Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT

Efficient AC1-Matching using Constraints

J.Avenhaus, J.Denzinger, T.Hoffmann

SEKI Report SR-92-03 (SFB)

Efficient AC1-Matching using Constraints

J.Avenhaus, J.Denzinger, T.Hoffmann

SEKI Report SR-92-03 (SFB)

Efficient AC1-Matching using Constraints

J. Avenhaus, J. Denzinger, T. Hoffmann
FB Informatik
Universität Kaiserslautern
Postfach 3049
6750 Kaiserslautern
E-mail : {avenhaus, denzinge}@informatik.uni-kl.de

Abstract :

We present an algorithm for the matching problem modulo the theory AC1 of associativity, commutativity and a unit element. Straight forward matching algorithms create a large number of branching points for backtracking. To reduce the enormous search space we propose to postpone the solution of subproblems and instead to approximate the solutions by constraints. This allows one to have a global view on the whole problem and so to reduce the large number of branching points. We demonstrate the algorithm by some examples and give hints how the general idea - contraint propagation - can be used for other theory matching algorithms and how term constraints from calling functions of our algorithm can be incorporated.

1. Introduction

As a basic operation used for simplification in deduction systems, efficient matching is very important for a good performance of these systems. In the last years the tendency to put more and more knowledge into the basic inference rules used by deduction systems resulted in the need for efficient theory matching and theory unification algorithms. Although for most of the interesting theories it is possible to simulate theory matching by theory unification, this solution of the matching problem is in most cases not very efficient.

In this paper we present an algorithm for AC1-matching, that is matching with function symbols that are commutative, associative and have a neutral element, which uses constraint propagation to limit unnecessary branching. We extend the technique developed for AC-matching in [DG88]. Since AC1 is a collapsing theory we have to solve new problems. On the other hand it turned out that the idea of using constraints is more powerful for the more complex problem of AC1-matching than for AC-matching. We outline some ideas of how to use this technique for matching modulo other theories.

The paper is structured as follows. After this section we give the

definitions needed throughout the paper and a description of the theoretical and implementational background of theory, especially AC1-matching. In section 3 we present an inference system for AC1-matching and sketch the proof of its correctness and completeness. In section 4 we describe, how we handle and process our constraints and how the inference system can be transformed into an algorithm. In the same section we also demonstrate the efficiency of this algorithm by some examples. In section 5 we outline how to apply our method for matching modulo other theories.

2. Notations and background

2.1 Notations

Let F be a finite set of function symbols of fixed arity and V be a denumerable set of variables disjoint from F. The set of terms over F and V is denoted by T(F,V). By V(t) we denote the set of all variables occuring in a term t. Top(t) is the top-level symbol of t. A substitution σ is a mapping from V to T(F,V) with finite domain Dom(σ). It is extended to a mapping from T(F,V) to T(F,V) in the usual way.
Given F and V, the equational theory $=_E$ induced by a set E of equations over T(F,V) is defined to be the smallest congruence on T(F,V) containing E and closed under substitution. Two terms t_1 and t_2 are said to be E-equivalent iff $t_1 =_E t_2$ holds. A substitution σ is an E-match from the pattern t to the example s iff $\sigma(t) =_E s$.
In the following we always assume F to be partitioned into disjoint sets F_{AC1}, F_1 and F_{NAC1}, where F_{AC1} is a non-empty set of binary operators and F_1 is the set of units for F_{AC1}, i.e. $F_1 = \{1_f \mid f \in F_{AC1}\}$. The set E of equations consists of the AC1-axioms for the operators of F_{AC1}, i.e.

$$E := AC1 := \{f(x,y) = f(y,x)\ ,\ f(f(x,y),z) = f(x,f(y,z))\ ,\ f(x,1_f) = x \mid f \in F_{AC1}\}.$$

We use the following notations. With $f,g,f_1,\ldots$ we will denote elements of F_{AC1}, $1_f,1_g,1_{f1},\ldots$ are the corresponding members of F_1 and $h,k,h_1,\ldots$ are elements of F_{NAC1}. For variables we use $x,y,z,x_1,\ldots$ and elements of T(F,V) are denoted by $t,t_1,t_2,\ldots$ Finally $s,s_1,\ldots$ denote ground terms, i.e. terms s with $V(s) = \emptyset$.

When using theory matching as an operation in deduction systems pattern and example term have no variables in common. Therefore we will assume in the following, that all example terms are ground terms.

2.2 Related results

The first work on theory matching was done by Hullot [[Hu79]] for the theory AC. A similar framework was later used by Mzali in his works on ACI-, AC-, A-, C-matching [[Mz85]] and D-matching [[Mz86]]. Nipkow used in [Ni89] the concept of variable abstraction, introduced by Stickel in [St81] for unification, for combining matching algorithms

and dealing with collapsing theories. Kapur and Narendran [[KN90]] have shown that AC1-matching is NP-complete. Despite of this our algorithm is fast on most inputs of practical interest.
The framework of Mzali consists of four steps to perform in order to solve a given matching problem, some of them several times :

- computation of a normalform for an efficient decision of $=_E$
- decomposition of problems t = s where top[t] is free
- merging of found substitutions into the remaining problems and
- mutation of problems t = s with top[t] is a theory symbol.

