INSTITUT FÜR INFORMATIK

Implementing a Unification Algorithm for Protocol Analysis with XOR

Max Tuengerthal

Bericht Nr. 0609
Mai 2006

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Implementing a Unification Algorithm for Protocol Analysis with XOR

Max Tuengerthal

Bericht Nr. 0609
Mai 2006

e-mail: mtu@informatik.uni-kiel.de

Implementing a Unification Algorithm for Protocol Analysis with XOR

Max Tuengerthal
Christian-Albrechts-Universität zu Kiel
mtu@informatik.uni-kiel.de

Abstract

Unification algorithms are central components in constraint solving procedures for security protocol analysis. For the analysis of security protocols with XOR a unification algorithm for an equational theory including ACUN is required. While such an algorithm can easily be obtained using general combination methods such methods do not yield practical unification algorithms. In this work, we present a unification algorithm for an equational theory including ACUN which performs well in practice and is well-suited as a subprocedure in constraint solving procedures for security protocols with XOR. Our algorithm contains several optimizations which make use of the specific properties of the equational theories at hand. The efficiency of our implementation is demonstrated by experimental results.

1 Introduction

Many methods and tools for the fully automatic analysis of security protocols are based on a technique called constraint solving (see, e.g., [13, 8]), which as a central component involves a unification algorithm. The first methods and tools for the analysis of security protocols assumed the message space to be a free term algebra. However, this is a too idealized assumption in case the protocols employ operators involving algebraic properties, such as the exclusive or (XOR), an operator frequently used in security protocols. In [5, 9] it was shown that the security, more precisely secrecy and authentication, of protocols is still decidable w.r.t. a bounded number of sessions, even NP-complete [5], when taking algebraic properties of XOR into account. However, these results do not yield practical algorithms. A first algorithm based on constraint solving and tailored towards efficient implementation was proposed by Chevalier [4]. However, a prerequisite for this algorithm to be of practical use is a unification algorithm for a combination of the equational theory ACUN (modeling algebraic properties of XOR) and an equational theory E_{std} modeling public/private keys which works well in practice. The goal of the present work is to provide such an algorithm.

A unification algorithm for $E = E_{\text{std}} \cup \text{ACUN}$ can easily be obtained by the general combination method proposed by Baader and Schulz [1], since unification algorithms for E_{std} and ACUN exist. However, this unification algorithm would be highly non-deterministic and therefore not directly suitable for practical use. Several optimizations have been proposed. First, Baader and Schulz [1] already suggested simple optimizations. More sophisticated optimizations, called iterative and deductive method, were presented by Kepser and Richts [11], who exploit concrete properties of the theories, like collapse-freeness, to limit the non-determinism. Another combination method, along with optimizations, was proposed by Boudet [3]. However, the settings in all of these works are still quite general and their optimizations do not suffice for our purposes.

In this paper, we propose a unification algorithm for the theory E which combines unification algorithms for E_{std} and ACUN but compared to the more general combination methods mentioned above uses specific properties of the equational theories for further optimizations. Our optimizations drastically reduce the number of non-deterministic choices, in particular those for variable identification and linear orderings. This is important for reducing both the runtime of the unification algorithm and the number of unifiers in the complete set of unifiers. We emphasize that obtaining a "small" set of unifiers is essential for the efficiency of the constraint solving procedure within which the unification algorithm is used.

Outline of the Paper. In the first two sections we will give a short introduction to unification theory. Section 2 describes the notion needed to talk about syntactical unification. In Section 3 the concepts of syntactical unification are extended to the more general case of equational unification. The general combination method of Baader and Schulz is presented in Section 4. The steps in their procedure are explained and justified by examples. In Section 5 and 6 we first present the specific problem and then the main result, i.e. the unification algorithm for the given equational theories. We proof that the algorithm is correct and produces complete sets of unifiers in Section 7. Since the main goal was to implement an efficient algorithm for unification, we will also talk about the concrete implementation and give some runtime examples in Section 8. We conclude in Section 9.

2 Syntactic Unification

This Section gives a short introduction to syntactic unification. For more details about syntactic unification refer to [2].

A set of function symbols F is called a *signature*. With $T(F,V)$ we denote the *term algebra* which is generated by a signature F and a set of variables V, i.e. every variable $x \in V$ is in $T(F,V)$ and for each n-ary functional symbol $f \in F$, note that possibly $n = 0$, and terms $t_1, \ldots, t_n \in T(F,V)$ the term $f(t_1, \ldots, t_n)$ is in $T(F,V)$. We call the terms in $T(F,V)$ *F-terms*. We call a nullary functional symbol a *constant*.

A *substitution* σ is a mapping from the set of variables V to the term algebra $T(F,V)$. The identity substitution, $v \mapsto v$ for all $v \in V$, is denoted by Id. We can apply a substitution σ to a term t, denoted $t\sigma$, defined inductively on the term structure:

$$t\sigma := \begin{cases} x\sigma & \text{if } t = x \in V, \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

The *composition* of two substitutions σ and θ, denoted $\sigma\theta$, is defined by $t(\sigma\theta) = (t\sigma)\theta$ for each term $t \in T(F,V)$.

A substitution can be represented explicitly by a set of bindings of variables, i.e.

$$\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$$

represents the substitution which maps x_i to t_i for $i = 1, \ldots, n$ and which maps y to y if $y \neq x_i$ for all $i = 1, \ldots, n$.

Two substitutions σ and θ are called *equal*, denoted by $\sigma = \theta$, if $x\sigma = x\theta$ for every variable $x \in V$. A substitution σ is *more general* than another substitution θ, denoted $\sigma \leq \theta$, if there exists a substitution ν such that $\sigma\nu = \theta$. Note that the relation $\leq$ is a quasi-ordering, i.e. reflexive and transitive, the *instantiation quasi-ordering*. We call a substitution σ *idempotent*, if $x\sigma\sigma = x\sigma$ for all variables x.

Definition 1 (unification problem, unifier, unifiable). A *unification problem* is a finite set of equations

$$\Gamma = \left\{ s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n \right\}.$$

A substitution σ is a *unifier* of Γ if $s_i\sigma = t_i\sigma$ for each $i \in \{1, \ldots, n\}$. If such a σ exists Γ is called *unifiable*. The set of all unifiers of Γ is denoted by $U(\Gamma)$.

Definition 2 (most general unifier). A substitution is a *most general unifier (mgu)* of Γ if it is a unifier of Γ and it is more general than every unifier of Γ.

Note that for a unifiable unification problem Γ, a most general unifier always exists (see [2]). An mgu σ of Γ is quasi "the" solution because every other unifier θ can be obtained by applying a substitution ν to σ. In other words: If σ is an mgu of Γ, then the set of all unifiers

$$U(\Gamma) = \{\sigma\nu \mid \nu \text{ substitution}\}\,.$$

3 Equational Unification

Like for syntactic unification we will use the same notations as [2].

With equational unification we want to solve a unification problem with respect to a given equational theory. For two terms s and t we are now looking for a substitution σ such that $s\sigma$ equals $t\sigma$ in the equational theory. At first we need some definitions.

Let F be a signature and V a set of variables, like in Section 2. A set of *identities* E is a subset of $T(F,V) \times T(F,V)$. We write identities in the form $s \approx t$. Now we are prepared to define an *equational theory* $=_E$. It is induced by a set of identities E; it is the least congruence relation (i.e. reflexive, symmetric and transitive) on the term algebra $T(F,V)$ that is closed under substitution and contains E. More formal, $=_E$ is the following set:

$$\begin{aligned} =_E \;:= \bigcap\{R \mid\ & R \text{ congruence relation on } T(F,V), E \subseteq R, \\ & \text{for each subst. } \sigma\colon s \approx t \in R \Rightarrow s\sigma \approx t\sigma \in R\} \end{aligned}$$

Example 3.

(i) Commutativity of a binary functional symbol f is described by the set of identities $\mathrm{C} = \{f(x,y) \approx f(y,x)\}$, where x and y are variables.

(ii) The properties of the binary XOR operator ($\oplus$) are modeled by the ACUN (Associativity, Commutativity, existence of Unity, Nilpotence) theory

$$\begin{aligned} \mathrm{ACUN} = \{x \oplus (y \oplus z) &\approx (x \oplus y) \oplus z, && \text{(A)} \\ x \oplus y &\approx y \oplus x, && \text{(C)} \\ x \oplus 0 &\approx x, && \text{(U)} \\ x \oplus x &\approx 0\} && \text{(N)} \end{aligned}$$

Definition 4 (E-unification problem, E-unifier, E-unifiable). For a given signature F and a set of identities E an *E-unification problem* over F is a finite set of equations

$$\Gamma = \left\{ s_1 \overset{?}{=}_E t_1, \ldots, s_n \overset{?}{=}_E t_n \right\}$$

between F-terms. A substitution σ such that $s_i\sigma \stackrel{?}{=}_E t_i\sigma$, $i = 1, \ldots, n$, is called an *E-unifier* of Γ. $U_E(\Gamma)$ is the set of all E-unifiers of Γ. A unification problem Γ is called *E-unifiable* iff $U_E(\Gamma) \neq \emptyset$.

