ALGEBRAIC MANIPULATIONS AS A UNIFICATION AND MATCHING STRATEGY FOR LINEAR EQUATIONS IN SIGNED BINARY TREES. APPLICATION TO LISP PROGRAM SYNTHESIS AND DERECURSIVATION

C. KIRCHNER, H. KIRCHNER, J.P. JOUANNAUD

CRIN-UNIVERSITE of NANCY I
UER de Sciences Mathematiques
Case Off. 140 - 54037 NANCY Cedex, FRANCE

ABSTRACT

We study the unification and matching problems in the signed binary trees theory. We show that any equation tl1=t2 can be transformed into an equivalent one x=t. If x does not occur in t then (x——> t) is the unique most general unifier in the theory (up to an isomorphism). We apply this technique to find recurrence relations between sets of terms. These relations are used to synthesize LISP programs from a set of input traces.
KEYWORDS : unification, matching, equation, unifier, substitution, rewrite rule, term, tree, program synthesis.

I INTRODUCTION

Signed binary trees have been introduced in [14] in order to solve equations which have no solution in binary trees. Such equations appear in program synthesis or transformation processes. This technique enables us to perform such synthesis or transformation processes in various situations. Our goal here is to study how equations may be solved in this theory.

Section II recalls both usual concepts of unification in an equational theory and definitions of signed binary trees considered as terms of an equational theory or as a term rewriting system.

Section III introduces the concept of equivalent equations and sets some basic tools used in the following sections.

Section IV describes and proves an algorithm which transforms an equation into an equivalent one (with less function symbols). Some properties of reduced equations are studied.

Section V is concerned with the resolution of linear equation (an equation is linear if there exists a variable x which has a unique occurrence in the terms of the equivalent reduced equation). We give a constructive caracterisation of the most general solution (up to an isomorphism) of such a linear equation.

The matching problem is then studied in section VI.

An example of application to program derecursivation is shown in section VII.

II ALGEBRAIC FRAME

We recall here some basic notions and results discussed in [13].

A. Signed binary trees

As we are dealing with terms (or trees), we first give some general definitions :

DEFINITION 1 : given a denumerable set X of variables symbols, a denumerable set A of atoms (or constants symbols i.e. functions with arity 0) and a finite set F of functions symbols (with non-zero arity), the algebra $\mathfrak{A}(X, AUF)$ is the inductively defined set :

** $X \subseteq \mathfrak{A}(X, AUF)$
** $A \subseteq \mathfrak{A}(X, AUF)$
** $\forall f \in F, \forall t_i \in \mathfrak{A}(X,AUF), 1 \leq i \leq arity(f) \Rightarrow f(...t_i...) \in \mathfrak{A}(X, AUF)$.

Elements of $\mathfrak{A}(X,AUF)$ are called terms. The set of variables of a term t is denoted by V(t), its size by $|t|$.

It is often convenient to consider terms as labelled trees in the following way [4].

DEFINITION 2 : *a term t is a mapping from a set of occurrences denoted D(t) which is a subset of N*, into the set XUAUF such that :

** $m \in D(t) \Rightarrow$ any prefix of m belongs to D(t).
** $m \in D(t) \Rightarrow m.i \in D(t)$ iff $1 \leq i \leq arity(t(m))$.

* The number of occurrence of x in t is the cardinal of the set $\{u \in D(t) \text{ s.t. } t(u)=x\}$.

It is easy to see that these definitions are equivalent. Notice that the empty word ϵ belongs to D(t) for any t and that $t(\epsilon)$ is the top symbol of t. In the following, we generally use the tree-notation

f instead of f(... t ...)
|
... t ...

DEFINITION 3 : the set $\hat{A} = \mathcal{A}(X, A \cup \{cons\})$ where cons is a binary function symbol, is called set of binary trees.

It is clear that we have in mind some kind of LISP interpretation of terms in $\hat{A}$ and we thus use capital letters for atoms and small letters for variables.

DEFINITION 4 : the set of signed binary trees on A is the algebra $\bar{A} = \mathcal{A}(X, A \cup \{-, cons\})$ with the set of axioms :

(A1) $\bar{\bar{u}} = u$ (A2) $\overline{cons(u,v)} = cons(\bar{v}, \bar{u})$

(A3) $cons(u, \bar{\bar{u}}) = v$ (A4) $cons(\bar{u}, cons(u,v)) = u$

Notice that these axioms look like the axioms of groups theory. In earlier versions [14], the set of axioms was a little more complicated.

It is proved in [13] that this set of axioms is in fact a consequence of the two last axioms A3, A4.

In the following, $\bar{\bar{x}}$ is sometimes abbreviated in $-x$ and signed binary trees are denoted by letters t,u,v,w.

B. The term rewriting system $\vec{A}$

Applying the KNUTH-BENDIX completion algorithm [15], [9] , axioms are used from left to right only and an infinite set of rules is generated which divides into two parts :

** rules 1 and 2 obtained from the two first axioms :

$--u \xrightarrow{1} u$; $-cons(u,v) \xrightarrow{2} cons(-v, -u)$

** 2 meta-rules obtained from the two remaining axioms :

$cons(s, cons(m(s), v)) \xrightarrow{III} v$; $cons(cons(u, m(s)), s) \xrightarrow{IV} u$

where s denotes a structure and m(s) its opposite mirror,defined as follows :

DEFINITION 5 : the set S of structures and the opposite mirror m(s) of a structure S are inductively defined by :

** $X \subseteq S$ and $m(x) = -x$
** $\forall x \in X$, $-x \in S$ and $m(-x) = x$
** $\forall s1, s2 \in S$, $V(s1) \cap V(s2) = \emptyset$ =>
$cons \in S$ and $m(cons(s1, s2)) = cons(m(s2), m(s1))$

EXAMPLE : s = cons(cons, u) is a structure and m(s) = cons(-u, cons)

We call type III (respectively type IV) rules the rules obtained from the meta-rule III (respectively IV) by substituting a structure to the variable structure s and its opposite mirror to m(s).