Because the first three steps are also important to our algorithm and the forth step is the one which introduces most of the problems concerning efficiency we will discuss them here more precisely.

Normalform for deciding $=_E$

A theory matching algorithm always must be able to determine wheater two terms are equal with respect to E. A standard way to do this is to generate a unique normalform for all E-equal terms and then to test the normalforms for syntactic identity. In [Mz85], there is a classification of parts of some theories and their requirements for the normalform. For example, if the associativity is part of the theory then the terms have to be flattened. The commutativity allows one to order the arguments. Idempotency in addition leads to a set representation.

For AC1-matching, this leads to ordered multisets as a data structure for the arguments of symbols of F_{AC1}. The multisets are obtained by flattening and sorting the arguments and by eleminating all occurences of the corresponding 1-element of a symbol in the set. So terms with top-level function in F_{AC1} have the form $f[t_1^{a1},...,t_n^{an}]$, where the a_i indicate the number of occurences of t_i. Further, as in the case of AC-matching [[DG88]], the ordering can enhence the efficiency of the algorithm by forcing easy parts of the problem to be solved first. We use an ordering on the arguments where the smallest components are the constants, then terms with top-level functions that are not AC1, then the terms with top-level functions that are AC1 and finally the variables. Within each class, the symbols are ordered lexicographically. Note that $f[s^0]$ denotes f[s] and f[s] denotes s.

Examples

$f[a,f[g[a,b],f[1_f,a]]]$ has the normalform $f[a^2,g[a,b]]$.

$f[x,g[f[x,a],g[1_f,g[a,f[a,x]]]]]$ has the normalform $f[g[a,1_f,f[a,x]^2],x]$.

$f[f[h[a,b],f[a,x]],f[f[g[x,b],h[b,a]],f[x,y]]]$ has the normalform $f[a,h[a,b],h[b,a],g[b,x],x^2,y]$.

Decomposition

Decomposition steps handle E-matching problems of the form $h[t_1,...,t_n] = h[s_1,...,s_n]$, i.e. problems where the top-level function symbols are

free. Such problems are decomposed into n smaller problems $t_i = s_i$, $i = 1,....n$. If h_1 and h_2 are different free function symbols then the problem $h_1[t_1,....t_n] = h_2[s_1,....,s_m]$ is unsolvable.

Merging

Normally a variable occurs in pattern terms several times. Thus, when the steps decomposition and mutation generate a subproblem $x = s$, the other problems can also have occurences of x in their patterns. By merging these occurences will be replaced by s. In this way, found solutions are propagated within the set of subproblems. Hopefully, the resulting problems then are easier to solve or the unsolvability of the problem is detectable.

Mutation

Besides the computation of normalforms the mutation steps are the only steps which deal with the underlying theory. In fact, mutation reflects the properties of the theory. Mutation steps are used, whenever the matching problem has the form $f[t_1,....,t_n] = s$, with f not free. For most theories there are many solutions of this atomic problem. All these solutions have to be examined and this results in the need for backtracking. Mutation steps for a theory are usually defined for the so-called variable-only case, i.e. all arguments t_i are variables x_i. For example, in the theory AC1 there are two different mutation steps. The first one is similar to the mutation for the theory AC. If the problem has the form

$$f[x_1^{a1},....,x_n^{an}] = f[s_1^{b1},....,s_m^{bm}]$$

then mutation has to generate the possible subproblems

$$x_1 = f[s_1^{c1},....,s_m^{cm}]$$
$$f[x_2^{a2},....,x_n^{an}] = f[s_1^{b1-c1*a1},....,s_m^{bm-cm*a1}]$$

where the c_i range from 0 to b_i div a_1.

Example :

$f[x^2,y^3,z] = f[a^2,b,c,d^5]$ generates the following 6 possible branches for x :

$x = 1_f$	$x = a$	$x = d$
$f[y^3,z] = f[a^2,b,c,d^5]$	$f[y^3,z] = f[b,c,d^5]$	$f[y^3,z] = f[a^2,b,c,d^3]$
$x = f[a,d]$	$x = f[d^2]$	$x = f[a,d^2]$
$f[y^3,z] = f[b,c,d^3]$	$f[y^3,z] = f[a^2,b,c,d]$	$f[y^3,z] = f[b,c,d]$

This effect may lead to a combinatorical explosion.

The second mutation step reflects the fact that AC1 is a so-called collapsing theory. This means, that certain partial solutions may lead to a change of the top-level symbol of a term.

Example :

Let $t \equiv f[h[y,a],x,y^2]$ and $\mu = \{x \leftarrow 1_f\ ,\ y \leftarrow 1_f\}$, then $\mu[t] =_{AC1} h[1_f,a]$.

Let $t \equiv f[g(a,y),x]$, $\mu_1 = \{x \leftarrow 1_f\}$ and $\mu_2 = \{x \leftarrow 1_f\ ,\ y \leftarrow 1_g\}$, then $\mu_1(t) =_{AC1} g(a,y)$ and $\mu_2(t) =_{AC1} a$.

