Hierarchical Combination of Unification Algorithms

Serdar Erbatur, Deepak Kapur, Andrew Marshall, Paliath Narendran, Christophe Ringeissen

Hierarchical Combination of Unification Algorithms (Extended Abstract)

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

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

1 Introduction

A critical question in unification theory is how to obtain a unification algorithm for the combination of non-disjoint equational theories when there exists unification algorithms for the constituent theories. The problem is known to be difficult and can easily be seen to be undecidable in the general case. Therefore, previous work has focused on identifying specific conditions and methods in which the problem is decidable. We continue the investigation in this paper, building on previous combination results, [2, 4, 7] and [6]. We are able to develop a novel approach to the non-disjoint combination problem. The approach is based on a new set of restrictions and combination method such that if the restrictions are satisfied the method produces an algorithm for the unification problem in the union of non-disjoint equational theories.

A full version of this paper [5] will appear in the proceedings of the 24th International Conference on Automated Deduction (CADE-24).

2 Preliminaries

We use the standard notation of equational unification [3] and term rewriting systems [1]. A Σ-rooted term is a term whose top symbol is in Σ. Let $\Sigma_{(1,2)} = \Sigma_1 \cap \Sigma_2$. An alien subterm of a $\Sigma_1 \setminus \Sigma_{(1,2)}$-rooted term t (resp. Σ_2-rooted term) is a Σ_2-rooted subterm (resp. $\Sigma_1 \setminus \Sigma_{(1,2)}$-rooted subterm) of t such that all its superterms are $\Sigma_1 \setminus \Sigma_{(1,2)}$-rooted (resp. Σ_2-rooted). 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. Given a signature $\Sigma' \subseteq \Sigma$, $\mathcal{S}|_{\Sigma'}$ denotes the set of Σ'-equations occurring in $\mathcal{S}$. A set of equations $\mathcal{S}$ is said to be in *standard form* over a signature Σ if and only if every equation in $\mathcal{S}$ is of the form $x =^? t$, where x is a variable and t is one of the following: (a) a variable different from x, (b) a constant, or (c) a term of depth 1 that contains no constants, i.e., function symbols applied to variables. It is not generally difficult to decompose equations of a given problem into simpler standard forms. A set of equations is said to be in *dag-solved form* (or *d-solved form*) if and only if they can be arranged as a list $x_1 =^? t_1, \ldots, x_n =^? t_n$

*Supported by the Department of Computer Science of the University at Albany and then INRIA Nancy-Grand Est during a portion of this work.
†Partially supported by the NSF grant CNS-0905222
‡Supported by an ASEE postdoctoral fellowship under contract to the NRL.
§Partially supported by the NSF grant CNS-0905286

where (a) each left-hand side x_i is a distinct variable, and (b) $\forall 1 \leq i \leq j \leq n$: x_i does not occur in t_j. Each x_i in this case is called a *solved variable*. A set of equations S is said to be in *Σ-solved form* if and only if it is in standard form and $S|_\Sigma$ is in dag-solved form. 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: (i) f does not appear on the left-hand side on any rule in R, (ii) f does not appear as the root symbol on the right-hand side of any rule in R, and (iii) there are no function symbols below f on the right-hand side of any rule in R.

3 Combination Procedure

We want to investigate conditions to build an $E_1 \cup E_2$-unification algorithm by using two algorithms A_1 and A_2 solving two different kinds of $E_1 \cup E_2$-unification problems, described below in Restriction 1 and Restriction 2 respectively. Consider an $E_1 \cup E_2$-unification problem $\mathcal{P}$ in standard form. The steps of the combination procedure (see Figure 2) are as follows: (1) run A_1 on the $\Sigma_1 \setminus \Sigma_{(1,2)}$-equations of $\mathcal{P}$; (2) run A_2 on the Σ_2-equations of the unification problem obtained at step (1); (3) collect the resulting problems that are in dag-solved form. As in disjoint combination [2], we perform as step (0) a variable identification in order to guess a priori all possible identifications of variables occurring in $\mathcal{P}$. For this approach to work we need to place some restrictions.

