SEKI-REPORT

Artificial
Intelligence
Laboratories

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

Efficient AC-Matching Using Constraint Propagation

B. Gramlich & J. Denzinger

SEKI Report SR-88-15

Efficient AC-Matching Using Constraint Propagation

Bernhard Gramlich, Jörg Denzinger
FB Informatik, Universität Kaiserslautern,
Pf. 3049, D-6750 Kaiserslautern,
West Germany

Abstract:

We present a new approach to associative-commutative (AC-) pattern matching using a technique of global constraint propagation which in many cases enables us to drastically cut down the search space for solutions.

Given a conjunction of simplified non-trivial AC-matching problems the main idea of our method consists in deducing and propagating constraints for possible variable assignments from **all** problems instead of choosing only **one** problem for processing next. Thus many failure branches of the search tree for solutions can be cut without expanding them. The control strategy for branching is designed such that nodes with a small branching degree are preferred. Incorporating some additional optimizations we get a new AC-matching algorithm which is especially well-suited for certain non-linear patterns and for a systematic early detection of unsolvability.

Moreover we point out the advantages of using unique ordered normal forms for AC-equivalent terms based on a special class of orderings on the function symbols and the variables.

Finally we sketch how our constraint propagation approach for AC-matching can be generalized to other kinds of E-matching and E-unification problems.

0. Introduction

Associative-commutative (AC-) matching and unification are fundamental problems within many fields and applications of automated reasoning including automated theorem proving, term rewriting theory, program verification and synthesis, specification analysis etc. From a practical point of view it is very important to provide efficient algorithms for these problems, because AC-unification and in particular AC-matching are in general very often performed operations, for example in systems for equational completion modulo an underlying theory consisting of AC-axioms for some operators. Of course one might use AC-unification algorithms to solve AC-matching problems but this has turned out to be too expensive in practice.

In [BeKaNa85] it has been shown that the AC-matching problem is NP-complete but has a polynomial upper bound for linear patterns. Moreover it is conjectured there that the problem remains in polynomial time as long as it is possible to put a bound on the number of times every variable can occur in a pattern. This is a strong hint that it might be very well worth-while looking for algorithms that are efficient in practical cases.

The main approaches for AC-matching presented in the literature are given by Hullot ([Hu79], [Hu8o]) and Mzali ([Mz83]. Thus, when describing our algorithm and its implementation we shall also point out the similarities and differences w.r.t. these former approaches.

The rest of this paper is organized as follows. In section 1 basic definitions and notations are given. The subproblem of AC-equivalence is dealt with in section 2. The central part of this paper in section 3 presents our AC-matching approach and clarifies the connection with the approaches of Hullot and Mzali. In section 4 we sketch how the idea of constraint propagation can be applied to other kinds of equational problems whereas the last section consists of concluding remarks and perspectives for future work in this and related fields.

1. Definitions

Let F be a finite set of function symbols of fixed arity and X be a denumerable set of variables disjoint from F. By F_i we mean the subset of F containing all function symbols of F with arity i. The set of all terms constructed over F and X as usual is denoted by T(F,X) or T for short. By V(t) we denote the set of all variables occurring in a term t. The size of t, denoted |t|, is the number of occurrences of function and variable symbols in t and top(t) denotes the top-level symbol of t. A substitution σ is a mapping from X to T that is the identity almost everywhere on X. Its unique homomorphic extension to an endomorphism on T (regarding T as a F-algebra) is also denoted by σ. The domain of a substitution σ is $D(\sigma):=\{x\in X|\sigma(x)\neq x\}$, the set of introduced variables of σ is $I(\sigma):=\bigcup_{x\in D(\sigma)}V(\sigma(x))$. Given F and X, the equational theory $=_E$ induced by a set E of equations over T is defined to be the smallest congruence on T containing E and closed under substitution. Two terms s and t are said to be E-equivalent iff $s=_E t$ holds. This notion is extended to substitutions by defining $\sigma=_E\sigma'$ iff $\forall t\in T:\sigma(t)=_E\sigma'(t)$ which is easily shown to be equivalent to $\forall x\in D(\sigma)\cup D(\sigma'):\sigma(x)=_E\sigma'(x)$. A substitution σ is an E-match from s to t iff $\sigma(s)=_E t$. The set of all E-matches from s to t, restricted to V(s), is denoted by M(s,t). This notion is extended to sets of

term pairs Γ via $M(\Gamma):=\{\sigma \mid \forall (s,t)\in\Gamma:\sigma\in M(s,t)\}$.

In the following we always assume F to be partitioned into $F=F_{AC}\cup F_{NAC}$, where F_{AC} is a non-empty set of binary operators. The set E of equations consists exactly of the AC-axioms for the operators of F_{AC}, i.e. $E := AC := \{f(x,y) =f(y,x),\ f(f(x,y),z)=f(x,f(y,z)) \mid f\in F_{AC}\}$.

2. AC-Equivalence

Since AC-equivalence testing is a basic operation frequently used in any AC-matching algorithm it is crucial to provide efficient solutions for that problem. Let us summarize therefore the main ideas and results concerning AC-equivalence in order to have a solid basis for presenting our solution to the AC-matching problem and discussing implementation issues (cf. [Hu79], [Hu80], [Mz83]).

A brute force method for deciding AC-equivalence of two terms s and t would consist in generating the (finite) equivalence class $[s]_{AC}$ of s and testing whether $t\in[s]_{AC}$ or vice versa. Clearly this is very inefficient in general. The completion method for generating a canonical term rewriting system equivalent to AC in order to get unique normal forms is not applicable since the commutativity axioms in AC cannot be oriented into rewrite rules without loosing the finite termination property. Nevertheless the form of the AC-axioms suggests another method for deciding AC-equivalence by abstracting from these properties. This is described in the following.

Proposition 1

For any $s, t \in T$, if $s =_{AC} t$ then $|s| = |t|$ and $top(s) = top(t)$.

Proof: Follows obviously from the form of the axioms AC. ■

Definition 1

Let us denote the set of terms constructed from $s_1,\ldots,s_n \in T$ using only the (binary) operator $f \in F_2$ by $T_f(s_1,\ldots,s_n)$, i.e.

$T_f(s_1) := \{s_1\}$ and

$T_f(s_1,\ldots,s_n) := \bigcup_{k=1,\ldots,n-1} f(T_f(s_1,\ldots,s_k),T_f(s_{k+1},\ldots,s_n))$ for $n \geq 2$,

where $f(T_1,T_2) := \{f(t_1,t_2) \mid t_1\in T, t_2\in T_2\}$. Analogously we define the right and left associative form constructed from $s_1,\ldots,s_n\in T$ using only f by

$R_f(s) := L_f(s) := s$ and

$R_f(s_1,\ldots,s_n) := f(s_1,R_f(s_2,\ldots,s_n))$, $L_f(s_1,\ldots,s_n) := f(L_f(s_1,\ldots,s_{n-1}),s_n)$ for $n\geq 2$.

Proposition 2

For any $s \in T$ with $top(s) = f \in F_2$ there exists a uniquely determined sequence $(s_1,\ldots,s_n)$ of terms s_i with $top(s_i) \neq f$ for $1\leq i\leq n$ such that $s \in T_f(s_1,\ldots,s_n)$.

Proof: By an easy induction on $|s|$ using Def.1. ■

Note that for a given $s \in T$ with $top(s) = f \in F_2$ the computation of the s_i, such that $s \in T_f(s_1,\ldots,s_n)$ and $top(s_i) \neq f$ for $1\leq i\leq n$, may be performed by (partially) flattening s w.r.t. f. For example, if $s = f(f(x,g(y,f(z,u))),f(v,w))$ then partial flattening w.r.t. f yields $s \in T_f(x,g(y,f(z,u)),v,w)$.

0. Introduction

Associative-commutative (AC-) matching and unification are fundamental problems within many fields and applications of automated reasoning including automated theorem proving, term rewriting theory, program verification and synthesis, specification analysis etc. From a practical point of view it is very important to provide efficient algorithms for these problems, because AC-unification and in particular AC-matching are in general very often performed operations, for example in systems for equational completion modulo an underlying theory consisting of AC-axioms for some operators. Of course one might use AC-unification algorithms to solve AC-matching problems but this has turned out to be too expensive in practice.

