An alternative for handling AC[1]

Matthias Fuchs
Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
67653 Kaiserslautern
Germany
E-mail: fuchs@informatik.uni-kl.de

March 1994

Abstract

A method for efficiently handling associativity and commutativity (AC) in implementations of (equational) theorem provers without incorporating AC as an underlying theory will be presented. The key of substantial efficiency gains resides in a more suitable representation of permutation-equations (such as f(x,f(y,z))=f(y,f(z,x)) for instance). By representing these permutation-equations through permutations in the mathematical sense (i.e. bijective functions $\sigma:\{1,..,n\}\rightarrow\{1,..,n\}$), and by applying adapted and specialized inference rules, we can cope more appropriately with the fact that permutation-equations are playing a particular role. Moreover, a number of restrictions concerning application and generation of permutation-equations can be found that would not be possible in this extent when treating permutation-equations just like any other equation. Thus, further improvements in efficiency can be achieved.

0. Introduction

Whenever a theorem proving system has to deal with functions that are both associative and commutative (AC), it will face certain specific problems. These problems largely depend on the method employed to cope with the AC-property. Basically there are two extremes of handling the AC-property.

The first extreme used for example by unfailing completion (see [BDP89]) consists in ignoring completely the fact that the AC-property holds for certain functions. Hence, associativity and commutativity and all the numerous permutation-equations creatable are treated like any other equation. But the unsophisticated generation and application of permutation-equations quite often causes the respective proving system to perform poorly, because permutation-equations allow a lot of reductions and critical pairing, entailing vast matching resp. unification efforts. But quite a considerable part of these reductions and critical pairing is unnecessary and thus represents redundant effort. Moreover, permutation-equations can be created in many different ways, what leads to immense costs for eliminating duplicates. We shall see in the following sections how these striking drawbacks can be compensated for by using an appropriately designed representation and adapted inference rules for these problematic permutation-equations.

The second extreme for dealing with AC is incorporating the AC-property into the proving

1. This work was supported by the Deutsche Forschungsgemeinschaft (DFG).

system, yielding proofs resp. proving modulo AC (e.g. [PS81]). This way of proceeding calls for sophisticated algorithms (such as AC-unification resp. the computation of complete sets of AC-unifiers (e.g. [St81]), reduction modulo AC etc.) in high complexity classes. These algorithms are theoretically well founded and sometimes even indispensable, e.g. when attempting to compute a complete set of rules (modulo AC). But in the case where we are not interested in computing complete sets, intending only to prove a given theorem, the algorithms in question are off-putting due to their complexity, and practical performance does not necessarily encourage their utilization.

Therefore, an alternative method will be proposed which is very close to the first extreme, yet causing impressive efficiency gains. The main reason for such improvements lies in removing the permutation-equations from the set of equations and in representing them by pairs of permutations (σ_1,σ_2). The generation process as well as the application of permutation-equations in this new format can then be conveniently designed so as to avoid redundancies and inefficiencies that were inherently present before. As we shall see, the advantages of the new representation are remarkable, giving rise to substantial speed-ups.
In the following sections the details will be presented. The first section will introduce the general representation principle and some of its properties that are of interest in subsequent sections. In the sequel, the generation of permutation-equations in the format described in the preceding section is to be discussed. This generation process consists mainly in an enumeration that can be done without redundant effort. After that, reduction and critical pairing will be investigated, which both are based on exchanging arguments, thus avoiding unification and matching. In that section, we shall also take a look at issues concerning subsumption under these altered conditions. Finally, we will discuss correctness and completeness of the presented method. A conclusive summary will bring this report to a close.
A comparison of run times for some problems with and without the use of our method can be found in the appendix.

1. The general representation principle

Proofs involving functions with the AC-property usually confront proof systems with a considerable number of permutation-equations such as f(x,f(y,f(z,u)))=f(z,f(x,f(u,y))), for instance. When employing methods for incorporating the AC-property as a theory, we trade the existence of these problematic permutation-equations for the necessity of highly complex and costly algorithms. Here, we do not want to use such methods and consequently have to admit permutation-equations. Hence we must analyze the essential difficulties caused by permutation-equations and we must find other ways to cope with them.
The main problem caused by permutation-equations is the fact that they can be used for a lot of reductions (entailing many matching efforts), and also create numerous new equations via critical pairing (causing unification efforts), especially when performing critical pairing among each other. This latter action is particularly responsible for a substantial amount of redundancy since only a small percentage of the critical pairs thus creatable actually are not yet available permutation-equations. As a consequence, expensive subsumption tests must be performed which allow to get rid of “doubles”. Furthermore, unification and matching which precede critical pairing resp. reduction are an awkward way of performing the essential effect of a permutation-equation consisting in the exchange of arguments according to the respective permutation. It therefore stands to reason to treat and represent permutation-equations by what they express: permutations.
A permutation is obviously represented by a bijective function $\sigma:\Re\rightarrow\Re$ with $\Re=\{1,..,n\}$

being a finite subset of the set of natural numbers. We define $D(\sigma)=|\Re|(=n)$. Furthermore, if $\sigma(i)=k_i$ for all $1\leq i\leq D(\sigma)=n$, then we write $\sigma=(k_1,..,k_n)$ for short.
Since every permutation-equation can be read from left to right and from right to left and thus stands for two permutations in general, we have to deal with pairs of permutations in order to replace permutation-equations properly.

