SEKI-PROJEKT

SEKI MEMO

Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
D-6750 Kaiserslautern
West Germany

Universität Karlsruhe
Institut für Informatik I
Postfach 6380
D-7500 Karlsruhe
West Germany

UNIFICATION OF IDEMPOTENT FUNCTIONS

P. Raulefs, J. Siekmann

MEMO SEKI-78-I

UNIFICATION OF IDEMPOTENT FUNCTIONS

P. Raulefs, J. Siekmann

Institut für Informatik I

Universität Karlsruhe

D-7500 Karlsruhe 1

West Germany

Abstract

A complete algorithm for terms involving idempotent and idempotent-commutative functions is presented. The main results are: The unification problem for both cases is decidable and the set of unifiers is finite for both problems.

Keywords and phrases

Automatic theorem proving, matching algorithms, T-unification, idempotence.

1. *Introduction*

For almost as long as attempts at proving theorems by machines have been made, a critical problem has been well known [1], [2], [11]: Certain equational axioms, if left without precautions in the data base of an automatic theorem prover (ATP), will force the ATP to go astray. In 1967, Robinson [15] proposed that substantial progress ("a new plateau") would be achieved by removing these troublesome axioms from the data base and building them into the deductive machinery.

Four approaches to cope with equational axioms have been proposed:

(1) To write the axioms into the data base, and use an additional rule of inference, such as paramodulation [16].

(2) To use special "rewrite rules" [6], [7], [28], [29].

(3) To design special inference rules incorporating these axioms [21].

(4) To develop special unification algorithms incorporating these axioms [12].

At least for equational axioms, the last approach (4) appears to be most promising, however it has the drawback that for every new set of axioms a new unification algorithm has to be found.

The theoretical basis for utilizing unification algorithms incorporating equational axioms has been develop by G. Plotkin [12]. Plotkin has shown that whenever an ATP is to be refutation complete, its unification procedure must satisfy conditions given as follows: Assume T is the theory (set of axioms) considered, and t_1, t_2 are terms to be unified; then, the following properties should hold for the set Ψ of unifiers of t_1 and t_2:

1. Ψ is correct, i.e. for every $\sigma \varepsilon \Psi$, $\sigma t_1 \overset{T}{=} \sigma t_2$ ($\overset{T}{=}$ denotes equality with respect to the theory T).
2. Ψ is complete, i.e. for any substitution δ with $\delta t_1 \overset{T}{=} \delta t_2$ there is some $\sigma \varepsilon \Psi$ s. t. there exists a substitution λ so that $\delta \overset{T}{=} \lambda \circ \sigma$.
3. Ψ is minimal, i.e. no unifier σ_1 in Ψ is an instance of some other unifier σ_2 in Ψ.

Minimality is a property that has often been overlooked in the literature:

If minimality is completely ignored we arrive at simply enumerating all substitutions and removing all that do not unify as an algorithm satisfying our requirements. Generating such a set of all unifiers, instead of a set of *most general* unifiers, essentially amounts to proving a theorem by the 'British Museum Algorithm' (i.e. by enumerating all Herbrand instances). Such procedures are called conservative in [16] and are distinctively different from proofs by the resolution principle at the 'lifted' most general level. However, minimality is more difficult to achieve than correctness and completeness.

Looking at unification of terms in first-order predicate calculus with an equational theory T, unification problems may be classified with respect to the cardinality of minimal and complete sets Ψ of unifiers:

(i) Ψ may always be a singleton: e.g. for $T = \emptyset$, that is for ordinary first order unification as in [1],[6],[14]. Another (trivial) problem in this class is the string matching problem for constant strings only, as encountered in string manipulation languages such as SNOBOL [31]. The nontrivial aspect of this problem is to find *efficient* algorithms [32],[33]. Another example is unification under homomorphism, isomorphism and automorphism [30].

(ii) Ψ may have more than one element but at most finitely many: examples are the theory of idempotence as well as idempotence plus commutativity presented in this paper. Other examples are unification under commutativity [19]; unification under associativity and commutativity [22], [9]; unification under associativity, commutativity and idempotence [9] and the one way unification problem for strings.

(iii) Ψ may sometimes be an infinite set: examples are unification under associativity [12], [19], [8].

This problem is equivalent to the problem of solving a set of equations over a free semigroup (the monoid problem) [23], [27], the decidability of which is an open problem now for over twentyfive years. Other problems in this class are unification under distributivity [24] and the unification problem for second order monadic logic [4], [5], [25].

(iv) Ψ may sometimes not exist at all, e.g. for unification in ω-order predicate calculus. In such cases there exist infinite chains of unifiers (ordered by increasing generality)

$$\delta_1 \sqsubseteq \delta_2 \sqsubseteq \delta_3 \sqsubseteq \cdots$$

with no upper bound [5], [26].

For unification problems where complete sets of unifiers are always finite, it is not necessarily important that the unification procedure returns a minimal set of unifiers, since dependent unifiers can always be checked off. In this case, minimality of the unification procedure comes down to be a matter of computational efficiency.

Unification problems have significance beyond automatic theorem proving: certain axioms define structures which closely resemble familiar datastructure (e.g. strings, bags, sets etc.), and most AI-languages have pattern matching algorithms for these cases built into their deductive machinery (see e.g. [3], [17], [18]).

Apart from the fact that these matching algorithms have without exception been designed ad hoc, i.e. without respect to completeness, minimality or sometimes even correctness, the basic question of whether a particular matching problem for a particular data structure is decidable has not been answered.

A little reflection will show that for very rich matching structures, as it has e.g. been proposed in MATCHLESS in PLANNER [3], the matching problem is undecidable. This presents a problem for the designer of such languages: on the one hand, very rich and expressive matching structures are desirable, since they form the basis for the invocation- and deduction mechanism. On the other hand, drastic restrictions will be necessary if matching algorithms are to be found. The question is just how severe do these restrictions have to be.

In this paper, algorithms are presented for theories of idempotence and for idempotence together with commutativity. Idempotent functions appear in group theory, practical examples being proofs about substitutions which are idempotent if they are in normal form (i)

The main results are that the unification problem for idempotence is decidable and Ψ_I is finite, but not a singleton in general. We did not concern ourselves with efficiency (in space and time) of the algorithm. We do believe that the tech-

---

(i) See e.g. the proof of Lemma 4.2.2. in this paper. A historical example is how Luckhams program verifier found the correctness of Robinson's unification algorithm only after the idempotence for unifiers had been added.

nique of [13] allowing for linear unification of first order predicate calculus terms, could be employed here as well.

Another interesting problem concerns the relation between these unification algorithms and the corresponding rewrite rules [7], which is however outside of the scope of this paper.

