Refined strategies
for semantic unification

Pier Giorgio Bosco, Elio Giovannetti and Corrado Moiso
CSELT - Centro Studi E Laboratori Telecomunicazioni
via Reiss Romoli 274 -10148 Torino - Italy

1. Introduction

In the last few years there has been a growing interest on "semantic" unification algorithms based on narrowing or paramodulation, particularly in connection with the attempts to integrate logic and functional programming languages on the basis of first-order-logic with equality. When an equational theory **E** can be put into the form of a canonical rewrite system **R**, the well-known results of [**FA**],[**HU**] ensure that a complete set of E-unifiers of a pair of terms (**t1,t2**), i.e. a complete set of solutions, in the theory **E**, of the equation **?-t1=t2**, can be found by exhaustively searching the space of all possible R-narrowing sequences of the fictitious term **t1=t2**. The practical interest of this method as a possible basis for logic+functional programming languages has been questioned [**GA**], because of its too wide generality, which translates into the huge amount of redundant search usually involved in it. Some refinements which try to reduce the search space have been proposed, from Hullot's one [**HU**] where only so-called "basic" chains are taken into consideration, to more recent ones, based on innermost strategy [**FR1**],[**FR2**], which require additional conditions on the theory **E** besides canonicity, if completeness is to be preserved.

An alternative approach to equation-solving in a theory can be found in the use, under particular conditions, of resolution instead of narrowing, after transformation of the theory into a set of "flat" Horn clauses, i.e. clauses where functional composition has been replaced by logical conjunction of "flat" literals.

This technique was first developed by Brand [**BR**] in the theorem proving domain, while in logic programming it was introduced mainly by [**BB**] and [**TA**], to achieve integration between logic and functional languages.

In this paper we show how this sort of "flat" or "surface" resolution [**CP**] can be applied to obtain a complete E-unification algorithm for canonical theories which do not necessarily present the distinction between functions and constructors (on which both [**BB**] and [**TA**] are based), and how it allows integration between logic and equational programming by means of a single computational mechanism.

The algorithm is shown to be more efficient than unconstrained (basic) narrowing, owing to the fact that the well-known SLD refinement of resolution (for Horn clauses) can be exploited to reduce the search space without losing completeness.

Moreover, a refined narrowing strategy is derived which is equivalent, in a well defined sense, to the SLD-resolution strategy, and thus has the same advantages in terms of redundancy elimination without requiring, as resolution, a previous flattening of the program.

The argument, based on the correspondence that can be established between narrowing sequences and resolution sequences, is roughly the following. As has been above recalled, narrowing is (the core of) a complete unification algorithm for canonical theories; if the program (i.e. the theory) and the equational goal (i.e. the pair of terms to be unified) are transformed into their flat forms, then for every narrowing chain there is a corresponding resolution chain, so the use of (linear) resolution instead of narrowing preserves completeness; it also preserves soundness, because the flattened program is logically equivalent to the original one. But the flattened program is a set of definite Horn clauses, and therefore SLD strategy - where we recall that S stands for *selection* - can be applied without losing completeness: which means that at each step one literal in the goal is selected, and only the further paths which start from resolution against that literal are explored, unlike in (unrestricted) narrowing, where all the possible choices of the subterm to be narrowed are explored.

Finally, this procedure is translated back into a narrowing algorithm where at each step one subterm is selected, and other subterm choices for the same step are not taken into consideration in the search.

The paper is organized as follows: in section 2 we briefly recall the well-known E-unification algorithm based on narrowing; in section 3 we describe the optimized strategy based on flat resolution; in section 4 we compare it with (unrestricted) narrowing; in section 5 we derive the corrispondingly refined narrowing strategy; in section 6 possible applications to the domain of logic/functional programming are considered; in section 7 we give some non-exhaustive indication of the related work in the field; finally, in section 8 we draw some conclusions.

We will use, among others, the following standard notations:

- occurrences are represented by sequences, possibly empty, of naturals;
- **t/u** is the subterm at the occurrence **u** of **t**;
- **t[u<--r]** is the term **t** with the subterm at the occurrence **u** replaced with **r**.