Example:
Let us consider the permutation-equation PE := f(x,f(y,z))=f(y,f(z,x)).
Reading it from left to right we recognize that x is moved from the first position to the third, y from the second position to the first and z from the third position to the second. Thus, viewed this way, PE can be represented by the permutation σ_1 with $\sigma_1(1)=3$, $\sigma_1(2)=1$, $\sigma_1(3)=2$ or $\sigma_1=(3,1,2)$ for short. Analogously, reading PE from right to left, we realize that σ_2 with $\sigma_2(1)=2$, $\sigma_2(2)=3$, $\sigma_2(3)=1$ or $\sigma_2=(2,3,1)$ for short is the conjoined permutation. By convention, PE as a whole is represented by $(\sigma_1,\sigma_2)=((3,1,2),(2,3,1))$. (Obviously, the order of those two tuples doesn't matter at all, since it is also irrelevant if we write f(x,f(y,z))=f(y,f(z,x)) or f(y,f(z,x))=f(x,f(y,z)). That means, we could as well have chosen (σ_2,σ_1) to represent PE.)

Notes:
- If (σ_1,σ_2) represents a permutation-equation, then $\sigma_1\cdot\sigma_2=\sigma_2\cdot\sigma_1=id$, i.e. $\sigma_2=\sigma_1^{-1}$ resp. $\sigma_1=\sigma_2^{-1}$.
- Applying a permutation σ to a term $f(t_1,..,t_n)$ results in the term $f(s_1,..,s_n)$, where $s_{\sigma(i)}\equiv t_i$.
- Be aware that the application of a permutation σ to a term $f(t_1,..,t_n)$, as it was just described, does *not* result in $f(t_{\sigma(1)},..,t_{\sigma(n)})$, but in $f(t_{\sigma'(1)},..,t_{\sigma'(n)})$, where $\sigma'=\sigma^{-1}$.

By choosing the pair-of-permutation format, we eliminate another nasty property of permutation-equations when being represented by the standard format.

Example:
PE_1:=f(x,f(y,z))=f(y,f(z,x))
PE_2:=f(x,f(y,z))=f(z,f(x,y))
PE_1 and PE_2 look quite different at first sight, but turn out to be identical modulo variable renaming (rename z to x, x to y and y to z in PE_2), a fact that has to be discovered by subsumption tests involving matching. These efforts are not necessary when representing PE_1 and PE_2 by their respective pairs of permutations ((3,1,2),(2,3,1)) and ((2,3,1),(3,1,2)), providing us with an unambiguous representation "modulo swapping sides". The full advantage will become obvious in connection with an ordering on permutations and the systematic generation of pairs of permutations as it will be introduced in the following section.

2. The systematic generation of pairs of permutations

We shall now come to know the procedure which provides us with the needed permutation-equations as pairs of permutations. This procedure is based on enumeration rather than critical pairing, what increases efficiency remarkably.
First of all, it must be notified that the associativity remains in its standard format, usually being incorporated as a rewrite rule $f(f(x,y),z)\rightarrow f(x,f(y,z))$ or $f(x,f(y,z))\rightarrow f(f(x,y),z)$. We need it in this position not only for normal forms (this is truly of secondary interest), but

especially for preserving completeness as we shall see later on in section 4. All other permutation-equations (including commutativity) are to be removed whenever they turn up (after having been generated by some critical pairing or by being part of the input). An enumeration process -whose presentation follows- is entrusted with the generation of the corresponding pairs of permutations. (How these pairs can be applied for reductions and critical pairing is the subject of the subsequent section 3.)
The enumeration process is founded on an ordering of the permutations, which is also handy for some optimization. We shall therefore take a look at this ordering first.
The chosen ordering is the obvious lexicographic ordering which is defined as follows:

Definition:
Let σ_1,σ_2 be two permutations.
$\sigma_1 <_P \sigma_2$ iff
(a) $D(\sigma_1)<D(\sigma_2)$ or
(b) $D(\sigma_1)=D(\sigma_2)$ and there is $1 \le k \le D(\sigma_1)$ with $\sigma_1(i)=\sigma_2(i)$ for all $1\le i<k$ and $\sigma_1(k)<\sigma_2(k)$
($<$ is the usual ordering on natural numbers.)

(Note: $<_P$ is total.)

It is straight forward to enumerate permutations by starting with the smallest one and computing one by one the immediate successors w.r.t. $<_P$. Since we are not interested in identities (id), i.e. permutations σ with $\sigma(i)=i$ for all $i\in\Re$, we skip these by proceeding to their immediate successor (1,..,n-2,n,n-1), where $n=D(\sigma)$. Furthermore, we certainly do not have to care about "permutations" with $D(\sigma)<2$. (Consequently, the starting permutation would be (2,1).)