The following chart provides a quick survey of unification problems as investigated so far:

Axioms	Problem decidable?	complete Algorithm exists?	ψ, the set of mgu's is:	ψ is minimal	investigators
A	decidable	yes	infinite	yes	[12.,19.,23.,34.]
C	decidable	yes	finite	no	[20.]
I	decidable	yes	finite	no	(this paper)
A+C	decidable	yes	finite	yes	[9.]
A+I	?	?	?	?	
C+I	decidable	yes	finite	no	(this paper)
A+C+I	decidable	yes	finite	yes	[9.]
D	?	yes	infinite	?	[24.]
D+A	undecidable	?	infinite	?	[35.]
H	decidable	yes	singleton	yes	[30.]
H+A	decidable	yes	singleton	yes	[30.]
H+A+C	?	yes	infinite	?	[30.]
D+A+C	undecidable	?	infinite	?	[35.]
ω-order terms	$\omega \geq 3$ undec.	yes	ψ does not exist	no	[5.]
first order terms	decidable	yes	singleton	yes	[1.6.14.]

The axioms are:		
	A (associativity)	$f(f(x,y),z) = f(x,f(y,z))$
	C (commutativity)	$f(x,y) = f(y,x)$
	I (Idempotence)	$f(x,x) = x$
	D (distributivity)	$f(x,g(y,z)) = g(f(x,y),f(x,z))$
	H (homomorphism)	$\varphi(x \bullet y) = \varphi(x) \circ \varphi(y)$

2. *Basic Concepts and Terminology*

2.1 *Terms*

We are concerned with unifying terms of first order logic containing idempotent binary functions. Since terms starting with different function symbols cannot be unified at all, we restrict our investigation to terms involving one function symbol only. To simplify our notation, we ignore function symbols and write (s,t) instead of f(s,t).

In the following let CONST = $\{a,b,c,a_1,b_1,c_1,\ldots\}$ be an arbitrary decidable set of *constant* symbols, and VAR = $\{x,y,z,x_1,y_1,\ldots\}$ be an infinite, decidable set of *variable* symbols s.t. CONST and VAR are disjoint. Constant and variable symbols are called *atoms* forming the set AT = CONST $\cup$ VAR.

The set TERM of all terms is the least set s.t.

(1) AT $\subset$ TERM and (2) if s, t $\in$ TERM then (s,t) $\in$ TERM.

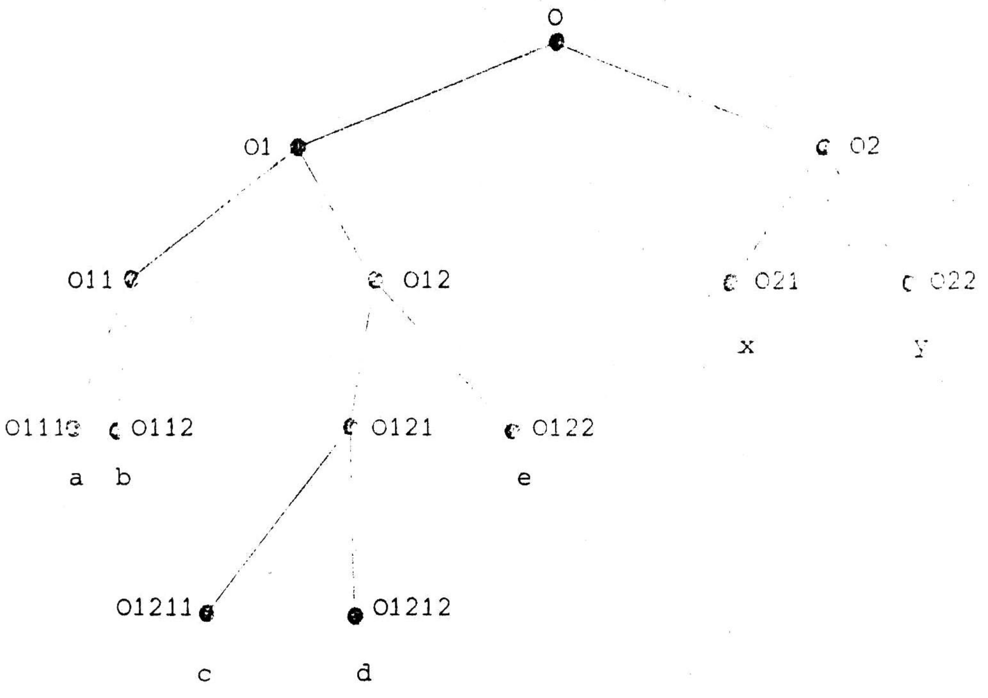
If an atom u occurs in a term t, we write u $\in$ t.

2.1.2. *Tree representation*

It is convenient to represent terms as labelled binary trees, using Dewey decimal labelling for marking nodes: the root node is labelled 0, and the left and right sons of a node labelled k are marked k1 resp. k2. For example, the labelled binary tree representation of the term

$$s = (((a,b),((c,d),e)),(x,y))$$

is:



We do not distinguish between terms and their tree representation.

2.1.3. Def.: If k marks a node in a term t then

sub(t,k)	is the largest subtree of t with root k;
lsub(t,k)	is the largest subtree in sub(t,k) with root k1;
rsub(t,k)	is the largest subtree in sub(t,k) with root k2.

Example: For the above term s:

sub(s,O2) = (x,y), lsub(s,O1) = (a,b), rsub(s,O1) = ((c,d),e).

2.1.4. *Normal forms*

A term t is in *normal form* iff there is no label k s.t. lsub(t,k) and rsub(t,k) are identical terms.

For example, ((a,a),b) is not in normal form, but (a,b) is. So, in a normal form term there are no subterms that can be collapsed into a single term because of idempotence. Obviously, each term has a unique normal form. NF[t] denotes the normal form of a term t. For example, NF[(((a,a),b),(a,b))] = (a,b).

2.2. *Substitutions*

A *substitution* is a finite set $\{(x_1 \leftarrow t_1),\dots,(x_n \leftarrow t_n)\}$ of substitution components $(x_i \leftarrow t_i)$ with $x_i \in$ VAR, $t_i \in$ TERM $(1 \leq i \leq n)$ s.t. x_k and x_j are different variable symbols for $k \neq j$.

A substitution $\sigma = \{(x_1 \leftarrow t_1),\dots,(x_n \leftarrow t_n)\}$ is applied to a term t by consistently replacing each occurence of x_i in t by t_i $(1 \leq i \leq n)$, and σt denotes the resulting term.

Any two substitutions σ and τ can be composed to the substitution $\sigma \circ \tau$ so that for any term t, $(\sigma \circ \tau)t \equiv \sigma(\tau t)$, where "$\equiv$" denotes symbolwise identity between terms.

For any two substitutions σ and τ,

a. $\sigma \sqsubseteq \tau$ iff there is a substitution λ s.t. $\forall t \in$ TERM.