In the syntactic case the instantiation quasi-ordering on unifiers helped to determine how a solution of a unification problem can be represented, namely by an mgu. The relation can be adapted to the equational case.

Definition 5 (instantiation quasi-ordering modulo E). Let E be an equational theory. The substitution σ is *more general modulo E* than the substitution θ, $\sigma \leq_E \theta$, iff a substitution ν exists such that $x\sigma\nu =_E x\theta$ for every variable x.

The major difference between syntactic and equational unification is that we do not necessarily have a most general E-unifier in the equational case. For example, if we look at the theory C, see Example 3, which describes commutativity of f, the problem $\{f(x,y) \stackrel{?}{=}_C f(a,b)\}$, where x and y are variables and a and b are constants, has two C-unifiers: $\sigma = \{x \mapsto a, y \mapsto b\}$ and $\theta = \{x \mapsto b, y \mapsto a\}$. Because σ and θ are not comparable according to $\leq_{\mathrm{C}}$ a most general C-unifier for this problem does not exist. Instead, by a most general unifier, the set of E-unifiers can be represented by a complete set of unifiers, which we will now define.

Definition 6 ((minimal) complete set of E-unifiers). A *complete set of E-unifiers* of an E-unification problem Γ is a set C of idempotent E-unifiers of Γ such that for each $\theta \in U_E(\Gamma)$ there exists $\sigma \in C$ with $\sigma \leq_E \theta$.

Additionally we call a complete set C of E-unifiers *minimal* if two distinct elements are incomparable w.r.t. $\leq_E$, i.e if $\sigma \leq_E \theta$, $\sigma, \theta \in C$ then $\sigma = \theta$.

Note that a minimal complete set of E-unifiers need not always exist and if one does it is not necessarily finite. An E-unification problem Γ over a signature F has type *unitary*, *finitary* or *infinitary* if the minimal complete set of E-unifiers has size one, is finite or is infinite, respectively. It is of type *zero* if a minimal complete set of E-unifiers does not exist. A theory E w.r.t. the signature F has *unification type unitary* if all E-unification problems over F have type unitary. It is *finitary* if all E-unification problems over F have type unitary or finitary. It is *infinitary* if no E-unification problem over F has type zero. Otherwise E has type *zero*.

The ACUN-theory of Example 3 has type unitary over the signature $\{0, \oplus\}$, but it has type finitary over the signature $\{0, \oplus, f, a, b\}$ where a and b are constants and f is an n-ary functional symbol and n is at least 1 (see [10]). It can be seen that the ACUN-theory is at least finitary by looking at problem $\Gamma = \{f(x) \oplus f(y) \stackrel{?}{=}_{\mathrm{ACUN}} f(a) \oplus f(b)\}$. The set of all unifiers is $U_{\mathrm{ACUN}}(\Gamma) = \{\{x \mapsto a, y \mapsto b\}, \{x \mapsto b, y \mapsto a\}\}$. Since these substitutions are not comparable w.r.t. $\leq$ the set $U_{\mathrm{ACUN}}(\Gamma)$ is the minimal complete set of unifiers.

From the ACUN-theory one sees that it is important to be aware of the signature F which is used, not only the equational theory. An E-unification problem Γ is

- *elementary* if the signature F contains exactly the symbols in E,
- an E-unification problem *with constants* if all functional symbols $f \in F$ of arity larger than 0 occur in E, i.e. F contains the symbols in E and at most additional constants.
- *general* otherwise, i.e. F may contain arbitrary symbols.

We can extend these notions to the type of an equational theory in the obvious way.

As shown in [10], the ACUN-theory is unitary w.r.t. elementary unification and unification with constants. But it is finitary w.r.t. general unification.

An *E-unification algorithm* is an algorithm which takes an E-unification problem Γ and computes a finite complete set of E-unifiers. Of course, such an algorithm can only exist if the theory is unitary or finitary.

The following lemma about idempotent unifiers is needed in the proof of correctness of our algorithm in Section 7.

Lemma 7. *Suppose E is an arbitrary equational theory, Γ an E-unification problem and $\sigma \in U_E(\Gamma)$ idempotent. Then for each $\theta \in U_E(\Gamma)$, $\sigma \leq_E \theta$ iff $x\sigma\theta =_E x\theta$ for all variables x.*

Proof. The direction $\Leftarrow$ is clear. Conversely, we have $\sigma \leq_E \theta$, so, we find a substitution λ such that $x\sigma\lambda =_E x\theta$ for each variables x. For every variable x we have

$$x\theta =_E x\sigma\lambda = x\sigma\sigma\lambda =_E x\sigma\theta.$$

□

4 Combination of Unification Algorithms

For two equational theories E and E' where it is easy to find an E/E'-unification algorithm, it might be hard to find one for the union of the equational theories $E \cup E'$. This holds even if the theories are disjoint, i.e. do not share functional symbols. The same problem occurs if we allow additional free functional symbols that are not in E or the considered signature. An E-unification algorithm cannot directly be used for this more general setting.

Baader and Schulz [1] describe a combination method, or decomposition algorithm, to solve the problems discussed above. But instead of looking at E-unification problems with constants, we have to consider a generalization for this. An E-unification problem Γ with *linear constant restrictions* is a E-unification problem with constants and additionally a linear ordering $<$ on the set of variables and constants $V \cup C$. Now, a solution of Γ is an E-unifier σ, which fulfills the following additional restriction: For each variable x and constant $c \in F$ with $x < c$ the constant c may not occur in the term $x\sigma$.

The combination method works for disjoint equational theories $E_1, \ldots, E_n$ where unification algorithms A_i, which solve E_i-unification problems with linear constant restriction, exist, i.e. A_i produces finite complete sets of E_i-unifiers which fulfill the restrictions of a linear ordering. It combines $A_1, \ldots, A_n$ to a (elementary) unification algorithm A for the combined theory $E = E_1 \cup \cdots \cup E_n$.

4.1 Motivation and Example

We want to motivate the algorithm by an example. But first we need some additional notation about terms.

Definition 8 (E-term, pure, alien subterm). Let E be a theory.

(i) A term is called an *E-term* if it is a variable or it is of the form $f(t_1, \ldots, t_n)$ with $f \in E$.

(ii) An E-term is called *pure* if it only contains symbols of E and variables.

(iii) A subterm t of an E-term s is an *alien subterm* if it is not an E-term and all proper superterms of t in s are E-terms.

Let $E_{\text{free}} = \{f(x) \approx f(x)\}$ be a free theory and ACUN the theory mentioned in Example 3. Let A_{free} and A_{ACUN} be the unification algorithms for E_{free} and ACUN, respectively, which we want to combine to solve the combined theory $E = E_{\text{free}} \cup \text{ACUN}$. Let $\Gamma = \{s_1 \stackrel{?}{=}_E t_1, \ldots, s_n \stackrel{?}{=}_E t_n\}$ be the given (elementary) E-unification problem, i.e. the terms s_i and t_i are build over symbols in E and variables.

Now, we will describe the problems that occur if we try to solve E-unification problems. The solutions to these problems will directly lead to the combination method of [1].

Problem 1 (non-pure terms): Since symbols of ACUN and E_{free} might be mixed in terms s_i and t_i it is not possible to apply the algorithms A_{free} or A_{ACUN} directly to any of these terms, since the algorithm A_{free} is not able to handle symbols of ACUN and A_{ACUN} is not able to handle symbols of E_{free}, vice versa.

Solution: We have to transform the given problem Γ into a problem Γ_1 with purified terms which is equivalent to Γ, i.e. Γ_1 has the same set of unifiers, $U_E(\Gamma) = U_E(\Gamma_1)$. To transform Γ into Γ_1 we need to apply the following procedure until it cannot be applied anymore:

If $s \stackrel{?}{=}_E t$ is an equation in Γ and the term s contains an alien subterm s_1, then replace s_1 in s by a new variable x and obtain the term s'. Replace the equation $s \stackrel{?}{=}_E t$ by $s' \stackrel{?}{=}_E t$ and add the new equation $x \stackrel{?}{=}_E s_1$.

If this is done all terms will be pure and Γ_1 is equivalent to Γ.

Example: The given E-unification problem is

$$\Gamma = \{x \stackrel{?}{=}_E f(x \oplus y)\}$$

where x and y are variables. The E_{free}-term $f(x \oplus y)$ contains the alien subterm $x \oplus y$ which will be replaced by z, according to the solution of Problem 1. So,

$$\Gamma_1 = \{x \stackrel{?}{=}_E f(z), z \stackrel{?}{=}_E x \oplus y\}.$$

Problem 2 (non-pure equations): Although all terms in Γ_1 are pure, it might contain non-pure equations, i.e. equations $s \stackrel{?}{=}_E t$ where s is an ACUN-term and t an E_{free}-term.