In section 1 we argued that permutation-equations obviously are to be represented by *pairs* of permutations (σ_1,σ_2). Since $\sigma_1\cdot\sigma_2=\sigma_2\cdot\sigma_1=id$ for all permutation-equations (i.e. $\sigma_2=\sigma_1^{-1}$ resp. $\sigma_1=\sigma_2^{-1}$), it suffices to enumerate the σ_1's and to compute σ_2 according to $\sigma_2\cdot\sigma_1=id$ (what can be done in time $O(D(\sigma_1))$). Due to the ordering $<_P$ on permutations we have a very efficient possibility to check whether a pair of permutations (σ_1,σ_2) generated this way has already been created before (to be exact, we have to check whether (σ_2,σ_1) is already available; in this case (σ_1,σ_2) is redundant because it does not bear new information. This fact will become clear when the application of pairs of permutations for reduction and critical pairing is introduced in section 3): We know that whenever a permutation $\delta_1<_P\sigma_1$, then (δ_1,δ_2) must have been enumerated before (σ_1,σ_2). Hence, if $\sigma_2<_P\sigma_1$, (σ_2,σ_1) is already existing and consequently (σ_1,σ_2) can be ignored. (The test "$\sigma_2<_P\sigma_1$?" can also be performed in time $O(D(\sigma_1))$.)
It remains to devise an algorithm for computing the immediate successor of a permutation σ w.r.t. $<_P$. The algorithm is based on the following observation:
Let $I_\sigma=\{i\in\{1,..,D(\sigma)-1\} \mid \sigma(i)<\sigma(i+1)\}$.
If I_σ is empty, then $id_{D(\sigma)+1}=(1,..,D(\sigma),D(\sigma)+1)$ is the immediate successor of σ w.r.t. $<_P$. (Since we want to skip identities, we shall later choose in this case $(1,..,D(\sigma)-1,D(\sigma)+1,D(\sigma))$ as "immediate" successor.)
If I_σ is not empty, then let $m=\max(I)$, $k=\min(\{\sigma(i) \mid m<i\le D(\sigma)\wedge\sigma(i)>\sigma(m)\})$ (k exists because $m<D(\sigma)$ and $\sigma(m)<\sigma(m+1)$). Choose furthermore j so that $\sigma(j)=k$.
For all $1\le x\le D(\sigma)$, let $\sigma_{succ}(x)$ be

$\sigma(x)$	if $1\le x<m$,
k	if $x=m$,
$\sigma(m)$	if $x=j$

$\sigma(D(\sigma)-x+m+1)$ else.

Obviously, $\sigma <_P \sigma_{succ}$ since $\sigma(i)=\sigma_{succ}(i)$ for all $1 \le i < m$ and $\sigma(m) < k = \sigma_{succ}(m)$.
We still have to prove that σ_{succ} is an *immediate* successor, i.e. for all permutations δ with $\sigma <_P \delta$, $\delta = \sigma_{succ}$ or $\sigma_{succ} <_P \delta$.

Proof:
If $I_\sigma = \emptyset$, then $\sigma_{succ} = id_{D(\sigma)+1}$ clearly is the immediate successor of σ.
If $I_\sigma \neq \emptyset$, then $D(\sigma)=D(\sigma_{succ})$. Let δ be a permutation satisfying $\sigma <_P \delta$. If $D(\delta) > D(\sigma) = D(\sigma_{succ})$, we are done.
The case $D(\delta) < D(\sigma)$ is inadmissible because of the prerequisite $\sigma <_P \delta$.
Therefore $D(\delta)=D(\sigma)=D(\sigma_{succ})=:n$. Since we demanded $\sigma <_P \delta$, there is $m' \in \{1,..,n\}$ with $\sigma(i)=\delta(i)$ for all $1 \le i < m'$ and $\sigma(m') < \delta(m')$.

(a) $m > m'$: consequently $\sigma_{succ}(m') = \sigma(m') < \delta(m')$ and thus $\sigma_{succ} <_P \delta$.
(b) $m < m'$: contradiction to $\sigma <_P \delta$ since due to the choice of m $\sigma(i) > \sigma(i+1)$ for all $m < i < n$.
(c) $m = m'$: We know that $\sigma(i)=\sigma_{succ}(i)=\delta(i)$ for all $1 \le i < m$, $\sigma_{succ}(m) > \sigma(m)$ and $\delta(m) > \sigma(m)$. Due to the choice of m, $\sigma(i) > \sigma(i+1)$ for all $m < i < n$. Furthermore, $\sigma_{succ}(m)=k$, $k > \sigma(m)$ and for all $m < i \le n$ $\sigma(i) > \sigma(m)$ implies $k \le \sigma(i)$ (❁). Moreover, $\sigma_{succ}(i) < \sigma_{succ}(i+1)$ for all $m < i < n$ (✯). Three cases arise:
    (i) $\delta(m) > \sigma_{succ}(m)$; In this case, $\sigma_{succ} <_P \delta$.
    (ii) $\delta(m) < \sigma_{succ}(m)$; Here, $\delta(m) < k \wedge \delta(m) > \sigma(m) \wedge \delta(m) \in \{\sigma(i) \mid m < i \le n\}$, what contradicts (❁) or the prerequisite $\sigma <_P \delta$.
    (iii) $\delta(m) = \sigma_{succ}(m)$; We then have $\delta = \sigma_{succ} \vee \sigma_{succ} <_P \delta$ since (✯) holds, and because of $\{\delta(i) \mid m < i \le n\} = \{\sigma_{succ}(i) \mid m < i \le n\}$. ❐

Algorithmic formulation:

```
PERMSUCC
input: σ=(i_1,..,i_n);
m:=n-1;
while (m>0) and (i_m>i_m+1) do
   m:=m-1;
if (m≤0) then
   return σ_succ=(1,..,n-1,n+1,n) /* identity skipped */
else begin
   j:=m+1;
   k:=i_m+1;
   q:=m+1;
   while (q<n) do
   begin
      if (i_q>i_m) and (i_q<k) then
      begin
         k:=i_q;
         j:=q
      end;
      q:=q+1
   end;
   return σ_succ=(i_1,..,i_m-1,k,i_n,..,i_j+1,i_m,i_j-1,..,i_m+1)
end;
```

PERMSUCC computes the immediate successor of σ (skipping identities) in time $O(D(\sigma))$.