   $\sigma t \equiv (\lambda \circ \tau)t$

b. $\sigma \sim \tau$ iff $\sigma \sqsubseteq \tau$ and $\tau \sqsubseteq \sigma$.

Clearly, $\sqsubseteq$ is a partial order so that $\sim$ is an equivalence relation on substitutions.

2.2.1. *Normal form substitutions*

A substitution $\sigma = \{(x_1 \leftarrow t_1), \dots, (x_n \leftarrow t_n)\}$ is in *normal form* iff $x_i \notin t_j (1 \leq i,j \leq n)$, i.e. no variable to be replaced occurs in any of the terms to be substituted. It can be shown that any substitution σ can be transformed into a substitution σ' in normal form s.t. $\sigma \sim \sigma'$ [8].

2.3. *Unifiers*

A substitution σ is a *unifier* of two terms s and t iff $\sigma t = \sigma s$, i.e. σ makes s and t equal. It can be shown [8] that for any unifiable set of terms there is a unifier in normal form. σ is a *most general unifier (mgu)* of terms s and t iff for any other unifier δ of s and t

$$\delta \sqsubseteq \sigma .$$

2.3.1. *R- and I-unifiers*

Two terms s and t are equal with respect to idempotence iff $NF(s) \equiv NF(t)$. In this case, we write $s \overset{I}{=} t$ or simply $s = t$.

For any two terms s and t, a substitution σ is called

(a) *I-unifier* iff $\sigma s \overset{I}{=} \sigma t$, and

(b) *R-unifier* iff $\sigma s \equiv \sigma t$.

R-unifiers are generated by Robinson's unification algorithm [14].

2.3.2. *Unification problems*

Given an equational theory T, the problem of T-unifying two first order terms s and t is denoted as $<T,<s,t>>$.

In this paper, we consider two theories:

a. $T_I = \{(x,x) = x\}$ containing the law of idempotence only;

b. $T_{CI} = \{(x,x) = x, (x,y) = (y,x)\}$ containing both the laws of idempotence and commutativity.

When the theory T is understood from the context, we simply write $<s,t>$ for a T-unification problem.

2.4. *Notation*

We specify algorithms in terms of the familiar λ-notation, sugared with *let* ... *in* clauses to declare abbreviations. If $\lambda x.h$ specifies a function, fix $x.h$ denotes its least fixpoint. Least fixpoints are convenient to specify and prove properties about recursive algorithms. For any tuple $x = (x_1,...,x_n)$, $x{\downarrow}i$ denotes the i-th component of x.

3. *Algorithm for I-Unification*

3.1. *Intuitive overview*

To obtain a feeling for the problem, let us first consider two examples of unifying two terms with respect to the theory T_I (idempotence only).

3.1.1. Example: $<x,(x,a)>$

x and (x,a) are clearly not unifiable in the sense of Robinson [14], as the variable x of the first term also occurs in the second term. Under idempotence, however, both terms are unifiable with mgu $\sigma = \{x \leftarrow a\}$, since $\sigma x \overset{I}{=} \sigma(x,a)$.

In Example 3.1.1. the set of all mgu's is a singleton. The next example shows that this is not always so.

3.1.2. Example

$$<(x,y),(a,b)> .$$

This unification problem has two mgu's:

$\sigma_1 = \{x \leftarrow a, y \leftarrow b\}$, and the independent, less obvious mgu

$\sigma_2 = \{x \leftarrow (a,b), y \leftarrow (a,b)\}$.

Example 3.1.2. leaves open the possibility of having an infinite number of mgu's for some I-unification problem. We will show that this is not so, but this fact is not obvious at all.

3.1.3. Intuitive idea of the algorithm

Example 3.1.2. already suggest a possible unification algorithm: Consider the idempotent expansions such as, for the term (a,b):

$$\{(a,b) , ((a,a),b) , (a,(b,b)) , ((a,a),(b,b)) , ((a,b),(a,b)) ,\ldots\} .$$

First, generate the two sets of all idempotent expansions of both terms, and then unify the Cartesian product of both sets with respect to R-unification. The algorithm presented in this paper is, in a sense, dual to this procedure, although it is much more efficient by generating less subproblems.

Our algorithm unifying two terms <s,t> is split up into two interlocking parts: the *collapsing phase* and the *R-unification phase*.

(1) In the collapsing phase, we look for subterms (r_1,r_2) in s so that r_1 and r_2 can be R-unified by some substitution ρ to r. Then, applying ρ to s causes each occurrence of the subterm (r_1,r_2) to "collapse" to r.

Example: s = (a,((x,b),y)) can be collapsed in three ways:

1. $\rho_1 = \{x \leftarrow b\}$ yields $\rho_1 s = (a,((b,b),y)) \overset{I}{=} (a,(b,y))$;

2. $\rho_2 = \{y \leftarrow (x,b)\}$ yields $\rho_2 s = (a,((x,b),(x,b))) \overset{I}{=} (a,(x,b))$

3. $\rho_3 = \{(x \leftarrow b)(y \leftarrow b)\}$ yields $\rho_3 = (a,((b,b),b)) \overset{I}{=} (a,b)$

The collapsing phase generates the two sets of all possible collapses of s and t. The Cartesian product of these two sets is a finite set of pairs of terms, each constituting a new unification problem to be handed over to the R-unification phase.

(2) In the R-unification phase, all unification problems resulting from the collapsing phase are solved by the algorithm RUNIFY. Essentially, RUNIFY follows the idea of Robinson's unification algorithm except for the way an atom and a nonatomic term is unified. RUNIFY returns a success/failure message (SUCC/FALL) and a substitution (empty upon failure).

3.2. Collapsing phase

In the collapsing phase we determine the set of all "collapses" that can be constructed from two terms to be unified initially. Any such collapse is obtained from collapsing individual nodes.

3.2.1. Collapsing individual nodes

The following definition makes use of the algorithm RUNIFY to be presented below. RUNIFY is applied to two terms as arguments and tries to unify them in a certain sense, returning a pair (m,σ). m is the success/failure message SUCC resp. FAIL, depending on whether RUNIFY could successfully unify its argument terms; σ is the unifying substitution resp. $\emptyset$ for $m = \text{FAIL}$.

3.2.1. Def.: The *collapse of a node k in a term t* is defined by coll(t,k) := *let* (m,σ) := RUNIFY[lsub(t,k), rsub(t,k)]
in if m = SUCC *then* $(\sigma[\text{lsub}(t,k)],\sigma)$
else $(\text{sub}(t,k),\emptyset)$

The collapse of a node in a term is the pair consisting of the term obtained after having collapsed, and the associated substitution as generated from RUNIFY.