Solution: To purify these equations it is sufficient to replace $s \stackrel{?}{=}_E t$ by the two equations $x \stackrel{?}{=}_E s$ and $x \stackrel{?}{=}_E t$ where x is a new variable.

By splitting non-pure equations we obtain Γ_2, which is still equivalent to Γ since it is equivalent to Γ_1. Γ_2 contains only pure terms and equations.

Example: Problem 2 does not appear in Γ_1 since all equations are already pure. So,

$$\Gamma_2 = \Gamma_1 = \{x \stackrel{?}{=}_E f(z), z \stackrel{?}{=}_E x \oplus y\}.$$

Problem 3 (incompatible variable instantiations): If we would simply apply the unification algorithms A_{free} and A_{ACUN} to the E_{free} and

ACUN-equations we could run into problems of incompatible variable instantiations. A unifier for the ACUN problem might instantiate the same variable like a unifier of the free problem. These unifiers are not, or not easily, combinable.

Example: Consider problem $\Gamma_2 = \{x \stackrel{?}{=}_E f(z), z \stackrel{?}{=}_E x \oplus y\}$. A most general unifier of $x \stackrel{?}{=}_E f(z)$ is $\{x \mapsto f(z)\}$ and one of $z \stackrel{?}{=}_E x \oplus y$ is $x \mapsto y \oplus z$. Variable x gets a value in the ACUN-theory and in the free-theory. Combining them would need E-unification of $f(z)$ and $y \oplus z$ which is as difficult as the problem we originally started with.

Solution: A solution for this problem is to guess properties of an E-unifier of Γ_2. Of course, such a unifier need not exist, but if it does, each variable will remain free or will be assigned to another variable, an ACUN-term or an E_{free}-term.

A theory index, ACUN or free, is assigned to each variable, which will ensure that a variable with the label ACUN (an ACUN-*variable*) will stay free in the unifiers produced by A_{free} and vice versa. The unification algorithms A_{ACUN} and A_{free} are able to deal with arbitrary constants, so, it is easy to ensure that ACUN-variables stay free in A_{free} by treating them as constants.

Because we do not know which variable has to get which label, we have to try all possibilities. Choosing theory indices can also be seen as a partition of size two on the set of variables. So, we obtain a set of problems

$$\{(\Gamma_2, p_1), \ldots, (\Gamma_2, p_n)\}$$

where $p_1, \ldots, p_n$ are all possible partitions of size two on the set of variables.

Problem 4 (cyclic dependencies): Choosing theory indices for each variable does not prevent cyclic dependencies. If the unifier σ for one theory assigns the term s to the variable x, i.e. $x \mapsto s \in \sigma$, and the unifier θ for the other theory assigns the term t to the variable y, then it is still not possible to combine these two unifiers if t contains x and s contains y.

Example: Consider problem

$$(\Gamma_2, p) = (\{x \stackrel{?}{=}_E f(z), z \stackrel{?}{=}_E x \oplus y\}, \{\{x\}, \{y, z\}\}).$$

So, x has the index free and y and z have the index ACUN. We could get the unifiers $\{x \mapsto f(z)\}$ and $z \mapsto x \oplus y$. These unifiers are not combinable due to cyclic dependencies between x and z.

Solution: We introduce a linear ordering $<$ on the set of variables. The meaning of $x < y$ is that y must not be a subterm of the instantiation of x. This is, like in the solution of problem 3, a guessing of properties of a unifier for the given problem. We have to consider all possible linear orderings $<_1, \ldots, <_m$ on the set of variables.

The set of problems is now

$$\begin{array}{ccc} (\Gamma_2, p_1, <_1) & \cdots & (\Gamma_2, p_n, <_1) \\ \vdots & \ddots & \vdots \\ (\Gamma_2, p_1, <_m) & \cdots & (\Gamma_2, p_n, <_m) \end{array}$$

Problem 5 (unsolvability of equations): Choosing theory indices leads to another problem. Some equations might become unsolvable since two variables have to be treated as constants. But sometimes these equations would be solvable if two variables would be identified.

Example: Consider the already purified E-problem

$$\Gamma' = \{x \stackrel{?}{=}_E f(z), y \stackrel{?}{=}_E f(u), 0 \stackrel{?}{=}_E x \oplus y\},$$

where x, y, z and u are variables.

The variables x and y have to get the index free. Otherwise, unification in the free-theory would fail immediately. Then, x and y are treated as *different* constants in the equation $0 \stackrel{?}{=}_E x \oplus y$, so, unification fails in the ACUN-theory. But Γ' has a unifier, namely

$$\{x \mapsto f(u), y \mapsto f(u), z \mapsto u\}.$$

We would have found this by identifying x and y in the beginning.

Solution: To overcome the problem of unsolvability of equations it again helps to guess a property of the E-unifier. We guess the variable identification, i.e. an arbitrary partition on the set of variables. Variables in the same class are treated identically and can be replaced by a representative.

Now, we have justified all the steps of the general combination algorithm that computes a complete set of E-unifiers and is introduced in the next section.

4.2 The General Combination Method

The algorithm introduced by Baader and Schulz [1] takes an elementary $(E_1 \cup E_2)$-unification problem Γ. Then the following steps are executed.

Step 1: **variable abstraction**. Purify non-pure terms like in the solution to Problem 1. Obtain Γ_1.

Step 2: **split non-pure equations**. Split non-pure equations like in the solution to Problem 2. Obtain Γ_2

Step 3: **variable identification**. For each partition on the variables proceed with the following steps. Obtain Γ_3 by identifying variables in the same equivalence class, i.e. replacing them by a representative.

Step 4: **choose theory indices**. For each partition p of size two on the set of variables proceed with the following steps.

Step 5: **choose linear ordering**. For each linear ordering $<$ on the set of variables proceed with the following steps.

Step 6: **split system**. The given system is $(\Gamma_3, p, <)$. Γ_3 is now split into two systems $\Gamma_{4,1}$ and $\Gamma_{4,2}$. Where $\Gamma_{4,1}$ contains the equations with E_1-terms and $\Gamma_{4,2}$ the others.

Step 7: **solve systems**. The partition p is of the form $\{V_1, V_2\}$. Every variable in $\Gamma_{4,1}$ that is not in V_1 is replaced by a new constant. We obtain $\Gamma_{5,1}$ and $\Gamma_{5,2}$ alike.

The unification algorithms A_1 and A_2 are applied to $(\Gamma_{5,1}, <)$ and $(\Gamma_{5,2}, <)$, respectively. The results are two complete sets of unifiers C_1 and C_2 where each unifier respects the ordering $<$.

Step 8: **combine unifiers**. If C_1 or C_2 are empty this choice of identification, theory indices and linear ordering has failed. But if both are not empty we can combine the unifiers of C_1 with the ones of C_2 to obtain a unifier of Γ. Therefore, the constants introduced in Step 7 have to be replaced again by the original variables. Each two unifiers $\theta \in C_1$ and $\nu \in C_2$ can be combined to a substitution σ due to the ordering $<$ (see below). Adding the identification constraints of Step 3 to σ provides an $(E_1 \cup E_2)$-unifier of Γ.

In Step 8 two unifiers are combined. The next definition clarifies how this can be done.

Definition 9 (combination of unifiers). [1] Let $<$ be a linear ordering on the set of variables V, V_1 and V_2 a partition of V and σ_i a unifier of $(\Gamma_i, <, V \setminus V_i)$ $(i = 1, 2)$. Then the *combined unifier* σ of σ_1 and σ_2, denoted by $\sigma_1 \odot \sigma_2$, is defined by induction on $<$.

Let x be the least variable. If i is the index of x, i.e. $x \in V_i$, define $x\sigma := x\sigma_i$.

Now, consider variable x with index $i \in \{1, 2\}$, i.e. $x \in V_i$, and assume that $y\sigma$ is already defined for all variables y with $y < x$. Due to the linear ordering $<$ and σ_i satisfies the linear constant restriction, $y < x$ for each $y \in Var(x\sigma_i)$. So, we can define $x\sigma := x\sigma_i\sigma$.

Theorem 10. *[1] The combined $(E_1 \cup E_2)$-unifiers, produced by the combination method above, form a complete set of $(E_1 \cup E_2)$-unifiers of the unification problem Γ.*

4.3 Drawbacks of the General Approach

The major disadvantage of the general combination method is that it is highly non-deterministic. The parts of choosing a variable identification (Step 3), theory indices (Step 4) and a linear ordering (Step 5) are processed in advance and this will lead to a huge number of systems that have to be handled in Step 7. Also for very simple unification problems the number of systems may grow tremendously.