It is now a simple task to write an algorithm for computing the "next" pair of permutations when given a pair (σ_1,σ_2).

```
repeat
    σ1:=PERMSUCC(σ1);
    for i:=1 to D(σ1) do
        σ2(σ1(i)):=i;
    if (σ2<Pσ1) then
        found:=false
    else
        found:=true
until found;
return (σ1,σ2);
```

Consequently, the computation of the next pair of permutations can also be done in time $O(D(\sigma_1))$.

Notes:

If there is no fact known to the proof system containing a subterm which has an AC function symbol as top-level symbol and whose arity when flattened exceeds n, then the enumeration process of pairs of permutations for the respective AC function symbol can be stopped when that threshold n is reached (i.e. when the first pair (σ,σ') with $D(\sigma)>n$ is generated). Naturally, the enumeration must continue if n increases.

Furthermore, the enumeration of pairs of permutations must be integrated into the proof procedure. The best way to accomplish this seems to be by interleaving the enumeration process with some suitable inference, e.g. enumerate the next pair of permutations after having selected and worked on $n\geq 1$ critical pairs. At this point, note that the generation of pairs of permutations and their applications as described in the following section 3 can naturally be regarded as inferences themselves. So, there will be no inconsistency in notation when the generally preferred way of describing a proof system by a set of inference rules is employed.

We have learned so far how permutation-equations can be represented and generated. In the next section, we shall turn our attention to the vital operations "reduction" and "critical pairing", outlining the way they can be performed utilizing pairs of permutations instead of permutation-equations in term form. We shall see that further efficiency increasing measures can be taken on account of the more appropriate representation.

3. Reduction and critical pairing

Since we remove all permutation-equations (except the rule or equation expressing associativity) from the current system of equations during a proof, replacing them gradually by pairs of permutations, we have to provide adapted strategies for reduction and critical pairing in order to be able to preserve completeness. For this purpose we shall work with *flat terms*.

A flat term corresponding to a term $t\equiv f(s_1,s_2)$ can be obtained by applying the following transformation:

$\text{flat\_term}(f(s_1,..,s_n))=f(s_1,..,s_n)$ iff there is no s_i with $s_i\equiv f(t_1,t_2)$, where f is an AC function symbol and $n\geq 2$.

Otherwise, $\text{flat\_term}(f(s_1,..,s_n))=\text{flat\_term}(f(s_1,..,s_{i-1},t_1,t_2,s_{i+1},..,s_n))$.

The concept of flat terms together with pairs of permutations allow to execute reductions, critical pairing and subsumption tests conveniently without having to employ matching or unification, thus contributing a great deal to remarkable efficiency gains. Details will be explained in the subsequent discussions.

3.1. Reduction

Reducing a given term t with a given equation $s_1=s_2$ generally involves finding a match τ and a subterm $t|p$ of t so that $\tau(s_A)\equiv t|p$, and replacing $\tau(s_A)$ by $\tau(s_B)$ results in a term $t[p\leftarrow\tau(s_B)]$ which is smaller w.r.t. a given reduction ordering > ($s_A\in\{s_1,s_2\}$, $s_B\in\{s_1,s_2\}-\{s_A\}$). This principle naturally also applies if $s_1=s_2$ is a permutation-equation. But in this special case the matching effort is not necessary. It is sufficient to find a subterm $t|p$ of t with $t|p\equiv f(t_1,t_2)$, where f is the AC function symbol $s_1=s_2$ expresses permutations for, and there is no subterm $t|q$ of t so that $t|q\equiv f(v_1,v_2)$ and p=q.1 or p=q.2. Let then t' be the flat term corresponding to $t|p$, i.e. $t'=flat\_term(t|p)=f(t_1',..,t_n')$. Furthermore, (σ,σ') be the pair of permutations superseding $s_1=s_2$. If $D(\sigma)\neq n$, then there is no point in trying to use (σ,σ') for an attempt to reduce t' resp. $t|p$. If $D(\sigma)>n$, then a range of pairs of permutations (ϕ,ϕ') have already been enumerated with $D(\phi)=n$ that will take effect. Otherwise, if $D(\sigma)<n$, then we might as well wait until the proper pairs (δ,δ') with $D(\delta)=n$ will appear, instead of taking care of subterms $\bar{t}$ of $t|p$ with $flat\_term(\bar{t})=f(\bar{t}_1,..,\bar{t}_m)$ and $m=D(\sigma)$ or even using not completely flattened terms. Both alternatives would cause unnecessary and therefore redundant effort. This restriction is compensated for by the fact that permutations are as well related by some kind of “sub-permutation” relation. Take for instance (2,1) which is comprised in (2,1,3) or (1,3,2).