PROPOSITION 1 [14] : A is finitely terminating and confluent. The normal form of any term t is computed in $O(|t| \log |t|)$ time in the worst case.

C. Unification in $\bar{A}$

We first recall some classical definitions and results related to unification in equational theories.

DEFINITION 6 : a substitution σ is a mapping from X to $\bar{A}$ equal to identity except on a finite subset of X called its domain and denoted by $D(\sigma)$. The restriction σ_V of σ to a subset V of X is equal to σ on V and equal to identity on X\V. Thus $D(\sigma_V) = D(\sigma) \cap V$. Substitutions are extended to terms by morphism.

In the following, letters σ, ρ and μ denote substitutions and Σ the set of substitutions.

DEFINITION 7 : the equality $=_A$ is the finest congruence over $\bar{A}$ containing all pairs $(\sigma(g_i), \sigma(d_i))$ for each axiom $g_i = d_i$. As a consequence, $t1 =_A t2 \Rightarrow \forall \sigma, \sigma t1 =_A \sigma t2$.

PROPOSITION 2 [8] : $t1 =_A t2$ iff t1 and t2 have the same normal form.

We define now a quasi-ordering on terms and extend it to substitutions :

DEFINITION 8 :
$\forall t \in \bar{A}, \forall t' \in \bar{A}$, $t \leq t'$ iff $\exists \sigma$ s.t. $t' =_A \sigma t$

DEFINITION 9 : let V be any finite subset of X.
$\sigma_V \leq \sigma'_V$ (or $\sigma \leq_V \sigma'$) iff $\exists \rho \in \Sigma$ s.t. $\forall x \in V, \sigma' x =_A \rho(\sigma x)$.
$\sigma_V \simeq \sigma'_V$ (or $\sigma \simeq_V \sigma'$) iff $\sigma \leq \sigma'$ and $\sigma' \leq_V \sigma$

The classical following proposition, easy to prove, tells us that $\leq$ works well on classes of terms for $=_A$ congruence :

PROPOSITION 3 :
$\forall t, t', t1, t1'$: $t =_A t1$ and $t' =_A t1' \Rightarrow (t \leq t' \text{ iff } t1 \leq t1')$

DEFINITION 10 : we call unifier of two terms t and t', any substitution σ such that $\sigma t =_A \sigma t'$. Let $U(t,t')$ be the set of all unifiers. We call equation in $\bar{A}$ any pair of terms (t,t'). A solution of the equation (t,t') is a unifier of t and t'. The equation (t,t') is also denoted $t =_A t'$.

A unifier σ of two terms t and t' (a solution of the equation $t =_A t'$) is said to be minimal iff $\forall \sigma'$ such that $\sigma' \leq \sigma$ with $V = V(t) \cup V(t')$ $\Rightarrow \sigma' \notin U(t,t')$.

We are in fact interested in a basis of the set of solutions of the equation $t =_A t'$. Such a basis is called minimal complete set of unifiers and has been defined by HUET [7].

DEFINITION 11 : a minimal complete set of unifiers of two terms t and t' (a basis of the set of solutions of the equation $t =_A t'$) denoted UP(t,t') satisfies the three following properties :
(1) $UP(t,t') \subseteq U(t,t')$

(2) $\forall \sigma \in U(t,t'), \exists \rho \in UP(t,t')$ s.t. $\rho_V \leq \sigma_V$ with V= V(t)UV(t') (completeness).
(3) $\forall \sigma,\sigma' \in UP(t,t'), \sigma \neq \sigma' \Rightarrow \sigma_V \not\leq \sigma'_V$ (minimality).

REMARK : we do not use the technical condition which appears in [10] , [7], [6]. It is well known that many theories have infinite (sometimes undecidable) minimal complete sets of unifiers. We shall see that it is not the case in $\bar{A}$. This problem is solved in the next section by the way of algebraic manipulations allowed by the axioms.

III EQUIVALENT EQUATIONS

In this section, we show how an equation t=t' can be transformed into an equivalent one, in practice a simpler one. The following results can be proved without difficulty.

DEFINITION 12 : two equations t=t' and u=u' are equivalent iff U(t,t')=U(u,u'), i.e. they have the same set of solutions.

PROPOSITION 4 : if t=u and t'=u' then the two equations t=t' and u=u' are equivalent.

COROLLARY : let t and t' be two terms of $\bar{A}$ with $\bar{t}',\bar{t}$ as normal forms, then the equation t=t' is equivalent to the equation $\bar{t}=\bar{t}'$.

Using these general results, we shall now study the equations between signed binary trees.

PROPOSITION 5 : $\forall t \in \bar{A}, \forall t' \in \bar{A}$, the equations t=t' and -t=-t' are equivalent.

LEMMA 1 : $\forall t,t',t'' \bar{A}, t'=t''$ iff cons = cons (t, A(t', t''))

iff cons = cons (t', A(t, t''))

PROPOSITION 6 : $\forall t,t',t'' \in \bar{A}$, the equations t'=t'', cons = cons (t, A(t', t'')) , cons = cons (A(t, t'), t'') are equivalent.