Another problem is that the systems often only differ slightly. Therefore, solving one system will very often produce a less general unifier than solving another system. This will lead to a complete set of unifiers which is far from minimal.

The problems described above explain why the general combination method in this form cannot be used practically without optimizations. Among others Baader and Schulz [1] themselves and Kepser and Richts [11] describe optimizations of the combination algorithm to make it suitable for practical use. The main idea of the optimizations of Kepser and Richts is to first make all non-deterministic decisions for one component in order to detect failures as soon as possible (iterative method) and to use constraints obtained by solving one component for reducing the number of remaining non-deterministic choices (deductive method).

Their approaches can be used for our specific equational theories, but further optimizations, which are tuned to the concrete problem at hand (see Section 5), will lead to a more efficient algorithm, which we will present in Section 6.

5 Our Specific Problem

In order to analyze security protocols employing the XOR operator with a constraint-solving decision procedure, as proposed by Y. Chevalier in [4], we need a unification algorithm that produces complete sets of unifiers for the following equational theory:

$$E = E_{\text{std}} \cup \text{ACUN}$$

where

$$\begin{aligned} E_{\text{std}} = \{ \{x\}_y &\approx \{x\}_y, && \text{(symm. encryption)} \\ \{x\}_y^p &\approx \{x\}_y^p, && \text{(public key encryption)} \\ x &\approx (x^{-1})^{-1}, && \text{(inverse, used for public keys)} \\ \langle x, y\rangle &\approx \langle x, y\rangle \} && \text{(pairing)} \end{aligned}$$

contains the free binary functional symbols $\{\cdot\}.$ and $\langle\cdot,\cdot\rangle$ and the unfree unary functional symbol $\cdot^{-1}$ (modeling a mapping between public and private keys) and possibly other free symbols of arbitrary arity, especially constants.

The other part is the ACUN-theory, which is already known from Example 3.

$$\begin{aligned} \text{ACUN} = \{ x \oplus (y \oplus z) &\approx (x \oplus y) \oplus z, && \text{(A)} \\ x \oplus y &\approx y \oplus x, && \text{(C)} \\ x \oplus 0 &\approx x, && \text{(U)} \\ x \oplus x &\approx 0 \} && \text{(N)} \end{aligned}$$

Both theories are unitary, i.e. for unifiable problems a most general unifier always exists, and it is easy to provide unification algorithms that compute an mgu for a given problem. For the ACUN-theory this is shown in [10]. Although the theories themselves are unitary their union E is not. In [10] it is even shown that general ACUN-unification is NP-complete. Theorem 10 implies that E is finitary.

The theory E_{std} is very close to a free theory. This allows many optimizations as we will see in the next section. One of the advantages of using a combination algorithm is that slight changes in one theory only effect the unification algorithm used for this theory and not the one for the other and the combination algorithm. One could possibly think of a theory E_{std} where pairing is associative, call it E'_{std}.

$$\begin{aligned} E'_{\text{std}} = \{ & \{x\}_y \approx \{x\}_y, \\ & \{x\}^p_y \approx \{x\}^p_y, \\ & x \approx (x^{-1})^{-1}, \\ & \langle x, \langle y, z\rangle\rangle \approx \langle\langle x, y\rangle, z\rangle \} \end{aligned}$$

Because E'_{std} is no longer unitary but finitary, it is sensible for future extensions to make the more general assumption that the unification algorithm for E_{std} produces a complete set of unifiers instead of an mgu.

6 The Algorithm

At first, we summarize the main optimizations of our algorithm compared to the algorithm by Baader and Schulz. Our optimizations employ specific properties of the equational theories under consideration and they reduce both the runtime and the size of complete unification sets. In Section 6.3, we give the algorithm in detail.

6.1 Overview of the Optimizations

The main optimizations can be split into four parts.

Simplified iterative and deductive method. Similar to Kepser and Richts, we employ the idea of the iterative and deductive method but apply it only once to E_{std}. That is, we first solve the E_{std}-unification problem without any constraints. If this fails, the original problem is unsolvable. Otherwise, we obtain an mgu σ_{std} used in subsequent steps to reduce the number of non-deterministic choices. Since typically the ACUN-unification problem will not yield further constraints, we postpone solving this unification problem to a later point.

Hierarchy of variable identifications. A major new optimization in our algorithm is that we do not have to iterate over all possible variable identifications. If unification for both Γ_{std} and Γ_{ACUN} succeeds for some variable identifications p and p' where p is more general than p', then the combined unifier for p is more general than the one for p'. This allows us to traverse the tree of variable identifications in a breadth-first manner and skip all less general variable identifications once we succeed in solving the problem for a more general one.

Reduce number of choices of indices. Most theory indices can be determined from σ_{std}. If a variable is instantiated by a term with a collapse-free top-symbol, then this variable has to be a constant in Γ_{ACUN}. On the other hand, if x is not instantiated by σ_{std} and if there exists no variable y with $y\sigma_{\text{std}} = x^{-1}$, then it does not matter whether x is treated as a constant in Γ_{std} or not. In fact, a non-deterministic choice of theory indices must only be made for variables x and y such that $x\sigma_{\text{std}} = y^{-1}$ and $y\sigma_{\text{std}} = y$. Of course, not both can be constants in Γ_{std}, so it suffice to choose one of them.

Reduce number of choices of linear orderings. Instead of choosing an arbitrary linear ordering on V (see Section 4.2), we first deduce (deterministically) a partial ordering $<_{po}$ from σ_{std} such that $x <_{po} y$ iff y occurs in $x\sigma_{\text{std}}$. Now, the important observation is that once we have found a solution of the ACUN-unification problem w.r.t. a linear ordering $<$ which extends $<_{po}$, we do not need to try other linear orderings.

6.2 Preliminaries

Before we can state the optimized algorithm in detail, we need some more definitions. We introduce a partial ordering on the variable identifications.

Definition 11 (partial ordering on variable identifications). The variable identification p_1, which is a partition on the set of shared variables, is called *more general*, denoted $p_1 \leq p_2$, than the variable identification p_2, iff p_2 can be obtained from p_1 by merging equivalence classes of p_1. Notice that this relation is reflexive.

Example 12. (i) For every partition p on $\{x_1, \ldots, x_n\}$ it holds

$$\{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\} \leq p \leq \{\{x_1, x_2, \ldots, x_n\}\}.$$

(ii) The partition $\{\{x\}, \{y, z\}, \{u\}\}$ is more general than $\{\{x\}, \{y, z, u\}\}$ while it is not comparable to $\{\{x\}, \{y\}, \{z, u\}\}$.

(iii) The whole relationship graph for three and four variables is given in Figure 1 and 2, respectively. The empty set $\emptyset$ refers to the most general partition and $x = y$ means the partition $\{\{x, y\}, \{z\}\}$ in the three variable case and $\{\{x, y\}, \{z\}, \{u\}\}$ in the four variable case.

Recall: An (elementary) ACUN-unification problem Γ_{ACUN} is a set of equations with terms which only contain variables and the symbol $\oplus$ or 0.

Let Γ_{ACUN} be an ACUN-unification problem, C a set of variables and $<$ a linear ordering. Then we define:

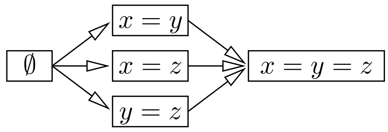

Figure 1: Variable identification graph with three variables

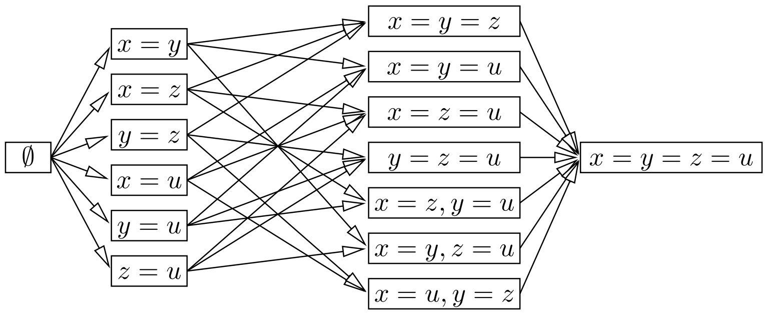

Figure 2: Variable identification graph with four variables

(a) $(\Gamma_{\text{ACUN}}, C)$ is an ACUN-unification problem with constants, where the variables in C have to be treated as constants.

(b) $(\Gamma_{\text{ACUN}}, <, C)$ is an ACUN-unification problem with linear constant restriction. The set of unifiers of $(\Gamma_{\text{ACUN}}, <, C)$ is

$$U(\Gamma_{\text{ACUN}}, <, C) = \{\sigma \in U(\Gamma_{\text{ACUN}}) \mid x\sigma = x \text{ for all } x \in C \text{ and} \\ \text{if } x < y \text{ then } y \text{ is not a subterm of } x\sigma\}$$

6.3 Detailed Description

We will describe the algorithm as a whole, in this section.