To deal with this problem, we need the following step. We start with the problem

$$f(x_1^{a1},\ldots,x_n^{an}) = s\ ,\quad \text{where } top[s] \neq f.$$

Then we get the following n possible solutions :

$$x_1^{a1} = s \qquad\qquad x_n^{an} = s$$
$$x_2 = \ldots = x_n = 1_f \quad \ldots \quad x_1 = \ldots = x_{n-1} = 1_f$$

Note, that all cases $x_i^{ai} = s$, where $a_i \neq 1$, are unsolvable.

Example :

For $f(x^2,y,z) = h(a,b)$ we get the following 2 branches :

$$y = h(a,b) \qquad\qquad z = h(a,b)$$
$$x = z = 1_f \qquad\qquad x = y = 1_f$$

In the non-variable case we either have the atomic problem $s_1 = s_2$ - which is solvable iff $s_1 \equiv s_2$ - or we have to apply variable abstraction to reduce the problem to the variable case.

Mutation generates with every branch a local solution of a problem. Merging propagates this solution to the other problems generated earlier. An important source of inefficiency is the fact, that many of the branches are detected to be unsolvable only very late. This is due to the locality of this method. Human beings would detect the unsolvability often very early because they have a global view on all subproblems. Therefore we will use constraint propagation to deal with this problems.

There are many extensions to mutation, for example in [Mz85] or [Ni89], that help to avoid some of the branching in the non-variable case. For example, the ground subterms in the pattern have also to appear in the example term, if the problem is solvable. Therefore the same ground term in the pattern and the example can be deleted. This provides fewer possibilities for branching. Our inference system $\mathcal{D}_{AC1}$ will also include these extensions before the problems are used to generate constraints.

3. An inference system for AC1-matching with constraints

The idea to use constraints for enhancing the efficiency of a matching algorithm is to postpone the solution of an atomic problem $f(x_1,\ldots,x_n) = s$ and instead to create constraints for the solutions. The constraints should be sharp and easy to propagate. Of course, not every solution of the constraints has to be a solution of the matching subproblems. But in many cases one may early detect that the constraints are unsolvable and so the matching problems are unsolvable also. This results in a reduction of the search space and so saves computing time. Of course,

the form of the constraints will depend on the underlying theory E.

Our inference system works on a triple of sets (E,C,Sol), where E is a set of AC1-matching problems to solve, Sol is a substitution that describes the partial solution so far and $C = C_V \cup C_E$ consists of a set of constraints for variables C_V, each of the form $C_x = x \in S$ or $C_x = x \in [f(s_1^{a1},...,s_n^{an})]$, and a set of verification equations C_E. S is a set of ground terms and by $[f(s_1^{a1},...,s_n^{an})]$ we denote the set of flattened terms

$$\{f(s_1^{c1},...s_n^{cn}) \mid f \in F_{AC1} \text{ and } 0 \le c_i \le a_i \text{ for } i = 1,...,n\}.$$

We call $[f(s_1^{a1},...,s_n^{an})]$ an intervall. C_E is used to check whether solutions of C_V are also solutions of the matching problem.

In the following inference system $\mathcal{D}_{AC1}$ the ideas of section 2.2 for solving the AC1-matching problem are made precise. The rules "Trivial equation", "Decomposition" and "Explicit solution" are our notion of the steps decomposition and merging of 2.2 . "Simplification" gets rid of ground terms as arguments of AC1-functions. "Variable abstraction" is a well know method to deal with terms that do not belong to the theory, in our case AC1. Variable abstraction was introduced by Stickel in [St81] for AC-unification and used by Nipkow in [Ni89] as central idea to combine matching algorithms for different theories. The rule "Variable arguments" generates new constraints.

The inference system $\mathcal{D}_{AC1}$

Trivial equation

$$\text{a)}\ \frac{\{s=s\} \cup E\ ,\ C\ ,\ Sol}{E\ ,\ C\ ,\ Sol} \qquad \text{b)}\ \frac{\{s=s'\} \cup E\ ,\ C\ ,\ Sol}{Fail}\ \text{if } s \neq s'$$

Decomposition

$$\text{a)}\ \frac{\{k(t_1,...,t_n) = k(s_1,...,s_n)\} \cup E\ ,\ C\ ,\ Sol}{\{t_i = s_i \mid i = 1,...,n\} \cup E\ ,\ C\ ,\ Sol}$$

$$\text{b)}\ \frac{\{k(t_1,...,t_n) = s\} \cup E\ ,\ C\ ,\ Sol}{Fail}\ \text{if } top[s] \neq k$$

Explicit solution (Merging)

$$\text{a)}\ \frac{\{x=s\} \cup E\ ,\ C\ ,\ Sol}{\sigma[E]\ ,\ \sigma[C - C_x]\ ,\ Sol \cup \sigma}\ \text{if } \sigma = \{x \leftarrow s\} \text{ and } \sigma \text{ satisfies } C$$

b] $\{x=s\} \cup E \ , \ C \ , \ Sol$

$$\frac{}{\text{Fail}} \quad \text{if } \sigma = \{x \leftarrow s\} \text{ and } \sigma \text{ does not satisfy } C$$