PROPOSITION 7 : $\forall t',t'' \in \bar{A}, \forall x \in V(t')UV(t''), \exists t \in \bar{A}$ such that the equation t'=t'' is equivalent to the equation x=t.

The proof is constructive and uses a structural induction on t'.See [17].

EXAMPLE : Using the proposition 7, the equation cons = cons (x, A(B, cons(A, y))) is equivalent to the equations :

x = cons (cons(-A, B), cons(A, y)) ; y = cons (-A, cons(-B, cons(x, A)))

IV SIMPLIFICATION OF EQUATIONS

A. Reduced equations

Some equations with a great number of nodes labelled with cons symbols are equivalent to equations with less cons symbols. For example :

cons = x <=> cons = cons <=>

cons = cons <=> cons = cons <=>

x = cons (-B, -C)

Notice that the simplified form of an equation is not unique. For example

x = cons <=> cons = -B <=> C = cons <=> ...

DEFINITION 13 : considering equations as terms of an algebra, let us define the relation $\rightarrow$ by the three rewrite rules :

cons = cons (x, y) --> y = z ; (1)

cons = cons (y, x) --> y = z ; x = x --> $\emptyset$ (2) (3)

($\emptyset$) is the empty equation).

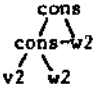
We say that e' immediately simplifies e iff e --> e'.

The last example shows that irreducible equations (for -->) are equivalent to reducible ones. We shall first design an algorithm able to compute the reducible equation e' equivalent to the equation e, when it exists.In our example, let d denote the duplicated term cons (z, A)

We see that d belongs to the left side of the first equation, while the second occurrence of d is split into 3 parts, -z and -A in the left-hand side and x in the right one. We thus can use the proposition 6 and 4 to build the second occurrence of d in the right side and then delete it. At each step, the part of d which goes from left to right across the = symbol is smaller than d. If such a full d exists at the beginning on the left-hand side of the equation, it remains at its place until the second d appears in the right-hand side.

DEFINITION 14 : we define now the relation |==> by the rewrite rules : cons = cons (v1, w1, v2, w2) |==> (1) v1 = cons (v2, w2) ; cons = cons (cons(m(w1), v1, w1), v2, w2) |==> (2) w1 = cons (m(v1), cons(v2, w2))

Let us denote by |==| the symetric closure of |==> and by <==| the inverse of |==>. Notice that we use the term cons (cons(v2, w2), m(w1)) which is in normal form

if w1≠w2, instead of the term  , which is

not in normal form, according to the lemma proved in [13] : $-t \xrightarrow[A]{} m(t)$.

DEFINITION 15 : let E be the set of equations t1 = t2 where t1 and t2 are terms of $\vec{A}$. Terms of E are called equational terms. We define the opposite mirror m(t1=t2) of the equational term t1 = t2 as the equational term m(t2)=m(t1).

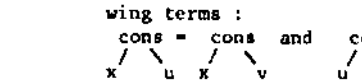
LEMMA 2 : $t1=t2 \vdash\!\!\Rightarrow t1'=t2'\Leftarrow\!\!\vdash t1''=t2'' \Rightarrow (t1''=t2'')=(t1=t2)$ or $(t1''=t2'') \vdash\!\!\Rightarrow\!\!\vdash m(t1=t2)$.

LEMMA 3 : $t1=t2 \vdash\!\!\overset{n}{\Rightarrow}\!\!\vdash t1'=t2' \Rightarrow m(t1=t2) \vdash\!\!\overset{n}{\Rightarrow}\!\!\vdash m(t1'=t2')$

Proof : by induction on n. See [17] for details.

From now up, we consider the only equational terms t1=t2 such that t1 and t2 are in normal form for $\vec{A}$. Let us denote by R the set of equational terms t1=t2 reducible for --> (and such that t1 and t2 are normal forms of $\vec{A}$).

Terms of R are thus instances of the two following terms :



where x,u and v are variables which can be instanciated by irreducible terms of $\vec{A}$.

Notice that $m(e) \in R$ iff $e \in R$.

We are now able to prove the following important result.

PROPOSITION 8 : if $t1=t2 \vdash\!\!\overset{p}{\Rightarrow}\!\!\vdash e \in R$, where t1 and t2 are irreducible terms of $\vec{A}$ with $|t1| > |t2|$, then there exists a sequence of equational terms : $(t1=t2)=(t1^0=t2^0) \vdash\!\!\Rightarrow t1^1=t2^1 \vdash\!\!\Rightarrow \ldots \vdash\!\!\Rightarrow t1^{p-1}=t2^{p-1} \vdash\!\!\Rightarrow t1^p=t2^p$ with $t1^p=t2^p \in R$ and $\forall\, 0 \le i \le p-1, |t1^i| > |t2^i|$ and $(t1^p=t2^p)=e$ or m(e) in normal form.

Proof : the proof is long, technical, and not easy. It uses an induction on the length n of $\vdash\!\!\Rightarrow$. See [17]. □

We are now ready to design and prove the simplification algorithm.

B. Simplification algorithm

DEFINITION 16 : given $e=(t1=t2) \in E$, with t1 and t2 in normal form for $\vec{A}$, let C(e) denote the set of equations equivalent to e modulo the equivalence generated by --> and $\vdash\!\!\Rightarrow$,i.e.: $C(e) = \{e' \in E / e' \ (\overset{*}{\rightarrow}.\overset{*}{\leftarrow}.\ \overset{*}{\vdash\!\!\Rightarrow}.\overset{*}{\Leftarrow\!\!\vdash}\ )^* e\}$.