Algorithm (`unif_combi`).

Input:

- an E-unification problem $\Gamma = \{s_1 \stackrel{?}{=}_E t_1, \ldots, s_n \stackrel{?}{=}_E, t_n\}$,
- an E_{std}-unification algorithm A_{std} which computes complete sets of unifiers and
- an ACUN-unification algorithm A_{ACUN} which computes an ACUN-unifier σ for a given problem Γ_{ACUN}, a partial linear ordering $<_{po}$ and a set of variables *const* with the following property: there exists a linear ordering $<$ which extends $<_{po}$ and σ is a most general ACUN-unifier of $(\Gamma_{\text{ACUN}}, <, \mathit{const})$.

  (We give such an algorithm in Appendix A)

Output: A complete set of E-unifiers C for Γ. So, C is empty iff Γ is not E-unifiable.

Step 1: **purification and splitting**. First the problem Γ is purified. Additionally it is directly split into E_{std}-equations Γ_{std} and ACUN-equations Γ_{ACUN}.

Step 2: **solve Γ_{std} with A_{std}**. Applying A_{std} on Γ_{std} will produce a complete set of E_{std}-unifiers C_{std} for Γ_{std}.

If C_{std} is empty then return the empty set, because the unification problem Γ is not solvable.

Else, proceed with each $\sigma_{\text{std}} \in C_{\text{std}}$.

Step 3: **choose variable identification**. The algorithm starts with the most general partition on the variable set p and traverses the graph of variable identifications in a breath first manner. If all possible partitions are processed, return C.

Step 4: **add identification constraints** (std). Pick a representative x of each equivalence class in p and for each variable y in the class of x call A_{std} with $\sigma_{\text{std}} \cup \{x \stackrel{?}{=}_E y\}$ (since σ_{std} is already in solved form, this can be done very efficient). While $x \mapsto t \in \sigma_{\text{std}}$ is interpreted as the equation $x \stackrel{?}{=}_E t$. The result of A_{std} is C'_{std}.

If C'_{std} is empty, go to Step 3 and try the next variable identification. Unification in the standard theory will fail for each variable identification p' which is less general than p, so mark p' such that it is not tested.

Else, proceed with each $\sigma'_{\text{std}} \in C'_{\text{std}}$.

Step 5: **pre variable identification**. For each two shared variables x and y with $x\sigma'_{\text{std}} =_{E_{\text{std}}} y\sigma'_{\text{std}}$ merge their equivalence classes in p.

Step 6: **add identification constraints** (ACUN). Pick a representative x of each equivalence class in p and for each variable y in the class of x add the equation $0 \stackrel{?}{=}_E x \oplus y$ to Γ_{ACUN}.

Step 7: Call `process_std_unifier` with σ'_{std}, Γ_{ACUN} and A_{ACUN}.

If it returns "No Solution" go to Step 3 and try the next variable identification.

Else, let σ be the returned E-unifier and add σ to C, the set of E-unifiers for Γ. Mark the variable identifications which are less general than p, such that these are not tested any more, if `process_std_unifier` did not return "No Solution" for any $\sigma'_{\text{std}} \in C'_{\text{std}}$. They would only produce less general unifiers. This is proved in Section 7.

The following algorithm is used in Step 7 of algorithm `unif_combi`. The variable identification choice is already made and `process_std_unifier` will basically perform the choice of theory indices and linear ordering and will combine the unifiers.

Algorithm (`process_std_unifier`).

Input:
- a substitution σ_{std},
- an ACUN-unification problem Γ_{ACUN} and
- an ACUN-unification algorithm A_{ACUN} with the same requirements as in algorithm `unif_combi`.

Output: An E-unifier for $\sigma_{\text{std}} \cup \Gamma_{\text{ACUN}}$, while each $x \mapsto t \in \sigma_{\text{std}}$ is interpreted as a unification problem $x \stackrel{?}{=}_E t$, or "No Solution" if $\sigma_{\text{std}} \cup \Gamma_{\text{ACUN}}$ is not E-unifiable without identifying variables.

Step 1: **first (det.) choice of theory indices**. In this deterministic step the algorithm tries to identify variables which are limited to E_{std} or ACUN, i.e. where we in fact do not have a choice.

The set of variables V shared between σ_{std} and Γ_{ACUN} is split into three parts V_{std}, V_{ACUN} and $V_{\text{uncertain}}$. A variable $x \in V$ belongs to V_{std} if a term t exists such that t has a collapse-free top symbol (i.e. $\{\cdot\}_{\cdot}$, $\{\cdot\}^p_{\cdot}$, $\langle \cdot, \cdot \rangle$ or it is a constant) and $x\sigma_{\text{std}} =_{E_{\text{std}}} t$ or $x\sigma_{\text{std}} =_{E_{\text{std}}} t^{-1}$

The variable x belongs to $V_{\text{uncertain}}$ if a variable $y \in V$ exists such that $x\sigma_{\text{std}} =_{E_{\text{std}}} y^{-1}$ or $y\sigma_{\text{std}} =_{E_{\text{std}}} x^{-1}$.

The set of ACUN variables is the rest: $V_{\text{ACUN}} = V \setminus (V_{\text{std}} \cup V_{\text{uncertain}})$.

If the standard theory would only contain collapse-free symbols, i.e. not $\cdot^{-1}$, we would not have any uncertain variables, but could decide the index of each variable.

Step 2: **second (non-det.) choice of theory indices**. If $V_{\text{uncertain}}$ is empty we have no choices. Otherwise, we choose a theory index for each variable in $x \in V_{\text{uncertain}}$ by adding x to V_{std} or V_{ACUN}. So, afterward $V = V'_{\text{std}} \cup V'_{\text{ACUN}}$.

If we look closely at the uncertain variables, we see that these always occur in pairs and that at least one has to be in V'_{std}. An uncertain variable x is instantiated by y^{-1} and its partner is y, or vice versa. If both would belong to V'_{ACUN}, unification in the standard case would no longer be possible. Furthermore, only one variable needs to be in V'_{std} since one of them can easily be treated as a constant. So, it makes sense to collect uncertain variables in pairs. When choosing indices we only have two choices per pair (x, y), namely x is standard and y is ACUN or the other way round.

From σ_{std} one can obtain σ'_{std} which respects the choice $(V'_{\text{std}}, V'_{\text{ACUN}})$, i.e. for each variable $x \in V'_{\text{ACUN}}$ it holds $x\sigma'_{\text{std}} = x$.

Step 3: **deduce partial linear ordering**. From the substitution σ'_{std} we can easily deduce a partial linear ordering $<_{po}$ over the set of shared variables V with the following property: It holds $x <_{po} y$ iff x is a subterm of $y\sigma'_{\text{std}}$.

Step 4: **solve ACUN problem**. Try to compute an ACUN-unifier σ_{ACUN} for Γ_{ACUN} by applying the algorithm A_{ACUN} to $(\Gamma_{\text{ACUN}}, <_{po}, V'_{\text{std}})$. If this fails because Γ_{ACUN} is not unifiable according to the restrictions of $<_{po}$ and V'_{std} then we skip this system and proceed with the next choice of indices (Step 2), if there is any. If there are no more choices left, i.e. if solving the ACUN problem failed for every choice, then return "No Solution".

Step 5: **combine unifiers**. Combining σ_{ACUN} and σ'_{std} produces an E-unifier σ for Γ (see Definition 9). Return σ.

We do not have to go back to Step 2 and test the other possible choices of theory indices since this would only produce equivalent unifiers w.r.t. $\leq_E$ (see Section 7 for a proof).

7 Soundness and Completeness

In order to prove soundness and completeness, we need some more definitions and lemmas which talk about ACUN-unifiers and about the combination of ACUN- and E_{std}-unifiers. Especially, when such unifiers are more general than others and how this relation might be extended to the combination of unifiers.

Interesting results are Lemma 16 and Lemma 18. The first proves that a most general ACUN-unifier of a problem with linear constant restrictions is already a most general ACUN-unifier of the same problem without any restrictions. The latter shows that if σ_{std} and σ_{ACUN} is more general than

σ'_{std} and σ'_{ACUN}, respectively, then also the combination of σ_{std} and σ_{ACUN} is more general than the one of σ'_{std} and σ'_{ACUN}.

At first, we give a definition of a *matrix representation* of an ACUN-unification problem.

Definition 13. (a) Suppose $M \in \{0,1\}^{n \times m}$. Define M_i to be the i-th row of M.

(b) Suppose $A \in \{0,1\}^m$ and $x = (x_1, \ldots, x_m)^T$. Then $Ax := \bigoplus_{j:\, A_j = 1} x_j$.