In [BeKaNa85] it has been shown that the AC-matching problem is NP-complete but has a polynomial upper bound for linear patterns. Moreover it is conjectured there that the problem remains in polynomial time as long as it is possible to put a bound on the number of times every variable can occur in a pattern. This is a strong hint that it might be very well worth-while looking for algorithms that are efficient in practical cases.

The main approaches for AC-matching presented in the literature are given by Hullot ([Hu79], [Hu8o]) and Mzali ([Mz83]. Thus, when describing our algorithm and its implementation we shall also point out the similarities and differences w.r.t. these former approaches.

The rest of this paper is organized as follows. In section 1 basic definitions and notations are given. The subproblem of AC-equivalence is dealt with in section 2. The central part of this paper in section 3 presents our AC-matching approach and clarifies the connection with the approaches of Hullot and Mzali. In section 4 we sketch how the idea of constraint propagation can be applied to other kinds of equational problems whereas the last section consists of concluding remarks and perspectives for future work in this and related fields.

1. Definitions

Let F be a finite set of function symbols of fixed arity and X be a denumerable set of variables disjoint from F. By F_i we mean the subset of F containing all function symbols of F with arity i. The set of all terms constructed over F and X as usual is denoted by T(F,X) or T for short. By V(t) we denote the set of all variables occurring in a term t. The size of t, denoted |t|, is the number of occurrences of function and variable symbols in t and top(t) denotes the top-level symbol of t. A substitution σ is a mapping from X to T that is the identity almost everywhere on X. Its unique homomorphic extension to an endomorphism on T (regarding T as a F-algebra) is also denoted by σ. The domain of a substitution σ is $D(\sigma):=\{x\in X|\sigma(x)\neq x\}$, the set of introduced variables of σ is $I(\sigma):=\bigcup_{x\in D(\sigma)}V(\sigma(x))$. Given F and X, the equational theory $=_E$ induced by a set E of equations over T is defined to be the smallest congruence on T containing E and closed under substitution. Two terms s and t are said to be E-equivalent iff $s=_E t$ holds. This notion is extended to substitutions by defining $\sigma=_E\sigma'$ iff $\forall t\in T:\sigma(t)=_E\sigma'(t)$ which is easily shown to be equivalent to $\forall x\in D(\sigma)\cup D(\sigma'):\sigma(x)=_E\sigma'(x)$. A substitution σ is an E-match from s to t iff $\sigma(s)=_E t$. The set of all E-matches from s to t, restricted to V(s), is denoted by M(s,t). This notion is extended to sets of

term pairs Γ via $M(\Gamma):=\{\sigma \mid \forall (s,t)\in\Gamma:\sigma\in M(s,t)\}$.

In the following we always assume F to be partitioned into $F=F_{AC}\cup F_{NAC}$, where F_{AC} is a non-empty set of binary operators. The set E of equations consists exactly of the AC-axioms for the operators of F_{AC}, i.e. $E := AC := \{f(x,y) =f(y,x),\ f(f(x,y),z)=f(x,f(y,z)) \mid f\in F_{AC}\}$.

2. AC-Equivalence

Since AC-equivalence testing is a basic operation frequently used in any AC-matching algorithm it is crucial to provide efficient solutions for that problem. Let us summarize therefore the main ideas and results concerning AC-equivalence in order to have a solid basis for presenting our solution to the AC-matching problem and discussing implementation issues (cf. [Hu79], [Hu80], [Mz83]).

A brute force method for deciding AC-equivalence of two terms s and t would consist in generating the (finite) equivalence class $[s]_{AC}$ of s and testing whether $t\in[s]_{AC}$ or vice versa. Clearly this is very inefficient in general. The completion method for generating a canonical term rewriting system equivalent to AC in order to get unique normal forms is not applicable since the commutativity axioms in AC cannot be oriented into rewrite rules without loosing the finite termination property. Nevertheless the form of the AC-axioms suggests another method for deciding AC-equivalence by abstracting from these properties. This is described in the following.

Proposition 1

For any $s, t \in T$, if $s =_{AC} t$ then $|s| = |t|$ and $top(s) = top(t)$.

Proof: Follows obviously from the form of the axioms AC. ■

Definition 1

Let us denote the set of terms constructed from $s_1,\ldots,s_n \in T$ using only the (binary) operator $f \in F_2$ by $T_f(s_1,\ldots,s_n)$, i.e.

$T_f(s_1) := \{s_1\}$ and

$T_f(s_1,\ldots,s_n) := \bigcup_{k=1,\ldots,n-1} f(T_f(s_1,\ldots,s_k),T_f(s_{k+1},\ldots,s_n))$ for $n \geq 2$,

where $f(T_1,T_2) := \{f(t_1,t_2) \mid t_1\in T, t_2\in T_2\}$. Analogously we define the right and left associative form constructed from $s_1,\ldots,s_n\in T$ using only f by

$R_f(s) := L_f(s) := s$ and

$R_f(s_1,\ldots,s_n) := f(s_1,R_f(s_2,\ldots,s_n))$, $L_f(s_1,\ldots,s_n) := f(L_f(s_1,\ldots,s_{n-1}),s_n)$ for $n\geq 2$.

Proposition 2

For any $s \in T$ with $top(s) = f \in F_2$ there exists a uniquely determined sequence $(s_1,\ldots,s_n)$ of terms s_i with $top(s_i) \neq f$ for $1\leq i\leq n$ such that $s \in T_f(s_1,\ldots,s_n)$.

Proof: By an easy induction on $|s|$ using Def.1. ■

Note that for a given $s \in T$ with $top(s) = f \in F_2$ the computation of the s_i, such that $s \in T_f(s_1,\ldots,s_n)$ and $top(s_i) \neq f$ for $1\leq i\leq n$, may be performed by (partially) flattening s w.r.t. f. For example, if $s = f(f(x,g(y,f(z,u))),f(v,w))$ then partial flattening w.r.t. f yields $s \in T_f(x,g(y,f(z,u)),v,w)$.

Proposition 3

Let $s_i, t_i \in T$ with $s_i =_{AC} t_i$ for $1 \le i \le n$ and $f \in F_{AC}$ be given. Then for any $s \in T_f(s_1,\ldots,s_n)$, $t \in T_f(t_1,\ldots,t_n)$ we have $s =_{AC} t$.

Proof: By an easy induction on n we get for the right (or left) associative forms $s =_{AC} R_f(s_1,\ldots,s_n)$ and $t =_{AC} R_f(t_1,\ldots,t_n)$, which, by the assumption $s_i =_{AC} t_i$ for $1 \le i \le n$ implies $s =_{AC} t$. ■

Proposition 4

Let $s_i, t_i \in T$ with $s_i =_{AC} t_i$ for $1 \le i \le n$, a permutation π of $\{1,\ldots,n\}$ and $f \in F_{AC}$ be given. Then for any $s \in T_f(s_1,\ldots,s_n)$, $t \in T_f(t_{\pi(1)},\ldots,t_{\pi(n)})$ we have $s =_{AC} t$.

Proof: Using Prop.3 it suffices to show that $R_f(s_1,\ldots,s_n) =_{AC} R_f(s_{\pi(1)},\ldots,s_{\pi(n)})$ for all permutations π of $\{1,\ldots,n\}$. But this is obvious since by $f(x,y) =_{AC} f(y,x)$, $f(x,f(y,z)) =_{AC} f(y,f(x,z))$ we may exchange any two neighbour arguments s_i, s_{i+1} in $R_f(s_1,\ldots,s_n)$. ■

Proposition 5

Let $f \in F_{AC}$, $s_i, t_j \in T$ with $top(s_i) \neq f \neq top(t_j)$ for $1 \le i \le m$, $1 \le j \le n$ and $s \in T_f(s_1,\ldots,s_m)$, $t \in T_f(t_1,\ldots,t_n)$ be given. Then $s =_{AC} t$ iff $m = n$ and there exists a permutation π of $\{1,\ldots,m\}$ with $t_i =_{AC} s_{\pi(i)}$ for all i, $1 \le i \le m$.

Proof: The "if"-direction is provided by Prop.4. For the "only-if"-direction assume that $s \in T_f(s_1,\ldots,s_m)$, $t \in T_f(t_1,\ldots,t_n)$ satisfy the assumptions of the proposition. The form of the axioms AC implies that any s' that can be derived from s in one step using AC has the following form:

$s' \in T_f(s_1,\ldots,s_{i-1},s_i',s_{i+1},\ldots,s_m)$ with $s_i =_{AC} s_i'$ and $top(s_i) = top(s_i')$ if the proof step is applied inside some s_i, or

$s' \in T_f(s_1,\ldots,s_m)$, if associativity for f is applied, but not inside one of the s_i, or

$s' \in T_f(s_1,\ldots,s_{p-1},s_q,\ldots s_{r-1},s_p,\ldots s_{q-1},s_r,\ldots,s_m)$ for $s' \in T_f(s_1,\ldots,s_p,\ldots s_q,\ldots,s_r,\ldots,s_m)$ and some p, q, r with $1 \le p < q \le r \le m$, if commutativity for f is applied, but not inside one of the s_i.

By an inductive argument (for equational proofs using AC) we conclude for any s", that $s =_{AC} s"$ implies $s" \in T_f(s_{\pi(1)}",\ldots,s_{\pi(m)}")$ with $s_i =_{AC} s_{\pi(i)}"$ for all i, $1 \le i \le m$ and some permutation π of $\{1,\ldots,m\}$. In particular for s" := t we get, using Prop.2, m = n and $t_i = s_{\pi(i)}" =_{AC} s_i$ for all i, $1 \le i \le n$. ■

In the following, if $s \in T_f(s_1,\ldots,s_n)$ with $f \in F_{AC}$ and $top(s_i) \neq f$ for $1 \le i \le n$, we will write $R(s) := f[s_1,\ldots,s_n] := R_f(s_1,\ldots,s_n)$ to denote the right associative form of s (constructible by partial flattening w.r.t. f). The s_i will be referred to as (top-level) arguments of s. After these auxiliary considerations we can give a first characterization of AC-equivalence.

Lemma 1

Let $s, t \in T$ be given. Then $s =_{AC} t$ iff either

(i) $s = t \in X$, or

(ii) $s = g(s_1,\ldots,s_m)$, $t = g(t_1,\ldots,t_m)$, $g \in F_{NAC}$ and $s_i =_{AC} t_i$ for $1 \le i \le m$, or

(iii) $R(s) = f[s_1,\ldots,s_m]$, $R(t) = f[t_1,\ldots,t_m]$, $f \in F_{AC}$ and there exists a permutation

π of $\{1,\ldots,m\}$ with $s_i =_{AC} t_{\pi(i)}$ for $1 \le i \le m$.

Proof: The interesting "only-if"-direction is easily proved by induction on $|s|$ using Prop.1, Prop.5 and the form of the axioms AC. ■

This result now provides a recursive test for AC-equivalence. But due to the partial flattening and the search for adequate permutations involved in (iii) it may still be expensive in general. It is obvious from Lemma 1 that the AC-equivalence class [s] of any term s is uniquely determined by

g and the sequence $([s_1],\ldots,[s_m])$, if $s = g(s_1,\ldots,s_m)$, $g \in F_{NAC}$, and

f and the multiset $\{[s_1],\ldots,[s_m]\}$, if $R(s) = f[s_1,\ldots,s_m]$, $f \in F_{AC}$.

This fact may be used to define an ordering on AC-equivalence classes as follows.

Definition 2 (orderings for AC-equivalence classes, cf. [Hu80])

Let $<_{F \cup X}$ be a partial ordering on $F \cup X$. The partial ordering $<$ on $T/=_{AC}$ is given by

$[s] < [t]$ iff either

(i) $top(s) <_{F \cup X} top(t)$, or

(ii) $s = g(s_1,\ldots,s_m)$, $t = g(t_1,\ldots,t_m)$, $g \in F_{NAC}$ and $([s_1],\ldots,[s_m]) <_{lex} ([t_1],\ldots,[t_m])$, or

(iii) $R(s) = f[s_1,\ldots,s_m]$, $R(t) = f[t_1,\ldots,t_n]$, $f \in F_{AC}$ and $\{[s_1],\ldots,[s_m]\} << \{[t_1],\ldots,[t_n]\}$,

where $<_{lex}$ denotes the lexicographic extension of $<$ to sequences of AC-equivalence classes and $<<$ denotes the usual multiset extension of $<$.

Lemma 2 (cf. Lemma 3.7 in [Hu80])

If $<_{F \cup X}$ is a total ordering on $F \cup X$, then $<$ is total on $T/=_{AC}$.

Proof: see [Hu80]. ■

Let us assume from now on that $<_{F \cup X}$ is total. As described by Hullot one may use the above result to get a unique normal form for any AC-equivalence class by recursive regrouping of arguments. For any s with $R(s) = f[s_1,\ldots,s_n]$ we construct t with $R(t) = f[t_1,\ldots,t_n]$ such that $s =_{AC} t$ and

(i) $\forall$ i, $1 \le i \le n$: $t_i =_{AC} s_{\pi(i)}$ for some permutation π of $\{1,\ldots,n\}$, and

(ii) $\forall$ i, $1 \le i \le n$: $[t_i] < [t_{i+1}]$ or $t_i =_{AC} t_{i+1}$.

Let $ORD((s_1,\ldots,s_n)) = (t_1,\ldots,t_n)$ be any regrouping of $(s_1,\ldots,s_n)$ satisfying (i) and (ii). Then we get an ordered normal form ONF(s) for any term s by recursively defining:

ONF(s) := if $s \in X$ then s else

if $s = g(s_1,\ldots,s_n)$, $f \in F_{NAC}$ then $g(ONF(s_1),\ldots,ONF(s_n))$ else

if $s = f[s_1,\ldots,s_n]$, $f \in F_{AC}$ then $f[ORD(ONF(s_1),\ldots,ONF(s_n))]$

Ordered normal forms now provide an efficient syntactic test for AC-equivalence.

Lemma 3 (cf. Lemma 3.8 in [Hu80])

For any terms s, t we have: $s =_{AC} t$ iff $ONF(s) = ONF(t)$.

Proof: see [Hu80]. ■

We conclude this section with an algorithm FONF that produces an explicit flattened

representation of ordered normal forms by combining flattening and regrouping of arguments according to <. The auxiliary function FONF' needs an additional parameter op indicating that flattening is performed w.r.t. op. ε is a dummy operator needed for initialization.

```
function FONF'(s:T, op:F_AC ∪ {ε}) :=
if s ∈ F_0 ∪ X then if op ∈ F_AC then [ s ] else s
else  if s = g(s_1,...,s_n),  g ∈ F_n\F_AC, n ≥ 1
    then  if op ∈ F_AC
          then [  g(FONF'(s_1,ε),...,FONF'(s_n,ε)) ]
          else g(FONF'(s,ε),...,FONF'(s_n,ε))
    else  (* s = f(s_1,s_2), f ∈ F_AC *)
          if f = op
          then MERGE(FONF'(s_1,f), FONF'(s_2,f))
          else  if op ∈ F_AC
                then  [ f MERGE(FONF'(s_1,f), FONF'(s_2,f)) ]
                else f MERGE(FONF'(s_1,f), FONF'(s_2,f)).
```

For any term s the flattened ordered normal form FONF(s) is constructed now by FONF'(s,ε). The use of a MERGE algorithm is very natural in the above context since the two argument lists are already sorted (increasingly w.r.t. <) by construction. In our implementation of AC-matching we use flattened ordered normal forms based on a special class of orderings < which may be described as follows: The underlying total ordering $<_{F \cup X}$ is chosen such that $c <_{F \cup X} g <_{F \cup X} f <_{F \cup X} x$ for any $c \in F_0$, $g \in F_{NAC} \backslash F_0$, $f \in F_{AC}$ and $x \in X$. The advantages of this class of orderings are discussed in the next section.

In [Hu80] the usage of ordered normal forms as abbreviation for the corresponding right associative normal form is suggested without giving implementation details whereas in [Mz83] it is claimed that ordered normal forms are not really necessary. This is clearly true from a theoretical point of view, but from an implementation point of view it is not obvious at all how to get efficient implementations of AC-equivalence and AC-matching using (unordered) multiset representations of terms involving AC-operators.

3. AC-Matching

A general scheme for E-matching problems consists in transforming an initial problem Γ into an equivalent disjunction (in a sense to be made precise) of problems $\Gamma_1 \vee \ldots \vee \Gamma_n$ where the Γ_i are in "solved form", i.e. the solutions are obvious. This is achieved by repeated application of so-called merging, decomposition and mutation rules. Details of such general approaches may be found for example in [MaMo82], [Ki85], [Mz86], [Co88], [Sc88]. Essentially merging means propagating parts of possible solutions into the rest of the problem, whereas decomposition means to decompose the problem into a conjunction of some "smaller" problems or to deduce that a problem has no solution. These steps are in a sense of purely syntactical nature as soon as it is clear which function symbols are "decomposable" w.r.t. a given (arbitrary) E and which pairs of function

symbols are "exclusive" w.r.t. E (allowing clash-rules, cf. [Ki85]). In contrast to these rules mutation deals with problems that cannot be simplified any more using decomposition and merging. In fact adequate mutation steps heavily depend on the underlying equational theory defined by E .

From an abstract point of view such a method for solving E-matching problems may be represented by a tree whose nodes consist of problems (including a special symbol FAIL denoting unsolvability) and whose arcs are labelled by (parts of) substitutions. Applying the rules corresponds to constructing this "E-matching tree" starting from the tree that consists only of a root node containing the initial problem. If the method is correct all solutions of the initial problem are then given by the substitutions built along the paths leading from the root to all leaves consisting of (trivially solved) empty problems (cf. [Hu79]).

We will now give an abstract description of the basic AC-matching algorithm common to the approaches of Hullot ([Hu79], [Hu80]), Mzali ([Mz83]) and ours. Merging and decomposition are grouped together into a simplification phase, whereas mutation is performed within a so-called reduction phase.

Note that w.l.o.g. we may assume $V(s) \cap V(t) = \emptyset$ for any given initial AC-problem s=t. If this is not the case one may rename the variables of t accordingly or introduce new constants for them. The solutions of s=t bijectively correspond to the solutions of the transformed problem in a straightforward way. Moreover we assume in the following w.l.o.g. that terms with an AC top-level operator are in right associative form.

Simplification may be described by a function SIMPLIFY with two arguments Γ and S, where Γ is the non-trivial part and S the trivial or solved part of the AC-matching problem $\Gamma \cup S$. It returns an equivalent problem $\Gamma' \cup S'$ which has been simplified as far as possible or 'FAIL' in the case of unsolvability. The "environment" argument S is considered as a substitution with its domain denoted by D(S).

function SIMPLIFY (Γ, S) :=
  if $\Gamma = \emptyset$ then (Γ,S)
  else (* $\Gamma = \{s=t\} \cup \Gamma_1$ *)
    if $s \in X$ then if $s \notin D(S)$ then SIMPLIFY(Γ_1, $S \cup \{s=t\}$)
      else if $S(s) =_{AC} t$ then SIMPLIFY(Γ_1, S) else 'FAIL'
    else if $top(s) \neq top(t)$ then 'FAIL'
      else if $s = g(s_1,\ldots,s_m)$, $t = g(t_1,\ldots,t_m)$, $g \in F_{NAC}$
        then SIMPLIFY ($\Gamma_1 \cup \{s_i=t_i \mid 1 \le i \le m=n\}$, S)
        else (* $s = f[s_1,\ldots,s_m]$, $t = f[t_1,\ldots,t_n]$, $f \in F_{AC}$ *)
          ($\Gamma' \cup \{s=t\}$, S') where (Γ', S') := SIMPLIFY (Γ_1, S)

Note that the test on AC-equivalence is needed whenever we encounter a pair s=t in Γ such that s is a variable that has already got a value in the environment.

Lemma 4:

SIMPLIFY is terminating and solution preserving, i.e.

(a) SIMPLIFY (Γ, S) = 'FAIL' $\Rightarrow$ M ($\Gamma \cup$ S) = $\emptyset$, and

(b) SIMPLIFY (Γ, S) = (Γ', S ') $\Rightarrow$ M ($\Gamma \cup$ S) = M ($\Gamma' \cup$ S')

Proof: Termination is guaranteed by decreasing problem sizes $| (\Gamma, S) | := \Sigma_{s=t \in \Gamma} | s |$ in any recursive call. Correctness, i.e. (a) and (b), easily follows from the correctness of the merging and decomposition rules for the case E = AC. ■

For example, starting from Γ: g(g(x,x),f(y,z)) = g(g(f(a,b),f(b,a)),f(c,d)) with F_{AC} = {f} and an empty environment S simplification yields Γ': f[y,z] = f[c,d] and S': x = f[a,b].

IF $F_{AC}=\emptyset$ then SIMPLIFY provides an abstract version of the usual matching algorithm without an underlying theory. If SIMPLIFY(Γ,S)=(Γ',S') with $\Gamma' \neq \emptyset$ then any problem of Γ' has the form $f[s_1,\ldots,s_m]=f[t_1,\ldots,t_n]$, $f \in F_{AC}$. We assume now as black box a function REDUCE that transforms such simplified problems (Γ',S') into a set of reduced problems such that the solutions are preserved and the complexity of the problem has decreased, i.e.

(1) $M (\Gamma' \cup S') = \bigcup_{(\Gamma'',S'') \in \text{REDUCE } (\Gamma',S')} M (\Gamma'' \cup S'')$, and

(2) $\forall$ (Γ'',S'') $\in$ REDUCE (Γ',S') : $| (\Gamma'',S'') | <_N | (\Gamma',S') |$, where $<_N$ is the usual ordering on natural numbers. Using such a function REDUCE we get an abstract version for AC-matching as follows:

function AC-MATCH-ALL' (Γ ,S) :=
  if SIMPLIFY(Γ, S) = ($\emptyset$, S')
  then S'
  else if SIMPLIFY(Γ, S) = 'FAIL'
    then $\emptyset$
    else $\bigcup_{(\Gamma',S') \in \text{REDUCE(SIMPLIFY}(\Gamma,S))}$AC-MATCH-ALL'($\Gamma'$,S').

function AC-MATCH-ALL(s = t) := AC-MATCH-ALL'({ s = t }, $\emptyset$).

Termination and correctness of this AC-matching algorithm are assured by

Lemma 5:

For any reduction function REDUCE satisfying (1) and (2) AC-MATCH-ALL is terminating for any problem s=t and computes the set M(s,t) of all AC-matches from s to t.

Proof: Termination (of AC-MATCH-ALL and AC-MATCH-ALL') is again guaranteed by decreasing problem sizes $|(\Gamma,S)|$ in any recursive call of AC-MATCH-ALL' using Lemma 4 and property (2). Correctness follows from the correctness of SIMPLIFY, property (1) and the following considerations. Any non-trivial problem part Γ of a solvable problem eventually becomes empty and the corresponding trivial part S always remains in solved form, i.e. any s=t $\in$ S has the form x=u, u is different from x and x does not occur anywhere else in S. Thus any final S may indeed be considered as a (solving) substitution representing all solutions S' with $S' =_{AC} S$. ■

In close analogy to AC-MATCH-ALL, using backtracking, one may specify an algorithm AC-MATCH-ONE which, for a given AC-matching problem s=t, returns the first AC-match found

and indicates unsolvability otherwise. This is indeed sufficient in many applications like completion modulo AC. In both cases it is crucial for the efficieny of AC-matching how the reduction process is performed and controlled. Let us now see how these problems are solved in the traditional approaches and what we suggest instead.

3.1 Reduction via Local Partitioning

Given a simplified matching problem Γ Hullot ([Hu79], [Hu80]) and Mzali ([Mz63]) essentially proceed as follows.

1.) Choose one problem s=t of Γ.

2.) Perform "partitioning" for s=t.