Example:
$t\equiv f(a,f(b,c))$, f be AC.
Using commutativity ((2,1),(2,1)), what is not recommendable here, yields f(f(b,c),a) resp. f(b,f(c,a)) (assuming that the equation representing associativity is incorporated as rewrite rule $f(f(x,y),z)\rightarrow f(x,f(y,z))$) and f(a,f(c,b)). f(b,f(c,a)) would be produced when not flattening completely, f(a,f(c,b)) by considering the subterm f(b,c).) The same results will also be attained by applying the “left” sides of ((3,1,2),(2,3,1)) resp. ((1,3,2),(1,3,2)) to the totally flattened term f(a,b,c).

Consequently, by proceeding in the outlined manner, the number of reductions which must be considered as wasted effort can be cut down considerably.

So, let us suppose that $D(\sigma)=n$. We then have two possibilities to employ (σ,σ') which are equivalent to what could also be done if we disposed of $t|p$ and $s_1=s_2$. Applying the permutation σ resp. σ' to t' we obtain $r_1\equiv f(t'_{\sigma'(1)},..,t'_{\sigma'(n)})$ resp. $r_2\equiv f(t'_{\sigma(1)},..,t'_{\sigma(n)})$. t is reduced to $u_1\equiv t[p\leftarrow\Psi(r_1)]$ resp. $u_2\equiv t[p\leftarrow\Psi(r_2)]$ if $t>u_1$ resp. $t>u_2$. $\Psi(t)$ denotes a structured (and possibly completely interreduced) term corresponding to the flat term t. (If we have the associativity as a rewrite rule $f(f(x,y),z)\rightarrow f(x,f(y,z))$ resp. $f(x,f(y,z))\rightarrow f(f(x,y),z)$, then it is recommendable to have Ψ build the right- resp. left-parenthesized form to avoid further reductions.)

3.2. Critical pairs

With the principles just learned from the reduction of a term t with a pair of permutations in mind, the generation of critical pairs by (virtually) overlapping a permutation-equation (represented by a pair of permutations (σ,σ'), therefore “virtually”) into an equation t=s or

s=t or a rule $t \to s$ imposes no major problems.
Analogously to the reduction case discussed in 3.1, we are looking for places p in t so that $t|p=t' \equiv f(t_1,t_2)$, where f is AC and (σ,σ') is related to f, $flat\_term(t')=f(t_1',..,t_n')$ and $n=D(\sigma)$. (The reason for the latter condition is the same as before.) When attempting a reduction, we were contemplating the two possible outcomes of an application of (σ,σ'), namely $r_1 \equiv f(t'_{\sigma'(1)},..,t'_{\sigma'(n)})$ and $r_2 \equiv f(t'_{\sigma(1)},..,t'_{\sigma(n)})$, and we were interested in establishing $t>u_1 \equiv t[p \leftarrow \Psi(r_1)]$ or $t>u_2 \equiv t[p \leftarrow \Psi(r_2)]$. For the creation of critical pairs these comparisons with the reduction ordering > are to be changed to $\neg(u_1>t)$ resp. $\neg(u_2>t)$. But apart from that, critical pairing is similar to reducing, and we obtain the critical pairs $u_1=s$ resp. $u_2=s$, provided that $\neg(u_1>t)$ and $\neg(u_2>t)$ respectively.
There is no need for unification as it would be the case if the permutation-equations were not handled separately.
Furthermore, we do not need to generate critical pairs by overlapping into permutation-equations if the used reduction ordering meets certain requirements (see section 4). The equations and rules derived from such an operation can then as well be attained by overlapping into associativity (which still *is* a member of the set of rules or equations as we have already stipulated), in combination with reductions and overlapping into the resulting terms, using pairs of permutations in the described way.
Hence, we dispose in this case of further restrictions allowing to avoid redundancy and computational effort.
Section 4 will explain more exactly the effects of the presented method on correctness and completeness, especially regarding the latter w.r.t. the constraints proposed so far. But before that, let us investigate concisely subsumption in connection with the representation of permutation-equations as pairs of permutations.

3.3. Subsumption

Subsumption generally stands for precluding formulas that are subsumed by other more general ones. It is not an indispensable feature of proving systems, but often serves the desirable purpose of reducing the amount of formulas kept by a system. This can have a deciding influence on the performance of proving systems. For this reason we should cover this subject from a point of view induced by the new method.
For the case that is interesting for us in this context, namely the subsumption of an equation by another one, we have in general the following definition.

Definition:
s=t *is subsumed* by u=v iff there is a place p in both s and t, and there is a substitution τ (a match), so that $s|p=\tau(\bar{u})$, $s[p \leftarrow \tau(\bar{v})] \equiv t$, where $\bar{u} \in \{u,v\}$, $\bar{v} \in \{u,v\}-\{\bar{u}\}$.

(Note: For $p=\varepsilon$ we have $\tau(\bar{u})=\tau(\bar{v}) \equiv s=t$.)