Simplification

a] $$\frac{\{f[s^a, t_1^{a1}, \ldots t_n^{an}] = f[s_1^{b1}, \ldots, s^b, \ldots, s_m^{bm}]\} \cup E \ , \ C \ , \ Sol}{\{f[t_1^{a1}, \ldots, t_n^{an}] = f[s_1^{b1}, \ldots, s^{b-a}, \ldots, s_m^{bm}]\} \cup E \ , \ C \ , \ Sol} \quad \text{if } 1 \leq a \leq b$$

b] $$\frac{\{f[s^a, t_1^{a1}, \ldots t_n^{an}] = f[s_1^{b1}, \ldots, s^b, \ldots, s_m^{bm}]\} \cup E \ , \ C \ , \ Sol}{\text{Fail}} \quad \text{if } a > b$$

c] $$\frac{\{f[s, t_1^{a1}, \ldots, t_n^{an}] = s\} \cup E \ , \ C \ , \ Sol}{\{f[t_1^{a1}, \ldots, t_n^{an}] = 1_f\} \cup E \ , \ C \ , \ Sol}$$

d] $$\frac{\{f[s^a, t_1^{a1}, \ldots, t_n^{an}] = s\} \cup E \ , \ C \ , \ Sol}{\text{Fail}} \quad \text{if } a > 1$$

e] $$\frac{\{f[s^a, t_1^{a1}, \ldots, t_n^{an}] = s'\} \cup E \ , \ C \ , \ Sol}{\text{Fail}} \quad \text{if } s \neq s' \text{ and } top[s'] \neq f$$

Variable abstraction of alien subterms

$$\frac{\{f[t_1^{b1}, \ldots, t_m^{bm}, x_1, \ldots, x_n] = s\} \cup E \ , \ C \ , \ Sol}{\{f[x_1, \ldots, x_n, u_1^{b1}, \ldots, u_m^{bm}] = s\} \cup E \ , \ \{t_j = u_j \mid j = 1, \ldots, m\} \cup C \ , \ Sol}$$

if t_j not in V and not ground
where $u_j \in V_{aux}$ new, i.e. $V_{aux} \cap V = \emptyset$

Variable arguments

a] $$\frac{\{f[x_1, \ldots, x_n, y_1^{a1}, \ldots, y_m^{am}] = s\} \cup E \ , \ C \ , \ Sol}{\sigma[E] \ , \ \sigma[C'] \ , \ Sol \cup \sigma} \quad \text{if } top[s] \neq f \text{ and } a_j > 1$$

where $\sigma = \{y_j \leftarrow 1_f \mid j = 1, \ldots, m\}$
$C'_V = C_V \cup \{x_i \in \{1_f \ , \ s\} \mid i = 1, \ldots, n\}$
$C'_E = C_E \cup \{f[x_1, \ldots, x_n] = s\}$

b] $$\frac{\{f[x_1^{a1},\ldots,x_n^{an}] = f[s_1^{b1},\ldots,s_m^{bm}]\} \cup E \ , \ C \ , \ Sol}{E \ , \ C \cup \{x_i \in [f[s_1^{c1i},\ldots,s_m^{cmi}]] \mid i = 1,\ldots,n\} \cup \{f[x_1^{a1},\ldots,x_n^{an}] = f[s_1^{b1},\ldots,s_m^{bm}]\} \ , \ Sol}$$

where $c_{ij} = b_j \text{ div } a_i$

The following rules deal with the solution of easy constraints and the detection of unsolvable constraints. Note, that in same cases the solution of a constraint generates new AC1-matching problems.

Pruning constraints

a] $$\frac{E \ , \ C \ , \ Sol}{\sigma[E] \ , \ \sigma[C - C_x] \ , \ Sol \cup \sigma} \quad \text{if } C_x = x \in \{s\} \text{ and } \sigma = \{x \leftarrow s\}$$

b] $$\frac{E \ , \ C \cup \{f[x^a] = f[s_1^{b1},\ldots,s_n^{bn}]\} \ , \ Sol}{\text{Fail}} \quad \text{if } \exists i : \neg a \mid b_i$$

c] $$\frac{E \ , \ C \cup \{f[x^a] = f[s_1^{b1},\ldots,s_n^{bn}]\} \ , \ Sol}{\sigma[E] \ , \ \sigma[C] \ , \ Sol \cup \sigma} \quad \text{if } a \mid b_i \text{ for all } i = 1,\ldots,n$$

where $c_i = b_i \text{ div } a$

$\sigma = \{x \leftarrow f[s_1^{c1},\ldots,s_n^{cn}]\}$

d] $$\frac{E \ , \ \{t=u \ , \ u=s\} \cup C \ , \ Sol}{\{t=s\} \cup \sigma[E] \ , \ \sigma[C] \ , \ Sol} \quad \text{if } u \in V_{aux}$$

where $\sigma = \{u \leftarrow s\}$

e] $$\frac{E \ , \ C \ , \ Sol}{\text{Fail}} \quad \text{if C unsatisfyable}$$

We are going to show that the inference system $\mathcal{D}_{AC1}$ is correct.