Example: For t = (((a,b),((c,d),e)),(x,y)) (tree representation in section 2.1.2.),we obtain:

coll (t,O2) = [(((a,b),((c,d),e)),(y,y)),{$x\leftarrow y$}]
(since x and y trivially unify by {$x\leftarrow y$}).

coll (t,O1) = [t,$\emptyset$]
(since (a,b) and ((c,d),e) cannot be unified)

coll (t,O) = [(((a,b),((c,d),e)),((a,b),((c,d),e))),
{$x\leftarrow$(a,b),$y\leftarrow$((c,d)e)}]

Note that the normal form of the above terms shows the intuitive reason for the 'collapsing': e.g.

NF(coll (t,02)) = [(((a,b),((c,d),e)),y), {x←y}]

NF(coll (t,0)) = [((a,b),((c,d),e)), {x←(a,b),y←((c,d),e)}]

For technical reasons (in Def. 3.2.2. below) we did not include the normal form already in this definition but delayed the actual "collapsing" until last.

3.2.2. Collapsing terms

The intention of the following definition is to obtain all possible collapses of a term t. This is achieved by determining for each set $L = \{k_1,...,k_n\}$ of individual nodes the collapse of k_n, in the collapse of k_{n-1},..., in the collapse of k_1 in t.

Let labels(t) be the set of all labels marking nodes in t.

Example: For t = (((a,b),((c,d),e)),(x,y)) as in the above example, labels(t) = {0,01,02,011,012,021,022,0111,0112, 0121,0122,01211,01212}.

3.2.2. Def.: The *collapse of a term t* is the set collapse(t) of terms obtained as follows:

(1) For any subset $L \subseteq$ labels(t) with $L = \{k_1,...,k_n\}$, let

σ_1 := coll(t,k_1)↓2 and t_1 := coll(t,k_1)↓1;

σ_2 := coll(t_1,k_2)↓2 and t_2 := coll(t_1,k_2)↓1;

.

.

.

σ_n := coll(t_{n-1},k_n)↓2 and t_n := coll(t_{n-1},k_n)↓1.

Let

$t_L := NF(t_n)$, and

$\sigma_L :=$ *if* σ_i = NIL for some i,

$1 \leq i \leq n$ *then* NIL *else* $\sigma_1 \bullet \sigma_2 \bullet \ldots \bullet \sigma_n$.

(2) collapse(t) := $\{(t_L, \sigma_L) \mid L \subseteq \text{labels}(t)\}$.

That is, collapse(t) returns a set of pairs with first elements being terms and second elements substitutions.

3.3. R-Unification phase

R-unification follows the operations and flow of control of Robinson's unification algorithm [14] until atomic terms are encountered. The algorithm UNIAT unifies pairs of terms where at least one term is an atom. UNIAT is called from the R-unification algorithm RUNIFY.

3.3.1. Notation

1. For any domain D, and variable x, x:D indicates that x takes values from D.
2. For any term t,

   const(t) := $\{c \in \text{CONST} \cap t\}$ denotes the set off all constants occurring in t;

   var(t) := $\{x \in \text{VAR} \cap t\}$ denotes the set of all variables occurring in t.
3. For any term t and $k \in \mathbb{N}^*$, brother(t,k) denotes the subterm in t having the brother node of k in t as its root node; if no such node exists for t, brother(t,k) is the empty term.
4. AT, TERM, and SUBST denote the domains of atoms, terms and

substitutions.

5. disagree(s,t) denotes the disagreement set of s and t (see [14]).

3.3.2. Unifying atoms and terms

When applied to an atom u and a term t, UNIAT returns a pair consisting of a success/failure message and a substitution.

```
UNIAT := λ u:AT t:TERM.
case u (1) u ∈ CONST : if {u} ∪ const(t) ≠ {u} then (FAIL,∅)
                       else (SUCC,{x←u|x ∈ var(t)})

          {The atom u is a constant. If t contains a constant
           other than u, then u and t are not unifiable; other-
           wise, every variable in t is substituted with u
           (see Examples 3.3.2.1,2 below)}

       (2) u ∈ VAR:
          case u
          (2.1) u ∉ t: (SUCC,{u←t})
          (2.2) u ∈ t: let l ∈ {j|u ∉ sub(t,j)
                                and brother(t,j) ≡ u} in
                       let σ := {u←sub(t,l)} in
                       if NF[sub(t,l)] ≡ NF[σt] then (SUCC,σ)
                                                else (FAIL,∅)

          {The atom u is a variable. If u is not occurring in
           t, the substitution {u←t} immediately substitutes
           t for u without changing t (see Example 3.3.2.3).
           However, if u is a variable that does occur in t,
```

1. *Look for a subtree q in t s.t. u does not occur in q but the brother of q in t is an occurrence of u.*
2. *Let σ := {u←q}; if the normal form of σt and the normal form of q are identical, then σ unifies u and t; otherwise, u and t are not unifiable (see Examples 3.3.2.4,5).}*

The following examples illustrate how UNIAT works and do also exhibit some particulars of unification under idempotence.

3.3.2.1. Example: UNIAT [c,((c,d),x)]
Since c is a constant, case (1) applies. As const[((c,d),x)] = {c,d} ≠ {c}, UNIAT returns (FAIL,∅).

3.3.2.2. Example: UNIAT [c,((c,x),y)]
Again c is a constant and we are in case (1). Now const[((c,y),y)] = {c}, hence UNIAT returns (SUCC,{x←c,y←c}).

3.3.2.3. Example: UNIAT [x,(a,b)]
This is the trivial case (2.1) with UNIAT returning (SUCC,{x←(a,b)}).

3.3.2.4. Example: UNIAT [x,(((a,b),x),x)]
This is case (2.2) with the variable x occurring in the term. (a,b) is the largest subterm with brother x but not containing x itself, so the substitution σ = {x←(a,b)} is generated.
σx ≡ NF[σ(((a,b),x),x)] ≡ (a,b) UNIAT terminates with (SUCC,σ).

2.3.2.5. Examples: UNIAT [x,(((a,b),x),c)]

Again case (2.1) applies as in the previous example, and the same substitution $\sigma = \{x \leftarrow (a,b)\}$ is generated. But $\sigma x \overset{I}{\neq} NF[\sigma(((a,b),x),c)] \equiv ((a,b),c)$ so that UNIAT returns (FAIL, $\emptyset$).