LEMMA 4 : if $t1=t1'$ and $t2=t2'$ then $C(t1=t2) = C(t1'=t2')$.

Proof : we prove that $(t1=t2)\ (\overset{*}{\rightarrow}.\overset{*}{\leftarrow}.\ \overset{*}{\vdash\!\!\Rightarrow}.\overset{*}{\Leftarrow\!\!\vdash}\ )^* (t1'=t2')$. □

PROPOSITION 9 : given any equation $e=(t1=t2)$ with $|t1| \ge |t2|$, t1 and t2 in normal form for $\vec{A}$, the following algorithm REDUIRE computes a minimal element of C(e), i.e. which is not equivalent (for $\vdash\!\!\Rightarrow$) to a reducible one (for -->)

REDUIRE (t1=t2)

CASE (0) t1 is an atom (a variable or a constant) AND t1=t2 THEN ∅,

(1) e=(cons = cons) OR (cons = cons)

THEN REDUIRE (u1=u2).

(2) e=(cons = x) OR (cons = x)

THEN IF u1 > u2 THEN REDUIRE (u1=m(u2)) ELSE REDUIRE (m(u2)=u1).

(3) $|t1| \le |t2|$ THEN (t1=t2) is the result.

(4) t1= cons AND $|v| > |w|$ THEN REDUIRE(v,cons)


(5) ELSE REDUIRE (w= cons)


ENDCASE

END REDUIRE.

REMARK : the aim of case (2) is to express the fact that in proposition 8, the equational term $t1^p=t2^p$ may verify $|t1^p| < |t2^p|$. It is possible to avoid this case (2) but cases (1) and (4) become much more complicated.

Proof : REDUIRE terminates obviously, because the size of the left term of the equation argument of REDUIRE is strictly decreasing at each recursive call.

The proof of correctness uses proposition 8, see [17].

This algorithm is very performant : it is easy to convince oneself that it stops after a few calls when the equation e is not reducible and we have proved that its complexity is linear in the worst case [17].

C. Properties of reduced equations

Before studying how equivalent reduced equations are related, we first prove :

PROPOSITION 9 : $\rightarrow / \overset{*}{\vdash\!\!\Rightarrow}$ is confluent.

Proof : $\rightarrow / \overset{*}{\vdash\!\!\Rightarrow}$ is locally confluent and finitely terminating. □

COROLLARY 2 : if $\bar{e}1$ and $\bar{e}2$ are irreducible elements of C(e), then $V(\bar{e}1)=V(\bar{e}2)$ and $\forall x$, $O(x,\bar{e}1)=O(x,\bar{e}2)$, where O(x,e) denotes the number of occurrences of x in e.

Proof : it follows from proposition 9 that $\bar{e}1 = \bar{e}2$.

V LINEAR EQUATIONS

In this section, we show that for linear equations, the minimal complete set of unifiers has only one element, which is easy to compute.

A. Definitions

DEFINITION 17 : we say that a term t in normal form is linear in a variable x iff O(x,t)=1.

We say that an equation e is linear in x iff O(x,e)=1.

We can remark first that a non-linear equation can be equivalent to a reduced linear one, as in the example of section IV.

Secondly, for two equations e and e' deduced one from the other by some manipulations of section III, their reduced equivalent equations $\bar{e}$ and $\bar{e}'$ are linear or non-linear at the same time : it is an obvious consequence of the corollary 2.

So the property of linearity can be extended to the whole set C(e).

We shall say that C(e) is linear in x iff $\bar{e}$ is linear in x. So we suppose from now on that we are dealing with reduced equations.

We use in what follows the classical notion of linear contexts.

DEFINITION 18 : let t a term in $\bar{A}$, oc an element of D(t) and s the subterm of t at the occurence oc ; then $C_{t,oc}[\ ]$ denotes the context such that $C_{t,oc}[s]=t$, that is the term obtained by substituting [ ] to s at occurrence oc in t and $C^{-1}_{t,oc}[\ ]$ the context such that $C^{-1}_{t,oc}[t] =_{\bar{A}} s$.

With this notation, proposition 7 becomes : if x is a variable at the occurrence oc in t1 and if t1=t2 is an equation linear in x, then t1=t2 is equivalent to the equation $x=C^{-1}_{t1,\,oc}[t2]$.

B. Minimal complete set of unifiers for a linear equation

PROPOSITION 11 : let t1=t2 be an equation linear in x and $x=C^{-1}_{t1,oc}[t2]$ the equivalent equation. Then $\{\sigma\}$, where $\sigma=(x\longrightarrow t=C^{-1}_{t1,oc}[t2])$, is a minimal complete set of unifiers of t1 and t2.

Proof :* σ is clearly an unifier of x and t since x does not belong to V(t).

* if σ belongs to U(x,t), then $\sigma'\sigma \underset{\bar{A}}{=} \sigma'$ because $\sigma'\sigma(x) \underset{\bar{A}}{=} \sigma'(t) \underset{\bar{A}}{=} \sigma(x)$ and for $y\neq x$, $\sigma'\sigma(y) \underset{\bar{A}}{=} \sigma'(y)$.