2. The traditional E-unification algorithm based on narrowing.

Let the equational theory **E** consist of a canonical (i.e. confluent and terminating) rewrite system **R**, and **(t1,t2)** be the pair of terms for which a *complete* set of E-unifiers has to be computed, i.e. **?- t1=t2** is the equation that has to be solved in **E** - recall that a set **S** of E-unifiers of **(t1,t2)** is *complete* iff for every unifier σ there is in **S** a unifier σ' such that $\sigma =_E \alpha\sigma'$, i.e. a unifier σ' E-equal to σ or "more general" than σ ($\sigma' \leq_E \sigma$).

Then the narrowing-based E-unification algorithm is, informally, the following (where **eq** is a functor not occurring in the terms of **R**, which represents the equality symbol "="):

unify(t1,t2,R) =
current-goal := eq(t1,t2)
current-subst := empty-subst
1. execute **don't-know-non-deterministically** 2 or 3
2. let eq(t1',t2') = current-goal
if t1' and t2' syntactically unify with mgu σ
then (σ current-subst) is a E-unifier of t1 and t2
3. select **don't-know-non-deterministically**
a non-variable subterm **current-goal/u** of current-goal
and a rule **l-->r**
s.t. **current-goal/u** and **l** syntactically unify;
let σ = mgu (current-goal/u, l)
current-goal := σ(current-goal[u<--r])
current-subst := σ(current-subst)
goto 1.

The first refinement, introduced by [**HU**], makes use of the notion of *basic* narrowing, i.e., roughly, a narrowing that may only reduce, at each step, a subterm whose outermost functor is a descendant of (a subterm at) a non-variable occurrence of the initial term.

If **O'(t)** is the set of non-variable occurrences of **t**, and **occ** is the set of the "narrowable" occurrences, the refined algorithm is:

unify1(t1,t2,R) =
current-goal := eq(t1,t2)
current-subst := empty-subst
occ:= O'(eq(t1=t2))
1. execute **don't-know-non-deterministically** 2 or 3
2. let eq(t1',t2') = current-goal
if σ is the mgu of t1' and t2'
then (σ current-subst) is a E-unifier of t1 and t2
3. select **don't-know-non-deterministically**
a subterm **current-goal/u** of current-goal with **u** belonging to **occ**
and a rule **l-->r**
s.t. **current-goal/u** and **l** syntactically unify;
let σ = mgu (current-goal/u, l)
current-goal := σ(current-goal[u<--r])
current-subst := σ(current-subst)
occ:=(occ - {v | u≤v, i.e. u is a prefix of v})∪{u.v | v is in O'(r)}
goto 1.

Both algorithms explore all possible choices, respectively, of non-variable subterms and of subterms at basic occurrences.

In the following, narrowing will always be intended as basic narrowing, to which additional refinements will be added.

3. Flat resolution

Narrowing and resolution are two inference rules based on the same sort of mechanism, namely unification between a "piece" of the goal (a subgoal in resolution, a subterm of an equation side in narrowing) and a "piece" of a rule (a clause head in resolution, the lhs of a rewrite rule in narrowing), then application of the unifier both to the whole goal and to the whole rule, and finally creation of a goal by "sticking together" (the instances of) the two "remaining pieces" of the goal and the "rule".

The main difference between narrowing and resolution lies in that the former, to try unification, takes from the goal any subterm of the selected literal, while the latter only takes the whole litteral and so the whole argument-terms.

A transformation of narrowing into resolution is then possible if - both in goal and in rules - terms are "flattened out", i.e. subterms are unnested so that resolution is allowed to apply unification to them; to this end, a new auxiliary variable is introduced for each subterm, and all the new "flat" subterms become conditions, i.e. the body of an equational Horn clause, so that in later steps resolution can apply unification to them too.

Actually, for resolution to be able to mimic narrowing, the lhs's of the rules need not be flattened, because after one of them is unified with the goal's subterm being reduced, its occurrence is replaced, in narrowing, by the respective rhs, and so it is no longer available for reduction in the next steps.

In conclusion, the flattening procedure, unlike the ones described in [**BB**] and [**TA**], on the one hand does not leave constructor-terms unflattened, because the distinction between constructors and functions is not necessarily present, on the other hand, owing to the above remark, does not flatten the lhs's of the rules even if nested functors are present.

The algorithm, for rewrite rules **l-->r** and goals **?-t1=t2** is then, with a sloppy but hopefully self-explaining notation, the following:

flat(l-->r) = if r is a variable (also occurring in l)
then l=r
else l=Z :- flatterm(r,Z) (where Z is a new variable)
flatgoal(t1=t2) = flatterm(t1,Z),flatterm(t2,Z) (where Z is a new variable)
flatterm(f(t1,...,tn),Z) =
let $t_{i1},...t_{iq}$ be the non-variable arguments
let Zi=ti if ti is a variable
Zi=Xk if ti is t_{ik} in
f(Z1,...,Zn)=Z,flatterm(t_{i1},X1),....flatterm(t_{iq},Xq)

If **R** is a (canonical) rewrite system, the corresponding "flat resolution" program $\mathbf{R_{flat}}$ is, of course, the set of Horn clauses obtained by flattening all the

rewrite rules of **R**.

Every goal submitted to the system has first to be flattened out; then it can be solved by means of the sole use of SLD-resolution as if the equality symbol were an ordinary predicate, provided that the clause **X=X** is added to the program $\mathbf{R_{flat}}$: resolution against **X=X** corresponds to the final step of the narrowing-based unification algorithm, where the two sides of the goal are syntactically unified.

The transformation is sound because the trasformed program is a logical consequence of the original one and of the equality axioms.

For the proof of completeness of flat resolution, we need the following

Lemma: If $\mathbf{t}\text{-->}^*_R \mathbf{s}$ through a basic derivation, then there exists a substitution σ, whose domain is the set of the new variables introduced in flattening the term **t**, such that $\sigma\mathbf{Z}=\mathbf{s}$, and

$R_{flat} \cup \{X=X\} \models \sigma$ flatterm(t,Z).

The lemma can be proved by induction on the length of the derivation

$t=t0 \text{-->}_R t1 \text{-->}_R \ldots\ldots\ldots \text{-->}_R tn=s.$

We are then able to prove the

Proposition (completeness of flat resolution):

If $\mathbf{R} \models \sigma\mathbf{t1}=\sigma\mathbf{t2}$, then there is an SLD-resolution computation which solves the goal **?-flatgoal(t1=t2)** with respect to the program $\mathbf{R_{flat}} \cup \{\mathbf{X=X}\}$ and yields a solution τ such that $\tau \leq_R \sigma$, i.e. $\mathbf{R} \models \sigma =_R \alpha\tau$, where α is any (possibly empty) substitution.

Proof:

Owing to the canonicity of R, every substitution σ can be normalized to a $\sigma_{nor} = \{X:=q \mid q$ is the normal form of $\sigma(X)\}$. From $\mathbf{R} \models \sigma\mathbf{t1}=\sigma\mathbf{t2}$ follows then

$\mathbf{R} \models \sigma_{nor}\mathbf{t1}=\sigma_{nor}\mathbf{t2}$; so there must exist a term **t** such that :

$\sigma_{nor}\mathbf{t1}\text{-->}^*_R\mathbf{t}$ and $\sigma_{nor}\mathbf{t2}\text{-->}^*_R\mathbf{t}$

where the two derivation chains -->^*_R are basic (see [HU] lemma 3). Then, by our previous lemma, there are two substitutions $\rho\mathbf{1}$ and $\rho\mathbf{2}$ such that:

$\mathbf{R_{flat}} \cup \{\mathbf{X=X}\} \models \rho\mathbf{1}(\mathbf{flatterm}(\sigma_{nor}\mathbf{t1},\mathbf{Z}))$ with $\rho\mathbf{1}(\mathbf{Z})=\mathbf{t}$ and

$\mathbf{R_{flat}} \cup \{\mathbf{X=X}\} \models \rho\mathbf{2}(\mathbf{flatterm}\ (\sigma_{nor}\mathbf{t2},\mathbf{Z}))$ with $\rho\mathbf{2}(\mathbf{Z}) = \mathbf{t}$

putting $\rho = \rho 1 \cup \rho 2$ we have

$\mathbf{R_{flat}} \cup \{\mathbf{X=X}\} \models \rho(\mathbf{flatterm}\ (\sigma_{nor}\mathbf{t1},\mathbf{Z}),\mathbf{flatterm}\ (\sigma_{nor}\mathbf{t2},\mathbf{Z}))$

Due to the normalization of σ_{nor}, we can write:

$\mathbf{R_{flat}} \cup \{X=X\} \models \rho\ \sigma_{nor}$**(flatterm (t1,Z),flatterm (t2,Z))**

i.e. $\mathbf{R_{flat}} \cup \{X=X\} \models \rho\ \sigma_{nor}$**flatgoal(t1=t2)**

Completeness of SLD-resolution ensures that there is a resolution sequence starting from **?-flatgoal(t1=t2)** which yields a result $\tau' \leq \rho\sigma_{nor}$; then

$\tau = \tau'/Var(t1=t2) \leq \sigma_{nor} =_R \sigma$, i.e. $\tau \leq_R \sigma$, q.e.d.