3.1 Restrictions

We present a less technical overview of the required restrictions (see [5] for more details). Consider two *subterm collapse-free* equational theories such that $\Sigma_1 \cap \Sigma_2 = \Sigma_{(1,2)} \neq \emptyset$. Fresh variables created by A_1 that could cause the need for reapplication of the rules of A_1 during the application of A_2 are called "Ping-Pong Variables" as it implies a back-and-forth mechanism between the algorithms.

Restriction 1. *(on Algorithm A_1) Let $\mathcal{P}$ be a set of $\Sigma_1 \setminus \Sigma_{(1,2)}$-equations. Algorithm A_1 computes a set of problems $\{\mathcal{Q}_k\}_{k\in K}$ such that*

$$\bigcup_{k\in K} CSU_{E_1\cup E_2}(\mathcal{Q}_k) \text{ is a } CSU_{E_1\cup E_2}(\mathcal{P}) \text{ and for each } k \in K:$$

(i) $\mathcal{Q}_k$ consists of $(\Sigma_1 \setminus \Sigma_{(1,2)})$-equations and $\Sigma_{(1,2)}$-equations.

(ii) $\mathcal{Q}_k$ is in standard form and $(\Sigma_1 \setminus \Sigma_{(1,2)})$-solved form.

(iii) No fresh variable occurring in a nonvariable $\Sigma_{(1,2)}$-term in $\mathcal{Q}_k$ appears as solved in $\mathcal{Q}_k$.

(*CSU* stands for "complete set of unifiers".)

Note that A_1 is a special type of algorithm that returns a "partial" solution to an $E_1 \cup E_2$-unification problem. A_1 is needed to solve some portion of the problem, namely the $\Sigma_1 \setminus \Sigma_{(1,2)}$-pure, but a standard E_1-unification algorithm is not sufficient, even for $\Sigma_1 \setminus \Sigma_{(1,2)}$-pure $E_1 \cup E_2$ problems.

Example 3.1. Let $E_1 := \{h(a, x, y) = g(x * y),\ h(b, x, y) = g(y * x)\}$ and let E_2 be the commutative theory for $*$. Then, $h(a, a, z) =^? h(b, a, b)$ is not solvable in E_1 but is in $E_1 \cup E_2$.

Restriction 1 ensures that completeness is not lost by ensuring that a CSU for all the partial solutions is a CSU for the original problem.

To explain Restriction 1(iii), let us note that A_1 may generate fresh $\Sigma_{(1,2)}$-equations, e.g. $z =^? f(x, y)$ where $f \in \Sigma_{(1,2)}$, together with some $\Sigma_1 \setminus \Sigma_{(1,2)}$-equations, e.g. $x =^? s, y =^? t$. If A_2 later generates $x =^? y$, then this may lead to the reapplication of A_1, to solve $x =^? s, x =^? t$. If x and y are from the initial set of variables, this problem can be discarded without loss of generality, since the variable identification performed initially generates another unification problem where x and y are already identified. The problem remains if x or y are fresh variables, "ping-pong variables". In order to avoid it, we introduce Restriction 1(iii), where only the occurrences of fresh variables are restricted.

Restriction 2. *(on Algorithm A_2)*
Algorithm A_2 computes a finite complete set of 2-pure unifiers of 2-pure $E_1 \cup E_2$-unification problems.

We restrict A_2 to compute only 2-pure substitutions. This is used to avoid possible reapplications of A_1 after A_2.

Restriction 3. *(Errors)*

(i) A $\Sigma_1 \setminus \Sigma_{(1,2)}$-rooted term cannot be $E_1 \cup E_2$-equal to a Σ_2-rooted term.

(ii) $E_1 \cup E_2$ is subterm collapse-free.

