem Proving in First-order Theories with Equality. *Machine Intelligence 4, American Elsevier, pp. 135-150.* (1969)

Rulifson J.F., Derksen J.A. and Waldinger R.J. QA4 : a Procedural Calculus for Intuitive Reasoning. *Technical Note 73, A.I. Center, SRI, Menlo Park.* (Nov. 1972)

Siekmann J. Unification and Matching Problems. *Ph. D. thesis, Memo CSM-4-78, University of Essex,* (1978)

Siekmann J. and Szabo P. Universal Unification in Regular Equational ACFM Theories. *CADE 6th, New-York.* (June 1982)

Slagle J.R. Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity. *JACM 21, pp. 622-642.* (1974)

Stickel M.E. A Complete Unification Algorithm for Associative-Commutative functions. *4th International Joint Conference on Artificial Intelligence, Tbilisi.* (1975)

Stickel M.E. Unification Algorithms for Artificial Intelligence Languages. *Ph. D. thesis, Carnegie-Mellon University.* (1976)

Stickel M.E. A Complete Unification Algorithm for Associative-Commutative Functions. *JACM 28,3 pp 423-434.* (1981)

Venturini-Zilli M. Complexity of the Unification Algorithm for First-Order Expressions. *Calcolo XII, Fasc. IV, p361-372.* (October December 1975)

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique