Asymmetric Unification: A New Unification Paradigm for Cryptographic Protocol Analysis*

Serdar Erbatur[1], Santiago Escobar[2], Deepak Kapur[3], Zhiqiang Liu[4], Christopher Lynch[4], Catherine Meadows[5], José Meseguer[6], Paliath Narendran[1], Sonia Santiago[2], and Ralf Sasse[7]

[1] University at Albany-SUNY, Albany, NY, USA
se@cs.albany.edu, dran@cs.albany.edu
[2] DSIC-ELP, Universitat Politècnica de València, Spain
sescobar@dsic.upv.es,ssantiago@dsic.upv.es
[3] University of New Mexico, Albuquerque, NM, USA
kapur@cs.unm.edu
[4] Clarkson University, Potsdam, NY, USA
liuzh@clarkson.edu, clynch@clarkson.edu
[5] Naval Research Laboratory, Washington DC, USA
meadows@itd.nrl.navy.mil
[6] University of Illinois at Urbana-Champaign, USA
meseguer@illinois.edu
[7] Institute of Information Security, ETH Zurich, Switzerland
ralf.sasse@inf.ethz.ch

Abstract. We present a new paradigm for unification arising out of a technique commonly used in cryptographic protocol analysis tools that employ unification modulo equational theories. This paradigm relies on: (i) a decomposition of an equational theory into (R, E) where R is confluent, terminating, and coherent modulo E, and (ii) on reducing unification problems to a set of problems $s =^? t$ under the constraint that t remains R/E-*irreducible*. We call this unification method *asymmetric unification* because of the asymmetric irreducibility constraint. We first present the generic asymmetric unification, and then outline an approach for converting conventional unification algorithms to asymmetric ones, demonstrating it for *exclusive-or* with uninterpreted function symbols. We demonstrate how asymmetric unification can improve the performance of cryptographic protocol analysis tools by running the algorithm on a set of benchmark problems. We also give results on the complexity and decidability of asymmetric unification.

1 Introduction

The symbolic analysis of cryptographic protocols has been one of the most successful applications of model-checking to security. In such an analysis, messages

* Portions of this paper appeared in a 6-page abstract of the same title [7] in the informal proceedings of UNIF'11.

are symbolic terms constructed out of function symbols and variables. Message terms often satisfy some equational properties: e.g. that decryption with a key cancels out encryption with the same key or that a symbol satisfies exclusive-or properties. Also, the network is assumed to be under the control of a hostile intruder who can read and modify all traffic, perform any operation available to a legitimate principal, and may be in league with a set of corrupted principals, and thus have access to their keys.

Protocol execution paths are usually computed by unifying messages received with messages sent. Since equational properties are usually involved, the unification must be *modulo* the equational theory describing those properties. The following strategy to achieve unification in protocol analysis, which we call *variant-based unification,* is used in one form or another by many cryptographic protocol analysis tools, including ProVerif [3], OFMC [2], Maude-NPA [6] and Tamarin [14] (see [6] for a detailed comparison). The equational theory is decomposed into (R, E), where R is a set of sort-decreasing rewrite rules that are confluent, terminating, and coherent modulo E. Given two terms m_1 and m_2 to be unified, complete sets of *irreducible variants* of m_1 and m_2 with respect to (R, E) are computed,[8] and each irreducible variant of m_1 is E-unified with each irreducible variant of m_2. Any unifier that results in either side of the equation being reducible using R modulo E is discarded as redundant. If the complete set of irreducible variants is guaranteed to be finite (that is, (R, E) has the *finite variant property* [4]), this gives a finitary unification procedure [8].

Example 1. Let us consider the following equational theory (Σ, E, R) for the exclusive-or theory, where R consists of the following equations oriented into rules,[9] and E contains the associativity and commutativity (AC) axioms for $\oplus$:

$$X \oplus 0 = X \qquad X \oplus X = 0 \qquad X \oplus X \oplus Y = Y$$

For term $t = M \oplus M$, $(0, id)$ is the only variant. For term $s = X \oplus Y$, the set of its most general variants is

$$\begin{array}{ll}\{\,(X \oplus Y, id), & \\ \;(Z, \{X \mapsto 0, Y \mapsto Z\}), & (Z, \{X \mapsto Z, Y \mapsto 0\}), \\ \;(Z, \{X \mapsto Z \oplus U, Y \mapsto U\}), & (Z, \{X \mapsto U, Y \mapsto Z \oplus U\}), \\ \;(0, \{X \mapsto U, Y \mapsto U\}), & (Z_1 \oplus Z_2, \{X \mapsto U \oplus Z_1, Y \mapsto U \oplus Z_2\})\}\end{array}$$

since any possible variant of s is an instance of one of the terms according to the substitution. For term $u = X \oplus n(A, r)$, the set of its most general variants is

$$\{(X \oplus n(A, r), id), (Z, \{X \mapsto n(A, r) \oplus Z\}), (0, \{X \mapsto n(A, r)\})\}.$$

[8] A set V of R/E-irreducible variants is a *complete set of variants* of term t with respect to (R, E) iff for any substitution θ there is a $(u, \rho) \in V$ such that the R/E-canonical form $t\theta{\downarrow_{R/E}}$ of $t\theta$ satisfies: $t\theta{\downarrow_{R/E}} =_E u\rho$ (more in Section 2).

[9] Note that the first two equations are not AC-coherent, but adding the third equation (with variable Y) is sufficient to recover that property (see [17, 5]).

Now, given the unification problem $Y \oplus n(B, r') = X \oplus n(A, r)$ arising in [6] for a simple protocol, the set of irreducible variants for each side is similar to the variants shown above for term u and the pairwise AC-unification of them gives the following substitutions as solutions to the unification problem:

$$
\begin{array}{ll}
\{X \mapsto n(B, r') \oplus Z, Y \mapsto n(A, r) \oplus Z\} & \\
\{X \mapsto n(A, r) \oplus Y \oplus n(B, r')\} & \{Y \mapsto n(B, r') \oplus X \oplus n(A, r)\} \\
\{X \mapsto n(A, r), Y \mapsto n(B, r')\} & \{X \mapsto n(A, r) \oplus Z, Y \mapsto n(B, r') \oplus Z\}
\end{array}
$$

However, there is only one most general unifier for the exclusive-or theory, $\{X \mapsto n(A, r) \oplus Y \oplus n(B, r')\}$.

The use of variant-based unification is motivated by two key features. First, it is *theory-generic* and can be applied to many of the theories and combinations of theories that arise in cryptographic protocol analysis. Second, it makes possible many *state space reduction techniques* common in cryptographic protocol analysis tools that require messages to be in *irreducible form*. This is the case, for example, when states in which certain subterm patterns appear are discarded. For example, Maude-NPA discards as unreachable any state in which the intruder learns a term containing a nonce before that nonce is generated. Consider a case, discussed in [6] in which the term learned is of the form $n(A, r) \oplus X$, where $\oplus$ satisfies the equational theory of exclusive-or and $n(A, r)$ is a nonce. If X is instantiated to $n(A, r)$ later in the search, the term reduces to 0, but variable X may appear in other positions so that the nonce could not have been generated, making this instantiation impossible; this is represented in our approach as an irreducibility constraint.