Note that in the next section we give a modularity result for subterm collapse-freeness. That is, we give a non-empty family of theories for which subterm collapse-freeness is a modular property.

The failure rules associated with Restriction 3 are given in Figure 1.

Conflict: If $s(\epsilon) \in \Sigma_1 \setminus \Sigma_{(1,2)}$ and $t(\epsilon) \in \Sigma_2$: $\{v =^? s, v =^? t\} \cup P \longrightarrow Fail$
Cycle: If P contains a cycle: $P \longrightarrow Fail$

Figure 1: Failure rules

It can be shown based on these restrictions that Algorithm $\mathfrak{C}$ is a complete unification algorithm for the combined theory. More precisely, if A_1 and A_2 are algorithms satisfying Restrictions 1 through 3, then Algorithm $\mathfrak{C}$ is sound and complete, which also means that $E_1 \cup E_2$-unification is finitary.

4 A Class of Hierarchically Combinable Theories

In this section we define a non-empty class of theories for which Restrictions 1 through 3 can be satisfied. We assume that E_1 *and* E_2 *are subterm collapse-free.* The class is then defined by the following additional properties.

1. Properties of E_1:
   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 equational theory.

Some of the steps below are non-deterministic, hence lead to computation paths. Throughout each path, the failure rules of Figure 1 are applied eagerly after each step. Let $\mathcal{V} = Var(\mathcal{P})$.

Step 0: Variable Identification

Guess a partition on $\mathcal{V}$ and set variables in each subset equal to each other. This requires adding fresh equations of type $u =^? w$, where $u, w \in \mathcal{V}$ and u and w belong to the same subset in the partition. Let us denote the enumeration of the partitions of $\mathcal{V}$ as $\pi_1, \pi_2, \ldots, \pi_m$. Once a partition is selected, the unification problem $\mathcal{P}$ is modified by adding the fresh equations between variables to the unification problem. We denote the modified problems by $\mathcal{P}^{\pi_i}$, $1 \leq i \leq m$.

Step 1: Run A_1

We apply A_1 to the $\Sigma_1 \setminus \Sigma_{(1,2)}$-equations of $\mathcal{P}^{\pi_i}$. If A_1 fails for all π_i, report failure and stop. Otherwise, we now have a modified, by A_1, set of unification problems from $\mathcal{P}^{\pi_i}$, $\mathcal{P}_1^{\pi_i}, \ldots \mathcal{P}_n^{\pi_i}$ for $n \geq 1$, such that all the $\Sigma_1 \setminus \Sigma_{(1,2)}$-equations in each $\mathcal{P}_j^{\pi_i}$, are in $(\Sigma_1 \setminus \Sigma_{(1,2)})$-solved form.

Step 2: Run A_2

Run Algorithm A_2 on the 2-pure equations of each $\mathcal{P}_j^{\pi_i}$. If A_2 equates variables in $\mathcal{V}$ not equated by π_{i_j} discard that particular $\mathcal{P}_j^{\pi_i}$. If none of the remaining $\mathcal{P}_1^{\pi_i}, \ldots \mathcal{P}_n^{\pi_i}$ exit with success, return failure. Otherwise, return success.

Figure 2: $\mathfrak{C}$: Unification Algorithm for the combined theory $E_1 \cup E_2$.

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.

Example 4.1. The equational theory $\mathcal{E}_{AC}$ considered in [6] satisfies all the above properties. The axioms of $\mathcal{E}_{AC}$ are as follows:

$$exp(exp(x, y), z) = exp(x, y \circledast z) \ (1) \quad (x \circledast y) \circledast z = x \circledast (y \circledast z) \ (3)$$
$$exp(x * y, z) = exp(x, z) * exp(y, z) \ (2) \quad x \circledast y = y \circledast x \ (4)$$

Here, $\mathcal{E}_{AC} = E_1 \cup E_2$ where $E_1 = \{(1), (2)\}$ and $E_2 = \{(3), (4)\}$.

