Hierarchical Combination of Matching Algorithms

(Extended Abstract)

Serdar Erbatur[1], Deepak Kapur[2] *, Andrew M Marshall[3] †, Paliath Narendran[4] ‡, and Christophe Ringeissen[5]

[1] Università degli Studi di Verona (Italy)
[2] University of New Mexico (USA)
[3] Naval Research Laboratory (USA)
[4] University at Albany, SUNY (USA)
[5] LORIA – INRIA Nancy-Grand Est (France)

1 Introduction

A critical question in matching and unification is how to obtain an algorithm for the combination of non-disjoint equational theories when there exist algorithms for the constituent theories. In recent work ([4]) we were able to develop a new approach to the unification problem in the combination of non-disjoint theories. The approach is based on a new set of restrictions, for which we can identify a set of properties on the constituent theories, such that theories characterized by these properties satisfy the restrictions and thus can be combined using the new algorithm. The main properties of this class are: a hierarchical organization of E_1 and E_2, R_1 is a left-linear, convergent rewrite system corresponding to E_1, and the shared symbols are "inner constructors" of R_1.

Here we consider the matching problem in this new hierarchical framework. Due to the more restricted nature of the matching problem we obtain several improvements over the unification problem. One of the improvements is that we are able to relax several restrictions we assumed for the unification problem. Key among these discarded restrictions is a restriction on the type of new variables created by the unification algorithm for the first theory in the hierarchical organization. In the unification setting it was necessary to restrict variables which could cause reapplication of the first unification algorithm, denoted as "ping pong" variables. This tricky restriction can be avoided if most general solutions can be expressed without any new variable. Because matching problems in regular (variable-preserving) theories have only ground solutions, we can remove this assumption.

An additional improvement is obtained when constructing a general matching algorithm for the first theory which satisfies the restrictions of the hierarchical framework. In the unification case a general procedure was developed but due to the generality of unification problem, termination had to be checked for each theory. However, for the matching problem we are able to exploit an interesting relation to the work done on syntactic theories [5, 6, 3]. By assuming a newly defined *resolvent* property we are able to construct a terminating and thus general matching algorithm which can be used in the hierarchical framework for any theory satisfying the restrictions. The algorithm we present can be seen as an extension of the work done for matching in disjoint unions of regular/syntactic theories [7, 8, 9].

*Partially supported by the NSF grant CNS-0905222
†ASEE postdoctoral fellowship under contract to the NRL.
‡Partially supported by the NSF grant CNS-0905286

2 Preliminaries

We use the standard notation of equational unification [2] and term rewriting systems [1]. A term t is *linear* if each variable of t occurs only once in t. Given a first-order signature Σ, and a set E of Σ-axioms (i.e., pairs of Σ-terms, denoted by $l = r$), the *equational theory* $=_E$ is the congruence closure of E under the law of substitutivity. By a slight abuse of terminology, E will be often called an equational theory. An axiom $l = r$ is *variable-preserving* if $Var(l) = Var(r)$. An axiom $l = r$ is *linear* (resp. *collapse-free*) if l and r are linear (resp. non-variable terms). An equational theory is *variable-preserving* (resp. linear/collapse-free) if all its axioms are variable-preserving (resp. linear/collapse-free). An equational theory E is *finite* if for each term t, there are finitely many terms s such that $t =_E s$. A theory E is *subterm collapse-free* if and only if for all terms t it is not the case that $t =_E u$ where u is a strict subterm of t. Note that a subterm collapse-free theory is necessarily variable-preserving and collapse-free.

A Σ-equation is a pair of Σ-terms denoted by $s =^? t$. When t is ground, $s =^? t$ is denoted by $s \leq^? t$ and called a match-equation. A unification (resp. matching) problem P is a set of equations (resp. match-equations). An E-unifier of P is a substitution σ such that $s\sigma =_E t\sigma$ for each equation $s =^? t$ in P.

For a convergent rewrite system R we define a constructor of R to be a function symbol f which does not appear at the root on the left-hand side of any rewrite rule of R. We define an inner constructor to be a constructor f that satisfies the following additional restrictions: (1) f does not appear on the left-hand side on any rule in R. (2) f does not appear as the root symbol on the right-hand side of any rule in R. (3) there are no function symbols below f on the right-hand side of any rule in R. We consider two equational theories E_1 and E_2 built over the signatures Σ_1 and Σ_2. Let $\Sigma_{(1,2)} = \Sigma_1 \cap \Sigma_2$. In [4], we introduce a hierarchical framework for a union of equational theories $E_1 \cup E_2$ such that E_1 is given by a convergent rewrite system R_1 for which $\Sigma_{(1,2)}$-symbols are inner constructors. In [4], we study the unification problem in $E_1 \cup E_2$. In this work, we now consider the matching problem.