Such a strategy, although it has clear advantages, introduces performance costs due to the fact that the attempt to unify each pair of generated irreducible variants can lead to inefficiency, both because of the time it takes to generate all irreducible variants of both terms and because the size of the most general set of unifiers may be larger than optimal, as shown in Example 1. The latter also causes the state space to be larger than expected, since each produced unifier generally results in the creation of a new state. However, it may be possible to relax the irreducibility conditions on messages. For example, Maude-NPA only requires *received* messages to be in irreducible form. This led to the formulation in [6] of the concept of *contextual symbolic reachability analysis* in which irreducible variants, together with associated irreducibility constraints, are computed on only some of the terms appearing in a state. In [6] this was proved sound and complete with respect to *state reachability analysis achieved via equational unification*.

However, contextual symbolic reachability analysis opens up a new problem: how best to unify two terms, one of which must satisfy an irreducibility constraint. Indeed, the only instance of an asymmetric unification algorithm we could find was a modified variant-based unification, called *asymmetric variant-based unification*, which is similar to variant-based unification described above except that no variant is computed for the side with an irreducibility constraint.

Example 2. Following Example 1, for the *asymmetric* unification problem $Y \oplus n(B, r') = X \oplus n(A, r)$ where $X \oplus n(A, r)$ is irreducible, the solutions computed by asymmetric variant-based unification are:

$$\{X \mapsto n(B, r') \oplus Z, Y \mapsto n(A, r) \oplus Z\} \quad \{Y \mapsto n(B, r') \oplus X \oplus n(A, r)\}$$

However, there is only one most general asymmetric unifier for the exclusive-or theory: $\{Y \mapsto n(B, r') \oplus X \oplus n(A, r)\}$.

This problem, which we call *asymmetric unification* has, to the best of our knowledge, not been investigated before. Thus we ask the question: Is it possible to find asymmetric unification algorithms that can be used in cryptographic protocol analysis and are more efficient than asymmetric variant-based unification?

With this question in mind, we study asymmetric unification as a problem in its own right. After some preliminaries necessary to understanding the paper in Section 2, Section 3 gives a formal definition of asymmetric unification and shows its relation to variant-based unification. Section 4 outlines a general procedure for converting a symmetric algorithm to an asymmetric one, and applies it to exclusive-or with uninterpreted function symbols. In Section 5 we study the complexity and decidability of asymmetric unification, and show there are theories for which symmetric unification is decidable and asymmetric unification is undecidable. Section 6 gives some experimental results on an implementation of this algorithm for asymmetric exclusive-or in Maude-NPA, comparing its performance with the asymmetric variant-based unification, and provides evidence that variant-based unification is far from optimally efficient but theory-generic. Section 7 concludes the paper and discusses future work.

2 Preliminaries

We follow the classical notation and terminology from [16] for term rewriting, and from [13] for rewriting logic and order-sorted notions. We assume an order-sorted signature $\mathbf{\Sigma} = (S, \leq, \Sigma)$ with poset of sorts $(S, \leq)$. We also assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s} \in \mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ countably infinite. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding order-sorted term algebras. For a term t, $Var(t)$ denotes the set of variables in t. A *substitution* $\sigma \in \mathcal{S}ubst(\Sigma, \mathcal{X})$ is a sorted mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where the domain of σ is $Dom(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $Ran(\sigma)$. The identity substitution is *id*. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$. A *Σ-equation* is an unoriented pair $t = t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *equational theory* (Σ, E) is a pair with Σ an order-sorted signature and E a set of Σ-equations. The *E-subsumption* preorder $t \sqsupseteq_E t'$ (meaning that t is *more general* than t' modulo E) holds between terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$ iff there is a substitution σ such that $t\sigma =_E t'$; such a substitution σ is called an *E-match* from t' to t. For substitutions σ, ρ and a set of variables V we define $\sigma =_E \rho$ (over V) if $x\sigma =_E x\rho$ for all $x \in V$; and $\sigma \sqsupseteq_E \rho$ (over V) if there is a substitution η such that $(\sigma\eta)|_V =_E \rho|_V$. We say σ is *equivalent to* ρ if $\sigma \sqsubseteq_E \rho$ and $\rho \sqsubseteq_E \sigma$. An *E-unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_E t'\sigma$. For $\mathit{Var}(t) \cup \mathit{Var}(t') \subseteq W$, a set of substitutions $\mathit{CSU}_E^W(t = t')$ is said to be a *complete* set of unifiers for the equality $t = t'$ modulo E away from W iff: (i) each $\sigma \in \mathit{CSU}_E^W(t = t')$ is an E-unifier of $t = t'$; (ii) for any E-unifier ρ of $t = t'$ there is a $\sigma \in \mathit{CSU}_E^W(t = t')$ such that $\sigma|_W \sqsupseteq_E \rho|_W$ (i.e., there is a substitution η such that $(\sigma\eta)|_W =_E \rho|_W$); and (iii) for all $\sigma \in \mathit{CSU}_E^W(t = t')$, $\mathit{Dom}(\sigma) \subseteq (\mathit{Var}(t) \cup \mathit{Var}(t'))$ and $\mathit{Ran}(\sigma) \cap W = \emptyset$.

A *rewrite rule* is an oriented pair $l \to r$, where $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple (Σ, E, R) with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. The rewriting relation on $\mathcal{T}_\Sigma(\mathcal{X})$, written $t \to_R t'$ or $t \to_{p,R} t'$ holds between t and t' iff there exist $p \in \mathit{Pos}_\Sigma(t)$, $l \to r \in R$ and a substitution σ, such that $t|_p = l\sigma$, and $t' = t[r\sigma]_p$. The relation $\to_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $=_E; \to_R; =_E$. A relation $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \to_{p,R,E} t'$ (or just $t \to_{R,E} t'$) iff there is a non-variable position $p \in \mathit{Pos}_\Sigma(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. The transitive (resp. transitive and reflexive) closure of $\to_{R,E}$ is denoted $\to^+_{R,E}$ (resp. $\to^*_{R,E}$). A term t is called $\to_{R,E}$-irreducible (or just R, E-irreducible) if there is no term t' such that $t \to_{R,E} t'$. For $\to_{R,E}$ confluent and terminating, the irreducible version of a term t is denoted by $t{\downarrow}_{R,E}$. In order to guarantee the approximation of $\to_{R/E}$-reducibility by $\to_{R,E}$-reducibility, we require that R is a set of sort-decreasing rewrite rules that are confluent, terminating, and coherent modulo E (see [9, 17, 5]). We call (Σ, E, R) a *decomposition* of an order-sorted equational theory (Σ, G) if $G = R \uplus E$ and R and E satisfy these four conditions. Given a decomposition (Σ, E, R) of an equational theory, (t', θ) is an R, E-*variant* [8] (or just a variant) of term t iff $t\theta{\downarrow}_{R,E} =_E t'$ and $\theta{\downarrow}_{R,E} =_E \theta$. A decomposition (Σ, E, R) has the *finite variant property* [8] (also called a *finite variant decomposition*) iff for each Σ-term t, a complete set of its most general variants is finite.