(c) Suppose $A \in \{0,1\}^{n \times m}$, $B \in \{0,1\}^{n \times k}$, $x = (x_1, \ldots, x_m)^T$ a vector of variables and $y = (y_1, \ldots, y_k)^T$ another vector of variables. Then

$$Ax \stackrel{?}{=}_{\text{ACUN}} By := \bigcup_{i=1}^{n} \{A_i x \stackrel{?}{=}_{\text{ACUN}} B_i y\}$$

Every ACUN-unification problem with constants $(\Gamma_{\text{ACUN}}, C)$ induces two 0-1-matrices A and B and vectors of variables x and y such that $(Ax \stackrel{?}{=}_{\text{ACUN}} By, C)$ is equivalent (i.e. has the same set of unifiers) as $(\Gamma_{\text{ACUN}}, C)$ and x contains no variable of C and y contains only variables of C.

Lemma 14. *Every* ACUN-*unification problem with linear constant restriction is unitary.*

In order to proof this theorem we present an algorithm which generates a most general unifier and prove its correctness. Our algorithm is similar to the one in [10], which computes an mgu for ACUN-unification problems with constants.

Algorithm (`unify_ACUN`).

Input: an ACUN-unification problem Γ_{ACUN}, a linear ordering $<$ and a set of variables C

Output: a most general unifier of $(\Gamma_{\text{ACUN}}, <, C)$, if it is unifiable. "No Solution" otherwise.

Step 1: Compute a matrix representation A, B, x, y of Γ_{ACUN}, such that $x_{i+1} < x_i$, $y_{i+1} < y_i$, $x_i \notin C$ and $y_i \in C$ (for all i). Let n be the number of rows in A and B.

Step 2: Apply Gaussian elimination on $(A|B)$ in the field GF(2), i.e. modulo 2. This produces two matrices A' and B' such that A' is a triangular matrix and $U(Ax \stackrel{?}{=}_{\text{ACUN}} By) = U(A'x \stackrel{?}{=}_{\text{ACUN}} B'y)$.

Step 3: For each variable x set $x\sigma := x$.

Step 4:

1 **for** $i = n$ **downto** 1 **do begin**
2   **if** $(Var(A'_i x) = \emptyset$ **and** $Var(B'_i y) \neq \emptyset)$ **then**
3   return "No Solution"
4   **else if** $(Var(A'_i x) \neq \emptyset)$ **then begin**
5         $u := \max_< Var(A'_i x)$
6   **if** $(Var(B'_i y) \neq \emptyset$ **and** $u < \max_< Var(B'_i y))$ **then**
7   return "No Solution"
8   **else**
9             $u\sigma := B'_i y \oplus \bigoplus_{v \in Var(A'_i x)\setminus\{u\}} v\sigma$
10   **end**
11 **end**

Step 5: Return σ.

The following lemma states that the algorithm `unify_ACUN` is sound and complete.

Lemma 15. *(a) The algorithm* `unify_ACUN` *returns a unifier of* Γ_{ACUN} *if* $(\Gamma_{\mathrm{ACUN}}, <, C)$ *is unifiable and "No Solution" otherwise.*

(b) If the algorithm `unify_ACUN` *returns a unifier* σ*. Then* σ *is more general than every unifier of* $(\Gamma_{\mathrm{ACUN}}, <, C)$.

Proof. (a) Assume that $(\Gamma_{\mathrm{ACUN}}, <, C)$ is unifiable and let θ be an ACUN-unifier of it. It is easy to see that θ is a unifier of $Ax \stackrel{?}{=}_{\mathrm{ACUN}} By$ and of $A'x \stackrel{?}{=}_{\mathrm{ACUN}} B'y$, too. So, for each $i = 1, \ldots, n$ it holds $(A'_i x)\theta =_{\mathrm{ACUN}} (B'_i y)\theta = B'_i y$. Therefore, $Var(A'_i x) \neq \emptyset$ or $Var(B'_i y) = \emptyset$ $(i = 1, \ldots, n)$, which implies that line 3 is never reached.

Suppose $Var(A'_i x) \neq \emptyset$, $Var(B'_i y) \neq \emptyset$ and $u := \max_< Var(A'_i x) < \max_< Var(B'_i y) =: a$ (i.e. line 7 will be reached). Then $v \leq u < a$ for each $v \in Var(A'_i x)$. Since a occurs in $B'_i y$, it also has to occur in $(A'_i x)\theta$. So, there exists $v \in Var(A'_i x)$ such that a occurs in $v\theta$. This is a contradiction to $v < a$ and θ respects $<$.

We have shown that if $(\Gamma_{\mathrm{ACUN}}, <, C)$ is unifiable, `unify_ACUN` will not return "No Solution".

Let σ be the output of `unify_ACUN`$(\Gamma_{\mathrm{ACUN}}, <, C)$, if it is not "No Solution". We will show that it is a unifier of $(\Gamma_{\mathrm{ACUN}}, <, C)$.

There exist variables $u_0, \ldots, u_l$ such that $A'_i x =_{\mathrm{ACUN}} u_0 \oplus \ldots \oplus u_l$ and $u_0 = \max_< Var(A'_i x)$. Then for each $i = 1, \ldots, n$:

$$(A'_i x)\sigma =_{\mathrm{ACUN}} B'_i y \oplus \Big(\bigoplus_{v \in Var(A'_i x) \setminus \{u_0\}} v\sigma \Big) \oplus u_1\sigma \oplus u_l\sigma =_{\mathrm{ACUN}} B'_i y.$$

The first equation holds by definition of σ in the algorithm and because A' is triangular. The second, because $Var(A'_i x) \setminus \{u_0\} = \{u_1, \ldots, u_l\}$. So, σ is a unifier of $A'x \stackrel{?}{=}_{\mathrm{ACUN}} B'y$ and it is easy to prove, that it is one of Γ_{ACUN}. It is easy to see that σ respects the linear ordering $<$ and that variables in C stay unassigned. So, σ is a unifier of $(\Gamma_{\mathrm{ACUN}}, <, C)$.

(b) Let θ be a unifier of $(\Gamma_{\mathrm{ACUN}}, <, C)$. It is easy to prove that θ is a unifier of $A'x \stackrel{?}{=}_{\mathrm{ACUN}} B'y$, too. By (a), we know that $\sigma :=$ `unify_ACUN`$(\Gamma_{\mathrm{ACUN}}, <, C)$ is a unifier of $(\Gamma_{\mathrm{ACUN}}, <, C)$, too. We prove $u\sigma\theta =_{\mathrm{ACUN}} u\theta$ for each variable u by induction on $<$.

Let u be the least variable according to $<$. If $u\sigma = u$, we are finished. Otherwise, line 9 was reached at some round i with $u = \max_< Var(A'_i x)$. So, $A'_i x = u$ and $u\sigma = B'_i y$ and therefore

$$u\theta = (A'_i x)\theta =_{\mathrm{ACUN}} (B'_i y)\theta =_{\mathrm{ACUN}} ((A'_i x)\sigma)\theta = u\sigma\theta.$$

Now, consider an arbitrary variable u and assume $v\sigma\theta =_{\mathrm{ACUN}} v\theta$ for all variables $v < u$. If $u\sigma = u$, we are finished. Otherwise, line 9 was reached at some round i with $u = \max_< Var(A'_i x)$. So,

$$\begin{aligned}
(u\sigma)\theta &=_{\mathrm{ACUN}} \Big(B'_i y \oplus \bigoplus_{v \in Var(A'_i x) \setminus \{u\}} v\sigma \Big)\theta \\
&=_{\mathrm{ACUN}} (B'_i y)\theta \oplus \bigoplus_{v \in Var(A'_i x) \setminus \{u\}} v\sigma\theta \\
&\stackrel{\text{ind. hyp.}}{=_{\mathrm{ACUN}}} (B'_i y)\theta \oplus \bigoplus_{v \in Var(A'_i x) \setminus \{u\}} v\theta \\
&\stackrel{(A'_i x)\theta =_{\mathrm{ACUN}} (B'_i y)\theta}{=_{\mathrm{ACUN}}} u\theta.
\end{aligned}$$

We have proved that σ is more general than θ. $\square$

This proves Lemma 14 and additionally that `unify_ACUN` produces a most general ACUN-unifier.

Lemma 16. *Suppose σ is an mgu of $(\Gamma_{\mathrm{ACUN}}, <, C)$. Then σ is already an mgu of Γ_{ACUN}.*

Proof. At first we prove the lemma for the mgu σ of $(\Gamma_{\mathrm{ACUN}}, <, C)$ returned by algorithm `unify_ACUN`. Consider $\theta \in U_{\mathrm{ACUN}}(\Gamma_{\mathrm{ACUN}})$. We prove $u\sigma\theta =_{\mathrm{ACUN}} u\theta$ for each variable u by induction on $<$.