Lemma

If $E \neq \emptyset$ then one rule of the inference system $\mathcal{D}_{AC1}$ is applicable to [E,C,Sol].

Proof :

By induction on the structure of possible flattened pattern terms. ■

Theorem

Let E be an AC1-matching problem :

a] If every derivation in $\mathcal{D}_{AC1}$ from $[E,\emptyset,\emptyset]$ ends with Fail, then E is unsolvable.

b] For every solution σ of E, there is a derivation in $\mathcal{D}_{AC1}$ from $[E,\emptyset,\emptyset]$ to $[\emptyset,C,Sol]$ such that $\sigma = \sigma' \cup Sol$ and σ' satisfies C.

Proof :

If we can show the following lemma, then an induction on the number of applications of inference rules of $\mathcal{D}_{AC1}$ proves the theorem.

Lemma

Let $[E_1,C_1,Sol_1]$ result from $[E_0,C_0,Sol_0]$ be application of a rule of $\mathcal{D}_{AC1}$. Then $\sigma|_V$ is part of a solution of $(E_1,C_1,Sol_1]$ iff $\sigma|_V$ is part of a solution of $[E_0,C_0,Sol_0]$.

Here $\sigma|_V$ denotes the restriction of σ to the variables in V.

The proof of the lemma for most of the inference rules is easy. We will show here the proofs for the rules Variable abstraction and Variable arguments.

Let μ be a solution of $[E_1,C_1,Sol_1]$ and τ a solution of $[E_0,C_0,Sol_0]$.

1] Variable abstraction

Then $E_0 = E \cup \{f[t_1^{b1},...,t_m^{bm},x_1^{a1},...,x_n^{an}] = s\}$, $C_1 = C_0 \cup \{t_j = u_j \mid j = 1,...,n\}$, $Sol_1 = Sol_0$ and $E_1 = E \cup \{f[x_1^{a1},...,x_n^{an},u_1^{b1},...,u_m^{bm}] = s\}$.

Let $\tau' = \tau \circ \{u_j \leftarrow t_j \mid j = 1,...,n\}$. Then $\tau'|_V = \tau|_V$.

$$\begin{aligned}\tau'[f[x_1^{a1},...,x_n^{an},u_1^{b1},...,u_m^{bm}]] &= \tau\circ\{u_j \leftarrow t_j \mid j = 1,...,n\}[f[x_1^{a1},...,x_n^{an},u_1^{b1},...,u_m^{bm}]]\\ &=_{AC1} \tau[f[t_1^{b1},...,t_m^{bm},x_1^{a1},...,x_n^{an}]]\\ &=_{AC1} s.\end{aligned}$$

So $\tau|_V$ is part of a solution of $[E_1,C_1,Sol_1]$.

It is obvious, that μ is a solution of $[E_0,C_0,Sol_0]$, because

$$\begin{aligned}\mu[f[x_1^{a1},...,x_n^{an},u_1^{b1},...,u_m^{bm}]] &= f[\mu[x_1^{a1}],...,\mu[x_n^{an}],\mu[u_1^{b1}],...,\mu[u_m^{bm}]]\\ &=_{AC1} f[\mu[t_1^{b1}],...,\mu[t_m^{bm}],\mu[x_1^{a1}],...,\mu[x_n^{an}]]\\ &=_{AC1} \mu[f[t_1^{b1},...,t_m^{bm},x_1^{a1},...,x_n^{an}]]\\ &=_{AC1} s.\end{aligned}$$

2] Variable arguments

a] μ is a solution for $[E_1,C_1,Sol_1]$, therefore $\mu[y_j] = 1_f$.

$$\begin{aligned}\mu[f[x_1,...,x_n,y_1^{a1},...,y_m^{am}] &=_{AC1} f[\mu[x_1],...,\mu[x_n],1_f,...,1_f]\\ &=_{AC1} \mu[f[x_1,...,x_n]]\\ &=_{AC1} s, \text{ because } f[x_1,...,x_n] = s \in C'_E.\end{aligned}$$

So μ is a solution of $[E_0,C_0,Sol_0]$.

$\tau[y_j] = 1_f$ for all $j = 1,...,m$, else let $\tau[y_k] = s'$. Then $top[\tau[f[x_1,...,x_n,y_1^{a1},...,y_m^{am}]]] = f$, which contradictes $top[s] \neq f$ and τ solution of $[E_0,C_0,Sol_0]$.

Therefore is σ part of τ, $\tau = \tau'\circ\sigma$. $\tau[\sigma[E]] = \tau'[\sigma[\sigma[E]]] = \tau'[\sigma[E]] = \tau[E]$.