3 Asymmetric Unification

We give a formal definition of asymmetric unification.

Definition 1 (Asymmetric Unification). *Given a decomposition* (Σ, E, R) *of an equational theory* $(\Sigma, E \cup R)$*, a substitution* σ *is an* asymmetric R,E-unifier *of a set* P *of* asymmetric equations $\{t_1 =_\downarrow t'_1, \ldots, t_n =_\downarrow t'_n\}$ *iff for each asymmetric equation* $t_i =_\downarrow t'_i$ *in* P*,* σ *is an* $(E \cup R)$*-unifier of the equation* $t_i = t'_i$ *and* $(t'_i{\downarrow}_{R,E})\sigma$ *is in* R,E*-normal form. A set of substitutions* Ω *is a* complete set of asymmetric R,E-unifiers *of* P *iff: (i) every member of* Ω *is an asymmetric* R,E*-unifier of* P*, and (ii) for every asymmetric* R,E*-unifier* θ *of* P *there exists a* $\sigma \in \Omega$ *such that* $\sigma \sqsupseteq_E \theta$ *(over* $\mathit{Var}(P)$*).*

In the following, we always assume that in every asymmetric equation $t =_{\downarrow} t'$, t' is in normal form; otherwise, we can always normalize t'.

Example 3. Consider the asymmetric unification problem $Y \oplus n(B, r') =_{\downarrow} X \oplus n(A, r)$ arising in [6] for a simple protocol demonstrating the usefulness of the contextual symbolic reachability analysis framework. Then, there is a most general $\oplus$-unifier $X \mapsto Y \oplus n(B, r') \oplus n(A, r)$. However, this is not an asymmetric unifier; but an equivalent $\oplus$-unifier is $Y \mapsto X \oplus n(B, r') \oplus n(A, r)$, which is the singleton most general asymmetric unifier.

For any $(E \cup R)$-unifier θ of P and substitution τ, $\theta\tau$ is also an $(E \cup R)$-unifier of P. But this is not necessarily the case for asymmetric R,E-unifiers.

Example 4. Consider Example 3 and the most general exclusive-or asymmetric unifier $Y \mapsto X \oplus n(B, r') \oplus n(A, r)$. If we apply the substitution $X \mapsto n(A, r)$ to the above unifier, the resulting substitution is no longer an asymmetric unifier of the original asymmetric unification problem.

The question now arises of how to produce such asymmetric algorithms that improve upon the generic variant-based algorithm described above. We discuss one such approach in the next section.

4 An Asymmetric Unification Algorithm for the Theory of Exclusive OR with Uninterpreted Function Symbols

There are two metrics to be considered when optimizing asymmetric unification algorithms for cryptographic protocol analysis. One of course is speed of execution. The other is the size of the most general set of unifiers. Each such unifier results in the production of a new state, so minimizing the size of this set helps to keep the size of the state space down.

One way of minimizing both execution time and mgu size is to convert a symmetric algorithm that has already been optimized for these features. In that case, we need to keep unifiers produced by the original algorithm whenever possible. We outline a general approach and illustrate it for exclusive-or of Example 1 together with uninterpreted function symbols, chosen because it is the simplest theory appearing in cryptographic protocol analysis that combines both cancellation rules and a non-trivial theory E in the decomposition (Σ, E, R).

Given a decomposition (Σ, E, R), and an asymmetric unification problem $\Gamma = \{t_1 =_{\downarrow} t'_1, \ldots, t_n =_{\downarrow} t'_n\}$, the key steps of the approach are:

1. First compute a complete finite set S of G-unifiers using a finitary unification algorithm for G. If S is empty, then there are no asymmetric unifiers.
2. For each such unifier σ from the previous step, check whether every $t'_i\sigma$ is in R,E-normal form. All such unifiers are retained also as asymmetric unifiers.
3. For a unifier σ such that some $t'_i\sigma$ is not in R,E-normal form, compute an equivalent equivalent asymmetric unifier if possible.

4. If both of the previous steps fail, this implies that σ or its equivalents cannot be asymmetric unifiers in their full generality. However, there may be some instances obtained by instantiating variables in them which are asymmetric unifiers. A complete set of instances of a given unifier is generated by suitably instantiating variables. This step may be expensive, so it is employed only as a last resort (as demonstrated in Table 4 of Section 6 using unification problems manually chosen to stress this point). For each such instance the above steps are repeated.

We explain below how steps (1)–(4) yield an asymmetric unification algorithm for exclusive or with uninterpreted symbols (*XOR*) from a symmetric one. Variables appearing in Γ are called *original variables* to distinguish them from new variables, called *support variables* by the inference rules. Variable x is said to be in *conflict* with a *simple term* s (i.e., a term that does not have $\oplus$ as its outermost symbol) if both x and s appear in some t'_i in Γ. The significance of conflicts is that a substitution of v cannot include s as a subterm, in order to ensure the irreducibility of the right side of equations in Γ.

We present the algorithm as a collection of inference rules on a triple of sets:

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma' \| \Upsilon' \| \Delta'},$$

where σ is an *XOR* unifier of Γ, Υ is a set of *constraint pairs* in which each member has the form (v, s), where a variable v is in conflict with s , and Δ is a set of disequations of the form $s \oplus t \neq^? 0$, with s and t having the same topmost uninterpreted function symbol.

A complete set of *XOR*-unifiers is first generated using an *XOR*-unification algorithm. For each *XOR* unifier σ, the algorithm starts with a triple $\sigma \| \emptyset \| \emptyset$. The algorithm may generate numerous branches, some of which lead to a dead end because either (i) no inference rule is applicable or (ii) the candidate for an *XOR* unifier violates a constraint in the second component or a disequation in the third component. Different branches can generate equivalent asymmetric unifiers or asymmetric unifiers which are instances of other asymmetric unifiers.