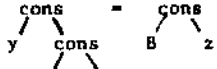
* The third condition of definition 11 is clearly true. □

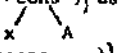
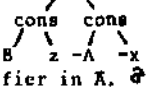
REMARK 1: when t1=t2 is also linear in $y\neq x$, $y\notin V(t1)$ for example, let oc_x and oc_y the respective occurrence of x and y in t1.

Then $\sigma 1=(x\longrightarrow C^{-1}_{t1,oc_x}[t2])$ and $2=(y\longrightarrow C^{-1}_{t1,oc_y}[t2])$ verify $\sigma 1 \underset{V}{\#} \sigma 2 (V=V(t1)\cap V(t2))$.

$\{\sigma 1\}$ and $\{\sigma 2\}$ are two distinct minimal complete sets of unifiers of t1 and t2.

REMARK 2: the most general unifier in $\hat{A}$ is not necessarily a minimal unifier in $\bar{A}$: for example the equation



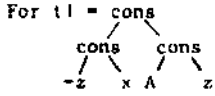
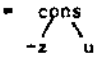
has $\hat{\sigma}=\{(y\longrightarrow B)(z\longrightarrow$  $)\}$ as most general unifier in $\hat{A}$, and $\bar{\sigma}=\{(y\longrightarrow$  $)\}$ as minimal unifier in $\bar{A}$. $\hat{\sigma}\,\bar{\sigma}=\hat{\sigma}$, but there is no ρ such that $\rho\hat{\sigma}=\bar{\sigma}$, because $\rho\hat{\sigma}(y)=\rho(B)=B$ and $\bar{\sigma}(y)\neq B$.

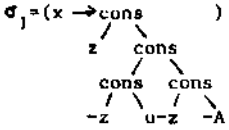
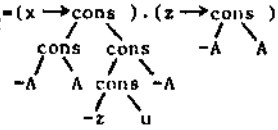
VI MATCHING IN $\bar{A}$

A. Definition and examples

DEFINITION 19 : in $\bar{A}$, matching a term t1 to a term t2 is defined as looking for a substitution σ such that $\sigma t1 \underset{\bar{A}}{=} t2$. When such a substitution exists, let us call match from t1 to t2 the substitution σ_V where V=V(t1).

Contrary to the first order theory without axioms, the following example shows that when a match exists, it is generally not unique.

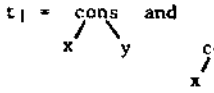
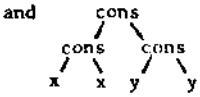
For t1 =  and t2 = 

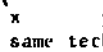
$\sigma_1=(x\longrightarrow$  $)$ and $\sigma_2=(x\longrightarrow$  $).(z\longrightarrow$ $)$

are two solutions to the matching problem.

REMARK : a strategy to find a match from t_1 to t_2 when t_1 is linear (for example in x) consists in using the results about unification :

a) renaming all the variables of t_2 with a one to one mapping $\mathcal{F}$ such that $V(t_1)\cap \mathcal{F}(V(t_2)) = \emptyset$, is always possible.

b) Since t_1 is linear in x, t_1 and $\mathcal{F}(t_2)$ have a minimal unifier $\sigma=(x\longrightarrow t)$ such that $\sigma t_1 \underset{\bar{A}}{=} \sigma(\mathcal{F}(t_2)) = \mathcal{F}(t_2)$ because $D(\sigma)\cap V(\mathcal{F}(t_2)) = \emptyset$. Thus, $\mathcal{F}^{-1}\sigma\, t_1 \underset{\bar{A}}{=} t_2$ and $\mathcal{F}^{-1}\sigma$ is a match from t_1 to t_2. But it is in general not minimal : for example, let t_1 =  and  ; $\rho=(x\longrightarrow$  $)$

is the match found with the above method and it is easy to verify that ρ is not smaller than $\mu=(x\longrightarrow$  $)(y\longrightarrow$  $)$ which cannot be obtained by the same technique. This remark justifies the following results.

From now on, F(t1, t2) denotes the set of all matches from t1 to t2, and we define a minimal complete set of matches.

DEFINITION 20 : a set of substitutions FP(t1, t2) is a minimal complete set of matches from t1 to t2 iff :

1) $FP(t1,t2) \subseteq F(t1,t2)$.

2) $\forall \sigma \in F(t1,t2)$, $\exists \rho \in FP(t1,t2)$ such that $\rho_V < \sigma_V$ with V=V(t1).

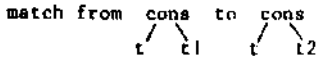
3) $\forall \sigma,\sigma' \in FP(t1,t2)\ \ \sigma\neq\sigma' \Rightarrow \sigma_V \not\leq \sigma'_V$.

B. Manipulations on equations and matching

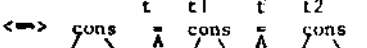
We should like to know what happens to the matchings when the former manipulations are applied to the terms. The following lemmas attempt to answer this question.

LEMMA 5 : let σ be a substitution and t any term verifying $\sigma t=t$. σ is a match from t1 to t2 if and

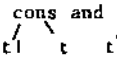
only if σ is a match from cons to cons



Proof : $\sigma t1 \underset{A}{=} t2 \Longleftrightarrow$



Obviously we get the same result with cons and cons



LEMMA 6 : σ is a match from t1 to t2 if and only if σ is a match from -t1 to -t2.