2.3.3. The algorithm RUNIFY

```
RUNIFY := {fix RU.$\lambda\sigma$:SUBST. $\lambda$s:TERM t:TERM.
          if disagree(s,t) = $\emptyset$
             then (SUCC,$\emptyset$)
             else let $(r_1,r_2)$ := disagree(s,t) in
                  let $r_i \in$ AT and $i \neq j$ for $i,j \in \{1,2\}$  in
                  {at least one of $r_1$ and $r_2$ is an atom, say $r_i$}
                  let $(m,\sigma_1)$ := UNIAT $r_i r_j$  in
                  if m = SUCC
                     then let $\sigma_2 := \sigma_1 \circ \sigma$  in
                          if $\sigma_2 s \overset{I}{=} \sigma_2 t$  then (SUCC,$\sigma_2$)
                          else RU[$\sigma_2$,$\sigma_2$s,$\sigma_2$t]
                     else (FAIL,$\emptyset$)
          } $\emptyset$
```

RUNIFY is a recursive algorithm unifying two arbitrary terms s and t. Recursion is specified in terms of the familiar fix-point notation that abstracts from particular implementations. RUNIFY is the least function constructing a unifier from the empty substitution $\emptyset$ and s and t in the following way:

First, we check the disagreement set of s and t; if it is empty, we are done. From the disagreement set $\{r_1, r_2\}$ we pick an atom r_i and apply UNIAT to r_i and r_j ($j \neq i$, i.e. the other member of $\{r_1, r_2\}$). If UNIAT $\{r_1, r_2\}$ successfully returns a substitution σ_1, σ_1 is appended to the substitution having been worked out so far (initially $\emptyset$), resulting in a substitution σ_2. If σ_2 already does the job, we are done; otherwise, the whole procedure is applied again to $\sigma_2, \sigma_2 s$, and $\sigma_2 t$. In case UNIAT fails, RUNIFY terminates with a failure.

3.4. Algorithm for Idempotent Unification

Combining 3.2. and 3.3, we obtain the following algorithm IUNIFY for unifying two terms with respect to idempotence:

IUNIFY := λs:Term t:Term.
{RUNIFY $r_1 r_2 \mid (r_1, r_2) \in$ (collapse s) x (collapse t)} .

Note that instead of collapsing we could employ the conceptually equivalent strategy of expansion. However, collapsing is more efficient with respect to both time (interlacing RUNIFY avoids redundant steps) and space.

4. Completeness of the Unification Algorithm

We show the completeness of our unification algorithm IUNIFY in two steps:

(1) First, we show that for any unifier δ of terms s and t not collapsing any non-leaf nodes ("<u>immediate unifier</u>"), RUNIFY applied to s and t successfully returns a unifier σ s.t. $\delta \sqsubseteq \sigma$. I.e., RUNIFY is complete with respect to immediate unifiers.

(2) We then show that for any arbitrary unifier δ of s and t we can find substitutions τ and τ' s.t.

1. $\tau s \in \text{collapse}(s)$ and $\tau' t \in \text{collapse}(t)$, and
2. there is an immediate unifier Θ with $\Theta\tau s \overset{I}{=} \Theta\tau' t$ and $\delta \sqsubseteq \Theta \cdot \tau \cdot \tau'$.

4.1. Partial completeness of RUNIFY

To aid our subsequent exposition, we single out nodes that can be "immediately" collapsed.

4.1.1. Def.: For any term t, the set ICnode(t) of *immediately collapsible nodes* in t is defined by:

$\forall k \in \text{labels}(t).$

$k \in \text{ICnode}(t)$ iff $\text{lsub}(t,k) \overset{I}{=} \text{rsub}(t,k)$.

4.1.1.1. Example: Intuitively, a node k is immediately collapsible iff its two subtrees are equal under idempotence. For t = (((a,a),(a,b)),(a,(a,b))), ICnode(t) = {0,011}.

4.1.2. Immediate unifiers

Taking NLnode(t) to be the set of all non-leaf nodes in a term t, an immediate unifier is a substitution not immediately collapsing any non-leaf nodes.

4.1.2.1. Def.: A unifier σ of two terms t_1 and t_2 is called an *immediate unifier* for t_1 and t_2 iff $\text{ICnode}(\sigma t_i) \cap \text{NLnode}(t_i) = \emptyset$ for $i = 1,2$.

4.1.2.2. Example: Let $s = (x,y)$ and $t = (a,b)$; then $\sigma = \{x \leftarrow a, y \leftarrow b\}$ is an immediate unifier, but $\sigma = \{x \leftarrow (a,b), y \leftarrow (a,b)\}$ is a unifier which is not immediate.

4.1.3. Lemma: Let σ be an immediate unifier for two terms t_1 and t_2, and $\text{disagree}(t_1,t_2) = \{s_1,s_2\}$.
Then $\sigma s_1 \equiv \sigma s_2$.

Proof: Suppose the lemma is false, i.e. $\sigma s_1 \not\equiv \sigma s_2$. Since we assumed σ to be a unifier, we know that $\sigma t_1 \overset{I}{=} \sigma t_2$, hence $NF[\sigma t_1] \equiv NF[\sigma t_2]$ and we conclude that σ must collapse at least one node in t_1 or t_2. This node cannot be a leaf node, contradicting our assumption that the unifier σ is immediate.

□

4.1.3.1. Remark: Lemma 4.1.3 is always tacitly assumed in the unification theorem of ordinary unification (see e.g. [14]). However, it is not obvious, and usually not true, for T-unification with certain equational theories T. From the above Example 4.1.2.2, we can see that restricting the lemma to immediate unifiers is crucial: The disagreement set for $<(x,y),(a,b)>$ is $\{x,a\}$, and for the non-immediate unifier $\sigma = \{x \leftarrow (a,b), y \leftarrow (a,b)\}$, $\sigma x \not\equiv \sigma a$.

4.1.4. Theorem: [Completeness of RUNIFY with respect to immediate unifiers.] Let δ be an immediate unifier of two terms t_1 and t_2. Then RUNIFY terminates successfully with RUNIFY $t_1 t_2$ = (SUCC,σ) and $\delta \sqsubseteq \sigma$.

Proof: The proof is done by computational induction [10] on the functional:

```
Runi := λRU.λσ:SUBST. λ s:TERM t:TERM.
        if disagree(s,t) = ∅
           then (SUCC,∅)
           else let $(r_1,r_2)$ := disagree(s,t) in
                let $r_i \in$ AT and $i \neq j$ for $i,j \in \{1,2\}$  in
                {at least one of $r_1$ and $r_2$ is an atom, say $r_i$}
                let $(m,\sigma_1)$ := UNIAT $r_i r_j$  in
                if m = SUCC
                   then let $\sigma_2 := \sigma_1 \circ \sigma$  in
                        if $\sigma_2 s \overset{I}{=} \sigma_2 t$ then $(SUCC,\sigma_2)$
                                          else $RU[\sigma_2,\sigma_2 s,\sigma_2 t]$
                   else (FAIL,∅)
```

We show inductively that for all $k \in \mathbb{N}$ and terms t_1 and t_2 there are substitutions σ_k with:

(1) $(\text{Runi}^k\ h)\ \emptyset\ t_1 t_2 = (\text{SUCC},\sigma_k)$,

where h maps any substitution σ and terms s_1, s_2 to $h\sigma s_1 s_2 := (\text{SUCC},\bot)$, taking SUBST to be a discrete domain with least element $\bot$.

(2) $\exists\lambda_k \cdot \delta = \lambda_k \circ \sigma_k$.