3 Hierarchical Combination for Matching

The key principle of the combination algorithm for matching is to purify only the left-hand sides of matching problems. Thus, this purification introduces a pending solved equation $X =^? t$. Since X occurs in a match-equation solved by A_1 or A_2, it will be instantiated by a ground term, say u, transforming eventually $X =^? t$ into a match-equation $t \leq^? u$. Hence, our rule-based procedures will generate equational problems involving also equations and not only match-equations. Fortunately, we assume the right properties to solve these equational problems by using only matching algorithms:

1. Properties of E_1: E_1 is finite, subterm collapse-free and R_1 is a left-linear, convergent term rewrite system corresponding to E_1.
2. Properties of E_2: E_2 is a linear, finite, collapse-free equational theory.
3. Properties of the shared symbols: If $f \in \Sigma_{(1,2)}$, then f is an *inner constructor* of R_1. If f and g are inner constructors of R_1, then f-rooted terms cannot be equated to g-rooted terms in E_2.

According to the above assumptions, we can show that $E_1 \cup E_2$ is finite, and so we could take a brute force approach to constructing a $E_1 \cup E_2$-matching algorithm [7]. However, we can use the constituent algorithms, A_1 and A_2 to construct a more efficient combination method.

We assume that A_1 and A_2 handle now left pure match-equations: A_1 handles match-equations whose left-hand sides are in $(\Sigma_1 \backslash \Sigma_{1,2})$, whilst A_2 handles match-equations whose left-hand sides are in Σ_2.

We first consider the question of constructing the A_1 algorithm. We show how such algorithms can be constructed for a family of theories related to the syntactic theories [5, 6, 3]. Therefore, we assume the following *resolvent* property for R_1.

Restriction 1. *(Algorithm A_1)*
Algorithm A_1 is a mutation-based algorithm as depicted in Figure 1, where R_1 is a resolvent *rewrite system; that is, any R_1-normal form can be reached by applying at most one rewrite step at the top position.*

Note, resolvent does not require that all paths from a term to its normal form use one topmost rewrite step, only that for each normal form there is at least one rewrite path with such a property. When R_1 is resolvent, the mutation-based A_1 algorithm presented in Figure 1 is sound and complete. For the second algorithm A_2, we simply assume it is a matching algorithm.

Restriction 2. *(Algorithm A_2)*
Algorithm A_2 is an E_2-matching algorithm.

Rule (i) **Mutate**

$$\{f(s_1, \ldots, s_m) \leq^? g(t_1, \ldots, t_n)\} \cup P$$
$$\longrightarrow \{s_1 =^? l_1, \ldots, s_m =^? l_m, r_1 \leq^? t_1, \ldots, r_n \leq^? t_n\} \cup P$$

If $f(l_1, \ldots, l_m) \to g(r_1, \ldots, r_n)$ is a fresh variant of a rule in R_1.

Rule (ii) **Matching Decomposition**

$$\{f(s_1, \ldots, s_m) \leq^? f(t_1, \ldots, t_m)\} \cup P$$
$$\longrightarrow \{s_1 \leq^? t_1, \ldots, s_m \leq^? t_m\} \cup P$$

Where $f \in \Sigma_1 \setminus \Sigma_{(1,2)}$.

Rule (iii) **Matching Clash**

$$\{f(s_1, \ldots, s_m) \leq^? g(t_1, \ldots, t_n)\} \cup P$$
$$\longrightarrow Fail$$

Where $f \in \Sigma_1 \setminus \Sigma_{(1,2)}, f \neq g$ and **Mutate** does not apply.

Figure 1: Mutation-based A_1 algorithm

3.1 The Matching Procedure - Hierarchical Combination

We give a new matching procedure for the hierarchical combination. It works as follows. First, we purify the left-hand sides of match-equations. After this purification step, we can easily distinguish which left-pure match-equations must be given to A_1 and A_2. Then, the solutions computed by A_1 and A_2 are combined using some replacement and merging rules.