The lemma completes the proof of the logical equivalence between the transformed and the original program and corresponds to the theorem in [**TA**]; the proposition is the proof of the completeness of the resolution on the flattened program as E-unification algorithm.

As a matter of fact, for every basic narrowing derivation there is a corresponding resolution derivation with the same result: a narrowing step performed, with the application of the rule **l --> r**, on the subterm (at the occurrence) **u** in the goal, translates into the resolution of the flat literal corresponding to the term **u** against the clause **flat(l-->r)**, and into the subsequent resolutions against **X=X** of all the literals corresponding to the subterms of **u** (i.e. to the occurrences **u.v**), because in the narrowing sequence these subterms disappear, the term **u** being replaced by the instance of the rhs of the rewrite rule, or become no longer narrowable; syntactical unification between the two sides of the equation translates in successive resolution against **X=X** of all the literals in the goal.

This translation is possible because at each narrowing step:

- there is a one-to-one correspondence between the elements of the set **occ** of narrowable occurrences in algorithm **unify1**, and the literals in the flattened goal.
- there is a one-to-one correspondence between **current-goal** in the algorithm **unify1**, and the flattened goal; in fact, the goal is always of the form **?-L,t1'=Z,t2'=Z**, where **L** is a set of literals, and there is a substitution σ, such that $\{X=X\} \models \sigma L$ and **current-goal=eq(**σ**t1',**σ**t2')**.

Thus every solution found by narrowing is found by flat resolution as well, and that explains again why we were able to extend to the latter the completeness result of [**HU**] holding for the former.

Moreover, the construction - which we will sketch in next section - of an onto-map from the set of all the basic narrowing sequences to the set of all the flat SLD-resolution sequences ensures, among other things, that no unsound solutions can be derived by SLD-resolution (or by generic resolution, due to the completeness of SLD strategy).

4. Narrowing and flat SLD-resolution

It is well known that, in case of Horn clauses, the SLD refinement of resolution preserves completeness, whatever be the rule for selecting the literal to be resolved. Once fixed this rule, there is an onto-map from a subset of the set of all linear resolution derivations, including all successful derivations, to the set of all SLD derivations: the inverse image of an SLD derivation is the set of all the derivations that only differ in the order of selection of the literals.

Let us see now what the meaning of this refinement is with respect to our translation of narrowing into resolution.

We start by mapping the set of successful narrowing sequences onto the set of successful resolution sequences: the inverse image of an SLD-resolution derivation is the set of all the narrowing derivations that satisfy the condition that a narrowing step with the rule **l-->r** is performed at the occurrence **u** of the goal iff the literal corresponding to **u** is resolved against clause **flat(l-->r)**.

Given for example the program

(unflattened)	(flattened)
f(a) --> a	f(a)=X :- a=X
g(X)-->X	g(X)=X

and the goal

?- f(X)=g(Y)	?- f(X)=Z,g(Y)=Z

the SLD-resolution derivation

?- f(X)=Z,g(Y)=Z ?- a=Z,g(Y)=Y ?- g(Y)=a ?-

yielding the solution {X:=a;Y:=a}, is the image of two narrowing sequences:

f(X)=g(Y) -»-> a=g(Y) -»-> a=Y result: {X:=a;Y:=a}

f(X)=g(Y) -»-> f(X)=Y -»-> a=Y result: {X:=a;Y:=a}

So, there are less successful flat SLD-resolutions than successful narrowing chains, without losing completeness.

If the selection rule adopted is the *innermost* rule, this map can be surjectively extended to the set of all the narrowing sequences; of course, in correspondence to failing (finitely or infinitely) narrowing sequences there will be failing resolution sequences.The innermost rule consists in always selecting an *innermost literal* , i.e. a literal of the form f(t1,..,tn)=t , where the ti's are not variables introduced during flattening.