Since this is obvious for $k = 0$, we assume inductively for any k

(*1) $\mathrm{Runi}^k\ h\emptyset t_1 t_2 = (\mathrm{SUCC}, \sigma_k)$ and

(*2) $\exists \lambda_k \cdot \delta = \lambda_k \circ \sigma_k$,

leaving to show that (1), (2) hold for k+1.

<u>Case 1:</u> $\sigma_k t_1 \equiv \sigma_k t_2$

Then $\mathrm{disagree}(\sigma_k t_1, \sigma_k t_2) = \emptyset$ so that $\mathrm{Runi}^{k+1}\ h\emptyset t_1 t_2 = (\mathrm{SUCC}, \sigma_k)$ by (*1), and $\lambda_{k+1} := \lambda_k$ does the job to prove (2).

<u>Case 2:</u> $\sigma_k t_1 \not\equiv \sigma_k t_2$

Let $\mathrm{disagree}(\sigma_k t_1, \sigma_k t_2) = \{r_1, r_2\}$ and, say, $u \equiv r_1 \in AT$. Then

$\mathrm{Runi}^{k+1}\ h\emptyset t_1 t_2 =$ *let* $(m, \Theta) = \mathrm{UNIAT}\ u\ r_2$ *in*

if $m = \mathrm{SUCC}$ *then* $(\mathrm{SUCC}, \Theta \circ \sigma_k)$ *else* $(\mathrm{FAIL}, \emptyset)$.

Our case analysis proceeds by considering u to be (a) a variable and (b) a constant.

<u>(a) $u \in Var$.</u>

(1) Let $u \in r_2$; from assumption (*2) and Lemma 4.1.3 we conclude (*) $\lambda_k u \equiv \lambda_k r_2$, so there is some term q with $u \leftarrow q \in \lambda_k$. Because all substitutions are assumed to be in normal form, λ_k does not substitute for any variable occurring in q, whence there is an occurrence of u in r_2 paired with a term equal to q as its brother. Taking $l \in \{j \mid u \notin \mathrm{sub}(r_2, j)$ and $\mathrm{brother}(r, j) = u\}$ to be the label of any such occurrence of u in r_2, we obtain for $\Theta := \{u \leftarrow \mathrm{sub}(r_2, l)\}$ that $\mathrm{sub}(r_2, l) = \Theta r_2$ and $\mathrm{UNIAT}\ u\ r_2 = (\mathrm{SUCC}, \Theta)$.

Let $\lambda_{k+1} := \lambda_k - \Theta$ and $\sigma_{k+1} := \Theta \circ \sigma_k$; then

$$\begin{aligned}\lambda_k &= \{u \leftarrow \lambda_k u\} \cup \lambda_{k+1} \\ &= \{u \leftarrow \lambda_k[\mathrm{sub}(r_2,1)]\} \cup \lambda_{k+1} \quad \text{by } (*) \\ &= \{u \leftarrow \lambda_{k+1}[\mathrm{sub}(r_2,1)]\} \cup \lambda_{k+1} \quad \text{since } u \notin \mathrm{sub}(r_2,1) \\ &= \lambda_{k+1} \circ \Theta.\end{aligned}$$

Hence $\delta = \lambda_k \circ \sigma_k$ by induction hypothesis (*2)

$$\begin{aligned}&= \lambda_{k+1} \circ \Theta \circ \sigma_k \\ &= \lambda_{k+1} \circ \sigma_{k+1}.\end{aligned}$$

(2) Let $u \notin r_2$; then UNIAT u r_2 = (SUCC,Θ) with $\Theta := \{u \leftarrow r_2\}$ so that for

$\lambda_{k+1} := \lambda_k - \Theta$ and $\sigma_{k+1} := \Theta \circ \sigma_k$ we obtain

$$\begin{aligned}\lambda_k &= \{u \leftarrow \lambda_k u\} \cup \lambda_{k+1} \\ &= \{u \leftarrow \lambda_k r_2\} \cup \lambda_{k+1} \quad \text{since } \lambda_k u = \lambda_k r_2 \text{ as in (1)} \\ &= \{u \leftarrow \lambda_{k+1} r_2\} \cup \lambda_{k+1} \quad \text{since } u \notin r_2 \\ &= \lambda_{k+1} \circ \Theta\end{aligned}$$

so that $\delta = \lambda_{k+1} \circ \sigma_{k+1}$ follows as in (1)

<u>(b) $u \in$ Const.</u>

If r_2 contains a constant different from u then u and r_2 are obviously not unifiable, contradicting inductive hypothesis (*2). Hence we obtain UNIAT u r_2 = (SUCC,Θ) with $\Theta := \{x \leftarrow u \mid x \in \mathrm{var}(r_2)\}$ and for $\lambda_{k+1} := \lambda_k - \Theta$ and $\sigma_{k+1} := \Theta \circ \sigma_k$ we conclude $\delta = \lambda_{k+1} \circ \sigma_{k+1}$ similarly to (a).

□

4.2. Completeness of IUNIFY

To establish the overall completeness of our unification algorithm we need two auxiliary lemmas about substitutions.

4.2.1. Lemma. Let ρ and σ be substitutions with $\rho \sqsubseteq \sigma$. Then, for any substitution τ : $\rho \circ \tau \sqsubseteq \sigma \circ \tau$.

Proof: Obviously, $\lambda \circ (\sigma \circ \tau) \sqsubseteq \sigma \circ \tau$ for any λ; assume in particular

$\rho = \lambda \circ \sigma$, whence:

$$\begin{aligned} \lambda \circ (\sigma \circ \tau) &= (\lambda \circ \sigma)\ \tau \\ &= \rho \circ \tau \\ &\sqsubseteq \sigma \circ \tau \end{aligned}$$

□

Unfortunately, Lemma 4.2.1 does not hold for composition from the left side. This is demonstrated by the example

$\rho := \{z \leftarrow c, x \leftarrow a\}$, $\sigma := \{z \leftarrow c\}$, and $\tau := \{x \leftarrow b\}$:

$\rho \sqsubseteq \sigma$ holds, but $\tau \circ \rho = \{z \leftarrow c, x \leftarrow a\} \not\sqsubseteq \{z \leftarrow c, x \leftarrow b\} = \tau \circ \sigma$. This fact makes the proof of the following lemma more complicated:

4.2.2. Lemma: For any substitutions δ, σ_1, and σ_2:
If $\delta \sqsubseteq \sigma_1$ and $\delta \sqsubseteq \sigma_2$ then $\delta \sqsubseteq \sigma_2 \circ \sigma_1$.

Proof: By assumption there exist λ_i s.t. $\delta = \lambda_i \circ \sigma_i (i = 1,2)$. Among all λ_1 with $\delta = \lambda_1 \circ \sigma$ choose λ_1 s.t. it is minimal with respect to $\sqsubseteq$. Let $\lambda_1 = \{v_n \leftarrow t_n \mid 1 \leq n \leq N\}$.