Naturally, we are interested in the case where u=v is a permutation-equation which is represented by a pair of permutations. We are here in the fortunate position of not having to care about the exact shape of u=v except the fact that it is *some* permutation-equation. As a consequence, the test whether an equation s=t is subsumed by some permutation-equation can be reduced to finding a place p in both s and t, so that $s|p \equiv f(s_1,s_2)$, $t|p \equiv f(t_1,t_2)$, $s[p \leftarrow t|p] \equiv t$, f is AC and $flat\_term(s|p)=f(s_1',..,s_n') \equiv f(t'_{\sigma(1)},..,t'_{\sigma(n)})$, where $f(t_1',..,t_n')=flat\_term(t|p)$ and σ is a permutation with $D(\sigma)=n$. Again, the avoidance of matching and the fact that we do not have to search the current set of equations for an equation u=v that subsumes s=t make subsumption testing in this case more efficient.

(Note: With this subsumption test, permutation-equations that may still be present in term form can be eliminated since the test will succeed at $p=\varepsilon$ whenever s=t is a permutation-equation for some AC symbol.)

4. Correctness and completeness

In this section we shall take a look at aspects of our method concerning correctness and completeness.

As for correctness, it is quite clear that we do not perform an illegal, i.e. theory distorting operation when permuting arguments of AC functions, taking the respective flat term as a starting point. We shall therefore not go any further into this matter, devoting our attention to the less easily understandable subject of completeness.

It is satisfactory for us to show that we do not jeopardize completeness by using the pair of permutation representation instead of permutations in term form.

It is not hard to see that we do not lose any necessary critical pair that can be generated by overlapping a permutation-equation into a rule or equation according to the proceeding described in 3.2. Thus, we only have to make sure that the restriction not to perform overlaps into permutation-equations does not destroy completeness. For this purpose, we shall demonstrate that any critical pair resulting from an overlap into a permutation-equation can also be attained without such an overlap. As we shall see in the sequel, we cannot show this in complete generality. The reduction ordering has to meet some requirements which, fortunately, the most commonly used reduction orderings (LPO, KBO, RPO and polynomial orderings) do meet.

In the following, we shall not give an exact proof, contenting ourselves with a sketch of the basic ideas. For simplifying the notation let us suppose that the associativity is included as the rewrite rule $(\ast)$ $f(f(x,y),z)\rightarrow f(x,f(y,z))$ so that we can assume every term to be in normal form w.r.t. to that rule. Furthermore, $f(t_1,t_2,..,t_n)$ be the short form for $f(t_1,f(t_2,..,f(t_{n-1},t_n)..)$.

When overlapping a rule or the left side of an equation $l\sim r$ (i.e. $\sim\in\{\rightarrow,=\}$) into a permutation-equation P_{EQ} $f(x_1,..,x_n)=f(x_{\sigma(1)},..,x_{\sigma(n)})$ (i.e. (σ^{-1},σ) in pair notation), a critical pair $CP=<\tau(f(x_1,..,x_k,r)),\ \tau(f(x_{\sigma(1)},..,x_{\sigma(n)}))>$ will be created, where τ is the mgu of $f(x_{k+1},..,x_n)$ and l. Moreover, $k\geq 1$, since otherwise a top-level overlap has been performed what is already covered by overlaps of permutation-equations into rules or equations. As we do not have to consider overlaps into variables, we also know that $k<n-1$. (In addition $\neg(\tau(f(x_1,..,x_k,r)) > \tau(f(x_1,..,x_n)))$ and $\neg(\tau(f(x_{\sigma(1)},..,x_{\sigma(n)}))>\tau(f(x_1,..,x_n)))$ since otherwise CP needs not be examined. See [KB70].) W.l.o.g. P_{EQ} and $l\sim r$ do not have variables in common. We can now be more specific about the shape of l: l must have the form $f(l_1,..,l_m)$ with $m\geq 2$ and none of the l_j has the function symbol f as top-level symbol. Two cases must be distinguished:

(1) $m+k\geq n$

(2) $m+k<n$

In case (1), w.l.o.g. we have $\tau(z)\equiv z$ for all variables occurring in l. By overlapping $l\sim r$ into $(\ast)$ at position p=1, we obtain $CP_1=<f(l_1,..,l_m,z_1),f(r,z_1)>$, where z_1 is a new variable. When making a rule or equation from CP_1 we can continue this kind of overlapping (regardless of the orientation of such a CP_i when transformed into a rule), finally coming up with $CP_k=<f(l_1,.., l_m,z_1,..,z_k),\ f(r,z_1,..,z_k)>$ (the z_i's denoting new variables). With CP_k we can produce CP via a combination of reductions of CP and CP_k and *top-level* overlaps of permutation-equations into one or both sides of CP_k.

In case (2), we employ the same operation as in case (1) what will provide us with $CP_k=$

$<f(l_1,..,l_m,z_1,..,z_k), f(r,z_1,..,z_k)>$. l_m must be a variable because $m+n<k$. Since $k \geq 1$, we may overlap (✶) into the rule or equation resulting from CP_k so that l_m matches the subterm $f(x,y)$ coming from (✶).

Note:

An essential prerequisite for this overlap is the condition $\neg(f(r,z_1,..,z_k) > f(l_1,..,l_m,z_1,..,z_k))$. The most commonly used reduction orderings (e.g. LPO, RPO, KBO and polynomial orderings) meet this requirement, provided that $\neg(r>l)$ what we implicitly assume because of $l \rightarrow r$ or $l=r$, not $r \rightarrow l$.

