Rule-Based Unification in Combined Theories and the Finite Variant Property

Ajay Eeralla, Serdar Erbatur, Andrew Marshall, Christophe Ringeissen

Rule-Based Unification in Combined Theories and the Finite Variant Property

Ajay K. Eeralla[1], Serdar Erbatur[2], Andrew M. Marshall[3], and Christophe Ringeissen[4⋆]

[1] University of Missouri, Columbia (USA)
[2] Ludwig-Maximilians-Universität München (Germany)
[3] University of Mary Washington (USA)
[4] Université de Lorraine, CNRS, Inria, LORIA (France)

Abstract. We investigate the unification problem in theories defined by rewrite systems which are both convergent and forward-closed. These theories are also known in the context of protocol analysis as theories with the finite variant property and admit a variant-based unification algorithm. In this paper, we present a new rule-based unification algorithm which can be seen as an alternative to the variant-based approach. In addition, we define forward-closed combination to capture the union of a forward-closed convergent rewrite system with another theory, such as the Associativity-Commutativity, whose function symbols may occur in right-hand sides of the rewrite system. Finally, we present a combination algorithm for this particular class of non-disjoint unions of theories.

Keywords: term rewriting, unification, combination, forward-closure

1 Introduction

Unification plays a central role in logic systems based on the resolution principle, to perform the computation in declarative programming, and to deduce new facts in automated reasoning. Syntactic unification is particularly well-known for its use in logic programming. Being decidable and unitary are remarkable properties of syntactic unification. More generally, we may consider equational unification, where the problem is defined modulo an equational theory E, like for instance the Associativity-Commutativity. Equational unification, say E-unification, is undecidable in general. However, specialized techniques have been developed to solve the problem for particular classes of equational theories, many of high practical interest. It is not uncommon to have such equational theories include Associativity-Commutativity, which is useful to represent arithmetic operators. Nowadays, security protocols are successfully analyzed using dedicated reasoning tools [5,4,13,18] in which protocols are usually represented by clauses in first-order logic with equality. In these protocol analyzers, equational theories are

⋆ This work has received funding from the European Research Council (ERC) under the H2020 research and innovation program (grant agreement No 645865-SPOOC).

used to specify the capabilities of an intruder [1]. To support the reasoning in these equational theories E, one needs to use E-unification procedures. When the equational theory E has the Finite Variant Property (FVP) [8], there exists a reduction from E-unification to syntactic unification via the computation of finitely many variants of the unification problem. When this reduction is used, we talk about variant-based unification. The class of equational theories with the FVP has attracted a considerable interest since it contains theories that are crucial in protocol analysis [14,7,6,9,19]. The concept of narrowing is another possible unification technique when E is given by a convergent term rewrite system (TRS). Narrowing is a generalization of rewriting which is widely used in declarative programming. It is complete for E-unification, but it terminates only in some very particular cases. A particular narrowing strategy, called folding variant narrowing, has been shown complete and terminating for any equational theory with the FVP [14]. When E has the property of being *syntactic* [16,20], it is possible to apply a rule-based unification procedure in the same vein as the one known for syntactic unification, which is called a mutation-based unification procedure. Unfortunately, being syntactic is not a sufficient condition to insure the termination of this unification procedure. Finally, another important scenario is given by an equational theory E defined as a union of component theories. To solve this case, it is quite natural to proceed in a modular way by reusing the unification algorithms available in the component theories. There are terminating and complete combination procedures for signature-disjoint unions of theories [21,3], but the non-disjoint case remains a challenging problem [12].

In this paper, we investigate the impact of considering an equational theory with the FVP in order to get a terminating mutation-based unification procedure and a terminating combination procedure for some non-disjoint unions of theories. Instead of directly talking about the FVP, we study the equivalent class of theories defined by forward-closed convergent TRSs [6]. Actually, a forward-closed convergent TRS is a syntactic theory admitting a terminating mutation-based unification procedure. Here, we consider the unification problem in the class of *forward-closed combinations* defined as unions of a forward-closed convergent TRS plus an equational theory over function symbols that may only occur in the right-hand sides of the TRS. To solve this problem we need a mutation procedure for the forward-closed component of the combination. Rather than reusing the mutation procedure given in [17] we develop a new mutation procedure which is more conducive to combination. By adding some standard combination rules, we show how to extend this new mutation procedure in order to solve the unification problem in forward-closed combinations.