4.1 Satisfying Restriction 1

The first and critical step to satisfying Restriction 1 is the construction of an inference system $\mathcal{G}$ such that the standard forms of leaves in the search tree generated by $\mathcal{G}$ are the $\Sigma_1 \setminus \Sigma_{(1,2)}$-solved forms corresponding to a complete set of $E_1 \cup E_2$-unifiers of the input problem, thus satisfying the majority of Restriction 1. In [5], we modify the general syntactic E-unification procedure $\mathcal{G}$ [3], but we still have to identify interesting cases for which $\mathcal{G}$ can be turned into a terminating algorithm. For the particular case of $\mathcal{E}_{AC}$, an algorithm A_1 satisfying Restriction 1 can be constructed from scratch as in [6].

4.2 Satisfying Restriction 2

Let t^ϕ denote the standard 2-abstraction of t, which corresponds to the 2-pure term obtained from t by replacing alien subterms of t with fresh variables ($E_1 \cup E_2$-equal alien subterms are replaced by the same fresh variable). It can be shown if $s \longleftrightarrow_{E_1 \cup E_2} t$, then $s^\phi =_{E_2} t^\phi$. Reasoning inductively it can further be shown for any 2-pure equation $s =^? t$ where $s\sigma =_{E_1 \cup E_2}$

$t\sigma$, that $s\sigma^\phi =_{E_2} t\sigma^\phi$. This implies that E_2-unification is sound and complete for solving 2-pure $E_1 \cup E_2$-unification problems. Hence, an E_2-unification algorithm provides an algorithm A_2 satisfying Restriction 2 since it computes a complete set of 2-pure unifiers.

4.3 Satisfying Restriction 3

Using the fact that E_2 is subterm collapse-free and the fact that if $s \longleftrightarrow_{E_1 \cup E_2} t$ then $s^\phi =_{E_2} t^\phi$, it can be shown that a $\Sigma_1 \setminus \Sigma_{(1,2)}$-rooted term cannot be $E_1 \cup E_2$-equal to a Σ_2-rooted term. Lastly, it can be shown that for any term t and any strict subterm u of t, if $t \to^+_{R_1} t'$, then $t' \neq_{E_2} u$. This fact leads directly to the result that $E_1 \cup E_2$ is subterm collapse-free.

As a final remark, it is important to note that our algorithm $\mathfrak{C}$ does not work in a symmetric way with respect to E_1 and E_2, contrary to combination algorithms where we assume that any $(\Sigma_1 \cap \Sigma_2)$-equality is valid in E_1 if and only if it is valid in E_2 [2, 4, 7]. Indeed, the algorithm A_1 partially solves the problem modulo $E_1 \cup E_2$ (and not only modulo E_1). Then, A_2 is in charge of solving modulo E_2 the remaining part that was kept unsolved by A_1. Our restrictions allow us to get all the solved forms without any further pingponging between A_2 and A_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 Klaus U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21(2):211 – 243, 1996.

[3] 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.

[4] Eric Domenjoud, Francis Klay, and Christophe Ringeissen. Combination techniques for non-disjoint equational theories. In *International Conference on Automated Deduction, (CADE-12)*, volume 814 of *LNCS*, pages 267–281. 1994.

[5] Serdar Erbatur, Deepak Kapur, Andrew M. Marshall, Paliath Narendran, and Christophe Ringeissen. Hierarchical combination. In *Proceedings of CADE-24*, volume 7898 of *LNAI*, pages 249–266. Springer, 2013.

[6] Serdar Erbatur, Andrew M. Marshall, Deepak Kapur, and Paliath Narendran. Unification over distributive exponentiation (sub)theories. *Journal of Automata, Languages and Combinatorics (JALC)*, 16(2–4):109–140, 2011.

[7] Christophe Ringeissen. Unification in a combination of equational theories with shared constants and its application to primal algebras. In *The 1st International Conference on Logic Programming and Automated Reasoning, volume 624 of LNAI*, pages 261–272. Springer, 1992.