As the result of the described overlap we get $CP_{k+1}=<f(l_1,..,l_{m-1},y_1,y_2,z_1,..,z_k), f(\tau_1(r),z_1,..,z_k)>$, where $\tau_1(l_m) \equiv f(y_1,y_2)$, $\tau_1(x) \equiv x$ for all $x \neq l_m$ and y_1, y_2 are new variables. Continuing this process by overlapping (✶) into the rule or equation made from CP_{k+i} so that one of the introduced (and new) variables $y_1,..,y_{i+1}$ is bound to $f(x,y)$ will eventually result in $CP_{k+n-m-k}=CP_{n-m}=<f(l_1,..,l_{m-1},y_1,..,y_{n-m-k+1},z_1,..,z_k),f(\tau_{n-m-k}(r),z_1,..,z_k)>$, where $\tau_{n-m-k}= f(y_1,..,y_{n-m-k+1})$ and $\tau_{n-m-k}(x) \equiv x$ for all $x \neq l_m$. CP_{n-m} can be transformed into CP as outlined in the discussion of case (1).

Note:

If a member in the chain of CP_j's is reducible, then the application of the same technique sketched above to the reducing rule will show that CP itself is finally reducible and hence not worth considering.

5. Summary

An important performance criterion of any proving system involving equality is its ability to cope with AC functions, i.e. functions that are both associative and commutative.Because the AC-property is responsible for a considerable amount of (partially redundant) effort spent on account of the so-called permutation-equations (e.g. $f(x,f(y,z))=f(y,f(z,x))$), it is most desirable to handle these efficiently.
The way we chose here to achieve this goal is founded on a representation of these permutation-equations that is more suitable than the usual term format. By representing them through pairs of permutations (σ_1,σ_2), a lot of efficiency increasing measures can be taken. Among the most striking improvements is the generation of the pairs of permutations via enumeration (instead of critical pairing), reduction, critical pairing and subsumption testing without having to employ unification or matching procedures, and the possibility to exploit efficiently the fact that overlaps into permutation-equations are not necessary, at least when utilizing "conventional" reduction orderings (KBO, LPO, RPO, polynomial orderings). It must be appended that the incorporation of associativity as a rule or equation in term form plays a vital role for not losing completeness. All other permutation-equations (including commutativity) are available as pairs of permutations only.
The advantage of this method resides in its conceptual simplicity due to its close relationship to the "simplest way to handle AC", i.e. not making any difference between permutation-equations and other equations at all. Nevertheless, the attained improvements are remarkable and encouraging, and the comparison of a range of proofs accomplished with and without the presented method corroborates this observation.

Appendix

The table depicted in this appendix compares the run times and some of the characteristics of proofs in pure equational logic accomplished by the DISCOUNT-system which is based on the unfailing Knuth-Bendix completion ([BDP89]).
There are two rows for each problem: the first row lists the results obtained when *not* employing the presented method, the second one those gained when using it.
It must be emphasized that in both cases the same strategy for selecting the next critical pair was used. That is, the only difference was the way the permutation-equations were dealt with.

Proof details:

reduction ordering: LPO (preorder given for each problem; lexicographic evaluation from left to right);
selection of next critical pair: select pair with minimal weight, where the weight for a critical pair $<u,v> = \Phi(u)+\Phi(v)$ with $\Phi(x)=1$, if x is a variable, else $\Phi(g(t_1,..,t_n))= 2+\Phi(t_1)+..+\Phi(t_n)$.
The *goal* is already negated and skolemized, and the skolem constants are also listed in the preorder of the LPO.

problem descriptions:
(The hirsh examples were taken from
Workshop on automated reasoning
Argonne laboratory, August 1989)

hirsh9.4

axioms: $o(x,y) = o(y,x)$
$o(o(x,y),z) = o(x,o(y,z))$
$n(o(n(o(x,y)),n(o(x,n(y))))) = x$
$a = n(o(n(b),c))$

goal: $a \neq n(o(n(o(o(b,a),c)),c))$

preorder: $n > o > a > b > c$

hirsh9.5

axioms: $o(x,y) = o(y,x)$
$o(o(x,y),z) = o(x,o(y,z))$
$n(o(n(o(x,y)),n(o(x,n(y))))) = x$
$a = n(o(b,c))$

goal: $a \neq n(o(n(o(o(n(b),a),c)),c))$

preorder: $n > o > a > b > c$

hirsh12

axioms: $o(x,y) = o(y,x)$
$o(o(x,y),z) = o(x,o(y,z))$
$n(o(n(o(x,y)),n(o(x,n(y))))) = x$
$n(o(a,n(b))) = n(a)$
$o(a,b) = b$