We use the following notation. The result of applying a substitution θ to $\Upsilon = \{(v_1, s_1), \cdots, (v_n, s_n)\}$ is $\Upsilon\theta = \{(v_i, s_i\theta \downarrow) | (v_i, s_i) \in \Upsilon\}$; we will rewrite $(v_i, t_1 \oplus \cdots \oplus t_n)$ to $(v_i, t_1), \cdots, (v_i, t_n)$. A substitution δ satisfies Υ iff δ satisfies every constraint pair in Υ, i.e., given a pair $(v, s) \in \Upsilon$, δ satisfies (v, s) iff $\delta(v) \oplus \delta(s)$ is irreducible using R, E (in this case the rules are the theory of XOR). If δ does not satisfy Υ, then δ violates Υ. Similarly, δ satisfies Δ iff δ satisfies every disequation $s \oplus t \neq 0 \in \Delta$, in other words $(s\delta \oplus t\delta)$ does not rewrite to 0.

The Inference System

All inference rules below are don't care nondeterministic rules. They are grouped as: **Splitting**, **Branching** and **Instantiation**. The algorithm runs in two phases. In the first phase, the **Splitting** and **Branching** rules are applied, attempting to generate an asymmetric *XOR* unifier equivalent to the original *XOR* unifier.

The **Splitting** rule is applied as much as possible to (i) move all toplevel original variables out of the range of an *XOR* unifier, while (ii) eliminating conflicts between original variables and subterms with which they appear in t_i's in Γ. Once it is no longer applicable, an *XOR* unifier equivalent to the original unifier is constructed such that its range only includes new variables at top levels. Then, branching rules are repeatedly applied attempting to eliminate conflicts between support variables with other variables and nonvariable subterms The **Non-Variable Branching** rule, which eliminates a conflict between a support variable and a nonvariable subterm, is repeatedly applied first. This is followed by (i) the **Auxiliary Branching** rule and (ii) the **Variable Branching** rule. The last two rules may not eliminate any conflicts; however they are helpful later during the second phase. In this first phase, if any of the branches yields an asymmetric *XOR* unifier, the algorithm terminates; it is not necessary to consider other branches as all asymmetric *XOR* unifiers from various branches are equivalent. Checking whether there is an asymmetric unifier equivalent to an *XOR* unifier is *NP-complete*, since monotone 1-in-3 SAT, an NP-complete problem, can be reduced to it.

If the first phase does not succeed in generating an equivalent asymmetric *XOR* unifier, all branches generated from the first phase must be considered in the second phase. Instantiation rules are now applied to generate instances of equivalent *XOR* unifiers. The **Decomposition Instantiation** rule generates instances of an *XOR* unifier so that the rules $x \oplus x \oplus y \mapsto y$ and $x \oplus x \mapsto 0$ are applicable, whereas the **Elimination Instantiation** rule generates instances by making support variables 0. It is possible that an *XOR* unifier generated by the **Elimination Instantiation** rule is equivalent to the original *XOR* unifier (since it may have been generated by instantiating a support variable to 0 implying that it was unnecessary to introduce that support variable).

If along a branch, a result of **Decomposition Instantiation** is not an asymmetric *XOR* unifier, the algorithm moves again to the first phase and applies **Splitting**, since some of the original variables underneath interpreted function symbols may get elevated to the top level in substitutions of original variables. **Elimination Instantiation** is repeatedly applied only after **Decomposition** cannot be applied any further. If the result is not an asymmetric *XOR* unifier, then the **Branching** rules are applied by returning to the first phase (**Splitting** is not applicable in this case.

The Splitting Rule

This rule transforms an *XOR* unifier σ into an equivalent *XOR* unifier σ' such that all the top variables in $Range(\sigma')$ are support variables.

$$\frac{[x \mapsto y \oplus S \oplus T] \cup \sigma \| \Upsilon \| \Delta}{([x \mapsto y \oplus S \oplus T] \cup \sigma) \circ \theta \| \Upsilon\theta \| \Delta\theta}$$

where $\theta = \{y \mapsto v \oplus S\}$ and v is a fresh support variable. The rule is applied only if (i) $x, y \in Vars(\Gamma)$ and (ii) $y \notin Vars(S)$.

Even though S and T can be chosen in any way, if x has a conflict at some simple term s in $S \oplus T$, then for efficiency in our implementation, we will put s into S, unless $y \in Vars(s)$. After **Splitting** there will be no top level original variables in the range of σ. So from now on, we assume that all the top variables which appear in the range of σ are support variables.

The Branching Rules

The main objective in applying the two branching rules is to try to transform an *XOR* unifier into an equivalent one without conflicts.

Non-Variable Branching. This rule considers the case that some original variable x has a conflict at some non-variable simple term s.

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta \| (\Upsilon[v'/v] \cup (v', s))\theta \| \Delta\theta \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v, s)\} \| \Delta\theta}$$

where there exists an assignment $[x \mapsto v \oplus s \oplus S] \in \sigma$ and $\theta = [v \mapsto v' \oplus s]$ with v' being a fresh support variable, under the conditions that x has a conflict at a simple nonvariable terms s in Γ where (i) $v \notin Vars(s)$ and (ii) $(v, s) \notin \Upsilon$.

Above, $\Upsilon[v'/v]$ means: replace all occurrences of the variable v in the first component of every pair in Υ by the variable v'. The first branch is used when the conflict between x and s is successfully resolved using v by introducing a new support variable v'; the second branch is used when that is not possible, thus leading to an additional constraint (v, s) implying that v and s are in conflict.

Auxiliary Branching. This rule is applied when an original variable conflict with another original variable in Γ and their substitutions in an *XOR* unifier share a common part.

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta \| (\Upsilon[v'/v] \cup (v', s))\theta \| \Delta\theta \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v, s)\} \| \Delta}$$

where $\theta = \{v \mapsto v' \oplus s\}$ with v' being a fresh support variable, and there exist two assignments $[x \mapsto v \oplus s \oplus S, y \mapsto v \oplus S']$ in σ. This rule is applied only if (i) x, y are in conflict in Γ, (ii) s is a simple non-variable term and $v \notin Vars(s)$ and (iii) $(v, s) \notin \Upsilon$.

The additional simple nonvariable term s in the substitution for x in an *XOR* unifier is used to possibly eliminate the conflict with a new variable v', which stands for the common shared part of x and y. The reader will notice that unlike the **Non-Variable Branching** rule, both branches after this rule still have conflicts in the substitutions of x and y which are in conflict in Γ. So this rule does not solve the conflict directly; it is preparing for the instantiation part.

Variable Branching. This rule is similar to the **Auxiliary Branching** rule and is applied when two original variables x and y have a conflict in Γ and share a common support variable v_1 in their substitutions in an *XOR* unifier. The key difference from the **Auxiliary Branching** rule is that instead of the substitution for x having a simple nonvariable term that is not in conflict with v_1, it has another support variable v_2. The common support variable v_1 is then split into two parts: the common part of x and y, represented by v_{12}, and the remaining parts of x and y, represented by v_1' and v_2', respectively.