The rest of the paper is organized as follows. Section 2 recalls the standard notions and Section 3 introduces the class of forward-closed theories. In Section 4, we present a terminating mutation-based unification procedure for forward-closed theories. In Section 5, we introduce forward-closed combinations. The related combination method is given in Section 6, by proving its termination and correctness. Finally, Section 8 discusses some limitations and possible extensions

of this work. Omitted proofs and additional unification procedures can be found in [10].

2 Preliminaries

We use the standard notation of equational unification and term rewriting systems [2]. Given a first-order signature Σ and a (countable) set of variables V, the set of Σ-terms over variables V is denoted by $T(\Sigma, V)$. The set of variables in a term t is denoted by $Var(t)$. A term t is *ground* if $Var(t) = \emptyset$. A term is *linear* if all its variables occur only once. For any position p in a term t (including the root position ϵ), $t(p)$ is the symbol at position p, $t|_p$ is the subterm of t at position p, and $t[u]_p$ is the term t in which $t|_p$ is replaced by u. A substitution is an endomorphism of $T(\Sigma, V)$ with only finitely many variables not mapped to themselves. A substitution is denoted by $\sigma = \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$, where the domain of σ is $Dom(\sigma) = \{x_1, \ldots, x_m\}$. Application of a substitution σ to t is written $t\sigma$. Given 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 is often called an equational theory). Equivalently, $=_E$ can be defined as the reflexive transitive closure $\leftrightarrow_E^*$ of an equational step $\leftrightarrow_E$ defined as follows: $s \leftrightarrow_E t$ if there exist a position p of s, $l = r$ (or $r = l$) in E, and substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$. An axiom $l = r$ is *regular* 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 *regular* (resp., linear/collapse-free) if all its axioms are regular (resp., linear/collapse-free). A theory E is *syntactic* if it has finite *resolvent presentation* S, defined as a finite set of axioms S such that each equality $t =_E u$ has an equational proof $t \leftrightarrow_S^* u$ with at most one equational step $\leftrightarrow_S$ applied at the root position. A Σ-equation is a pair of Σ-terms denoted by $s =^? t$ or simply $s = t$ when it is clear from the context that we do not refer to an axiom. An E-unification problem is a set of Σ-equations, $G = \{s_1 =^? t_1, \ldots, s_n =^? t_n\}$, or equivalently a conjunction of Σ-equations. The set of variables in G is denoted by $Var(G)$. A solution to G, called an *E-unifier*, is a substitution σ such that $s_i\sigma =_E t_i\sigma$ for all $1 \leq i \leq n$, written $E \models G\sigma$. A substitution σ is *more general modulo E* than θ on a set of variables V, denoted as $\sigma \leq_E^V \theta$, if there is a substitution τ such that $x\sigma\tau =_E x\theta$ for all $x \in V$. An E-unification algorithm computes a (finite) *Complete Set of E-Unifiers* of G, denoted by $CSU_E(G)$, which is a set of substitutions such that each $\sigma \in CSU_E(G)$ is an E-unifier of G, and for each E-unifier θ of G, there exists $\sigma \in CSU_E(G)$ such that $\sigma \leq_E^{Var(G)} \theta$. Given a unifiable equation $s =^? t$, a syntactic unification algorithm computes a unique most general unifier denoted by $mgu(s, t)$. A set of equations $G = \{x_1 =^? t_1, \ldots, x_n =^? t_n\}$ is said to be in *tree solved form* if each x_i is a variable occurring once in G. Given an idempotent substitution $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ (such that $\sigma\sigma = \sigma$), $\hat{\sigma}$ denotes the corresponding tree solved form. A set of equations is said to be in *dag solved form* if they can be arranged as a list $x_1 =^? t_1, \ldots, x_n =^? t_n$ 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. A set of equations $\{x_1 =^? t_1, \ldots, x_n =^? t_n\}$ is a *cycle* if for any $i \in [1, n-1], x_{i+1} \in Var(t_i)$, $x_1 \in Var(t_n)$, and there exists $j \in [1, n]$ such that t_j is not a variable. Given two variables x and y, $x = y$ is said to be *solved* in a set of equations G if x does not occur in $G \backslash \{x = y\}$. Then, x is said to be *solved* in G. Given two disjoint signatures Σ_1 and Σ_2 and any $i = 1, 2$, Σ_i-terms (including the variables) and Σ_i-equations (including the equations between variables) are called *i-pure*. For any $\Sigma_1 \cup \Sigma_2$-theory E, an E-unification problem is in *separate form* if it is a conjunction $G_1 \wedge G_2$, where G_i is a conjunction of Σ_i-equations for $i = 1, 2$. A term t is called a *Σ_i-rooted* term if its root symbol is in Σ_i. An *alien* subterm of a Σ_i-rooted term t is a Σ_j-rooted subterm s $(i \neq j)$ such that all superterms of s are Σ_i-rooted. A *term rewrite system* (TRS) is a pair (Σ, R), where Σ is a signature and R is a finite set of rewrite rules of the form $l \to r$ such that l, r are Σ-terms, l is not a variable and $Var(r) \subseteq Var(l)$. A term s *rewrites* to a term t w.r.t R, denoted by $s \to_R t$ (or simply $s \to t$), if there exist a position p of s, $l \to r \in R$, and substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$. A TRS R is *terminating* if there are no infinite reduction sequences with respect to $\to_R$. A TRS R is *confluent* if, whenever $t \to_R^* s_1$ and $t \to_R^* s_2$, there exists a term w such that $s_1 \to_R^* w$ and $s_2 \to_R^* w$. A confluent and terminating TRS is called *convergent*. In a convergent TRS R, we have the existence and the uniqueness of R-normal forms, denoted by $t{\downarrow_R}$ for any term t. A substitution σ is *normalized* if, for every variable x in the domain of σ, $x\sigma$ is a normal form. A convergent TRS R is said to be *subterm convergent* if for any $l \to r \in R$, r is either a strict subterm of l or a constant. To simplify the notation, we often use tuples of terms, say $\bar{u} = (u_1, \ldots, u_n)$, $\bar{v} = (v_1, \ldots, v_n)$. Applying a substitution σ to $\bar{u}$ is the tuple $\bar{u}\sigma = (u_1\sigma, \ldots, u_n\sigma)$. The tuples $\bar{u}$ and $\bar{v}$ are said E-equal, denoted by $\bar{u} =_E \bar{v}$, if $u_1 =_E v_1, \ldots, u_n =_E v_n$. Similarly, $\bar{u} \to_R^* \bar{v}$ if $u_1 \to_R^* v_1, \ldots, u_n \to_R^* v_n$, $\bar{u}$ is R-normalized if $u_1, \ldots, u_n$ are R-normalized, and $\bar{u} =^? \bar{v}$ is $u_1 =^? v_1 \wedge \cdots \wedge u_n =^? v_n$.

3 Forward Closure

In this section, we define the central notion of finite forward closure. To define the forward closure as in [6], let us first introduce the notion of redundancy. For a given convergent TRS R, assume a reduction ordering $<$ such that $r < l$ for any $l \to r \in R$ and $<$ is total on ground terms. Since (rewrite) rules are multisets of two terms, the multiset extension of $<$ leads to an ordering on rules, also denoted by $<$, which is total on ground instances of rules. A rule ρ is *strictly redundant in R* if any ground instance $\rho\sigma$ of ρ is a logical consequence of ground instances of R that are strictly smaller w.r.t $<$ than $\rho\sigma$. A rule ρ is *redundant in R* if ρ is strictly redundant in R or ρ is an instance of some rule in R. Given two rules $\rho_1 = (g \to d)$, $\rho_2 = (l \to r)$ and a non-variable position p of d such that $d|_p$ and l are unifiable, $Fwd(\rho_1, \rho_2, p)$ denotes the rule $(g \to d[r]_p)\sigma$ where $\sigma = mgu(d|_p, l)$. Forward closure steps are inductively defined as follows:

- $FC_0(R) = NR_0(R) = R$,

– $FC_{k+1}(R) = FC_k(R) \cup NR_{k+1}(R)$ where $NR_{k+1}(R)$ is the set of rules $\rho_3 = Fwd(\rho_1, \rho_2, p)$ such that $\rho_1 \in NR_k(R)$, $\rho_2 \in R$, p is a non-variable position of the right-hand side of ρ_1, and ρ_3 is not redundant in $FC_k(R)$.

The *forward closure* of R is $FC(R) = \bigcup_{k \geq 0} FC_k(R)$. A TRS R is *forward-closed* if $FC(R) = R$. A TRS is *forward-closed convergent* if it is both forward-closed and convergent.

Example 1. Any subterm convergent TRS has a finite forward closure. Subterm convergent TRSs are often used in the verification of security protocols [1], e.g., $\{dec(enc(x, y), y) \to x\}$ and $\{fst(pair(x, y)) \to x,\ snd(pair(x, y)) \to y\}$.