(4.2.2.1) $\delta \circ \lambda \sqsubseteq \delta$

Proof: From $\delta = \lambda_1 \circ \sigma_1$ we conclude $\forall n (1 \leq n \leq N).\ v_n \leftarrow t_n \in \delta$. This can be easily seen by assuming the contrary, i.e. $v_m \leftarrow t_m \notin \delta$ for some m, $1 \leq m \leq N$; then, for
$\lambda_1' := \lambda_1 - \{v_m \leftarrow t_m\}$ we obtain $\delta = \lambda_1' \circ \sigma_1$, contradicting the (minimal) definition of λ_1.

Hence we have:

$\lambda_1 \subseteq \delta$ (considered as sets of pairs) .

Thus follows:

$\delta \circ \lambda_1 \subseteq \delta \circ \delta$ (by $\circ$-multiplying δ from the left)

$= \delta$ (since δ is assumed in normal form).

Hence in particular:

$\delta \circ \lambda_1 \sqsubseteq \delta$.

(4.2.2.2) $\delta = \sigma_2 \circ \delta$.

Proof: $\delta = \delta \circ \delta$ (since δ assumed in normal form)

$= (\lambda_2 \circ \sigma_2) \circ (\lambda_1 \circ \sigma_1)$ (by definition)

$\sqsubseteq \sigma_2 \circ \lambda_1 \circ \sigma_1$

$= \sigma_2 \circ \delta$

$\sqsubseteq \delta$

(4.2.2.3) $\delta \sqsubseteq \sigma_2 \circ \sigma_1$

Proof: From: $\delta \circ \lambda_1 \sqsubseteq \delta$ by (4.2.2.1.)

follows (*) $(\delta \circ \lambda_1) \circ \sigma_1 \sqsubseteq \delta \circ \sigma_1$ by lemma 4.2.1.

Hence we have:

$\delta = \delta \circ \delta$ (since δ in NF)

$= \delta \circ (\lambda_1 \circ \sigma_1)$ (by definition)

$\sqsubseteq \delta \circ \sigma_1$ (by (*))

$= \lambda_2 \circ \sigma_2 \circ \sigma_1$ by definition

$\sqsubseteq \sigma_2 \circ \sigma_1$

□

From Lemma 4.2.2. we obtain immediately

4.2.3. Corollary: For any substitutions $\sigma_1,\ldots,\sigma_N (N \in \mathbb{N})$: If $\delta \sqsubseteq \sigma_n$ for $1 \le n \le N$ then $\delta \sqsubseteq \sigma_{i_1} \circ \ldots \circ \sigma_{i_N}$ for any permutation $(i_1,\ldots,i_N)$ of $(1,\ldots,N)$.

Our final result is the overall completeness of IUNIFY which is an immediate consequence of the next theorem:

4.2.4. Theorem: Let δ be any unifier for two terms t_1 and t_2. Then, there are substitutions τ_1 and τ_2 s.t.

(1) $\tau_i t_i \in \text{collapse}(t_i)$ $(i = 1,2)$, and

(2) RUNIFY applied to $\tau_1 t_1$ and $\tau_2 t_2$ successfully returns an immediate unifier Θ with $\delta \sqsubseteq \Theta \bullet \tau_1 \circ \tau_2$.

Proof: Let $\text{Icoll}_i := \text{ICnode}(\delta t_i)$ $(i = 1,2)$ be the sets of labels marking immediately collapsible nodes in δt_i, i.e. nodes "collapsed by δ". Then for any $k \in \text{Icoll}_i$ there is a substitution σ_{ki} s.t. $\text{sub}(\sigma_{ki} t_i, k) \equiv \text{sub}(\delta t_i, k)$ and $\sigma_{ki} t_i \in \text{collapse}(t_i)$. Because of Theorem 4.1.4., σ_{ki} is most general.

Let $\tau_i := \circ\{\sigma_{ki} \mid k \in \text{Icoll}_i\}$ be the combination of all such substitutions. Since τ_i combines all substitutions yielding collapses of nodes also collapsed by δ, all other substitutions carried out by δ do not result in collapses. Hence, there is a substitution Θ s.t. RUNIFY $(\tau_1 t_1)$ $(\tau_2 t_2)$ = (SUCC,Θ), and $\delta \sqsubseteq \Theta \circ \tau_1 \circ \tau_2$ again by Theorem 4.1.4.

□

4.2.5. Corollary: IUNIFY is complete; i.e. for any terms t_1 and t_2 with unifier $\delta : \delta(t_1) \overset{I}{=} \delta(t_2)$ there is a unifier σ generated by IUNIFY $t_1 t_2$ and some substitution λ such that

$$\delta \overset{I}{=} \lambda \circ \sigma \quad .$$

5. *Minimality*

The set of unifiers returned by IUNIFY is not minimal in general. For example, IUNIFY applied to (x,b) and $((a,y),b)$ returns the unifiers $\sigma_1 = \{x \leftarrow (a,y)\}$ and $\sigma_2 = \{x \leftarrow a, y \leftarrow b\}$. However, σ_2 is an instance of σ_1. IUNIFY can be improved with respect to minimality by restricting the set of collapses to be formed from "hot" nodes only:

5.1. Def.: Given a pair t_1 and t_2 of terms, the set of *hot nodes of t_i* with respect to $t_j (i = 1,2,\ i \neq j)$ is defined by

$$\text{Hnode}(t_i,t_j) := \{k \mid k \in \text{labels}(t_i) \text{ and } k \in \text{NLnode}(t_j)\}.$$

In other words, any node k of t_i is hot with respect to t_j if k also marks a non-leaf node in t_j.

Let IUNIFY_H be similar to IUNIFY except that forming collapses is restricted to hot nodes only. Then, following Section 4, it is straightforward to show:

5.2. Lemma: IUNIFY_H is complete.

IUNIFY_H applied to (x,b) and ((a,y),b) only returns the above unifier $\sigma_1 = \{x \leftarrow (a,y)\}$ but not $\sigma_2 = \{x \leftarrow a, y \leftarrow b\}$. However, IUNIFY_H is also not minimal in general. For example IUNIFY_H applied to ((x,a),b) and ((y,a),z) returns the unifiers $\tau_1 = \{z \leftarrow b, x \leftarrow y\}$ and $\tau_2 = \{z \leftarrow b, x \leftarrow a, y \leftarrow a\}$ but $\tau_2 \sqsubseteq \tau_1$.

We did not find a criterion ensuring minimality of an accordingly modified IUNIFY-algorithm. However, since the complete set of unifiers is always finite, substitutions that are instances of other unifiers can always be eliminated.

Consequently, any useful criterion guaranteeing minimality should be computationally cheaper than checking off instances of other unifiers.