$\tau[Sol \cup \sigma] = \tau[Sol]$ and it is obvious that τ is a solution of C'_V and C'_E. So τ is also a solution of $[E_1, C_1, Sol_1]$.

b] Because $f[x_1^{a1},...,x_n^{an}] = f[s_1^{b1},...,s_m^{bm}] \in C_1$ μ is solution to $[E_O, C_O, Sol_O]$. Because $f[x_1^{a1},...,x_n^{an}] = f[s_1^{b1},...,s_m^{bm}] \in E_O$ and $\tau[x_i] \in [f[s_1^{c1i},...,s_m^{cmi}]]$ due to the definition of intervall τ is solution to $[E_1, C_1, Sol_1]$. ∎

4. Processing of constraints and implementational issues

4.1 Processing of constraints

So far, we have presented an inference system that transforms AC1-matching problems into a set of constraints by decomposition and solving of easy subproblems. Further the system explains how very special constraints, i.e. of the form $x \in \{s\}$, can be solved or very special constraints can be detected as unsolvable. Now the questions remain, how to combine several constraints for a variable and how to sharpen constraints, if new substitutions are found. The answers to these questions will also show that constraints can help to solve AC1-matching problems more efficiently.

Our constraints represent possible substitutions for a variable in a compact form. All possible substitutions form a set. If we have two constraints C_x and C'_x for a variable x, then we have to compute the intersection $C''_x = C_x \cap C'_x$. The intersection of two intervalls may not be an intervall. Then we switch to an explicit set representation. Note that for intervall variable constraints, i.e. constraints for a variable in intervall form, the argument lists can be extended, if necessary. So $[f[s_1^{a1},...,s_n^{an}]]$ is the same as $[f[s^O, s_1^{a1},...,s_n^{an}]]$. Thus we have the following possible intersection situations :

- two variable constraints in set representation :
  normal intersection $\{s_1,...,s_n\} \cap \{s'_1,...,s'_m\}$.

- variable constraint in set representation with intervall constraint :
  $\{s'_1,...,s'_m\} \cap [f[s_1^{a1},...,s_n^{an}]] =$
  $\{s'' \mid s'' \in \{s'_1,...,s'_m\}$ and $[s'' \in \{s_1,...,s_n\}$ or $s'' = f[s_1^{c1},...,s_n^{cn}]$ with $0 \le c_i \le a_i]\}$.

- two intervall constraints with equal top-level symbol :
  $[f[s_1^{a1},...,s_n^{an}]] \cap [f[s_1^{b1},...,s_n^{bn}]] = [f[s_1^{c1},...,s_n^{cn}]]$
  where $c_i = \min(a_i, b_i)$ for all i.

- two intervall constraints with different top-level symbol :
  $[f[s_1^{a1},...,s_n^{an}]] \cap [g[s_1'^{b1},...,s_m'^{bm}]] =$
  $[\{s_1,...,s_n\} \cap [g[s_1'^{b1},...,s_m'^{bm}]]] \cup [\{s'_1,...,s'_m\} \cap [f[s_1^{a1},...,s_n^{an}]]]$.

So we have to convert the intervall representation to the set representation only when constraints with different top-level symbols

have to be combined or if the representation is $[f(s_1^0,...,s_n^0)]$, which means that the only solution of this constraint is 1_f. Note, that an implementation of this intersection rules can be efficiently done.

Examples :

1] $x \in \{a, b, c, 1_f, f(a,b)\}$ and $x \in [f(a^3,b^2,d)]$.
Combination leads to
$x \in \{a, b, 1_f, f(a,b)\}$.

2] $x \in [f(a^6,b^2,c^8,d^3,e^4)]$ and $x \in [f(a^2,b^4,d^6,e^2)]$.
Combination leads to
$x \in [f(a^2,b^2,d^3,e^2)]$.

3] $x \in [f(a^2,b^4,d^6)]$ and $x \in [g(1_f^2,a,b^4,f(a^2,b^3),f(b^4,e^2))]$.
Combination leads to
$x \in \{a, b\} \cup \{1_f, a, b, f(a^2,b^3)\} = \{1_f, a, b, f(a^2,b^3)\}$.

The inference rule Variable abstraction allows for a second way to restrict variable constraints. If we have a constraint of the form

$$f(t_1^{a1},...,t_n^{an},x_1^{an+1},...,x_k^{an+k}) = u,$$

where $f \in F_{AC1}$, $u \in V_{aux}$ and t_i not in V, then we can compute "at-least" requirements for the variable u. A substitution that solves this equation must substitute for u at least a term of the form $f(t_{i1}^{ai1},...,t_{il}^{ail})$, where $i_j \in \{1,...,n\}$ and $top(t_{ij})$ not in F_{AC1}. If there is an intervall variable constraint $u \in [f(s_1^{b1},...,s_m^{bm})]$, we get a combined constraint

$$u \in [f(s_1^{[c1,b1]},...,s_m^{[cm,bm]})],$$

with $c_i = 0$, if s_i not in $\{t_{i1},...,t_{il}\}$, or a_{ij} else. If $c_i > b_i$, then the constraints are not solvable.

If solutions for variables in $\{x_1,...,x_k\}$ or variables in $V(t_i)$ are found, the "at-least" constraints c_i have to be adjusted.