$n(g(x)) = x$

goal: $n(b) \neq n(o(a,n(a)))$

preorder: $n > o > g > a > b$

hirsh5

axioms: $o(x,y) = o(y,x)$

$o(o(x,y),z) = o(x,o(y,z))$

$n(o(n(o(x,y)),n(o(x,n(y))))) = x$

$n(n(x)) = x$

goal: $a \neq o(n(o(n(a),n(b))),n(o(n(a),b)))$

preorder: $n > o > a > b$

hirsh8a

axioms: $o(x,y) = o(y,x)$

$o(o(x,y),z) = o(x,o(y,z))$

$n(o(n(o(x,y)),n(o(x,n(y))))) = x$

$n(g(x)) = x$

$o(a,n(a)) = a$

goal: $n(a) \neq o(n(a),n(a))$

preorder: $n > o > g > a$

hirsh6

axioms: $o(x,y) = o(y,x)$

$o(o(x,y),z) = o(x,o(y,z))$

$n(o(n(o(x,y)),n(o(x,n(y))))) = x$

$o(c,d) = c$

$o(n(c),d) = n(c)$

$n(g(x)) = x$

goal: $n(n(a)) \neq a$

preorder: $n > o > g > a > c > d$

demo1

axioms: $f(a,f(b,c)) = b$

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

$f(x,y) = f(y,x)$

$f(b,f(c,x)) = f(a,f(b,x))$

goal: $f(a,f(cx,f(c,f(cy,b)))) \neq f(cx,f(cy,b))$

preorder: $f > c > cy > b > cx > a$

demo2

axioms: $f(a,f(b,c)) = b$

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

$f(x,y) = f(y,x)$

$f(b,f(c,x)) = f(a,f(b,x))$

goal: $f(a,f(cx,f(c,f(cy,b)))) \neq f(cx,f(cy,b))$

preorder: $f > b > cy > c > cx > a$

demo4

axioms: $f(h(x),f(r(x),x)) = h2(x)$

f(f(x,y),z) = f(x,f(y,z))
f(x,y) = f(y,x)

goal: f(cx,h2(f(cu,cv))) ≠ f(cu,f(cx,f(r(f(cu,cv)),f(cv,h(f(cu,cv))))))

preorder: f > r > h > h2 > cv > cu > cx

nonassring4a (non-associative ring theory: linearity of associator in first argument)

axioms: f(x,0) = x
f(x,f(y,z)) = f(f(x,y),z)
f(x,g(x)) = 0
f(x,y) = f(y,x)
h(x,h(y,y)) = h(h(x,y),y)
h(h(x,x),y) = h(x,h(x,y))
h(x,f(y,z)) = f(h(x,y),h(x,z))
h(f(x,y),z) = f(h(x,z),h(y,z))
a(x,y,z) = f(h(h(x,y),z),g(h(x,h(y,z))))

goal: a(f(d1,d2),d3,d4) ≠ f(a(d1,d3,d4),a(d2,d3,d4))

preorder: a > h > g > f > 0 > d1 > d2 > d3 > d4

The figures listed in the table on the following page denote
(1) number of rules generated
(2) number of equations generated
(3) number of critical pairs generated
(4) number of reductions performed
(5) run time (in seconds)
For (3) and (4), in the case where pairs of permutations are used, the number in parentheses indicates the number of critical pairs resp. reductions on account of permutations (performed by applying pairs of permutations).
The column “name” refers to the problems just described above.
The column “speed-up” displays the speed-up factor of our method.
The entry “> 10 min.” indicates that the proof was aborted (by the user) after 10 minutes.

Table 1:

name	(1)	(2)	(3)	(4)	(5)	*speed-up*
hirsh9.4	11	14	2167	2075	40.8 sec	
	12	0	(13) 173	(143) 218	0.45 sec	ca. 90
hirsh9.5	11	15	2456	2380	50.1 sec	
	12	0	(13) 168	(131) 206	0.45 sec	ca. 111
hirsh12	9	4	154	159	0.7 sec	
	10	0	(14) 32	(14) 43	0.11 sec	ca. 6
hirsh5	9	4	263	383	1.34 sec	
	9	0	(15) 86	(72) 246	0.23 sec	5.8
hirsh8a	-	-	-	-	> 10 min.	
	161	0	(798) 11931	(10673) 26577	106.6 sec	> 5.6
hirsh6	-	-	-	-	> 10 min.	
	127	2	(606) 9573	(9005) 16499	74.1 sec	> 8
demo1	16	23	4801	7359	142.6 sec	
	13	2	(142) 173	(122) 435	0.95 sec	ca. 150
demo2	18	33	7279	12832	229.9 sec	
	19	6	(279) 352	(199) 815	2.2 sec	104.5
demo4	7	15	2362	2823	51.3 sec	
	18	1	(337) 522	(331) 919	4.3 sec	ca. 12
nonassring4a	48	19	5217	10136	163.6 sec	
	49	5	(620) 1705	(471) 4613	22.9 sec	ca. 7

References

[BDP89] Bachmair, L.; Dershowitz, N.; Plaisted D.A.:
"Completion without failure"
Coll. on the resolution of equations in algebraic structures,
Austin, Texas 1987
Academic Press, 1989

[KB70] Knuth, D.E.; Bendix, P.B.:
"Simple word problems in universal algebra"
Computational algebra, J. Leach, Pergamon Press, 1970, pp 263-297

[PS81] Peterson, G.E.; Stickel, M.E.:
"Complete sets of reductions for some equational theories"
Journal of the association for computing machinery
Vol. 28, No 2, pp 233-264, April 1981

[St81] Stickel, M.E.:
"A unification algorithm for associative-commutative functions"
Journal of the association for computing machinery
Vol. 28, No 3, pp 423-434, July 1981