Example 2. The following TRSs are forward-closed convergent:

$\{f(x) + f(y) \to f(x * y)\}$	$\{g(h(x, y), z) \to h(x, y * z)\}$
$\{f(x) + y \to f(x * y)\}$	$\{d(e(x, a), a) \to x * a\}$
$\{exp(exp(a, x), y) \to exp(a, x * y)\}$	$\{pdt(pair(x, y)) \to x * y\}$

We will study the unification problem in a combination of any of these TRSs with an equational theory over $*$, such as $C = \{x * y = y * x\}$ (Commutativity) or $AC = \{x * (y * z) = (x * y) * z, x * y = y * x\}$ (Associativity-Commutativity).

It has been shown in [6] that for any convergent TRS R, R has a finite forward closure if and only if R has the finite variant property (FVP, for short). When a TRS has the FVP, any R-unification problem G admits a computable finite complete set of R-variants of G, say $V_R(G)$, such that solving G reduces to solve the syntactic unification problems in $V_R(G)$. In many cases, the computation of $V_R(G)$ can be prohibitive even with an efficient implementation of folding variant narrowing [14,9]. For these cases, it is interesting to have an alternative to this brute force method, possibly via a rule-based R-unification procedure that does not impose a full reduction to syntactic unification.

4 Rule-Based Unification in Forward-Closed Theories

To design a rule-based unification procedure for forward-closed theories, we basically reuse the *BSM* unification procedure initially developed for the class of theories saturated by paramodulation [17], where *BSM* stands for *Basic Syntactic Mutation*. The *BSM* procedure extends syntactic unification with some additional mutation rules applied in a *don't know* non-deterministic way. These mutation rules are parameterized by a finite set of axioms corresponding to a resolvent presentation (cf. Section 2). The resulting *BSM* unification procedure is similar to the mutation-based unification procedures designed for syntactic theories [16,20] but with the additional property of being terminating. To get termination, it makes use of boxed terms. Variables can be considered as implicitly boxed, and terms are boxed according to the following rules:

– Subterms of boxed terms are also boxed.
– Terms boxed in the premises of an inference rule remain boxed in the conclusion.

– When the "box" status of a term is not explicitly given in an inference rule, it can be either boxed or unboxed. For instance, each occurrence of f in the premise of **Imit** rule (cf. Figure 1) can be either boxed or unboxed.

Boxed terms allow us to focus on particular R-normalized solutions of a unification problem. Hence, we are interested in R-normalized solutions σ such that $t\sigma$ is R-normalized for each boxed term t occurring in the unification problem.

Definition 3. *Let G be a unification problem and σ be a substitution. We say that (G, σ) is* R-normalized *if σ is R-normalized, and for any term t in G, $t\sigma$ is R-normalized whenever t is boxed.*

Assuming a forward-closed convergent TRS R is sufficient to replay the correctness proofs of *BSM* originally stated for theories saturated by paramodulation. Thus, *BSM* can be rephrased by using directly a forward-closed convergent TRS R as input. In this setting, the equational theory of any forward-closed convergent TRS R is syntactic and a resolvent presentation is used as the parameter of *BSM* mutation rules. This leads to a *BSM* procedure providing a unification algorithm for forward-closed theories, detailed in [10].

In this paper, we are also interested in solving the unification problem in the union of a forward-closed theory R_1 and a non-disjoint theory E_2. For this more general problem we develop a new and simplified mutation-based unification algorithm called BSM'. The new algorithm simplifies conflicts and therefore we need only a single mutation rule and thus a simpler mutation algorithm overall. These changes in turn allow for simpler correctness proofs as there are fewer cases to check. The single mutation rule, called **MutConflict** in Figure 1, aims at applying rewrite rules in R instead of equalities in the resolvent presentation. This restriction to R is sufficient if there is no equation between two non-variable terms. This form of equations can be easily avoided by splitting such equation $s = t$ into two equations $x = s$ and $x = t$ involving a common fresh variable x. Thanks to this additional transformation called **Split** in Figure 1, the classical decomposition rule of syntactic unification is superfluous.

All the BSM' rules are given in Figure 1. Let $\mathbf{B}'$ be the subset of BSM' that consists of rules with boxed terms, i.e., **Imit**, **MutConflict** and **ImitCycle**. BSM' rules are applied according to the following order of priority (from higher to lower): **Coalesce**, **Split** and $\mathbf{B}'$, where all $\mathbf{B}'$ rules are applied in a non-deterministic way (using a "don't know" non-determinism). The BSM' unification procedure consists in applying repeatedly the BSM' rules until reaching normal forms. The procedure then only returns those sets of equations which are in dag solved form. The BSM' unification can be used as an equivalent alternative to *BSM*. Compared to *BSM*, the BSM' alternative has the advantage of being easily combinable as shown in Section 6.