Example :

$f(a,h(a,b),h(c,x),g(a^2),x,y) = u$ and
$u \in [f(a^2,c,h(a,b),h(c,d)^2,g(a^2)^2,g(a,b))]$
leads to
$u \in [f(a^{[1,2]},c^{[0,1]},h(a,b)^{[1,1]},h(c,d)^{[0,2]},g(a^2)^{[1,2]},g(a,b)^{[0,1]})]$.

So $\mu_1 = \{u \leftarrow f(a^2,h(a,b),g(a^2),g(a,b)\}$ is a solution for u, but
$\mu_2 = \{u \leftarrow f(a,c,h(c,d),g(a^2))\}$ is not, because of the constraint on h(a,b).

The intersection of "at-least" constraints is like before.

Examples :

1] $x \in \{a, b, c, 1_f, f(a,b)\}$ and $x \in [f(a^{[1,3]},b^{[0,2]},c^{[0,1]},d^{[0,1]})]$.
Combination leads to
$x \in \{a, f(a,b)\}$.

2] $x \in [f(a^{[1,6]},b^{[2,2]},c^{[0,8]},d^{[1,3]},e^{[0,4]})]$ and
$x \in [f(a^{[0,2]},b^{[0,4]},d^{[0,6]},e^{[0,2]})]$.
Combination leads to

$x \in [f[a^{[1,2]}, b^{[2,2]}, d^{[1,3]}, e^{[0,2]}]]$.

3] $x \in [f[a^{[0,2]}, b^{[0,4]}, d^{[0,6]}]]$ and
$x \in [g[1_f^{[0,2]}, a^{[0,1]}, b^{[0,4]}, f[a^2, b^3]^{[0,1]}, f[b^4, e^2]^{[0,1]}]]$.
Combination leads to
$x \in \{a, b\} \cup \{1_f, a, b, f[a^2, b^3]\} = \{1_f, a, b, f[a^2, b^3]\}$.

If an AC1-matching problem has been transformed by $\mathcal{D}_{AC1}$ into $[\emptyset, C, Sol]$ and $C \neq \emptyset$ then we have to solve the constraints. As stated before, not every substitution of $x \in S$ or $x \in [f[s_1^{a1}, \ldots, s_n^{an}]]$ is a solution of the matching problem. So a variable with the smallest number of possible substitutions is chosen and one of the solutions is propagated in C_E. If this leads to no Fail, the next "smallest" variable is chosen and so on. If a Fail occurs, using backtracking another solution for the variable is tried.

4.2 Implementation

Our implementation combines the inference system $\mathcal{D}_{AC1}$ with the constraint handling of 4.1, with some small changes. The rule "Variable abstraction" that postpones branching into the solution of the constraints is not used in this manner. Instead only such arguments t with $top[t] \in F_{AC1}$ are treated as variables. the possible subproblems generated by other arguments are tried at once. We do not want to process all branches, because in most systems only one and not all matches of a pattern to an example are needed. Therefore we select one possibility and do backtracking if this branch is not sucessful. As we also want to generate as fast as possible partial soulutions or constraints, we do a depth-first search. This requires a control that uses three stacks :

- a to-do stack, where all remaining problems of the branch are stored. Always the first problem of the stack will be tackled next [depth-first].
- a done stack, where all backtracking points with the proper partial solutions and pointers to the to-do stack and the constraint stack at this time are stored.
- a constraint stack, where all constraints and their adjustments are stored. An adjustment of a constraint is generated, when a substitution of a variable is propagated in the constraints. In every verification equation with this variable in its left hand side the variable is deleted from the left hand side and apropriate times the substitution from the right hand side. If this is not possible, we have a Fail. If it is possible, the constraints of all variables in such adjusted verification equations can be sharpened. This can be done very efficiently, because the right hand sides of the equations are the constraints [or a multiple of it]. Then the rule "Pruning constraints" is applied.

In fact, the first and the last stack are so-called cactus stacks, because they have several branches, but at every moment only one branch is active. Other branches can become alive after backtracking using the second stack.

The same kind of algorithm with these stacks has also proven well suited for AC-matching.

4.3 AC1-matching examples

We will demonstrate the efficiency of our algorithm by some examples.

a) $h[f[x,y],g[x,y]] = h[f[a,b^2],g[a^2,b]]$

Decomposition leads to

(1) $f[x,y] = f[a,b^2]$

(2) $g[x,y] = g[a^2,b]$.

So we get the constraints

$x,y \in [f[a,b^2]]$ from (1) and $x,y \in [g[a^2,b]]$ from (2).

Intersection of the constraints results in

$x,y \in \{a,b\}$.

For x we try

$\sigma_1 = \{x \leftarrow a\}$ which leads to $y = f[b^2]$ and $y = g[a,b]$ ↯ and

$\sigma_2 = \{x \leftarrow b\}$ which leads to $y = f[a,b]$ and $y = g[a^2]$ ↯.

So we have to try 2 possible substitutions to detect the unsolvability of the problem instead of 6 using mutation steps.

b) $f[a,g[b^3,x],x,y] = f[a,b,g[a,b],g[a,b^2],g[a,b^4]]$

Simplification deletes the term a in both sides :

$f[g[b^3,x],x,y] = f[b,g[a,b],g[a,b^2],g[a,b^4]]$.