Let u be the least variable according to $<$. If $u\sigma = u$, we are finished. Otherwise, line 5 in Step 4 of the algorithm would be reached with some i and $u = \max_< Var(A'_i x)$. Since u is the least variable, $A'_i x = u$. Assume $Var(B'_i y) \neq \emptyset$ then line 7 would have been reached. So, $Var(B'_i y) = \emptyset$. This means $u \stackrel{?}{=}_{\mathrm{ACUN}} 0 \in A'x \stackrel{?}{=}_{\mathrm{ACUN}} B'y$. $\theta \in U(\Gamma_{\mathrm{ACUN}}) = U(Ax \stackrel{?}{=}_{\mathrm{ACUN}} By) = U(A'x \stackrel{?}{=}_{\mathrm{ACUN}} B'y)$ implies $u\theta =_{\mathrm{ACUN}} 0\theta = 0$. By line 9, $u\sigma = 0$, so $u\sigma\theta = 0\theta = 0 =_{\mathrm{ACUN}} u\theta$.

Now, consider an arbitrary variable u and assume $v\sigma\theta =_{\mathrm{ACUN}} v\theta$ for all variables $v < u$. If $u\sigma = u$, we are finished. Otherwise, line 5 in Step 4 of the algorithm would be reached with some i and $u = \max_< Var(A'_i x)$. So, $Var(A'_i x) \neq \emptyset$ and $u = \max_< Var(A'_i x)$ and ($Var(B'_i y) = \emptyset$ or $u > \max_< Var(B'_i y)$). Since $A'_i x \stackrel{?}{=}_{\mathrm{ACUN}} B'_i y \in A'x \stackrel{?}{=}_{\mathrm{ACUN}} B'y$ and

$$\theta \in U(\Gamma_{\mathrm{ACUN}}) = U(Ax \stackrel{?}{=}_{\mathrm{ACUN}} By) = U(A'x \stackrel{?}{=}_{\mathrm{ACUN}} B'y),$$

we have $(A'_i x)\theta =_{\mathrm{ACUN}} (B'_i y)\theta$ and therefore

$$\begin{aligned}
u\theta &=_{\mathrm{ACUN}} (B'_i y)\theta \oplus \bigoplus_{v \in Var(A'_i x)\setminus\{u\}} \underbrace{v\theta}_{=_{\mathrm{ACUN}}\, v\sigma\theta,\text{ by ind. hyp.}} \\
&=_{\mathrm{ACUN}} (B'_i y)\theta \oplus \bigoplus_{v \in Var(A'_i x)\setminus\{u\}} v\sigma\theta \\
&= \underbrace{\Big(B'_i y \oplus \bigoplus_{v \in Var(A'_i x)\setminus\{u\}} v\sigma\Big)}_{=_{\mathrm{ACUN}}\, u\sigma,\text{ by def. in line 9}}\theta =_{\mathrm{ACUN}} u\sigma\theta.
\end{aligned}$$

Now, we consider an arbitrary mgu σ' of $(\Gamma_{\mathrm{ACUN}}, <, C)$ and prove that $u\sigma'\theta =_{\mathrm{ACUN}} u\theta$ for each variable u. By Lemma 7 we have $v\sigma'\sigma =_{\mathrm{ACUN}} v\sigma$ for each variable v, because σ', as an mgu, is idempotent and $\sigma' \leq_{\mathrm{ACUN}} \sigma$. It follows from the above

$$u\sigma'\theta =_{\mathrm{ACUN}} u\sigma'(\sigma\theta) = (u\sigma'\sigma)\theta =_{\mathrm{ACUN}} u\sigma\theta = u\theta$$

for each variable u. □

The next lemma follows by the definition of the combination of unifiers (Definition 9), directly.

Lemma 17. *Let $<$, V_1, V_2, σ_1 and σ_2 be like before and let $\sigma = \sigma_1 \odot \sigma_2$. Then $x\sigma = x\sigma_1\sigma$ and $x\sigma = x\sigma_2\sigma$ for all variables x.*

The following lemma basically shows that if σ_1 and σ_2 is more general than σ'_1 and σ'_2, respectively, then also the combination of σ_1 and σ_2 is more general than the one of σ'_1 and σ'_2 (under certain circumstances).

Lemma 18. *Suppose V is a set of variables, E_1 and E_2 arbitrary equational theories, Γ_i an E_i-unification problem, V_1^j, V_2^j a partition of V, $<^j$ a linear ordering on V and σ_i^j an E_i-unifier of $(\Gamma_i, <^j, V \setminus V_i^j)$ $(i, j \in \{1,2\})$ such that $x\sigma_i^1\sigma_i^2 =_{E_i} x\sigma_i^2$ for all variables x. Then*

$$\sigma^1 := \sigma_1^1 \odot \sigma_2^1 \leq_{E_1 \cup E_2} \sigma_1^2 \odot \sigma_2^2 =: \sigma^2.$$

Proof. Define $E := E_1 \cup E_2$. We prove $x\sigma^1\sigma^2 =_E x\sigma^2$ for all variables x by induction on $<^1$.

Let x be minimal w.r.t. $<^1$. If $x\sigma^1 = x$, we are finished. Suppose $x \in V_i^1$ with $i \in \{0,1\}$. By definition of $\odot$ we have

$$x\sigma^1\sigma^2 =_E x\sigma_i^1\sigma^2 \overset{\text{Lemma 17}}{=_E} x\sigma_i^1\sigma_i^2\sigma^2 =_E x\sigma_i^2\sigma^2 \overset{\text{Lemma 17}}{=_E} x\sigma^2.$$

Now, consider an arbitrary variable x and assume $y\sigma^1\sigma^2 =_E y\sigma^2$ for all variables $y <^1 x$. If $x\sigma^1 = x$, we are finished. Suppose $x \in V_i^1$ with $i \in \{0,1\}$. Since σ_i^1 satisfies the linear constant restriction $<^1$, $y <^1 x$ for each $y \in Var(x\sigma_i^1)$. By definition of $\odot$ we have

$$\begin{aligned} x\sigma^1\sigma^2 &=_E x\sigma_i^1\sigma^1\sigma^2 \overset{\text{ind. hyp.}}{=_E} x\sigma_i^1\sigma^2 \overset{\text{Lemma 17}}{=_E} x\sigma_i^1\sigma_i^2\sigma^2 \\ &=_E x\sigma_i^2\sigma^2 \overset{\text{Lemma 17}}{=_E} x\sigma^2. \end{aligned}$$

□

The preconditions $\sigma_i^1\sigma_i^2 =_{E_i} \sigma_i^2$ for $i = 1, 2$ of the above theorem can be abolished if we talk about idempotent unifiers, because by Lemma 7 these preconditions are already met, if σ_i^1 $(i = 1, 2)$ are idempotent unifiers.

Lemma 18 is the key to optimize the variable identification step. It is independent from the equational theories and might be useful in other implementations of a combination method.

We have all parts together now, to prove the correctness of the algorithm.

Theorem 19. *Every combined E-unifier returned by* `unif_combi` *is an idempotent E-unifier of the original problem Γ.*

Proof. Suppose σ is the result of `unif_combi` for the given problem Γ. To generate σ, the algorithm has chosen a variable identification, theory indices and a linear ordering. These choices could be made by the general combination method (see Section 4) as well. So, σ could be returned by the general combination method and since this method produces only unifiers of Γ, σ is an E-unifier of Γ.

By definition of σ, it is easy to see that σ is idempotent. $\square$

The more interesting theorem is the following.

Theorem 20. *The set of combined E-unifier C returned by* `unif_combi` *is a complete set of E-unifiers of* Γ.

Proof. Without loss of generality, we assume that E_{std} is a unitary theory and that the E_{std}-unification algorithm computes a most general unifier, not a complete set of E_{std}-unifiers.

By the above theorem, we already have that $C \subseteq U_E(\Gamma)$. What is left to show is that for each $\theta \in U_E(\Gamma)$, we find a unifier in C which is more general than θ. Since, the general combination method is correct, it suffices to show that for each unifier θ returned by the general combination method, we find a unifier in C which is more general than θ.

Consider the unifier $\theta = \theta_{\text{std}} \odot \theta_{\text{ACUN}}$, returned by the general combination method by choosing the corresponding variable identification p, the linear ordering $<$ and theory indices V_{std} and V_{ACUN}. So, θ_{std} is a unifier of $(\Gamma_{\text{std}} \cup p, <, V_{\text{ACUN}})$ and θ_{ACUN} is a unifier of $(\Gamma_{\text{ACUN}} \cup p, <, V_{\text{std}})$. With $\Gamma \cup p$ we denote the problem Γ where the identification constraints of p are added, i.e. if x and y are identified in p, $\Gamma \cup p$ contains the equation $x \stackrel{?}{=}_E y$.