Proof : $\sigma t1 \underset{A}{=} t2 \Longleftrightarrow - \sigma t1 \underset{A}{=} -t2 \Longleftrightarrow \sigma(-t1) \underset{A}{=} -t2$. □

LEMMA 7 : let t1,t2,t1',t2' be terms such that $t1=t2 \Longleftrightarrow t1'=t2'$ and σ a substitution such that $D(\sigma) \cap V(t2) = D(\sigma) \cap V(t2') = \emptyset$; then σ is a match from t1 to t2 if and only if σ is a match from t1' to t2'.

Proof : obvious, since σ is a unifier of both t1, t2 and t1', t2'

REMARK : it follows that some matches are lost during the manipulations, precisely these which contradict the hypothesis of lemma 7.
In the particular case of a term t1 linear in x, x do not occur in a subterm u of t1 removed from the left during the sequence of manipulations of proposition 7. Thus lemma 6 applies and $\sigma = (x \longrightarrow C^{-1}_{t1,ocx}[t2])$ is a match from t1 to t2. But some matches can exist from t1 to t2 which do not verify for any subterm u of t1, $\sigma u \underset{A}{=} u$.

When t1 is linear in x and $x \notin V(t2)$ we are in the case of lemma 7 with t1'=x and $t2' = C^{-1}_{t1,ocx}[t2]$ and so $\sigma = (x \longrightarrow C^{-1}_{t1,ocx}[t2])$ is the only match from t1 to t2. Here again linearity hypothesis leads to stronger results, and we are going to study these cases.

C. Linear terms

Under some restrictive conditions, we can prove the existence of a minimal complete set of matches reduced to only one element.

PROPOSITION 12 : if t1 is linear in x and $V(t1) \cap V(t2) \subseteq \{x\}$ then $\{\sigma\}$ where $\sigma = (x \longrightarrow t = C^{-1}_{t1,ocx}[t2])$ is a minimal complete set of matches from t1 to t2.

Proof : 1) $\sigma \in F(t1,t2)$ see lemma 5 and the remark above.

2) $\forall \rho \in F(t1,t2)$, as $\rho t1 \underset{A}{=} t2$, $\rho(C_{t1,ocx}[x]) = C_{\rho t1,ocx}[\rho x] \underset{A}{=} t2$ then $\rho x = C^{-1}_{\rho t1,ocx}[t2]$.
Let V=V(t1) and $\mu = \rho_{V \setminus \{x\}}$. When $y \in V \setminus \{x\}$: $\mu y = \rho y \Rightarrow C_{\rho t1,ocx}[\,] \equiv C_{\mu t1,ocx}[\,]$.
As $(V \setminus \{x\}) \cap V(t2) = \emptyset$ by hypothesis, $\mu(t2) = t2$ and on the other hand $\rho(t) = \mu(t)$. So let us show that : $\forall z \in V$, $\mu\sigma z \underset{A}{=} \rho z$.
* When $z \neq x$, $\mu\sigma z = \mu z = \rho z$
* When $z = x$, $\mu\sigma x = \mu t$ and $\rho x \underset{A}{=} C^{-1}_{\rho t1,ocx}[t2] = C^{-1}_{\mu t1,ocx}[\mu t2] = \mu[C^{-1}_{t1,ocx}[t2]] \underset{A}{=} \mu t$.

Thus $\sigma \leq \rho$

2) The condition 3 of definition 22 is obvious. □

PROPOSITION 13 : if t1 and t2 are linear in x and if $t = C^{-1}_{t1,ocx}[t2]$, then $\{\sigma = (x \longrightarrow t)\}$ is a minimal complete set of matches from t1 to t2.

Proof : similar to proposition 12 except for 2) : $\forall \rho \in F(t1,t2)$, let us find a substitution μ verifying $\forall z \in V(t1) = V : \mu\sigma z = \rho z$ i.e.
* When $z \neq x$: $\mu z \underset{A}{=} \rho z$
* When $z = x$: $\mu t \underset{A}{=} \rho x$

Since t1 and t2 are linear in x, μx occurs only one time in μt. Let $x' = \mu x$. The equation $\mu t \underset{A}{=} \rho x$ is linear in x' and has a solution $(x' \longrightarrow C^{-1}_{\mu t,ocx}[\rho x])$
So $\mu x \underset{A}{=} C^{-1}_{\mu t,ocx}[\rho x] \underset{A}{=} C^{-1}_{\rho t,ocx}[\rho x]$.

VII APPLICATION TO PROGRAM SYNTHESIS AND PROGRAM DERECURSIVATION

We use a derecursivation technique which is detailed in [16] : from a given recursive definition, a set of "computational traces" is computed first. Then recurrence relations are found by matching together the successive terms of the set of computational traces. Finally, the recurrence relations are transformed into tail-recursive programs using a synthesis theorem [16]. The derecursivation process is thus based on a synthesis technique.

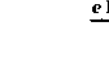
In what follows, we detail an example where the generalized matching techniques used in [16], [20] do not work : the use of signed binary trees will erase this difficulty.

Let us consider the following LISP program where F' has a recursive definition.

$F(x) \leftarrow F'(x,x,nil)$

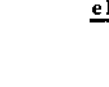
$F'(xx,x,z) \leftarrow$ if atom(cdr (xx)) then cons

else H [xx,x,F'(cdr(xx),x, cons)]



$H'[xx,x,z] \leftarrow$ if atom (cddr(xx)) then cdr(cdr(z))

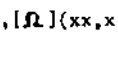
else H[cdr(xx),x, cons]



Let τ_P, be the functional which defines F' and τ_H the functional which defines H. We are interested in the chain of approximations of F (also called computational traces of F), i.e. in the chain $\tau_P^i, [\Omega]\ (x, x, nil)$, where Ω is the function undefined everywhere.