6. Unification of Commutative and Idempotent Terms

In [20] an algorithm, CUNIFY for the unification of commutative terms is presented. The problem there is to prove whether or not the set Ψ of mgu's is finite and to find a computationally cheap condition to ensure minimality.

Essentially the algorithm permutes all the arguments of the commutative function subject to certain conditions and then applys Robinson's unification algorithm to each permutation.

Replacing Robinson's Unification in CUNIFY by IUNIFY gives an algorithm for functions which are both commutative and idempotent.

The set Ψ of mgu's thus obtained is finite, however not minimal in general.

It may deserve mentioning that such a combination of algorithms is *not* always possible: the axioms for commutativity (C) and associativity (A) provide a counterexample in that the algorithm for a theory with both C and A is totally different from the algorithms for C and A alone: Ψ_C is finite, the unification problem for C is decidable; Ψ_A is infinite [8], [12], [19], the decision problem is an open problem now for over 25 years (i), Ψ_{C+A} is finite and the unification problem is decidable [9], [22].

(i) in its equivalent form as: the wordproblem over a free monoid, Löb's Problem, Markov's Problem; the "crossreference problem" for van Wijngaarden grammars; second order monadic unification; ...

7. *References*

1. [BE67]	Bennett, Easton, Guard, Settle. CRT-aided semiautomated mathematics. Techn. Report AFCRL 67-0167, 1967, Applied Logic Corp., Princeton.
2. [CO65]	S. Cook. Algebraic techniques and the mechanization of number theory. Techn. Rep. RM-4319-PR, 1965, Rand Corp., Santa Monica, Cal.
3. [HEW72]	C. Hewitt. Description and theoretical analysis of PLANNER. Ph.D.-Thesis, M.I.T., 1972, Art. Int. Lab., Cambridge.
4. [HUE75]	G. Huet. Unification in typed lambda calculus, "Springer Lecture Notes", No. 37, (ed) Goos, Hartmanis, 1975.
5. [HUE75]	G. Huet. A unification algorithm for typed λ-calculus, Theoretical Comp. Sci. 1.1., 1975.
6. [KB67]	D. E. Knuth, P. B. Bendix. Simple Word Problems in Universal Algebras, in "Computational Problems in Abstract Algebra", J. Leech (ed), Pergamon Press, Oxford 1970.
7. [LA77]	D. Lankford. Complete sets of reductions. Univ. of Texas, Automatic Theorem Proving Project, Austin, Texas, Techn. Rep. ATP-35, ATP-37, ATP-39, 1977.
8. [LS75]	M. Livesey, J. Siekmann. Termination and Decidability Results for String Unification. Essex University, Computing Centre, Memo CSM-12, 1975.
9. [LS76]	M. Livesey, J. Siekmann. Unification of Bags and Sets. Interner Bericht 3/76, Institut für Informatik I, Universität Karlsruhe, 1976.

10. [MA74] Z. Manna. Introduction to the Theory of Computation. Addison-Wesley, 1974.

11. [NE71] A. Nevins. A human-oriented logic for automatic theorem proving. JACM, vol 21, No. 4, 1971.

12. [PL72] G. Plotkin. Building in equational theories. Machine Intelligence, vol 7, 1972.

13. [PW72] M. S. Paterson, M. N. Wegman. Linear Unification. IBM Research Rept. 5804, 1976.

14. [ROB65] J. A. Robinson. A machine oriented logic based on the resolution principle. JACM:<u>12</u>, 1965.

15. [ROB67] J. A. Robinson. A review on automatic theorem proving. Symp. Appl. Math., vol 19, 1-18, 1967.

16. [RW73] G. Robinson, L. Wos. Maximal models and refutation completeness: Semidecision procedures in automatic theorem proving. In Boone et al. (eds.), "Word problems", North Holland, 1973.

17. [RDW72] Rulifson, Derksen, Waldinger, QA4: A procedural calculus for intuitive reasoning, Techn. Rep., Stanford Res. Inst., Nov. 1972.

18. [BFR76] Böhm, Fischer, Raulefs. Dialogs in Actor Nets. Universität Karlsruhe, Institut für Informatik I, 1976.

19. [SI75] J. Siekmann. String unification. Essex University, Memo CSM-7.

20. [SI76] J. Siekmann. Unification of commutative terms. Interner Bericht 2/76, Universität Karlsruhe, Institut für Informatik I.

21. [SL72] J. R. Slagle. ATP with built in theories including equality, partial ordering, and sets. JACM:<u>19</u>, 1972.

22. [ST75] M. Stickel. A complete unification algorithm for associative-commutative functions. Proc. 4th IJCAI, Tblisi, USSR, 1975.

23. [HM64] J. Hmelevskij. The solution of certain systems of word equations. Dokt. Akad. Nauk. SSR (1964), (1966), (1967), (Soviet Math. Dokl.).

24. [SZU78] P. Szabo, E. Unvericht. D-unification has infinitely many mgu's. Universität Karlsruhe, Institut für Informatik I, 1978.

25. [WI76] G. Winterstein. Monadic Second Order Unification. Universität Kaiserslautern, 1976.

26. [GO66] W. E. Gould. A matching procedure for ω-order logic. Scientific report No. 4, AFCRL-666-781, 1966.

27. [MA54] A. A. Markov. Trudy Mat. Inst. Steklov, No. 42, Izdat. Akad. Nauk SSR, 1954.

28. [STI77] M. F. Stickel, G. E. Peterson. Complete Sets of Reductions for Equational Theories with Complete Unification Algorithms. Dept. Comp. Sci., University of Arizona, Tucson, Techn. Rep., 1977.

29. [HUE77] G. Huet. Confluent Reductions: "Abstract Properties and Applications to Term Rewriting Systems", Pap. Rech. No. 250, IRIA Laboria, Rocquencourt, France, 1977.

30. [VO78] E. Vogel: Unifikationsalgorithmen für Morphismen. Diplomarbeit (forthcoming), Universität Karlsruhe, Institut für Informatik I, 1978.

31. [FGP64] P.J. Faber, R.E. Griswald, I.P. Polonsky: 'SNOBOL as String Manipulation Language'. JACM, vol 11, no. 2, 1966.

32. [FIP74] J. Fischer, S. Patterson: 'String Matching and other Products', MIT, Project MAC, Report 41, 1974.

33. [KMP74] Knuth, Morris, Pratt: 'Fast Pattern Matching in Strings', Stan-CS-74-440, Stanford University, Computer Science Dept., 1974.

34. [MAK77] G.S. Makanin: The Problem of Solvability of Equations in a Free Semigroup, Soviet Akad. Nauk SSSR, Tom 233, no. 2, 1977.

35. [SZA78] P. Szabó: 'The undecidability of the D+A-unification problem' (forthcoming), Universität Karlsruhe, Institut für Informatik I, 1978.