More generally, the above map extension is possible whenever the literals selected are always resolved without variables introduced in the lhs of the literal during flattening being bound; for example, an *outermost* selection rule, like the one used in LEAF [**BB**], would do as well. On the contrary, for an arbitrary selection rule the above extension would not work, because in general there would be failing resolution sequences that do not correspond to any narrowing, as in the following example.

Given the program

(flattened)	(unflattened)
f(a)=Z :- c=Z	f(a)-->c
f(b)=Z :- c=Z	f(b)-->c
a=Z :- b=Z	a --> b

and the goal

?- a=X, f(X)=Z, f(R)=Z	?- f(a)=f(R)

the SLD sequence

?- a=X, f(X)=Z, f(R)=Z

?- b=X, f(X)=Z, f(R)=Z

?- b=a, c=Z, f(R)=Z

failure

does not correspond to any narrowing sequence, because after the step

f(a)=f(R) -»-> f(b)=f(R)

the rule f(a)-->c cannot be applied to f(b) anymore. This kind of failure in resolution is actually the counterpart of a failure in trying unification between the term **f(b)** of the goal and the lhs **f(a)** of the rule **f(a)-->c**.

The comparison between the two algorithms shows that in SLD-resolution the number of successful computations leading to a same solution is smaller than in (unrestricted) narrowing, because all the narrowing derivations that only differ in the selection order of subterms are mapped into one SLD sequence corresponding to one selection order.

For example, given the program

(unflattened)	(flattened)
f(X) --> X	f(X) = X
h(h(X)) --> X	h(h(X))=X
h(a) --> a	h(a)=Z :- a=Z

if the goal is

?-f(h(R))=h(h(a)) ?- h(R)=X,f(X)=Z,h(Y)=Z,h(W)=Y,a=W

flat SLD-resolution (with an arbitrary selection rule) and (basic) narrowing compute respectively the following sets of solutions:

Computed value of R	number of successful paths		
	SLD	narrowing	basic-narrowing
a	3	20	11
h(a)	3	19	11
h(h(a))	1	6	2
h(h(h(a)))	1	2	1

The same property holds, in case of an innermost rule, for failing derivations too, as shown in the following example:

given the program

(unflattened)	(flattened)
f(X) --> g(X,X)	f(X) = Z :- g(X,X) = Z
h(a) --> a	h(a) = Z :- a = Z

if the goal is

?- f(h(R))=g(a,h(a)) ?- h(R)=X,f(X)=Z,a=Y,a=W,h(W)=T,g(Y,T)=Z

flat SLD-resolution with an innermost selection rule and (basic) narrowing computes the following answers:

	number of successful and failing computations	
	SLD	basic-narrowing
{X:= a}	1	3
failure	7	9

The innermost selection rule can be implemented by means of the leftmost selection rule (selection of the left-most literal, as in Prolog), provided that the "compilation" step puts the flat literals in the right order:

```
flat(l-->r) =  if r is a variable (also occurring in l)
                  then  l=r
                   else  l=Z :- flatterm(r,Z)   (where Z is a new variable)
```

flatgoal(t1=t2) = flat(t1,Z),flat(t2,Z) (where Z is a new variable)
flatterm(f(t1,...,tn),Z) =
let t_{i1},...t_{iq} be the non-variable arguments
let Zi=ti if ti is a variable
Zi=Xk if ti is t_{ik} in
flatterm(t_{i1},X1),....flatterm(t_{iq},Xq), f(Z1,...,Zn)=Z

5. A complete "selection" narrowing strategy

The above comparison between the two algorithms, which resulted in the estimation of a better efficiency of SLD-resolution with respect to the usual presentations of narrowing, is actually somewhat unfair, because the same "selection" strategy can be applied to narrowing too, so producing a refined, but still complete, algorithm which exactly "mimics" SLD-resolution (with an innermost selection strategy), in the sense that there is a one-to-one correspondence between the set of all the SLD sequences and the set of the narrowing sequences which obey to this refined strategy:

unify2(t1,t2,R) =
current-goal := eq(t1,t2)
occ := O'(eq(t1,t2)) - {<>}
current-subst := empty-subst
1. if occ = ∅ then let eq(t1',t2') = current-goal in
if σ is the mgu of t1' and t2'
then (σ current-subst) is a E-unifier of t1 and t2
else goto 2.
2.**don't-care-nondeterministically-select** from **occ**
an innermost occurrence **u**
(i.e. an occurrence which is not the prefix of any other occurrence in occ)
and **don't-know-nondeterministically-execute** 3 or 4.
3. occ := occ - {**u**}
goto 1.
{corresponding to resolution with X=X}
4. **don't-know-nondeterministically-select l-->r** from R
s.t. **l** and **current-goal/u** syntactically unify;
let σ = mgu (**current-goal/u** , **l**) in
current-goal:= σ(current-goal[**u<--r**])
current-subst:=σ(current-subst)
occ:= (occ-{u}) ∪ {u.v| v is in O'(r)}
goto 1.
{corresponding to resolution with flat(l-->r)}