Partitioning means to take into account any possibility of associating a couple of right hand side arguments to any left hand side argument. More formally this may be specified by the following mutation rule (for $s = f[s_1,\ldots,s_m]$, $t = f[t_1,\ldots,t_n]$ and $f \in F_{AC}$) :

$$\Gamma \cup \{ s = t \} \rightarrow \Gamma \cup \Gamma' \text{ for any } \Gamma' \in \text{PARTITIONS}(s, t),$$

where the function PARTITIONS(s, t) is defined as follows (cf. [Hu80]):

Let $\{P_1,\ldots,P_q\}$ with $P_i=[P_{i1},\ldots,P_{im}]$ be the set of all ordered partitions of the multiset $\{t_1,\ldots,t_n\}$ into m non-empty blocks. With $|P_{ij}|$ denoting the length of the block P_{ij} and $\tau_k(P_{ij})$ denoting the k-th element of P_{ij} (for $1\le i\le q$, $1\le j\le m$, $1\le k\le |P_{ij}|$) define

$t_j^i := \tau_1(P_{ij})$ for $|P_{ij}| = 1$, and

$t_j^i := f[\tau_1(P_{ij}),\ldots,\tau_r(P_{ij})]$ for $r := |P_{ij}| > 1$, and finally

PARTITIONS (s, t) := $\{ \{ s_1 = t_1^i,\ldots, s_m = t_m^i \} \mid 1 \le i \le q \}$.

This partitioning process is formally justified by Lemma 1. For the corresponding reduction function defined by

$$\text{REDUCE}(\Gamma \cup \{ s = t \}, S) := \bigcup_{\Gamma' \in \text{PARTITIONS}(s,t)} \{ (\Gamma \cup \Gamma', S) \}$$

(for $s = f[s_1,\ldots,s_m]$, $t = f[t_1,\ldots,t_n]$ and $f \in F_{AC}$) the required properties of correctness (1) and termination (2) are obviously satisfied. Let us give a simple example to make clear the definition above.

Example 1

For $F=\{a,b,c,f\}$, $F_{AC}=\{f\}$, s=f(x,y), t=f(a,f(b,c)) the AC-matching problem {s=t}, transformed into normal form, becomes {f[x,y]=f[a,b,c]} and by partitioning we get the disjunction of the following reduced problems: {x=f[a,b], y=c}, {x=f[a,c], y=b}, {x=f[b,c], y=a}, {x=a, y=f[b,c]}, {x=b, y=f[a,c]} and {x=c, y=f[a,b]}. Indeed, these six problems are already in solved form and thus they constitute a complete set of AC-matches for the original problem.

Taking into account the multiplicity of left and right hand side arguments (s^k denotes k times s) the partitioning process for the AC-matching problem s=t with $s=f[s_1^{k1},\ldots,s_m^{km}]$, $t=f[t_1^{l1},\ldots,t_n^{lm}]$ (the s_i, t_j are assumed to be pairwise distinct w.r.t. $=_{AC}$) may be optimized avoiding the generation of unsolvable problems: If for any i with $1\le i\le m$ s_i corresponds in a partition to $f[t_1^{vi1},\ldots,t_n^{vin}]$ the whole partition is already determined since two different associations for the same s_i would lead to a failure. This optimized partitioning with cardinality restrictions (cf. [Hu80]) obviously

corresponds to solving the following system of linear diophantine equations (cf. [Mz83])

$$l1 = k1 * v11 + \ldots + km * vm1$$
$$l2 = k1 * v12 + \ldots + km * vm2$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$ln = k1 * v1n + \ldots + km * vmn$$

in the unknowns vij, $1 \le i \le m$, $1 \le j \le n$ with the additional restriction that for any i not all vij are 0. This can be done separately for any diophantine equation since they are independent from each other.

Example 2

Let F, F_{AC} be as in example 1, $s=f[x^2,y^4]$, $t=f[a^4,b^6,c^2]$. To determine all possible partitions

$$x = f\,[a^{v11},b^{v12},c^{v13}],\ y = f\,[a^{v11},b^{v12},c^{v13}]$$

amounts to solving the following system of linear diophantine equations:

$$4 = 2 * v11 + 4 * v21$$
$$6 = 2 * v12 + 4 * v22$$
$$2 = 2 * v13 + 4 * v23$$

with the restriction $\Sigma_j vij > 0$ for i=1,2. For the first equation we get the solutions (v11,v21)=(0,1),(2,0), for the second (v12,v22)=(1,1),(3,0) and for the last (v13,v23)=(1,0). Taking care of the restrictions this leads to the three solutions of the matching problem

$$x = f\,[\,b, c\,],\ y = f\,[\,a, b\,]$$
$$x = f\,[\,b^3, c\,],\ y = a$$
$$x = f\,[\,a^2, b, c\,],\ y = b\,.$$

3.2 Reduction using Global Constraint Propagation

In our approach instead of local partitioning we shall use a kind of global reasoning to optimize the branching in the search tree. Having a closer look to the partitioning process described above a first important observation is that it can be split in a hierarchical way as follows. Given a simplified matching problem $\Gamma=\Gamma'\cup\{s=t\}$ with $s=f[s_1,\ldots,s_m]$, $t=f[t_1,\ldots,t_n]$ any possible partition of s=t that associates $f[t_1^{k1},\ldots,t_n^{kn}]$, $0 \le kj \le 1$, to some s_i may be constructed from the reduced problem $\Gamma'' := \Gamma' \cup \{\, s_i=f[t_1^{k1},\ldots,t_n^{kn}],\ f[s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_m]=f[u_1,\ldots,u_p]\,\}$ with $[u_1,\ldots,u_p] := [t_1,\ldots,t_n] \setminus [t_1^{k1},\ldots,t_n^{kn}]$ by partitioning $f[s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_m]=f[u_1,\ldots,u_p]$ in all possible ways. The formal justification of this connection is provided by

Lemma 6:

Let $s=f[s_1,\ldots,s_m]$, $t=f[t_1,\ldots,t_n]$ with $f \in F_{AC}$ and i, $1 \le i \le m$, be given. Then $s =_{AC} t$ iff there exists $[t_1^{k1},\ldots,t_n^{kn}] \subseteq [t_1,\ldots,t_n]$, $0 \le kj \le 1$, such that $s_i =_{AC} f[t_1^{k1},\ldots,t_n^{kn}]$ and $f[s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_m] =_{AC} f[u_1,\ldots,u_p]$, where $[u_1,\ldots,u_p] := [t_1,\ldots,t_n] \setminus [t_1^{k1},\ldots,t_n^{kn}]$ and the operators $\subseteq$ and $\setminus$ denote multiset inclusion and difference.

Proof: Follows easily from Lemma 1. ■

When using such a hierarchical strategy for branching in the search tree instead of ordinary

partitioning, it is useful from a heuristic point of view to choose an s_i with a minimal number of possible associations. Now the main idea of our approach is to collect as much restrictive information as possible for the associations for any left hand side argument s_i before branching. This is achieved by the following two steps:

1.) Consider not only one problem s=t but all problems of Γ simultaneously.
2.) Propagate constraints for the possible associations of all left hand side arguments of Γ.

Let us describe how this can be done. We assume that any problem s=t of Γ has only variables as left hand side arguments, i.e. has the form $f[x_1^{k1},\ldots,x_m^{km}]=f[t_1^{l1},\ldots t,_n^{ln}]$. We shall see later that it is indeed sufficient to deal with this "variable-only" case in practice.

Constraint propagation in the case of equal top-level AC-operators

Obviously any possible substitution for x_i must be of the general form

$$x_i=f[t_1^{m1},\ldots,t_n^{mn}]$$

and satisfy the conditions

$$\forall\, j,\ 1 \le j \le n : m_j \le (l_j \operatorname{div} k_i).$$

Now, if there is another $s'=t' \in \Gamma$ with $s'=f[\ldots,x_i^{ki'},\ldots]$ then any possible substitution for x_i must also satisfy the corresponding inequalities for s'=t'. Thus any further problem containing x_i as a left hand side argument provides more restrictive information.

Example 3: Assume we have given the simplified problem $\Gamma=\{(I),(II)\}$ with

$$(I)\quad f[\,x, y\,] = f[\,a^2, b^2, c^2\,]$$
$$(II)\quad f[\,x^2, z\,] = f[\,a, b^3, d^2]$$

Then we know that any possible substitution for x must be of the general form

$$f[\,a^{ma}, b^{mb}, c^{mc}, d^{md}\,].$$

From (I) we deduce the restrictions

$$ma \le 2,\ mb \le 2,\ mc \le 2,\ md = 0$$

and from (II)

$$ma = 0,\ mb \le 1,\ mc = 0,\ md \le 1.$$

Combination of both sets of necessary conditions yields

$$ma = mc = md = 0,\ mb \le 1,$$

which results in only one remaining possibility for x, namely x=b. Propagating this assignment into Γ we immediately get from (I) and (II) the only solution of the problem, namely

$$x = b$$
$$y = f[\,a^2, b, c^2\,]$$
$$z = f[\,a, b, d^2].$$

Choosing (I) for example for (optimized) local partitioning we would get 25 different branches to be investigated in the worst case since 24 of these branches will lead to a failure.

Constraint propagation in the case of different top-level AC-operators

Assume Γ contains two problems of the form

$$\text{(I)} \quad f[\ldots,x^i,\ldots] = f[\ s_1^{k1},\ldots,s_m^{km}\ ]$$
$$\text{(II)} \quad g[\ldots,x^j,\ldots] = g[\ t_1^{l1},\ldots,t_n^{ln}\ ],$$

where $f, g \in F_{AC}$, $f \neq g$ and both (I) and (II) have at least two different left hand side arguments. Then from (I) we can deduce that any possible assignment for x must be of the form

$$x = f[\ s_1^{u1},\ldots,s_m^{um}\ ] \text{ with } \forall\ p,\ 1 \le p \le m : up \le (kp \text{ div } i)$$

and (II) implies

$$x = g[\ t_1^{v1},\ldots,t_n^{vn}\ ] \text{ with } \forall\ q,\ 1 \le q \le n : vq \le (lq \text{ div } j).$$

If $\Sigma_{1\le p\le m} up > 1$ and $\Sigma_{1\le q\le n} vq > 1$ then this will obviously yield a clash of the form $f[\ldots]=g[\ldots]$. Thus we have only the following three possible cases left.

(a) $\Sigma_{1\le p\le m} up = \Sigma_{1\le q\le n} vq = 1$: This means that $x = s_p = t_q$ for some p, q with $1\le p\le m$ and $1\le q\le n$, i.e. the assignment for x must be an argument of both the first and the second right hand side.

(b) $\Sigma_{1\le p\le m} up = 1$, $\Sigma_{1\le q\le n} vq \ge 2$: In this case one has to choose for x any s_p with $top(s_p)=g$ such that the arguments of s_p are contained in $[\ t_1^{(l1 \text{ div } j)},\ldots,t_n^{(ln \text{ div } j)}\ ]$.

(c) $\Sigma_{1\le p\le m} up \ge 2$, $\Sigma_{1\le q\le n} vq = 1$: This case is symmetric to (b).

When there are common variables in two such problems with different top-level AC-operators, this will lead in general to very strong restrictions for the possible assignments for these common variables. Let us demonstrate the effect of this constraint propagation process with

Example 4: Assume we have given $F_{AC}=\{f,g\}$ and the simplified problem $\Gamma=\{(I),(II)\}$ with

$$\text{(I)} \quad f[\ x, y\ ] = f[\ a^2, c,\ g[a, b^2]\ ]$$
$$\text{(II)} \quad g[\ x^2, z] = g[\ a^3, b^7,\ f[a, b]^3\ ].$$

Investigating the possible assignments for x we get the following cases according to the above scheme:

(a) The only argument common to both the first and second right hand side is a, thus yielding one possibility, namely x = a.

(b) The only right hand side argument of (I) with top-level operator g is $g[a, b^2]$, for which we have to check whether one can combine the arguments a, b, b using (3 div 2) times a , (7 div 2) times b and (3 div 2) times f[a, b], which is indeed possible.

(c) The argument of f[a, b] obviously cannot be combined using two a's, one c and one $g[a,b^2]$.

Thus we have only two possibilities left for x, namely x = a or $x = g[a,b^2]$. Using (optimized) local partitioning for (I) in the example, we would get 10 branches to be investigated with 8 failure branches among them.

In the case of more than two non-trivial problems the constraints for the left hand side variables are incrementally sharpened in a straghtforward way according to the above two cases.

Control of Branching

Essentially the control of branching in our constraint propagation approach is determined by the following two steps (for the "variable-only" case):

(1) Compute (a representation of) the list of all possible assignments for any variable occurring in some left hand side argument list as described above.

(2) Choose a variable with a minimal number of possibilities for branching and backtrack in case of failure.

How do we actually represent these possible assignments for left hand side variables?

Case (1a): The top-level AC-operators of all subproblems in Γ containing x as left hand side argument are the same, say f. Then we compute an implicit representation of the form $x = f[t_1^{l1},\ldots,t_n^{ln}]$ meaning that any t_i can occur at most l_i times in the assignment. This yields a total number of $(\Pi_{i=1\ldots n}(l_i+1)) - 1$ possibilities. Thus in example 3 we get $y = f[a^2, b^2, c^2]$ resulting in 26 possibilities.

Case (1b): There are at least two subproblems containing x on the left with different top-level AC-operators. Then we get an explicit representation of the form $x = t_1,\ldots,t_p$ yielding p possibilities. In example 4, as shown above, this leads to the two possibilities $x = a$, $g[a,b^2]$.

Note that these computations actually only give approximations of the exact number of possibilities in the sense of upper bounds. This is due to the fact that the other variables occurring in the same argument list as x are ignored in the constraint propagation process for x. The method could be refined to compute exactly all possibilities in a way similar to the construction of ordered partitions with cardinality restrictions as described in [Hu80]. But since this would be rather expensive we renounce to do it in our current implementation.

If there are no variables occurring in more than one left hand side argument list the constraint propagation process may cause an additional overhead since the constraints for each subproblem are independent from each other. But we still have the advantage of choosing systematically a node with a small branching degree.

3.3 Optimizations and Implementation Issues

In order to get an efficient algorithm for AC-matching using our approach of global constraint propagation (as well as using local partitioning) it is crucial to incorporate additional optimizations and to deal efficiently with the AC-equivalence problem as mentioned before. Before pointing out the advantages of using ordered normal forms we describe now possible optimizations and their integration in our implementation.

(I) Global size criterion

One global optimization consists in a simple failure criterion. If $|s| > |t|$ then there cannot exist an AC-match from s to t. This size criterion is applicable to any subproblem generated. Although from a conceptional point of view this is indeed an optimization, we have not integrated this size test in our current implementation, because computing term sizes is rather

expensive if performed very often.

(II) **Optimizations in the reduction phase**

The main source of complexity of AC-matching comes in through the reduction process, which is responsible for the NP-completeness of the problem. Thus optimizations of the reduction process are very important. Let us assume in the following that we have given a simplified matching problem $\Gamma \cup S$ with non-empty Γ. That means every $s=t \in \Gamma$ has the form $s=f[s_1^{k1},\ldots,s_m^{km}]$, $t=f[t_1^{l1},\ldots,t_n^{ln}]$.

(IIa) **Length restrictions for argument lists**

Similar to the global size criterion for s=t the number of top-level arguments of S(s) must not be greater than the number of top-level arguments of t. Otherwise this clearly indicates a failure.

(IIb) **Deletion of common parts**

Any argument s_i of s with $V(s_i) \subseteq D(S)$ cannot be modified any more by additional substitution parts that might be generated later on in the actual branch of the search tree. Thus $S(s_i)$ must occur at least l_i times among the arguments of t. If this is not the case the whole problem $\Gamma \cup S$ is unsolvable and we can cut this branch of the search tree. Otherwise we simplify the problem s=t by deleting s_i respectively $S(s_i)$ l_i times in the argument list of s respectively t. At least for constant arguments s_i where the condition $V(s_i) \subseteq D(S)$ is vacuously satisfied this can be done efficiently. In our implementation deletion of constants is performed as early as possible within the simplification phase. Deletion of substituted variables is delayed until the variable-only case is reached (except case (IIe)).

(IIc) **Cardinality restrictions for reduction**

As already mentioned the multiplicities ki, lj of left and right hand side arguments s_i, t_j can be used to perform an optimized partitioning with cardinality restrictions (cf. [Hu80]) or equivalently to solve the corresponding system of diophantine equations ([Mz83]). In our approach the multiplicity information is used in the constraint propagation process to deduce combined constraints.

(IId) **Look ahead for top-level clashes**

Considering a non-variable left hand side argument s_i (implying $top(s_i) \neq f$) the corresponding right hand side after choosing a possible association of right hand side arguments for s_i must not have the form $f[u_1,\ldots,u_p]$ with $p \geq 2$ or $g(v_1 \ldots v_q)$ with $g \neq top(s_i)$. Otherwise this would result in a failure by simplification when processing later that newly generated subproblem. In other words reduction should always generate for such a non-variable left hand side argument s_i a subproblem of the form $s_i=t_k$ with $top(s_i)=top(t_k)$. In fact combining these conditions we can deduce that for the problem s=t as above any $g \in F \setminus \{f\}$ must occur as top-level symbol in the t_j at least as often as in the s_i, otherwise the problem has no solution. This consideration motivates the heuristic to perform an intelligent kind of branching focussing on the

non-variable left hand side arguments s_i until the variable-only case is reached. Of course we must keep track of the remaining possibilities of associating some t_j with the same top-level symbol to s_i when deciding to choose a specific one for branching. Again a plausible heuristic would consist in choosing such a non-variable s_i with a minimal number of possible associations for branching. Reasoning globally w.r.t. Γ and using again the constraint propagation techniques explained above this heuristic could still be refined. But since the restrictions for non-variable left hand side arguments are already relatively strong compared to variable arguments we did not implement this heuristic. Thus intelligent branching is performed according to the order of the left and right hand side arguments of the subproblem considered.

(IIe) **Trivial variable-only case**

If there is a problem s=t in Γ with $s=f[x^k]$, $t=f[t_1^{l1},\ldots,t_n^{ln}]$, $k\geq 2$ and $x\notin D(S)$, then the only possibility to solve s=t consists in reducing it to $x=f[t_1^{l1'},\ldots t_n^{ln'}]$ for $lj' := (lj \text{ div } k)$, provided that $(lj \bmod k) = 0$ holds for all j, $1\leq j\leq n$. Otherwise the problem is again unsolvable. In our approach we discover such situations as early as possible. Using local partitioning (with cardinality restrictions) this still depends on the choice of the subproblem to be partitioned.

The usage of unique ordered normal forms for AC-equivalence classes as described in section 1 strongly supports most operations in the AC-matching algorithm. The (potentially expensive) construction of ordered normal forms essentailly has to be done only once for the initial terms. Partial recomputations of ordered normal forms are only necessary when applying substitutions. Tests on AC-equivalence are reduced to linear syntactic equality tests of ordered normal forms, deletion operations and intelligent branching may be performed efficiently and also the constraint propagation computations profit of the fixed order of variable arguments. In particular the multiplicity information for left and right hand side arguments which is crucial for an efficient reduction process is a by-product of computing ordered normal forms.

Moreover the special class of orderings < we use (induced by $<_{F\cup X}$ with $c <_{F\cup X} g <_{F\cup X} f <_{F\cup X} x$ for any $c \in F_0$, $g \in F_{NAC}\backslash F_0$, $f \in F_{AC}$ and $x \in X$, see section 1) reflects exactly our strategy of reduction of simplified problems. First all constant left hand side arguments are eliminated by delete operations. This corresponds to the fact that the constants are the smallest arguments w.r.t. <. Then the non-variable left hand side arguments are handled by intelligent backtracking which is facilitated by the fact that non-variable arguments with the same top-level operator are neighbours in ordered argument lists. Reduction using constraint propagation is delayed until the variable-only case is reached. This is reflected by the fact that the variables are the greatest elements w.r.t. <.

3.4 An Extended Example

Instead of giving a detailed description of our implementation which would involve a rather complicated control structure for backtracking using a kind of nested stack we will present an extended example which demonstrates the main features of our algorithm.

Example 5

For $F_{AC} = \{f,g\}$, $F_{NAC} = \{a,b,c,h,k\}$ and $<_{F \cup X}$ given by the sequence a,b,c,h,k,f,g,u,x,y,z we consider the AC-matching problem in ordered normal form

$$h(f[\, g[\, k(u), k(v)\,], x, y\,], f[\, u, x^2, z]) = h(f[\, b^3, c,^2\, g[\, k(f[\, a, b\,]), k(f[\, a^2\,])\,]\,], f[\, a^6, b^2\,]).$$

Decomposition yields

(1) $f[\, g[\, k(u), k(v)\,], x, y\,] = f[\, b^3, c^2, g[\, k(f[\, a, b\,]), k(f[\, a^2\,])\,]\,]$ and

(2) $f[\, u, x^2, z] = f[\, a^6, b^2\,]$.

Intelligent branching for (1) now results in the following reduced problem:

(2) $f[\, u, x^2, z] = f[\, a^6, b^2\,]$.

(3) $g[\, k(u), k(v)\,] = g[\, k(f[\, a, b\,]), k(f[\, a^2\,])\,]$ and

(4) $f[\, x, y\,] = f[\, b^3, c^2\,]$.

By intelligent branching for (3) we get the two possibilities

(5) $k(u) = k(f[\, a, b\,])$

(6) $k(v) = k(f[\, a^2\,])$

or

(7) $k(u) = k(f[\, a^2\,])$

(8) $k(v) = k(f[\, a, b\,])$.

Choosing the first branch with (2), (4), (5) and (6) decomposition and deletion of substituted variables (here: u) leads to

(2') $f[\, x^2, z] = f[\, a^5, b\,]$

(4) $f[\, x, y\,] = f[\, b^3, c^2\,]$

(9) $u = f[\, a, b\,]$

(10) $v = f[\, a^2\,]$.

Now constraint propagation for (2') and (4) shows that there is no possible assignment left for x. Thus we have to backtrack and consider (2), (4), (7) and (8). Again using decomposition and deletion of substituted variables we get

(2'') $f[\, x^2, z] = f[\, a^4, b^2\,]$

(4) $f[\, x, y\,] = f[\, b^3, c^2\,]$

(11) $u = f[\, a^2\,]$

(12) $v = f[\, a, b\,]$.

Constraint propagation for (2'') and (4) results in only one remaining possibility for x, namely x=b. Thus by deletion of x respectively b in (2'') and (4) we immediately get the only solution (representant) of the original problem:

$$u = f[\, a^2\,], v = f[\, a, b\,], x = b, y = f[\, b^2, c^2\,], z = f[\, a^4\,].$$

4. Constraint Propagation for other kinds of Equational Problems

Constraint propagation techniques may also be successfully applied to other kinds of equational problems, for example E-matching and E-unification. We will sketch two cases, namely ACI-matching and AC-unification.

4.1 ACI-Matching using Constraint Propagation

In [Mz85] it has been shown how to construct an ACI-matching algorithm in the case where E=ACI consists of the associativity, commutativity and idempotency axioms for some function symbols from F. Similar to the AC-case one may define (ordered) flattened normal forms for ACI-equivalent terms. Argument lists of ACI-operators in such flattened normal forms are regarded not as multisets but as sets due to the idempotency axioms. Reduction of simplified non-trivial ACI-matching problems may be performed as follows (stated for the variable-only case):
Let $s=f\{x_1,\ldots,x_m\}$, $t=\{t_1,\ldots,t_n\}$, $f\in F_{ACI}$ and an ACI-matching problem set $\Gamma=\Gamma'\cup\{s=t\}$ be given.Then ACI-reduction means to apply in all possible ways the ACI-mutation rule

$$\Gamma' \cup \{s = t\} \rightarrow \Gamma' \cup \{x_i = f\{t_1^{ki1},\ldots,t_n^{kin}\} \mid 1\le i\le m\}$$

with the restrictions $0 \le kij \le 1$, $\Sigma_{j=1,\ldots n}\, kij \ge 1$ and $\Sigma_{i=1,\ldots m}\, kij \ge 1$ for all i,j, $1 \le i \le m$, $1 \le j \le n$. Instead of choosing this mutation rule we may also work globally w.r.t. Γ and propagate constraints for the possible assignments for all left hand side arguments. Thus we can cut many failure branches in advance if there are mutual dependencies .

Example 6

For $F_{ACI} = \{f\}$ we consider the ACI-matching problem

$$h(f(x, y), h(f(x, z), f(y, z))) = h(f(a, b), h(f(a, c), f(b, c))).$$

Transformed into (ordered) normal form decomposition yields

(I) $f\{x, y\} = f\{a, b\}$,

(II) $f\{x, z\} = f\{a, c\}$ and

(III) $f\{y, z\} = f\{b, c\}$.

Mutation for (I) would give the 7 subproblems

I.1 $\{x = a, y = b\}$

I.2 $\{x = a, y = f\{a, b\}\}$

I.3 $\{x = b, y = a\}$

I.4 $\{x = b, y = f\{a, b\}\}$

I.5 $\{x = f\{a, b\}, y = a\}$

I.6 $\{x = f\{a, b\}, y = b\}$ and

I.7 $\{x = f\{a, b\}, y = f\{a, b\}\}$.

Thus in the worst case we would have to investigate the 7 reduced problems I.i $\cup$ {II,III}, i=1,...7. Using global constraint propagation we can deduce from the general form of all possible assignments for the left hand side variables

$$x = f\{a^{xa}, b^{xb}, c^{xc}\},\ y = f\{a^{ya}, b^{yb}, c^{yc}\} \text{ and } z = f\{a^{za}, b^{zb}, c^{zc}\}$$

with coefficients 0 or 1 the constraints

$$xb = xc = 0,\ xa = 1,\ ya = yc = 0,\ yb = 1,\ za = zb = 0,\ zc = 1.$$

This immediately leads to the only solution (representant) of the original problem

$$\{x = a, y = b, z = c\}.$$

4.2 AC-Unification using Constraint Propagation

It is well-known that there exists a close connection between solving AC-unification problems and solving corresponding homogeneous linear diophantine equations over $\mathbb{N}\setminus\{0\}$ (see [LiSi76], [HeSi85], [St81], [Fa83] or [Fo85] for details). For the simplified non-trivial AC-unification problem

$$(*)\quad f[s_1^{k1},\ldots,s_m^{km}] = f[t_1^{l1},\ldots,t_n^{ln}] \quad \text{with } f \in F_{AC}$$

the corresponding diophantine equation in the unknowns x_i, y_j is

$$(**)\quad \Sigma_{i=1,\ldots,m}\, x_i\, k_i \;=\; \Sigma_{j=1,\ldots,n}\, y_j\, l_j.$$

From the solutions of (**) a complete set of AC-unifiers can be constructed. If we have to solve a conjunction of simplified non-trivial AC-unification problems of the form (*), we can apply constraint propagation roughly in the following way: Instead of choosing a single problem and computing a (finite) solution basis for the corresponding diophantine equation we can also consider some or all problems simultaneously. Solving the corresponding system of diophantine equations as a whole may be much more efficient in cases where there are dependencies between these diophantine equations in the sense of common unknowns. This effect is demonstrated by

Example 7

For $F_{AC} = \{f\}$ consider the AC-unification problem

$$h(f(f(x,x),f(x,x)),f(y,f(y,z))) = h(f(y,z),f(f(x,x),x)).$$

Transformation into normal form and decomposition yields the conjunction of the two subproblems

(I) $f[x^4] = f[y,z]$ and

(II) $f[y^2,z] = f[x^3]$.

The corresponding diophantine equations to be solved are

(I*) $4x' = y' + z'$ and

(II*) $2y' + z' = 3x'$.

Solving (I*) leads to 5 basic solutions for (x',y',z'), namely (1,0,4), (1,1,3), (1,2,2), (1,3,1) and (1,4,0). Associating a new variable to each basic solution and considering any combination of these basic solutions (satisfying certain restrictions) we get a complete set of AC-unifiers for (I) of cardinality 29. But none of these solutions is compatible with (II). Thus the problem is unsolvable. This can be detected much more efficiently by solving (I*) and (II*) simultaneously.Using elimination techniques from linear algebra we get for example from (I*): z'=4x'-y', which implies together with (II*): x'+y'=0. This equation obviously has no solution over $\mathbb{N}\setminus\{0\}$. Therefore the initial AC-unification problem is also unsolvable.

Of course the construction of a correct and terminating AC-unification algorithm which incorporates the constraint propagation ideas described requires a much more detailed investigation which is beyond the scope of this work. In particular one has to be able to handle systems of linear diophantine equations in a systematic and efficient way. Moreover new results on AC-unification should also be taken into account (cf. [Ki87], [Bü88], [LiCh88]).

5. Conclusion

We have shown how to apply constraint propagation techniques for reducing the search space of AC-matching problems. The resulting algorithm is well-suited for a certain class of non-linear patterns and for a systematic early detection of unsolvability. It has also been sketched how to apply such techniques to other kinds of equational problems like E-matching and E-unification for certain E. A promising perspective for future work might be to develop good criteria for deciding in which cases global reasoning (using constraint propagation) or local reasoning (handling a single subproblem) will be better in terms of efficiency. This could lead to a new kind of efficient "hybrid" algorithms for the equational problems in mind.

Acknowledgements. We would like to thank J.Avenhaus, H.-J.Bürckert, K.Madlener, J.Müller and M.Schmidt-Schauss for their hints and criticisms on earlier drafts of this paper.

This work was supported by the the Deutsche Forschungsgemeinschaft, SFB 314: Künstliche Intelligenz - Wissensbasierte Systeme.

References

[BeKaNa85] Benanav, D., Kapur, D., Narendran, P.: Complexity of Matching Problems, Proc. of 1st Conf. on Rewriting Techniques and Applications, Dijon, France, May 1985, Lecture Notes in Computer Science 202

[Bü88] Bürckert, H.J.: AC-Unification is AC1-Disunification, 2nd Workshop on Unification, Val d'Ajol, France, 1988

[Co88] Comon, H.: Unification et Disunification. Theories et Applications, PHD Thesis, University of Grenoble, France, 1988

[Fa84] Fages, F.: Associative-commutative unification, Proc. of 7th Conference on Automated Deduction, Napa Valley, California, USA, 1984, Springer Verlag

[Fo85] Fortenbacher, A.: An Algebraic Approach to Unification under Associativity and Commutativity, Proc. of 1st Conf. on Rewriting Techniques and Applications, Dijon, France, May 1985, Lecture Notes in Computer Science 202

[HeSi85] Herold, A., Siekmann, J.: Unification in Abelian Semigroups, Memo SEKI-85-III, Universität Kaiserslautern, 1985

[Hu79] Hullot, J.M.: Associative-commutative Pattern Matching, Proc. of IJCAI-79, Tokyo, Japan, August 1979

[Hu80] Hullot, J.M.: Compilation de Formes Canoniques dans les Theories Equationnelles, These de 3eme Cycle, Universite de Paris Sud, Orsay, France, 1980

[Ki85] Kirchner, C.: Methodes et outils de conception systematique d'algorithmes d'unification dans les theories equationnelles, These d'Etat de l'Universite de Nancy I, France, 1985

[Ki87] Kirchner, C.: Methods and Tools for Equational Unification, Proc. of the

Colloquium on the Resolution of Equations in Algebraic Structures, May 1987, Austin, Texas

[LiCh88] Lincoln, P., Christian, J.: Adventures in Associative-Commutative Unification (A Summary), Proc. of 9th CADE, Argonne, USA, 1988

[LiSi76] Livesey, M., Siekmann, J.: Unification of Bags and Sets, Technical Report, Institut für Informatik I, Universität Karlsruhe, 1976

[MaMo82] Martelli, A., Montanari, U.: An efficient unification algorithm, ACM Transactions on Programming Languages and Systems, 4/2, 1982

[Mz83] Mzali, J.: Algorithme de Filtrage Associatif Commutatif, Internal Report 83 R 060, Centre de Recherche en Informatique de Nancy, France, 1983

[Mz85] Mzali, J.: Filtrage Associatif, Commutatif ou Idempotent, Internal Report 85 R 25, Centre de Recherche en Informatique de Nancy, France, 1985

[Mz86] Mzali, J.: Matching with Distributivity, Proc. of 7th Conference on Automated Deduction, Oxford, England, July 1986, Lecture Notes in Computer Science 230

[Sc88] Schmidt-Schauss, M.: Unification in a Combination of Arbitrary Disjoint Equational Theories, Proc. of 8th CADE, Argonne, USA, 1988

[St81] Stickel, M.E.: A unification algorithm for associative-commutative functions, Journal of the Association for Computing Machinery, 28, 1981