Theorem 4. *If R is a forward-closed convergent TRS, then the BSM' unification procedure provides an R-unification algorithm.*

Theorem 4 is subsumed by Theorem 11 that will be presented in Section 6.

Coalesce $\{x = y\} \cup G \vdash \{x = y\} \cup (G\{x \mapsto y\})$
where x and y are distinct variables occurring both in G.

Split $\{f(\bar{v}) = t\} \cup G \vdash \{x = f(\bar{v}), x = t\} \cup G$
where t is a non-variable term and x is a fresh variable.

Imit $\bigcup_i \{x = f(\bar{v}_i)\} \cup G \vdash \{x = \boxed{f(\bar{y})}\} \cup \bigcup_i \{\bar{y} = \bar{v}_i\} \cup G$
where $i > 1$, $\bar{y}$ are fresh variables and there are no more equations $x = f(\dots)$ in G.

MutConflict $\{x = f(\bar{v})\} \cup G \vdash \{x = \boxed{t}, \boxed{\bar{s}} = \bar{v}\} \cup G$
where a fresh instance $f(\bar{s}) \to t \in R$, $f(\bar{v})$ is unboxed, and (there is another equation $x = u$ in G with a non-variable term u or $x = f(\bar{v})$ occurs in a cycle).

ImitCycle $\{x = f(\bar{v})\} \cup G \vdash \{x = \boxed{f(\bar{y})}, \bar{y} = \bar{v}\} \cup G$
where $f(\bar{v})$ is unboxed, $\bar{y}$ are fresh variables and $x = f(\bar{v})$ occurs in a cycle.

Fig. 1. BSM' rules

5 Forward-Closed Combination

Along the lines of hierarchical combination [12], we study a form of non-disjoint combination defined as a convergent TRS R_1 combined with a base theory E_2. The TRS R_1 must satisfy some properties to ensure that $E = R_1 \cup E_2$ is a conservative extension of E_2. We focus here on cases where it is possible to reduce the E-equality between two terms into the E_2-equality of their R_1-normal forms. In addition, we assume that R_1 is forward-closed. The following definition clearly introduces the forward-closed combinations studied in the rest of the paper.

Definition 5. *Let Σ_1 and Σ_2 be two disjoint signatures. A* forward-closed combination *(FC-combination, for short) is a pair (E_1, E_2) such that*

- *E_1 is an equational $\Sigma_1 \cup \Sigma_2$-theory given by a forward-closed convergent TRS R_1 whose left-hand sides are Σ_1-terms;*
- *E_2 is a regular and collapse-free equational Σ_2-theory;*
- *for any terms s, t, we have (i) $s =_{E_1 \cup E_2} t$ iff $s \downarrow_{R_1} =_{E_2} t \downarrow_{R_1}$, and (ii) if $s =_{E_2} t$ then s is R_1-reducible iff t is R_1-reducible.*

Let us discuss the ingredients of the above definition. First of all, it is important to note that Σ_1 and Σ_2 are disjoint signatures. Thus, the TRS is a standard rewrite system defined on the signature $\Sigma_1 \cup \Sigma_2$ where Σ_2-symbols can occur only in right-hand sides. For this TRS, we do not have to rely on the notions of E_2-confluence and E_2-coherence introduced for class rewrite systems [15].

Proposition 6. *Assume Σ_1, Σ_2 and E_2 are given as in Definition 5. If E_1 is an equational $\Sigma_1 \cup \Sigma_2$-theory given by a forward-closed convergent TRS whose left-hand sides are linear Σ_1-terms, then (E_1, E_2) is an FC-combination.*

Example 7. Consider R_1 as any TRS mentioned in Example 2 and $\Sigma_2 = \{*\}$. An FC-combination is defined by R_1 together with any regular and collapse-free Σ_2-theory E_2, such as C or AC.

From now on, we assume $E = E_1 \cup E_2$ and (E_1, E_2) is an FC-combination given by a forward-closed convergent TRS R_1.

6 Unification in Forward-Closed Combinations

We now study how the BSM' unification procedure can be combined with an E_2-unification algorithm to solve the unification problem in $E = E_1 \cup E_2$.

VA $\{s = t[u]\} \cup G \vdash \{s = t[x], x = u\} \cup G$
where u is an alien subterm of t, x is a fresh variable, and u is boxed iff $t[u]$ is boxed.

Solve $G_1 \wedge G_2 \vdash \bigvee_{\sigma_2 \in CSU_{E_2}(G_2)} G_1 \wedge \hat{\sigma}_2$
if $G_1 \wedge G_2$ is a separate form, G_2 is E_2-unifiable and not in tree solved form, where w.l.o.g $\forall \sigma_2 \in CSU_{E_2}(G_2)\ \forall x \in Dom(\sigma_2)$, ($x\sigma_2$ is a variable) $\Rightarrow x\sigma_2 \in Var(G_2)$.