The algorithm **unify2** achieves exactly the same elimination of redundant computations with respect to unconstrained (basic) narrowing as flat SLD-resolution does, but without requiring the presence of the "compilation" phase with the associated introduction of new variables.

The existence of a one-to-one mapping between innermost-SLD derivations and the derivations obtained with this new algorithm can be easily proved by induction on the length of the sequences.

The algorithm **unify2** is sound because it explores only a subset of the paths followed by **unify1**; so the correctness of flat resolution is confirmed.

Completeness, of course, is direct consequence of completeness of SLD-resolution. An important point to remark is that completeness is only guaranteed if, as in point 3 of the above algorithm, among all the possible narrowing reductions of the subterm selected, also the "null" reduction is taken into account, i.e., also the paths are explored where the subterm selected at each step, instead of being reduced, is excluded from the possible reducible terms for all the subsequent subpaths. That corresponds to resolution against **X=X,** which is necessary in general, although it may be disposed of in particular cases, such as the one of *theories with constructors* : if a distinction is operated between data constructors, which never rewrite, and actual function symbols, and if moreover the actual functions are "everywhere defined", then if the flattening algorithm is modified so as not to apply to data terms, resolution against **X=X**, and therefore "null" narrowing, become useless.

The following is the Prolog program implementing **unify2**, which we are using to carry out comparative experiences:

- **preprocessing on rewrite rules to compute the set of non variable occurrences of rhs:**

```
init :- L-->R, occ(R,OR),assertz(rwr(L,R,OR)),fail.
init.                          ; OR denotes the set O'(R)
occ(T,0)   :- var(T),!.
                               ; 0 indicates that the corresponding term is not to be rewritten
occ(T,[1|Ann]) :- T =.. [F|ARG], occ1(ARG,Ann).
                               ; 1 indicates that the corresponding term is to be rewritten
occ1([],[]).
occ1([TH|TT],[A|Ann]) :-occ(TH,A),occ1(TT,Ann).
```

- **refined narrowing strategy:**

```
unify(T,S):-occ(T,OT),occ(S,OS),!,narrred(T,OT,TR),
                                   narrred(S,OS,SR),TR=SR.
narrred(T,0,T) :- !. ; 0 denotes that this is a term not to be narrowed
narrred(TC,[1|Oarg],TR) :- TC =.. [F|Arg], narrarg(Arg,Oarg,Narg),
                           ; before narrowing a term, its arguments must be narrowed
                                   TC1 =.. [F|Narg],
                           ; now TC1 is an innermost term
                                   narrterm(TC!,TR).
narrterm(T,T).     ; clause corresponding to step 3 of unify2
narrterm(TC,TR) :- rwr(TC,R,OR),narrred(R,OR,TR).
                           ; clause corresponding to step 4 of unify2
```

```
narrarg([],[],[]).
narrarg([A1|Arg],[O1|Oarg],[AR|ArgR]) :- narrred(A1,O1,AR),
                                         narrarg(Arg,Oarg,ArgR).
```

6. Application to Logic/Functional languages

The algorithms presented in the above sections can be most naturally applied to logic programming with equality. Consider a language like the one proposed in [**GM**], for which a program consists of a set of Horn clauses **p:-b1,...,bn**, where the bi's are either predicates or equality, and of an equational theory described by a canonical rewrite system: its execution strategy is the same as the one used for Horn clause programs without equality (Prolog), with syntactic unification replaced by a complete E-unification algorithm. The flattening procedure can be easily extended to deal with this kind of programs (a similar procedure is found in [**TA**]):

flatrule(l-->r) = if r is a variable (also occurring in l)
then l=r
else l=Z :- flatterm(r,Z) (where Z is a new variable)