$Solve_1$**: Run** A_1
We apply A_1 to match-equations having $\Sigma_1 \setminus \Sigma_{(1,2)}$-pure left-hand sides
$Solve_2$**: Run** A_2
We apply A_2 to match-equations having Σ_2-pure left-hand sides

RemEq:
$$\frac{\mathcal{P} \uplus \{t =^? t'\}}{\mathcal{P} \cup \{t \leq^? t'\}} \quad \text{if } t' \text{ is ground}$$

Rep:
$$\frac{\mathcal{P} \uplus \{t =^? t'[Y],\ Y \leq^? u\}}{\mathcal{P} \cup \{t =^? t'[u],\ Y \leq^? u\}}$$

Merge:
$$\frac{\mathcal{P} \uplus \{X \leq^? t,\ X \leq^? s\}}{\mathcal{P} \cup \{X \leq^? t\}} \quad \text{if } s =_{E_1 \cup E_2} t$$

Clash:
$$\frac{\mathcal{P} \uplus \{X \leq^? t,\ X \leq^? s\}}{Fail} \quad \text{if } s \neq_{E_1 \cup E_2} t$$

Figure 2: $\mathfrak{D}$: inference system for the combination of matching

The matching procedure is given as the inference system $\mathfrak{D}$ defined in Figure 2 by the set of inferences rules

$$\{Solve_1, Solve_2, \mathbf{RemEq}, \mathbf{Rep}, \mathbf{Merge}, \mathbf{Clash}\}.$$

We can easily verify that each rule in $\mathfrak{D}$ preserves the set of $E_1 \cup E_2$-solutions. This is clear for the rules in $\{\mathbf{RemEq}, \mathbf{Rep}, \mathbf{Merge}, \mathbf{Clash}\}$. Moreover, this is true by definition for $Solve_1$, and since E_2-matching is sound and complete for solving $E_1 \cup E_2$-matching problems whose left-hand sides are 2-pure, its is also true for $Solve_2$. Furthermore, it can be shown that normal forms with respect to $\mathfrak{D}$ are matching problems in solved form and that $\mathfrak{D}$ terminates for any input. This implies that the algorithm $\mathfrak{D}$ is sound and complete, which means that it provides an $E_1 \cup E_2$-matching algorithm.

Example 3.1. The following theory appears to be a good case-study for the above hierarchical combination method.

$$\mathcal{E}_{AC} = \left\{ \begin{array}{l} exp(exp(x,y),z) = exp(x, y \circledast z) \\ exp(x * y, z) = exp(x,z) * exp(y,z) \end{array} \right\} = E_1$$
$$\left\{ \begin{array}{l} (x \circledast y) \circledast z = x \circledast (y \circledast z) \\ x \circledast y = y \circledast x \end{array} \right\} = E_2$$

The theory $\mathcal{E}_{AC}$ has the following $AC(\circledast)$-convergent system:

$$R_1 = \left\{ \begin{array}{rcl} exp(exp(x,y),z) & \to & exp(x, y \circledast z) \\ exp(x * y, z) & \to & exp(x,z) * exp(y,z) \end{array} \right.$$

The main task is to construct an A_1 algorithm. It can be constructed from an instantiation of the mutation-based algorithm given in Figure 1. This leads to a set of matching inference rules dedicated to the particular case of R_1.

References

[1] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, New York, NY, USA, 1998.

[2] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

[3] Alexandre Boudet and Evelyne Contejean. On n-syntactic equational theories. In Hélène Kirchner and Giorgio Levi, editors, *Algebraic and Logic Programming*, volume 632 of *Lecture Notes in Computer Science*, pages 446–457. Springer Berlin Heidelberg, 1992.

[4] Serdar Erbatur, Deepak Kapur, Andrew M. Marshall, Paliath Narendran, and Christophe Ringeissen. Hierarchical combination. In Maria Paola Bonacina, editor, *Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 249–266. Springer Berlin Heidelberg, 2013.

[5] Jean-Pierre Jouannaud. Syntactic theories. In Branislav Rovan, editor, *Mathematical Foundations of Computer Science 1990*, volume 452 of *Lecture Notes in Computer Science*, pages 15–25. Springer Berlin Heidelberg, 1990.

[6] C. Kirchner and F. Klay. Syntactic theories and unification. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 270–277, Jun 1990.

[7] T. Nipkow. Proof transformations for equational theories. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 278–288, Jun 1990.

[8] Tobias Nipkow. Combining matching algorithms: The regular case. *J. Symb. Comput.*, 12(6):633–654, 1991.

[9] Christophe Ringeissen. Combining decision algorithms for matching in the union of disjoint equational theories. *Inf. Comput.*, 126(2):144–160, 1996.