Fig. 2. Additional rules for the combination with E_2

Consider the inference system for Basic Syntactic Combination, say BSC, given by **Coalesce**, **Split** and $\mathbf{B}'$ rules defining BSM' in Section 4, where f is now supposed to be a function symbol in Σ_1; plus the two additional rules given in Figure 2, namely **VA** and **Solve**. The rule **VA** applies the classical Variable Abstraction transformation [21,3,11] to purify terms and so to get a separate form, while **Solve** calls an E_2-unification algorithm to solve the set of Σ_2-equations in a separate form. The repeated application of rules in $\{\mathbf{Coalesce}, \mathbf{Split}, \mathbf{VA}, \mathbf{Solve}\}$ computes particular separate forms defined as follows.

Definition 8. *A separate form $G_1 \wedge G_2$ is* mutable *if **Coalesce** does not apply on $G_1 \wedge G_2$, G_1 is a set of Σ_1-equations $x = t$ (where x is a variable), and G_2 is a set of Σ_2-equations in solved form. A* compound solved form *is a mutable separate form in dag solved form.*

BSC rules are applied according to the following order of priority (from higher to lower): **Coalesce**, **Split**, **VA**, **Solve**, and $\mathbf{B}'$ where **Solve** computes each solution of the subproblem G_2 in a separate form $G_1 \wedge G_2$ and $\mathbf{B}'$ rules are applied on a separate form $G_1 \wedge G_2$ in a non-deterministic way as in Section 4. Due to the order of priority, **Solve** applies only if **Coalesce**, **Split**, **VA** are not applicable and $\mathbf{B}'$ rules apply only if $G_1 \wedge G_2$ is a mutable separate form. Notice that any compound solved form is in normal form w.r.t BSC.

Lemma 9. *Given an E-unification problem G as input, the repeated application of BSC rules always terminates.*

Proof. To prove termination we use a complexity measure, similar to the one in [17], which decreases, according to the lexicographic ordering, with each application of a rule. This reduction is illustrated in the following table.

Rule	m	n	i_1	i_2	p	q
Imit	$\geq$	$>$				
MutConflict	$>$					
ImitCycle	$>$					
Coalesce	$\geq$	$\geq$	$\geq$	$\geq$	$>$	
VA	$\geq$	$\geq$	$>$			
Split	$\geq$	$\geq$	$\geq$	$>$		
Solve	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$>$

Where m is the number of unboxed Σ_1-symbols; n is the number of Σ_1-symbols; i_1 is the sum of sizes of impure terms; i_2 is the number of equations $f(\bar{v}) = t$ such that $f \in \Sigma_1$ and t is a non-variable term; p is the number of variables in G that are unsolved and not generated by E_2-unification; and $q \in \{0, 1\}$ such that $q = 0$ iff G is a separate form $G_1 \wedge G_2$ and G_2 is in solved form. □

Given an E-unification problem G, $\texttt{BSC}(G)$ denotes the normal forms of G w.r.t BSC, which correspond to the compound solved forms of G. Following Lemma 9, the BSC unification procedure works as follows: apply the BSC rules on a given E-unification problem G until reaching normal forms, and return all the dag solved forms in $\texttt{BSC}(G)$. Below, we show that BSC rules are applied without loss of completeness. We denote by $G \xrightarrow[BSC]{} G'$ an application of a BSC rule to a unification problem G producing a modified problem G'.

Lemma 10. *If (G, σ) is R_1-normalized, $E \models G\sigma$ and G is not a compound solved form, then there exist some G' and a substitution σ' such that $G \xrightarrow[BSC]{} G'$, (G', σ') is R_1-normalized, $E \models G'\sigma'$ and $\sigma' \leq_E^{Var(G)} \sigma$.*

Hence BSC leads to a terminating and complete E-unification procedure.

Theorem 11. *Given any FC-combination (E_1, E_2) and an E_2-unification algorithm, BSC provides an $E_1 \cup E_2$-unification algorithm.*

According to Definition 5, an FC-combination can be obtained by considering an arbitrary forward-closed convergent TRS R_1 and the empty theory E_2 over the empty signature Σ_2. In that particular case, BSC reduces to BSM' and so the fact that BSC is both terminating and correct provides a proof for Theorem 4.