flatterm(f(t1,...,tn),Z) =
let $t_{i1},...t_{iq}$ be the non-variable arguments
let Zi=ti if ti is a variable
Zi=Xk if ti is t_{ik} in
flatterm(t_{i1},X1),....flatterm(t_{iq},Xq), f(Z1,...,Zn)=Z

flatclause(p(t1,...,pn) :- body) =
let $t_{i1},...t_{iq}$ be the non-variable arguments
let Zi=ti if ti is a variable
Zi=Xk if ti is t_{ik} in
p(Z1,...,Zn) :- flatterm(t_{i1},X1),....flatterm(t_{iq},Xq),flatbody(body)

flatboby(b1,...,bn) = flatgoal(b1),...,flatgoal(bn)

flatgoal(t1=t2) = flat(t1,Z),flat(t2,Z) (where Z is a new variable)

flatgoal(p(t1,...,tn)) =
let $t_{i1},...t_{iq}$ be the non-variable arguments
let Zi=ti if ti is a variable
Zi=Xk if ti is t_{ik} in
flatterm(t_{i1},X1),....flatterm(t_{iq},Xq), p(Z1,...,Zn)

Then, for every goal **G**, a complete set of solutions is found by normal SLD-resolution (provided that the clause **X=X** is added, as usual, to the flattened program).

Alternatively, the selection narrowing algorithm could be used to replace syntactical unification in the resolution procedure running on the original program.

The use of the proposed algorithms reduces the search space both in the unification phase and, by reducing the number of duplicated unifiers, in the overall

resolution procedure. Flat SLD-resolution has the advantage of not introducing a different computational mechanism, at the price of a previous compilation stage.

With respect to the amount of computation, flat SLD-resolution and selection narrowing are not equivalent in this context: the first unification algorithm avoids term duplications, and so duplications of narrowing steps introduced by resolution in the second unification algorithm.

This improvement is comparable with the one introduced by basic-chains in the narrowing algorithm.

In spite of this reduction of the search space, completeness is not lost: it is easy, but tedious, to prove that at least all the solutions in normal form (or substitutions more general than them) are computed: starting by a resolution with selection narrowing which computes one of the normalized solutions, it is possible to build a resolution chain where all the occurrences of the same term are narrowed in the same way: this sequence, in turn, is shown to be correspondent to a flat SLD-resolution sequence.

A transformation based on flattening has been used to implement a higher-order functional plus logic programming language [**BG**].

7. Related work

Related work on the subject has been constantly increasing during the last few years. Maybe the closest proposals to the kind of approach here advocated are LEAF [**BB**], already quoted above, and SLOG [**FR2**].

In the LEAF language, based on full Horn-clause logic with equality, the flat-resolution approach is adopted, along with a demand-driven selection strategy which allows to deal with infinite data.

SLOG, on the other hand, adopts the innermost-narrowing approach; as remarked in section 5, since the execution mechanism of SLOG does not admit a step equivalent to "null" narrowing, completeness is only preserved as long as the functions are "everywhere defined"; being the language based on conditional rewrite rules, the interpretation algorithm is also related to the one proposed in [**KA**] (conditional narrowing).

Other similar approaches are those of [**RE**],[**SU**], which use an outermost constructor-driven narrowing strategy: the algorithm in [**SU**] is incomplete, the E-unification algorithm in [**RE**] is based on a notion of equality, called "continous equality", different from the more usual algebraic equality considered in our work. As the pure outermost strategy is incomplete for algebraic equality, an almost-outermost E-unification algorithm was proposed in [**MM**].

The above languages, all based on theories with constructors, do not allow rules whose lhs's contain functional composition (e.g. f(f(X)) --> g(X)), which on the contrary are permitted in the algorithms we have presented. This limitation, which may be undesirable when working with specifications, is sensible if the systems are used as programming languages.

8. Conclusions and future work

As has been remarked in the introduction, the problem - which has a definite importance in connection with attempted extensions of logic programming - of moving from syntactical to so-called "semantical" unification, or E-unification, even though theoretically rather well understood, does not present, in practice, easy solutions. Even if the stepping from a decidable to a semi-decidable procedure is accepted because inherent in the nature of the problem, there still remains a lot of inefficiency researchers who have tried to use narrowing in programming or specification languages [**GA**],[**BE**] including ourselves, feel uncomfortable about.