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta \| \Upsilon' \theta \| \Delta\theta \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v_1, v_2)\} \| \Delta}$$

where σ includes $[x \mapsto v_1 \oplus v_2 \oplus S, y \mapsto v_1 \oplus S']$, $\theta = [v_1 \mapsto v_{12} \oplus v_1', v_2 \mapsto v_{12} \oplus v_2']$, v_{12}, v_1' and v_2' are fresh support variables, and $\Upsilon' = (\Upsilon[v_{12}/v_1][v_{12}/v_2] \cup \Upsilon[v_1'/v_1] \cup \Upsilon[v_2'/v_2] \cup \{(v_{12}, v_1'), (v_{12}, v_2'), (v_1', v_2'), (v_1', v_{12}), (v_2', v_{12}), (v_2', v_1')\}$. This rule is applied only if (i) x and y have a conflict in Γ and (ii) $(v_1, v_2) \notin \Upsilon$.

The first branch is the case when v_1 and v_2 have a common part, whereas the second branch is the case when v_1 and v_2 have nothing in common.

Instantiation Rules

The following instantiation rules are used for solving conflicts by instantiating support variables based on the equations $x + x \to 0$ and $x + 0 \to x$

Decomposition Instantiation. This rule is used to solve the case that some original variable x has a conflict with a simple nonvariable term t.

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta_1 \| \Upsilon\theta_1 \| \Delta\theta_1 \quad \bigvee \cdots \bigvee \quad \| \sigma \circ \theta_n \| \Upsilon\theta_n \| \Delta\theta_n \quad \bigvee \quad \sigma \| \Upsilon \| \Delta''}$$

where there exists an assignment $[x \mapsto s \oplus t \oplus S]$ in σ, x has a conflict with a simple nonvariable subterm s in Γ and s and t have the same topmost uninterpreted symbol; $\{\theta_1, \cdots, \theta_n\}$ is a complete set of *XOR* unifiers of $s \stackrel{?}{=} t$ and $\Delta'' = \Delta \cup \{s \oplus t \neq^? 0\}$.

Elimination Instantiation. This rule is used to solve the case that some original variable x has a conflict at some support variable v.

$$\frac{[x \mapsto v \oplus S] \cup \sigma \| \Upsilon \| \Delta}{([x \mapsto S] \cup \sigma) \circ \theta \| \Upsilon\theta \| \Delta\theta}$$

where $\theta = \{v \mapsto 0\}$, x and y are in conflict in Γ for some y. The rule is applied only if $y\sigma = v \oplus S'$ with S' having at least one subterm.

Because v maps to 0, all pairs (v, s) in Υ will be removed from Υ.

Theorem 1. *The asymmetric unification algorithm described above is sound, terminating, and complete.*

Proof. A sketch of the proof of soundness, termination, and completeness is given in Appendix A. A complete proof is given in [11].

5 Complexity and Decidability of Asymmetric Unification

It is easy to see that asymmetric R,E-unification is at least as hard as $E \cup R$-unification. However, nothing can be said about its asymmetric unifiers of a problem from its set of unifiers. The unification problem could have a nonempty set of unifiers, whereas the asymmetric unification problem need not have any asymmetric unifier. Or, the unification problem could have a single most general unifier, whereas the asymmetric unification problem has exponentially many solutions, as illustrated using the following asymmetric unification problem:

$$x_1 \oplus x_2 \oplus \ldots \oplus x_n =_{\downarrow} a_1 \oplus \ldots \oplus a_k, x_1 \oplus x_2 =_{\downarrow} x_1 \oplus x_2, \ldots x_1 \oplus x_m =_{\downarrow} x_1 \oplus x_m.$$

We show that there exist theories for which unification is decidable and asymmetric unification is undecidable. These results are obtained by using a restricted version of the Modified Post Correspondence Problem (MPCP)[10]. First, we define the theory $(\Sigma, \mathcal{R}_\mu)$ based on the MPCP version here and prove that unification modulo $\mathcal{R}_\mu$ (and hence asymmetric unification modulo $\mathcal{R}_\mu$) is undecidable by a reduction from MPCP. Moreover, matching modulo $\mathcal{R}_\mu$ is shown to be decidable and finitary. We use these facts to extend $(\Sigma, \mathcal{R}_\mu)$ to a theory for which unification is decidable but asymmetric unification is not.

Let $\Omega = \{a, b\}$, and let $P = \{(\alpha_i, \beta_i) \mid i = 1, \ldots, n\} \subseteq \Omega^+ \times \Omega^+$ be a finite set of pairs of non-empty strings over Σ. Then consider the following restricted version of the Modified Post Correspondence Problem (MPCP):

Instance: A non-empty string $\alpha \in \Omega^+$.
Question: Does there exist a sequence of indices $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k}\alpha = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$?

We construct $\mathcal{R}_\mu$ from this problem as follows. We start by defining the signature of $\mathcal{R}_\mu$ as $\Omega' = \Omega'_1 \cup \Omega'_3$ where $\Omega'_1 = \{a, b, 1, \ldots, n\}$ and $\Omega'_3 = \{f\}$. Thus Ω' has $n+2$ unary function symbols and one ternary function symbol. Additionally, we convert strings in the MPCP instance to terms as usual. For any string $w \in \Omega^*$, let $\widetilde{w}(x)$ denote the term formed by treating a and b as unary function symbols and the concatenation operator as function composition; in other words,

$$\widetilde{\lambda}(x) = x, \qquad \widetilde{au}(x) = a(\widetilde{u}(x)), \qquad \widetilde{bu}(x) = b(\widetilde{u}(x)).$$

For each pair (α_i, β_i) of the MPCP we create a rule

$$f(x,\, i(y),\, z) \to f(\widetilde{\alpha}_i(x),\, y,\, \widetilde{\beta}_i(z))$$

Let $\mathcal{R}_\mu$ be the set of all such rules, and let Σ be the set of symbols involved in creating them. This system is confluent and terminating: we observe that $\mathcal{R}_\mu$ is left-linear and has no critical pairs, hence is orthogonal. Thus the confluence of the system follows. In addition it is easy to show that $\mathcal{R}_\mu$ is terminating, since each application of rules of $\mathcal{R}_\mu$ decreases the number of occurrences of a symbol $j \in \{1, \ldots, n\}$ in a term. Finally, $(\Sigma, \emptyset, \mathcal{R}_\mu)$ is trivially sort-decreasing and coherent, since all symbols have the same sort, and E is empty.

[10] Both PCP and MPCP are known to be undecidable.

Lemma 2 *Matching modulo $\mathcal{R}_\mu$ is decidable and finitary.*

Proof. See Appendix B.

Lemma 3 *Let c be an arbitrary constant. The following unification problem has a solution if and only if the instance of the MPCP problem has a solution.*

$$f(\alpha(c), V, c) =^?_{\mathcal{R}_\mu} f(X, c, X)$$

Proof. See Appendix B.

We now extend $\mathcal{R}_\mu$ by adding a special constant $\perp$ (annihilator) such that, if it occurs in a term t, then t reduces to $\perp$. That is, we add the rules

$$a(\perp) \to \perp, \quad b(\perp) \to \perp, \quad f(x, y, \perp) \to \perp, \quad f(x, \perp, y) \to \perp,$$

$$f(\perp, x, y) \to \perp, \quad \text{and} \quad i(\perp) \to \perp, \quad i \in \{1, \ldots, n\}$$

Let $\mathcal{R}_\perp$ be the set of those new rules. Then we denote $\mathcal{R} = \mathcal{R}_\mu \cup \mathcal{R}_\perp$ the system extended by annihilator rules. Note that $\mathcal{R}$ is convergent as well.

Since equations where both sides contain variables can be trivially solved by setting the variables to $\perp$, we can show that

Theorem 4. *Unification modulo $\mathcal{R}$ is decidable.*

Proof. See Appendix B.

Theorem 5. *Asymmetric unification modulo $\mathcal{R}$ is undecidable.*

Proof. The key idea is that the problem $f(\alpha(c), V, c) =_{\downarrow\mathcal{R}}^? f(X, c, X)$ has a solution if and only if $f(\alpha(c), V, c)$ and $f(X, c, X)$ are unifiable modulo $\mathcal{R}_\mu$. The details are given in Appendix B.

6 Experiments with Unification Problems Arising in Protocol Analysis

We implemented a variant-based algorithm for XOR and an algorithm produced by applying the procedure outlined in Section 4 to the special-purpose XOR algorithm of [10] in Maude-NPA and experimentally compared their performance. We have run the experiments presented in this Section in an Intel Xeon machine with 4 cores and 24GB of memory, using Maude 2.7, which includes a built-in implementation of the variant generation.

Tables 1, 2 and 3 gather the results of unification problems from the following protocols: (i) the running protocol example of [6], referred as *ESORICS12*, (ii) the Wired Equivalent Privacy Protocol (WEPP) of [1], and (iii) the TMN protocol of [15, 12], respectively. Table 4 gathers the results of some more complex problems manually defined by the authors to stress the algorithms. Here each unification problem combines several subproblems, shown below the table. The ESORICS12, WEPP and TMN protocols were used in the experiments performed in [6], in order to compare the contextual symbolic reachability approach

Unif. Problem	T. A-V	# A-V	T. D-A	# D-A	% T.	% #
$NS_1 \oplus NS_2 =_{\downarrow} NS_3 \oplus N_A$	153	12	153	1	0	91
$NS_1 \oplus N_A =_{\downarrow} NS_2 \oplus NS_3$	137	5	121	1	11	80
$NS_1 \oplus NS_2 =_{\downarrow} NS_3 \oplus NS_4 \oplus NS_5$	286	54	116	1	59	98
$NS_1 \oplus NS_2 =_{\downarrow} NS_3 \oplus NS_4 \oplus N_A$	159	36	115	1	27	97
$NS_1 \oplus NS_2 =_{\downarrow} N_A$	127	4	114	1	10	75
$NS_1 \oplus NS_2 =_{\downarrow} null$	128	1	105	1	17	0
$NS_1 \oplus NS_2 =_{\downarrow} null \oplus NS_3$	130	7	105	1	20	85

Table 1. Unification Problems in ESORICS12 protocol.

Unif. Problem	T. A-V	# A-V	T. D-A	# D-A	% T.	% #
$M_1 \oplus M_2 =_{\downarrow} M_3 \oplus pair(V_1, M_4)$	51	12	44	1	13	91
$pair(V, rc4(V_1, kAB) \oplus ([N_A, c(N_A)]))$ $=_{\downarrow} pair(V_1, M_1)$	30	1	29	1	3	0
$M_1 \oplus M_2 =_{\downarrow} M_3 \oplus V_1$	33	12	32	1	3	91
$M_1 \oplus M_2 =_{\downarrow} M_3 \oplus ([N_1, c(N_2)])$	34	12	30	1	11	91
$M_1 \oplus M_2 =_{\downarrow} M_3 \oplus pair(V_1, pair(V_2, M_4))$	36	12	30	1	16	91

Table 2. Unification Problems in WEPP protocol.

presented in that paper with other approaches. However, the experiments presented in this Section are more focused on concrete unification problems that occur during the analysis of these protocols and the efficiency of asymmetric unification algorithms when solving them in terms of number of unifiers and execution time.

In each table the first and second columns show, respectively, the execution time (in milliseconds) and the number of unifiers obtained using the asymmetric variant-based unification algorithm. The third and fourth columns show, respectively, the execution time (in milliseconds) and the number of unifiers obtained using the special-purpose asymmetric unification algorithm for exclusive-or. Finally, the two last columns present a percentage that reflects the performance improvement of the special-purpose asymmetric unification algorithm with respect to the asymmetric variant-based algorithm in terms of execution time and number of unifiers obtained, respectively.

On the average the special-purpose asymmetric unification algorithm is about 8% faster than the variant-based one, and generates about 71% fewer unifiers. Note, however, that in many cases the improvement in the number of unifiers is more than 90%. Moreover the asymmetric variant-based unification algorithm does not provide a minimal set of unifiers, whereas the special-purpose asymmetric algorithm does in all our examples. Indeed, all the asymmetric unification problems extracted from protocols have a singleton most general asymmetric unifier, as shown in Tables 1, 2, and 3. However, as shown in Table 4, the special-purpose algorithm can sometimes be slower than the variant-based one, even when it generates a smaller most general set of asymmetric unifiers. The reason is that the post-processing step of the algorithm explained in Section 4 in which appropriate asymmetric unifiers are only instances of the computed unifiers is sometimes very expensive.

Unif. Problem	T. A-V	# A-V	T. D-A	# D-A	% T.	% #
$M_1 \oplus M_2 =_{\downarrow} M_3 \oplus M_4$	115	18	105	1	8	94
$M_1 \oplus M_2 =_{\downarrow} M_3 \oplus M_4 \oplus M_5$	5749	1	74	1	98	0
$M_1 \oplus M_2 =_{\downarrow} M_3 \oplus pair(M_4, M_5)$	71	12	71	1	0	91
$pair(M_1, M_2) =_{\downarrow} pair(M_3, M_4)$	65	1	70	1	-1	0
$M_1 \oplus M_2 =_{\downarrow} pair(M_3, M_4)$	67	4	71	1	0	91
$M_1 \oplus M_2 =_{\downarrow} null \oplus M_3$	66	7	70	1	-6	85

Table 3. Unification Problems in TMN protocol.

Unif. Problem	T. A-V	# A-V	T. D-A	# D-A	% T.	% #
$SP4 \wedge SP1 \wedge SP2$	422	4	68	3	83	25
$SP5 \wedge SP1 \wedge SP2$	408	24	131	7	67	70
$SP6 \wedge SP1 \wedge SP2$	416	100	491	15	-18	85
$SP7 \wedge SP1 \wedge SP2$	454	360	3732	31	-722	91
$SP8 \wedge SP1 \wedge SP2 \wedge SP3$	151387	3	47	1	99	66
$SP9 \wedge SP1 \wedge SP2 \wedge SP3$	153913	33	80	3	99	66
$SP10 \wedge SP1 \wedge SP2 \wedge SP3$	154137	201	157	7	99	96
$SP11 \wedge SP1 \wedge SP2 \wedge SP3$	154534	1053	349	15	99	98
$SP12 \wedge SP1 \wedge SP2 \wedge SP3$	160114	5073	829	31	99	99

Table 4. Other Unification Problems

SP1 = $M_1 \oplus M_2 =_{\downarrow} M_1 \oplus M_2$
SP2 = $M_1 \oplus M_3 =_{\downarrow} M_1 \oplus M_3$
SP3 = $M_1 \oplus M_4 =_{\downarrow} M_1 \oplus M_4$
SP4 = $M_1 \oplus M_2 \oplus M_3 =_{\downarrow} a \oplus b$
SP5 = $M_1 \oplus M_2 \oplus M_3 =_{\downarrow} a \oplus b \oplus c$
SP6 = $M_1 \oplus M_2 \oplus M_3 =_{\downarrow} a \oplus b \oplus c \oplus d$
SP7 = $M_1 \oplus M_2 \oplus M_3 =_{\downarrow} a \oplus b \oplus c \oplus d \oplus e$
SP8 = $M_1 \oplus M_2 \oplus M_3 \oplus M_4 =_{\downarrow} a$
SP9 = $M_1 \oplus M_2 \oplus M_3 \oplus M_4 =_{\downarrow} a \oplus b$
SP10 = $M_1 \oplus M_2 \oplus M_3 \oplus M_4 =_{\downarrow} a \oplus b \oplus c$
SP11 = $M_1 \oplus M_2 \oplus M_3 \oplus M_4 =_{\downarrow} a \oplus b \oplus c \oplus d$
SP12 = $M_1 \oplus M_2 \oplus M_3 \oplus M_4 =_{\downarrow} a \oplus b \oplus c \oplus d \oplus e$

7 Conclusions and Future Work

We have shown how asymmetric unification arises in a natural way when analyzing cryptographic protocols. We have investigated the complexity and decidability of the problem and shown that variant-based unification can be adapted to obtain a *theory-generic* asymmetric unification algorithm. We have also outlined an approach for converting symmetric algorithms to asymmetric ones and applied it to an exclusive-or algorithm. Our experimental results are encouraging, not only for increasing speed but for reducing the number of unifiers.

We plan to refine our procedures for converting algorithms by applying them to other theories of interest to cryptographic protocol analysis. We conjecture that our method for converting symmetric algorithms to asymmetric ones can be developed into an algorithm for certain classes of unification algorithms and will investigate this further. We will also investigate *combining* asymmetric algorithms, since combined theories are a common occurrence in cryptographic protocols.Variant-based narrowing lends itself relatively easily to such combination. Special-purpose asymmetric unification algorithms will not be as easy to combine, but we have been investigating combination techniques that take advantage of special properties of the theories of interest to cryptographic protocol analysis and plan to apply them in the asymmetric setting.

References

1. *IEEE 802.11 Local and Metropolitan Area Networks: Wireless LAN Medium Access Control (MAC) and Physical (PHY) Specifications.* 1999.
2. D. Basin, S. Mödersheim, and L. Viganò. An on-the-fly model-checker for security protocol analysis. In *In Proceedings of Esorics'03, LNCS 2808*, pp. 253–270. Springer, 2003.
3. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW*, pp. 82–96. IEEE Computer Society, 2001.
4. H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *RTA 2005*, *LNCS vol. 3467*, pp. 294–307. Springer, 2005.
5. F. Durán and J. Meseguer. A Maude coherence checker tool for conditional order-sorted rewrite theories. In *WRLA*, *LNCS vol. 6831*, pp. 86–103. Springer, 2010.
6. S. Erbatur, S. Escobar, D. Kapur, A. Liu, C. Lynch, C. Meadows, J. Meseguer, P. Narendran, S. Santiago, and R. Sasse. Effective symbolic protocol analysis via equational irreducibility conditions. In *Proc. ESORICS 2012*, *LNCS vol. 7459*, pp. 73–90. Springer, 2012.
7. S. Erbatur, S. Escobar, D. Kapur, Z. Liu, C. Lynch, C. Meadows, J. Meseguer, P. Narendran, and R. Sasse. Asymmetric unification: A new unification paradigm for cryptographic protocol analysis. In *UNIF 2011*, 2011. https://sites.google.com/a/cs.uni.wroc.pl/unif-2011/program.
8. S. Escobar, R. Sasse, and J. Meseguer. Folding variant narrowing and optimal variant termination. *J. Log. Algebr. Program.*, 81(7-8):898–928, 2012.
9. J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Comput.*, 15(4):1155–1194, 1986.
10. Z. Liu and C. Lynch. Efficient general unification for XOR with homomorphism. In *CADE 2011*, pp. 407-421, 2011.
11. Z. Liu. *Dealing Efficiently with Exclusive OR, Abelian Groups and Homomorphism in Cryptographic Protocol Analysis.* PhD thesis, Clarkson University, 2012. http://people.clarkson.edu/ clynch/papers/Dissertation_of_Zhiqiang_Liu.pdf.
12. G. Lowe and A.W.R.. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23:659–669, 1997.
13. J. Meseguer. Conditional rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.
14. B. Schmidt, S. Meier, C. J. F. Cremers, and D. A. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Proc. CSF 2012*, pp. 78–94. IEEE, 2012.
15. M. Tatebayashi, N. Matsuzaki, and D. Newman. Key distribution protocol for digital mobile communication systems. In *Proc. CRYPTO'89*, *LNCS vol. 435*, pp. 324–334. Springer, 1990.
16. TeReSe, editor. *Term Rewriting Systems.* Cambridge University Press, 2003.
17. P. Viry. Equational rules for rewriting logic. *Theor. Comp. Sci.*, 285(2):487–517, 2002.

A Sketch of Proof of Soundness, Termination, and Completeness of Asymmetric Exclusive-Or Algorithm

The soundness of the algorithm is easy to establish since we need to show that if an inference rule generates an asymmetric *XOR* unifier, then that unifier is either equivalent to an *XOR* unifier or an instance of an *XOR* unifier. Termination and completeness are nontrivial to prove; detailed proofs are given in [11]. Below we give an informal overview of the arguments.

For termination, it must be proved that the algorithm does not go into cycles as well as does not keep on introducing new variables in the first phase; the termination of the second phase is easy to establish. The intertwining of two phases also terminates if it can be proved that throughout the algorithm, only a bounded number of new variables are introduced by various rules. Only the Splitting and Branching rules introduce new variables. We thus first prove that they are applied only finitely often. We then complete the proof of the absence of cycles by proving that the Instantiation rules are applied only finitely often.

Intuitively, the number of new variables generated is bounded by (i) the number of all possible subsets of nonvariable subterms in the original problem and (ii) an original variable sharing exclusively with another original variable, two original variables, and so on. The substitution for any original variable x is an *XOR* of (i) a subset of nonvariable subterms appearing in the original problem and their instances due to the Decomposition Instantiation Rule, (ii) original variables with which x has no conflict and (ii) new variables standing for disjoint subsets of original subterms in the substitution of x different from substitutions of variables in conflict with x (much like v_{12}, the common part of x and y, and v'_1 and v'_2, the parts of x and y that are disjoint from each other in the Variable Branching rule). New variables also serve as placeholders to allow for generation of conflict-free instances of an *XOR* unifier in case that it does not have an equivalent asymmetric *XOR* unifier.

Once it is proved that the algorithm only introduces finitely many new variables (thus implying that the Splitting rule and the three Branching rules are only applied finitely many times), the proof of termination becomes easier since it only needs to be made sure that the two instantiation rules cannot be applied infinitely often. The Elimination Instantiation rule reduces the size of the triple since variables get instantiated to 0 and then simplified.

The Decomposition Instantiation rule reduces the number of simple terms in the substitutions for the original variables along the branch due to the unification of s, t in $x \mapsto s \oplus t \oplus S$ thus replacing $s \oplus t \oplus S$ by $\theta_i(S)$. For the branch in which the disequation $s \oplus t \neq^? 0$ is added, the set of instances of the original *XOR* unifier being investigated get reduced (the set of all possible instances of an *XOR* unifiers which have to be considered for investigating equivalent asymmetric *XOR* unifiers is finite since original variables only need to be instantiated by an *XOR* of a subset of finitely many nonvariable subterms, variable subterms and new variables).

The completeness proof is the most nontrivial as we need to show that no asymmetric xor unifier is dropped by the algorithm. One way to do this check is to ensure that every inference rule only prunes those instances of an *XOR* unifier which are not asymmetric.

The splitting rule does not do any pruning of instances of an *XOR* unifier; further, it is only applied to substitute an original variable.

The Non-Variable Branching rule considers two possible cases for generating an equivalent asymmetric xor unifier based on a variable v and a nonvariable substerm s appeaaring in the substitution of an original variable x that has a conflict with s: (i) v cancels s and (ii) v does not cancel s, which leads to the second branch. Since new variables are introduced, constraint sets are updated and new constraint sets are appropriately inherited. The Auxiliary Branching rule is similar to the Non-Variable Branching. The Variable Branching rule also considers two possible cases for generating an equivalent asymmetric *XOR* unifier based on resolving conflicts in the substitutions of two original variables. So no instances of an *XOR* unifier are discarded.

As stated above, in the use of the Splitting rule and the three branching rules, if a branch leads to an asymmetric *XOR* unifier, then there is no need to consider any other branches as either they do not produce an equivalent asymmetric unifier or do not generate a new asymmetric unifier.

Discarding of instances of an *XOR* unifier can take place only with the instantiation rules. The Decomposition Instantiation rule does not discard any instances of an xor unifier since the branching is done based on whether two nonvariable subterms s and t are *XOR* unifiable or not. The Elimination Instantiation rule discards instances of an *XOR* unifier by considering only the case when a new variable is made equal to 0, while not considering the case when that new variable is not equal to 0, but this is done only if no other way is possible.

B Proofs of Theorems in Section 5

We first prove the following lemma:

Lemma 6 *Let $\mathcal{R}$ be a convergent term rewriting system. If $\mathcal{R}^{-1}$ is terminating then every congruence class modulo $\mathcal{R}$ is finite.*

Proof. Assume that $[t]_{\mathcal{R}}$ is infinite and without loss of generality, t is in $\mathcal{R}$-normal form. Then there are infinitely many t' which are reducible to t modulo $\to_{\mathcal{R}}$. Thus t reduces to infinitely many t' through $\mathcal{R}^{-1}$-rewriting. However, since $\mathcal{R}^{-1}$ is terminating there is no infinite $\mathcal{R}^{-1}$-rewriting sequence starting from t. Therefore by König's Lemma, t has only finitely many $\mathcal{R}^{-1}$-successors. This leads to a contradiction. □

.

Proof of Lemma 2

Note that $\mathcal{R}_\mu^{-1}$ is terminating; hence by Lemma 6 for each term s, the congruence class $[s]_{\mathcal{R}_\mu}$ is finite. It was shown by Bürkert, Herold and Schmidt-Schauß[11] that if $\mathcal{R}$ is a theory where every congruence class is finite then the matching problem modulo $\mathcal{R}$ is decidable and is of matching type finitary. □

.

[11] H-J. Bürckert, A. Herold, and M. Schmidt-Schauß. On Equational Theories, Unification, and (Un)Decidability. *Journal of Symbolic Computation* 8(1/2): 3-49 (1989).

Proof of Lemma 3

The "if" part is straightforward: assume that $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k}\alpha = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$ for some indices $i_1,\ldots,i_k \in \{1,\ldots,n\}$. Then

$$\tau \;=\; \{X \mapsto \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}(c),\; V \mapsto i_k i_{k-1}\ldots i_1(c)\}$$

is a unifier for the unification problem. Note that we have

$$\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k}\alpha(c) \;=\; \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}(c) \quad \text{and thus}$$

$$\begin{aligned} f(\alpha(c),\tau(V),c) &\longrightarrow^*_{\mathcal{R}_\mu} f(\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k}\alpha(c),c,\beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}(c)) \\ &\equiv \;\; f(\alpha(\tau(X)),c,\tau(X)) \end{aligned}$$

Conversely, suppose θ is a solution for the above equation. Then the following necessarily holds: $\theta(f(\alpha(c), V, c)) = f(\alpha(c), \theta(V), c) \longrightarrow^!_{\mathcal{R}_\mu} f(\theta(X), c, \theta(X))$. Now a solution for the MPCP instance can be obtained from $\theta(V)$ as follows. Each rewrite step reveals an $i_j \in \{1,\ldots,n\}$ by deleting the top symbol from $\theta(V)$. Otherwise $\mathcal{R}_\mu$ does not apply to $f(\alpha(c), \theta(V), c)$ and hence we conclude that there exists no sequence of $i_1,\ldots,i_k \in \{1,\ldots,n\}$. Thus by using i_j's we form a solution to the MPCP problem. □