Example 12. Assume f, g, a are in Σ_1 and E_2 is the C theory for $*$. Consider the separate form $G = \{x = f(y), x = z * y\}$ and the following possible cases:

- If $R_1 = \{f(v) \to a\}$, then **MutConflict** can be applied on G and we get $\{x = \boxed{a}, x = z * y\}$, which is in normal form w.r.t BSC but not in dag solved form. So, it has no solution.
- If $R_1 = \{f(v) \to v * a\}$, then **MutConflict** can be applied on G and we obtain $\{x = \boxed{y * a}, x = z * y\}$. Then **Solve** leads to the solved form $\{x = y * a, z = a\}$.
- If $R_1 = \{g(v) \to v * a\}$, then G is in normal form w.r.t BSC but not in dag solved form. So, it has no solution.

Example 13. Let $R_1 = \{exp(exp(a,x),y) \to exp(a, x*y)\}$ and let E_2 be the AC theory for $*$. Consider the problem $G = \{exp(x_1,x_2) = exp(a, x_2 * x_3)\}$ and a run of BSC on G. Applying **VA** and **Split** leads to the mutable separate form $\{z_2 = exp(x_1,x_2),\ z_2 = exp(a,z_1),\ z_1 = x_2 * x_3\}$. At this point one possibility is to apply **MutConflict** (introducing z_3, z_4) followed by **Coalesce** (replacing z_4 by x_2), leading to $\{z_2 = \boxed{exp(a, z_3 * x_2)},\ x_1 = \boxed{exp(a,z_3)},\ x_2 = z_4,\ z_2 = exp(a,z_1),\ z_1 = x_2 * x_3\}$ Then **VA** applies, leading to $\{z_2 = \boxed{exp(a,z_5)},\ x_1 = \boxed{exp(a,z_3)},\ x_2 = z_4,\ z_2 = exp(a,z_1),\ z_1 = x_2 * x_3,\ z_5 = \boxed{z_3 * x_2}\}$, By applying **Imit** (introducing z_6, z_7) followed by **Coalesce** (replacing z_5, z_7 by z_1), we obtain $\{z_2 = \boxed{exp(z_6,z_1)},\ z_6 = a,\ z_7 = z_5,\ z_7 = z_1,\ x_1 = \boxed{exp(a,z_3)},\ x_2 = z_4,\ z_1 = x_2 * x_3,\ z_1 = \boxed{z_3 * x_2}\}$ At this point an AC-unification algorithm can be used to solve $\{z_1 = x_2 * x_3,\ z_1 = z_3 * x_2\}$. The AC-unifier $\{z_3 \mapsto x_3\}$ leads to an expected solution of G, which is $\{x_1 \mapsto exp(a, x_3)\}$.

To complete the family picture on unification in FC-combinations, it is also possible to develop brute force methods relying on a reduction to general E_2-unification. Unsurprisingly, a possible method is based on the computation of variants. Another variant-free method consists in the non-deterministic application of some mutation/imitation rules. These two methods are described in [10].

7 Implementation

When choosing to implement the algorithms developed in this paper, we have selected the Maude programming language[1]. Maude provides a nice environment for a number of reasons. First, it provides a more natural environment for expressing the rules of algorithms such as BSM'. Second, it has both variant generation and several unification algorithms, such as AC, built-in. Indeed, having both the variant-based unification and the rule-based unification developed here implemented in Maude is the best way to compare them in practice. In addition, having both approaches implemented offers alternatives for selecting the most suitable method for an application (for example, in cases when the number of variants is high). One can now easily switch between the most appropriate approach for their situation.

Implementation of the above procedures is ongoing[2]. Currently, the focus of the implementation is on the BSM' algorithm which in itself provides a new alternative method for solving the unification problem in forward closed theories. Significantly, once the forward closure of a system is computed, the implementation of BSM' provides a unification procedure for any problem in the theory. In other words, the computation of a forward closure can be reused for any unification problem for that theory. The implementation also takes advantage of the flexibility of Maude, allowing the rules of the BSM' procedure to be

[1] `http://maude.cs.illinois.edu/w/index.php/The_Maude_System`
[2] `https://github.com/ajayeeralla/BSM`

instantiated by a theory input to the algorithm via a Maude-module. This will also make the program easier to incorporate into a larger tools.

After the BSM' implementation the focus will be on the combination and experimentation. Due to the importance of AC in practical applications, we plan to focus on the case of forward-closed combinations with AC-symbols, for which it is possible to reuse the AC-unification algorithm implemented in Maude in a way similar to [11].