Variable abstraction results in

$f[u,x,y] = f[b,g[a,b],g[a,b^2],g[a,b^4]]$ (*)

and the constraint $g[b^3,x] = u$ which gives an at-least constraint

$u \in [g[b^{\geq 3}]]$.

The equation (*) yields to the constraints

$u,x,y \in [f[b,g[a,b],g[a,b^2],g[a,b^4]]$.

Because of the at-least constraint we get $u = g[a,b^4]$ and therefore

$g[b^3,x] = g[a,b^4]$.

Simplification results in the substitution

$x = g[a,b]$.

Propagating this substitution in the verification equation (*) we get

$f[g[a,b^4],g[a,b],y] = f[b,g[a,b],g[a,b^2],g[a,b^4]]$

or simplified

$y = f[b,g[a,b^2]]$.

So the found solution is

$\sigma = \{x \leftarrow g[a,b],\ y \leftarrow f[b,g[a,b^2]]\}$.

c] $f[a^2,g[a,x^2,y],g[x,z]] = f[a^3]$

Simplification results in

$f[g[a,x^2,y],g[x,z]] = a$.

Variable abstractions generates the equation

$f[u_1,u_2] = a$ [*]

and the [at-least] constraints $g[a,x^2,y] = u_1$ which means u_1 contains at least a, i.e. $u_1 \in [g[a^{\geq 1}]]$ and $g[x,z] = u_2$.

[*] leads to the constraints

$u_1,u_2 \in \{1_f,a\}$.

Combining this constraint with the at-least constraint for u_1, we get

$u_1 = a$

and a new equation

$g[a,x^2,y] = a$.

Simplification and the inference rule Variable arguments generate

$x = 1_g$ and $y = 1_g$.

Propagating this substitution into the verification equation and simplifying it yields directly to

$z = 1_f$

So the found solution is

$\sigma = \{x \leftarrow 1_g,\ y \leftarrow 1_g,\ z \leftarrow 1_f\}$.

5. Conclusion and future work

We have presented an AC1-matching algorithm using constraint propagation to reduce the necessary branching. We have shown that our ideas on constraints and theory matching presented earlier [[DG88]] for the AC case can also be applied to collapsing theories like AC1. Especially the collapsing cases can also be represented in constraints and be used to reduce the search. The algorithm can easily be adapted to new theories, the main task is to define the normalform and the representation and combination of the constraints.

For example, for the theory ACI, i.e. associativity, commutativity and idempotency, we would use ordered sets as arguments for ACI-functions in the normalform. Constraints are represented in the form $[f[s_1,...,s_n]]$, similar to the representation of AC1-constraints. Due to the idempotency, no element of the set occurs more than once. Combination of the constraints is similar to the AC1-case, too. When propagating substitutions into verification equations, terms on the right hand side are not deleted but marked. Then a verification equation is satisfied, if all terms on the right hand side are marked, when there are no more variables on the left hand side. This is correct, because the idempotency allows the multiple use of these terms.

For a theory like A [i.e. associativity], we have to use different representations and normalforms. Again, the normalform - the

argumentlist of an A-function symbol is a word - helps to find the variable constraints. Then the combination of constraints reduces to finding common prefixes, suffixes or subwords of the words representing the constraints.

Using constraints is a concept not only for building single theory matching algorithms, it can also provide a way to combine such algorithms efficiently. Within the borders of the theoretical results of Nipkow in [Ni89], using constraints can provide a similar gain of efficiency compared to Nipkow's algorithm as in the single theory cases.

A third advantage of the use of constraints in theory matching is the ability to start the matching operation with constraints obtained from calling functions of deduction systems. In the last years many efforts in the areas constraint deduction and constraint logic programming showed the usefulness of constraints. Our approach to theory matching allows to pass down term constraints in a basic operation, i.e. matching. This can result in more efficiency, again, because theory matches that are no solutions of the term constraints can be detected by the matching algorithm itself and need not be checked by higher levels of the deduction systems. Finally, when the constraints are used as result of the matching operation, we get a compact representation of all solutions of the problem

References :

[DG88] Denzinger, J. ; Gramlich, B. :
Efficient AC-Matching Using Constraint Propagation,
SEKI-Report SR-88-15, University of Kaiserslautern, 1988.

[KN90] Kapur, D. ; Narendran, P. :
Complexity of Associative-Commutative Unification Check and Related Problems,
Tech. Rep. 90-7, Dep. of Computer Science, SUNY, Albany, 1990.

[Mz85] Mzali, J. :
Filtrage Associatif, Commutatif ou Idempotent,
Int. Rep. 85 R 25, CRIN, Nancy, 1985.

[Mz86] Mzali, J. :
Matching with Distributivity,
Proc. 8th CADE, LNCS 230, Springer, 1986.

[Ni89] Nipkow, T. :
Combining Matching Algorithms : The Regular Case,
Proc. 3rd RTA, LNCS 355, Springer, 1989, pp. 343-358.

[St81] Stickel, M.E. :
A unification algorithm for associative-commutative functions,
J. ACM 28 (3), 1981, pp. 423-434.