For instance :

$\tau_P^2, [\Omega](xx,x,z) =$ Case atom (cdr(xx)) then



atom (cdr(cdr(xx))) then

else Ω

Let us now call predicates the terms occurring before the then keyword and fragments the terms occurring after the then keyword.

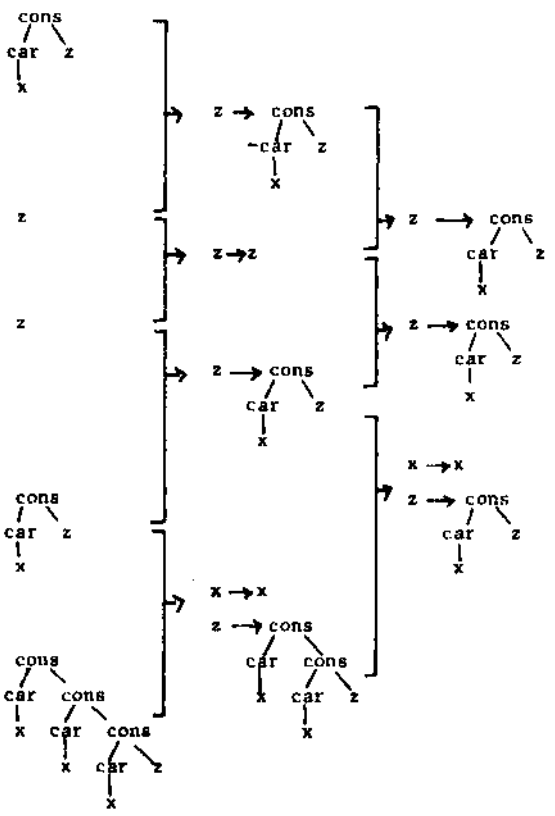
Predicates and fragments are terms of an abstract data type, the so called set of "S-expression", considered here as a term rewriting system by using the axioms from left to right only :

car (cons(x,y)) ⟶ x cdr(cons(x,y)) ⟶ y

We now use the term rewriting system to compute the normal forms of the predicates and the fragments :

$\tau_F^5[\Omega](xx,x,z)$ = case atom (cdr(xx)) then cons(car(x),z)
atom (cd^2r(xx)) then z
atom (cd^3r(xx)) then z
atom (cd^4r(xx)) then cons(car(x), z)
atom (cd^5r(xx)) then cons(car(x), cons(car(x),cons(car(x),z)))
else Ω

We now use our classical technique [11], [12] to infer a program from a sequence of traces, by matching the terms of the sequence. If we take the usual way to match terms, it is clearly impossible to get z starting from cons(car(x),z). But it is possible if we use the frame of the signed trees :



The substitution is now constant. We thus use the theorem 2 of [10] , and get the following tail recursive program :

FF'(xx,x,z) ⟵ If atom (cdr(xx)) then cons(car(x),z)
else FF'(cdr(xx),x,GG(xx,x,z))
GG(xx,x,z) ⟵ If atom (cddr(xx)) then cons(-car(x),z)
else GG(cdr(xx),x,cons(car(x),z)).

Notice that the theorem used says that the function FF' is the least upper bound of the chain of functions obtained by replacing inti [Ω] (xx, x,z) the predicates and the fragments ^{F}by their normal forms, provided that the recurrence relations found by matching the first five fragments are valid for all fragments. So, to prove that F' and FF' are equivalent programs, i.e. compute the same function, one only has to prove that the fragments of F^1 and those of FF' are equivalent. We know that it is true for the first five ones, but we have to prove that it is always true. It is clear that such a proof is not so hard and could be done by an automatic system.

Moreover the proof of equivalence of F' and FF' could be performed by an automatic system such as [1], [2] and [5]. There is in fact a stronger argument for the equivalence of F' and FF' : F' has a kind of polynomial behaviour and we believe in a not yet proved conjecture that polynomial functions on trees are characterized by a finite number of examples (or computational traces) and that they may be obtained from these computational traces by using a "differentiation" algorithm such as ours.

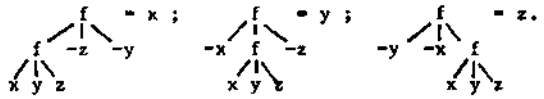
We thu9 have synthesized a non monotonic function (in the sense that the size of the fragments is a non monotonic function) using an auxiliary function : the - function. A good way to deal with this extra function would be to build the term rewriting system described in section III.2 in the LISP interpretor itself in order to compute in a systematic way the normal forms of signed S-expressions. We use in fact a classical LISP and we thus have to process the outputs of functions using the - function.