The algorithms presented in this paper are a step towards the discovery of methods more suitable to practical implementability. Our work also shows how narrowing and resolution, though being quite distinct inference rules, are actually two different versions of a common underlying mechanism, so that refinements found out for resolution can be applied to narrowing as well. The worse behaviour of the usual narrowing-based unification algorithms is therefore merely the result of these straightforward optimizations not being applied to them; when this handicap is eliminated, narrowing becomes as efficient as SLD-resolution.

Nevertheless, what is gained over flat SLD-resolution with this optimized narrowing strategy is a minor advantage, namely the absence of a quite simple compilation phase; that, in our opinion, does not seem to counterbalance the benefit of having just one inference rule, resolution, i.e. one computational model, for which efficient implementations are already available and more efficient ones based on specially-conceived architectures are being developed.

In this paper we have shown how an innermost selection strategy for flat SLD-resolution can easily be compiled; but an innermost selection rule in E-unification has the same drawbacks of innermost strategy in reduction (i.e. call by value): i.e. it performs unnecessary computations. We are therefore trying to define a complete "almost-outermost" strategy for flat SLD-resolutions, where "outermost" literals are selected, so that resolution with clauses different from X=X is only applied on request. This strategy, unlike the innermost case, cannot be implemented by means of a trivial compilation, because the atom selection order is not known statically.

Acknowledgement

This work has been partially sponsored by EEC through ESPRIT Project 415 "Parallel Architectures and Languages for Advanced Information Processing - a VLSI-directed Approach".

9. References

BB **R. Barbuti, M. Bellia, G. Levi** and **M. Martelli**, LEAF: A language which integrates logic, equations and functions. In *Logic Programming: Functions, Relations and Equations* , D. DeGroot and G. Lindstrom, Eds.

(Prentice-Hall, 1985), 201-238.

BE **D. Bert,** personal communication.

BG **P.G. Bosco** and **E. Giovannetti**, IDEAL: an Ideal DEductive Applicative Language. *Proc. 1986 Symp. on Logic Programming* (IEEE Comp. Society Press, 1986), 89-94.

BR **D. Brand**, Proving theorems with the modification method. *SIAM J. Comput.* 4 (1975), 412-430.

CP **P.T. Cox** and **T. Pietrzykowski**, Surface deduction: a uniform mechanism for Logic Programming. *Proc. 1985 Symp. on Logic Programming* (IEEE Comp. Society Press, 1985), 220-227.

FA **M. Fay**, First order unification in an equational theory. *Proc. 4th Workshop on Automated Deduction* (1979), 161-167.

FR1 **L. Fribourg**, A narrowing procedure for theories with constructors. Proc. 7th Int. Conf. on Automated Deduction, *LNCS 170* (Springer Verlag, 1984), 259-301.

FR2 **L. Fribourg**, SLOG: A logic programming language interpreter based on clausal superposition and rewriting. *Proc. 1985 Symp. on Logic Programming* (IEEE Comp. Society Press, 1985), 172-184.

GA **H. Gallaire**, Logic programming: further developments. *Proc. of the 1985 Symp. on Logic Programming* (IEEE Comp. Society Press, 1985), 88-96.

GM **J.A. Goguen** and **J. Meseguer**, Equality, types, modules and (why not?) generics for logic programming. *J. Logic Programming 1* (1984), 179-210.

HU **J.-M. Hullot**, Canonical forms and unification. *Proc. 5th Conf. on Automated Deduction, LNCS 87* (Springer Verlag, 1980), 318-334.

KA **S. Kaplan**, Fair conditional term rewriting systems: Unification, termination and confluence. *Technical Report no. 194* , University ofOrsay (1984).

MM **A. Martelli, C. Moiso** and **G.F. Rossi**, An algorithm for unification in equational theories. *Proc. 1986 Symp. on Logic Programming* (IEEE Comp. Society Press, 1986), 180-186.

RE **U.S. Reddy**, Narrowing as the operational semantics of functional languages. *Proc. 1985 Symp. on Logic Programming* (IEEE Comp. Society Press, 1985), 138-151.

SU **P.A. Subrahmanyam** and **J.-H. You**, FUNLOG = functions + logic: A

computational model integrating functional and logic programming. *Proc. 1984 Int. Symp. on Logic Programming* (IEEE Comp. Society Press, 1984), 144-153.

TA **H. Tamaki,** Semantics of a logic programming language with a reducibility predicate. *Proc. of the 1984 Symp. on Logic Programming* (IEEE Comp. Society Press, 1984), 259-264