8 Conclusion

In this paper we develop a rule-based unification algorithm which can be easily combined, even for some non-disjoint unions of theories, and does not require the computation of variants. By applying this rule-based unification algorithm, we present, in addition, a new non-disjoint, terminating combination procedure for a base theory extended with a non-disjoint forward-closed TRS. The new combination allows for the addition of such often used theories as AC and C.

Until now, we assume that the TRS is defined in a simple way, by using syntactic matching for the rule application. A possible extension would be to consider an equational TRS defined modulo the base theory, where equational matching is required for the rule application. Considering an equational TRS instead of a classical one, two natural problems arise. First, the possible equivalence between the finite forward closure and the FVP is an open problem when the TRS is equational. Second, another problem is to highlight a combination algorithm for solving unification problems modulo an equational TRS having the FVP.

References

1. Abadi, M., Cortier, V.: Deciding knowledge in security protocols under equational theories. Theor. Comput. Sci. **367**(1-2), 2–32 (2006)
2. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, New York, NY, USA (1998)
3. Baader, F., Schulz, K.U.: Unification in the union of disjoint equational theories: Combining decision procedures. Journal of Symbolic Computation **21**(2), 211 – 243 (1996)
4. Basin, D.A., Mödersheim, S., Viganò, L.: An on-the-fly model-checker for security protocol analysis. In: Snekkenes, E., Gollmann, D. (eds.) Computer Security - ESORICS 2003, 8th European Symposium on Research in Computer Security, Gjøvik, Norway, October 13-15, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2808, pp. 253–270. Springer (2003)
5. Blanchet, B.: Modeling and verifying security protocols with the Applied Pi calculus and proverif. Foundations and Trends in Privacy and Security **1**(1-2), 1–135 (2016)
6. Bouchard, C., Gero, K.A., Lynch, C., Narendran, P.: On forward closure and the finite variant property. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8152, pp. 327–342. Springer (2013)

7. Ciobâcă, S., Delaune, S., Kremer, S.: Computing knowledge in security protocols under convergent equational theories. J. Autom. Reasoning **48**(2), 219–262 (2012)
8. Comon-Lundh, H., Delaune, S.: The finite variant property: How to get rid of some algebraic properties. In: Giesl, J. (ed.) Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 3467, pp. 294–307. Springer (2005)
9. Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: Built-in variant generation and unification, and their applications in Maude 2.7. In: Olivetti, N., Tiwari, A. (eds.) Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9706, pp. 183–192. Springer (2016)
10. Eeralla, A.K., Erbatur, S., Marshall, A.M., Ringeissen, C.: Unification in non-disjoint combinations with forward-closed theories, available at `http://hal.inria.fr`
11. Eeralla, A.K., Lynch, C.: Bounded ACh Unification. CoRR **abs/1811.05602** (2018), `http://arxiv.org/abs/1811.05602`
12. Erbatur, S., Kapur, D., Marshall, A.M., Narendran, P., Ringeissen, C.: Hierarchical combination. In: Bonacina, M.P. (ed.) Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7898, pp. 249–266. Springer (2013)
13. Escobar, S., Meadows, C.A., Meseguer, J.: Maude-NPA: Cryptographic protocol analysis modulo equational properties. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures. Lecture Notes in Computer Science, vol. 5705, pp. 1–50. Springer (2007)
14. Escobar, S., Sasse, R., Meseguer, J.: Folding variant narrowing and optimal variant termination. J. Log. Algebr. Program. **81**(7-8), 898–928 (2012)
15. Jouannaud, J., Kirchner, H.: Completion of a set of rules modulo a set of equations. SIAM J. Comput. **15**(4), 1155–1194 (1986). https://doi.org/10.1137/0215084, `http://dx.doi.org/10.1137/0215084`
16. Kirchner, C., Klay, F.: Syntactic theories and unification. In: Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on Logic in Computer Science. pp. 270–277 (Jun 1990). https://doi.org/10.1109/LICS.1990.113753
17. Lynch, C., Morawska, B.: Basic syntactic mutation. In: Voronkov, A. (ed.) Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2392, pp. 471–485. Springer (2002)
18. Meier, S., Schmidt, B., Cremers, C., Basin, D.A.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 696–701. Springer (2013)
19. Meseguer, J.: Variant-based satisfiability in initial algebras. Sci. Comput. Program. **154**, 3–41 (2018)
20. Nipkow, T.: Proof transformations for equational theories. In: Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on Logic in Computer Science. pp. 278–288 (Jun 1990)
21. Schmidt-Schauß, M.: Unification in a combination of arbitrary disjoint equational theories. Journal of Symbolic Computation **8**, 51–99 (July 1989)