VIII CONCLUSION

Our goal was both the description of a synthesis system and the design of a mathematical theory which allows us to extend the concept of unification. Our first motivation was synthesis and derecursivation, and we are now still working to find a general algorithm to synthesize non monotonic "tree-polynomial";but we also think that our theory could have other applications and perhaps allow to design new unification algorithms. To reach such a goal, we first have to generalize signed binary trees, which is rather easy :

* For any unary function symbol f, we define a new symbol $\hat{f}$, with the axioms : $\hat{f}(x) = x$; $\hat{\hat{f}} = x$

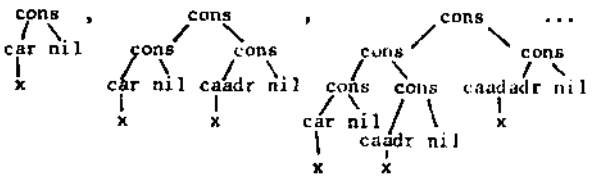
* For any ternary symbol f, we generalise axioms 3 and 4 :



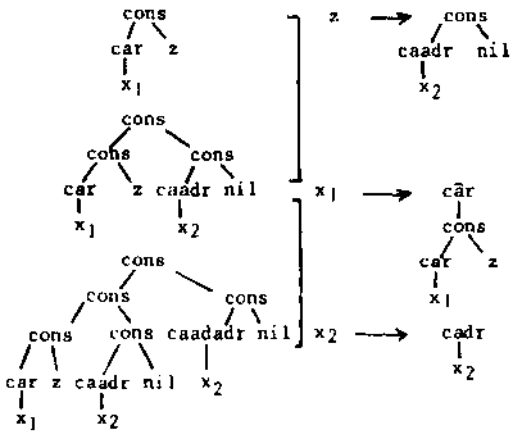
A axiom 2- like is introduced for any function symbol.

These axioms and the deduced rewriting rules system enable us to solve other interesting examples, as the following one.

As in [16], in order to synthesize the LISP program, defined on lists, which maps (A) to (A), (A(B)) to ((A),B), (A(B(C))) to (((A)B)C) and so on we first build the sequence of fragments :



To make the matching successfull, we have to generalize the constant nil in a variable z. Moreover, we have to distinguish occurrences of x which are substitued with different values. We thus obtain the following new set of fragments :



We get now the following tail recursive program :

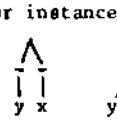
$F(x) \leftarrow G(x,x,nil)$

$G(x_1,x_2,z) \leftarrow$ If atom (cdr x_2) then cons(car(x_1),z)
else G(câr(cons(car(x_1),z)),cadr(x_2),cons(caadr(x_2), nil)).

where câr is some inverse function of car. It is easy to verify that this function F leads to correct computations, provided we have a LISP interpretor which performs the reductions defined by the term rewriting system $\vec{A}$.

A second point is the problem of non linear equations. Solutions of non linear equations on signed binary trees are infinite signed binary trees. But it is not easy to deal with both infinite trees and equations. Moreover results of [3] do not apply easily and must be adapted to our problems.

We finally want to outline the idea of using meta-rules in order to avoid loopings in the KNUTH-BENDIX algorithm : it seems to be a rather general feature which applies successfully in many cases, for instance in group theory, with the rule



. It would thus be interesting to study the looping cases and the corresponding meta-rules.

ACKNOWLEDGEMENTS

We thank the CNRS for financial support.

REFERENCES

[1] Burstall R.M., Darlington J."A transformation system for developping recursive programs". J.ACM 24, 44-67, 1977.

[2] Boyer R., Moore J. "A lemma driven automatic theorem prover for recursive function theory". 5th IJCAI, Boston, 1977.

[3] Courcelle B. "Arbres infinis et systemes d'equations". In R.A.I.R.O. Informatique theorique, vol. 13-1, 31-48, 1979.

[4] DONER, Trees acceptors and some of their applications", J. of computer system science 4, 406-451, 1970.

[5] Huet G., Hullot J.M. "Proofs by induction in equational theories with constructors", Proc. 21th symposium on foundation of computer science, 1980.

[6] Huet G. and Oppen D.C. "Equations and rewrite rules : a survey". In"Formal Languages : Perspectives and open problems". Ed.Book R., Academic Press, 1980.

[7] Huet G. "Resolution d'equation dans les langages d'ordre 1,2 ...". These d'Etat, Universite Paris VII, 1976.

[8] Huet G. "Confluent reductions : abstract properties and applications to term rewriting system". J.ACM 27,4, 797-821, 1980.

[9] Huet G. "A complete proof of the Knuth Bendix completion algorithm", INRIA report, 1980.

[10] Hullot J.M. "Canonical form and unification". Proc. 5th conference on automated deduction, Les Arcs, 1980.

[11] Jouannaud J.P., Kodratoff Y. "Characterisation of a class of functions synthesized from examples by a SUMMERS like method using the BOYER-MOORE-WEGBREIT matching technique". Proc. 6th IJCAI Tokyo, 1979.

[12] Jouannaud J.P., Kodratoff Y. "An automatic construction of LISP programs by transformation of functions synthesized from their input-output behaviour". In P.A.I.S.number 2, Michalski Editor, 1980.

[13] Jouannaud J.P., Kirchner C. and H., Picard M. "Signed trees : an algebraic frame for solving equations on binary trees. Application to LISP program synthesis". Rp. CRIN 80-R-30, 1980.

[14] Jouannaud J.P., Picard M., Kirchner C. and H. "Les arbres signes :un cadre algebrique pour la resolution d'equation dans les arbres". Proc. congres AFCET-Informatique Nancy, 1980.

[15] Knuth D., Bendix P. "Simple word problems in universal algebras". In "Computational problems in abstract algebra". Leech J. ed Pergamon Press, 1970.

[16] Kodratoff Y. and Jouannaud J.P. "Some specifications for the synthesis of lists programs". Proc. of the International Workshop on program construction. Chateau de Bonas, 1980.

[17] Kirchner C. and H., Jouannaud J.P. "Algebraic manipulations as a unification and matching strategy for linear equations in signed binary trees". Rp.CRIN n° 81-R-029, 1981.