Consider two cases:

(a) Our algorithm never reached the identification p (i.e. p is never chosen in Step 3 of `unif_combi`). Then either unification of Γ_{std} failed or p was marked in Step 7 of algorithm `unif_combi`. The first is not possible, since θ_{std} is a unifier of Γ_{std}. So, the latter has to be true. If p was marked, then Step 7 was reached for some identification $p' \leq p$ and the call of `process_std_unifier` successfully returned a combined unifier $\sigma = \sigma_{\text{std}} \odot \sigma_{\text{ACUN}}$ for some choices $<'$, V'_{std} and V'_{ACUN}.

    Then, σ_{std} is an mgu of $\Gamma_{\text{std}} \cup p'$. Since, $\theta_{\text{std}} \in U_{\text{std}}(\Gamma_{\text{std}} \cup p) \subseteq U_{\text{std}}(\Gamma_{\text{std}} \cup p')$, σ_{std} is more general than θ_{std}.

    Also, σ_{ACUN} is an mgu of $(\Gamma_{\text{ACUN}} \cup p', <, V'_{\text{std}})$. By Lemma 16 it is already an mgu of $\Gamma_{\text{ACUN}} \cup p'$. Since, $\theta_{\text{ACUN}} \in U_{\text{ACUN}}(\Gamma_{\text{ACUN}} \cup p) \subseteq U_{\text{ACUN}}(\Gamma_{\text{ACUN}} \cup p')$, σ_{ACUN} is more general than θ_{ACUN}.

    Lemma 18 implies, σ is more general than θ.

(b) Our algorithm reaches the identification p. So, Step 3 of `unif_combi` is reached and p is chosen. We will show that the call of `process_std_unifier` in Step 7 will return a unifier.

Assume that `process_std_unifier` returns "No Solution". Then by definition of `process_std_unifier` all linear orderings $<'$, such that $(\Gamma_{\text{std}} \cup p, <')$ is unifiable, have been tried. Since, θ_{std} solves $(\Gamma_{\text{std}} \cup p, <, V_{\text{ACUN}})$ also $<$ has been tried, i.e. `process_std_unifier` reached Step 4 with $<$, V'_{std} and V'_{ACUN}. Where V'_{ACUN} is as large as possible (again by definition of the algorithm). That means $V_{\text{ACUN}} \subseteq V'_{\text{ACUN}}$ and therefore $V'_{\text{std}} \subseteq V_{\text{std}}$. But, solving $(\Gamma_{\text{ACUN}}, <, V'_{\text{std}})$ failed. This is a contradiction to $\theta_{\text{ACUN}} \in U_{\text{ACUN}}(\Gamma_{\text{ACUN}}, <, V_{\text{std}}) \subseteq U_{\text{ACUN}}(\Gamma_{\text{ACUN}}, <, V'_{\text{std}})$.

So, `process_std_unifier` returned a unifier $\sigma = \sigma_{\text{std}} \odot \sigma_{\text{ACUN}}$ with σ_{std} mgu of $\Gamma_{\text{std}} \cup p$ and σ_{ACUN} mgu of $(\Gamma_{\text{ACUN}} \cup p, <', V'_{\text{std}})$ for some choices $<'$, V'_{std} and V'_{ACUN}. With the same arguments as used in case (a), it follows that σ is more general than θ. $\square$

8 Implementation Details and Benchmarks

Since the main task for this work was to actually implement an algorithm for unification in the presence of the XOR operator, we will explain some details of the implementation in this section. We also give some examples to show that unification can often be done very fast but also to show that there are inherent problems in some examples.

8.1 Implementation Details

Our algorithm has been implemented in the functional programming language Objective Caml [12]. The choice of the programming language was motivated by the following points:

- A main argument is that we already had a unification algorithm for the standard theory, which was fully working and implemented in Objective Caml. So, the combination of this algorithm and the combination algorithm was very easy. We could also reuse data structures and a lot of help functions, e.g. functions to turn terms into normal form.

- In order to limit the size of the substitutions we have to store terms in a dag-structure (directed acyclic graph structure). Objective Caml supports this naturally since it works with pointers on data structures and only copies pointers, not the actual data structure.

- The functional concept includes powerful pattern matching. This makes it very easy to traverse through terms and limits the size of the code.

- A unification algorithm of the ACUN-theory is based on solving an equational system modulo 2. Since Objective Caml also supports imperative programming it was easily possible to implement a Gaussian elimination algorithm.

We want to mention some of the data structures used:

- A unifier, or a substitution, is represented by a list of pairs of variables and terms. The pair (x, t) stands for $x \mapsto t$ in the substitution.

- To represent a partial ordering we used a list of pairs of a variable and a list of variables. The meaning of $(x, [y_1, y_2, \ldots, y_n])$ in the partial ordering is that it holds $x < y_1, x < y_2, \ldots, x < y_n$. The advantage of this representation is that it is easy to obtain all successors for a variable x. In this case the successors are $\{y_1, y_2, \ldots, y_n\}$. To check if y must not be a subterm of x is then reduced to a test whether y is an element of $\{y_1, y_2, \ldots, y_n\}$ or not. A drawback is that one has to be very careful by creating the partial ordering. Every transitive dependency has to be resolved.

- A partition, like it is used for a variable identification, was simply modeled by a list of a list of variables. For example $[[x; y; z]; [u; v]; [w]]$ represents the partition $\{\{x, y, z\}, \{u, v\}, \{w\}\}$. Of course, an empty list must not exist, i.e. $[\ldots; []; \ldots]$ is not allowed.

8.2 Tests

Table 1 and 2 summarizes some of our experimental results. It contains runtimes and sizes of complete sets of unifiers both with the optimization for variable identification turned on and off. (The other optimizations are harder to turn on and off in our implementation, which is why these optimization are always turned on.) These results show that our unification algorithm runs efficiently on many benchmarks and that our optimizations indeed reduce both runtime and size of complete sets unifiers. In fact, the size of the returned set of unifiers in the optimized version was always minimal. (However, we have no proof that this is always the case.)

Taking a closer look at Table 1, Problem 11 is motivated by a real security protocol which employs the XOR operator, it occurs in Bull's recursive authentication protocol [14]. Interestingly, while our algorithm quickly returns an mgu, the version of the algorithm with the optimization for variable optimization turned off does not come back with a solution within 30 minutes. The two versions of the algorithm also perform very differently on the

Table 1: Runtimes and sizes of complete sets of unifiers of sample unification problems: "size" denotes the size of the returned complete set of unifiers; "vi opt" stands for "variable identification optimization"; x, y, z, u are variables and a, b, c, d, e are constants. Runtime tests obtained on a 1.5 GHz Intel Pentium M processor.

no	unification problem	with vi opt time (msecs)	with vi opt size	without vi opt time (msecs)	without vi opt size
1	$x \oplus y \stackrel{?}{=}_E \{y\}_a \oplus b$	0.1	1	0.2	2
2	$x \stackrel{?}{=}_E \langle x \oplus y, \langle x \oplus z, y \oplus z\rangle\rangle$	0.2	1	1.0	5
3	$x \oplus a \stackrel{?}{=}_E \{x \oplus y\}_a$	0.2	1	0.3	2
4	$\langle a \oplus b, x\rangle \stackrel{?}{=}_E \langle x, a\rangle$	0.1	0	0.1	0
5	$\langle a \oplus y, x\rangle \stackrel{?}{=}_E \langle x, a\rangle$	0.1	1	0.2	2
6	$\{\langle x, \langle y, x \oplus y\rangle\rangle\}_{(z\oplus u)^{-1}} \stackrel{?}{=}_E z$	0.1	1	3.3	15
7	$\{\langle x, \langle y, x \oplus y\rangle\rangle\}_{(z\oplus a)^{-1}} \stackrel{?}{=}_E z$	9.1	0	9.1	0
8	$\{\langle x, \langle y, x \oplus y\rangle\rangle\}_{(z\oplus u)^{-1}} \stackrel{?}{=}_E z \oplus x$	0.3	1	3.9	15
9	$\{\langle x, \langle y, x \oplus y\rangle\rangle\}_{(z\oplus a)^{-1}} \stackrel{?}{=}_E z \oplus x$	10.3	0	10.3	0
10	$(x^{-1} \oplus y)^{-1} \stackrel{?}{=}_E y$	0.1	1	0.4	3
11	$\langle x, \langle \{x \oplus y\}_a, \{\{x \oplus y\}_a \oplus z\}_a\rangle\rangle \stackrel{?}{=}_E \langle \{b \oplus c\}_a, \langle \{\{b \oplus c\}_a \oplus d\}_a, \{\{\{b \oplus c\}_a \oplus d\}_a \oplus e\}_a\rangle\rangle$	3.3	1	> 30 min	
12	$\langle z, b\rangle \stackrel{?}{=}_E \langle a, b\rangle \oplus \langle x, \langle y, y\rangle\rangle \oplus \langle c^{-1}, \langle x^{-1}, c\rangle\rangle$	0.6	1	0.6	1
13	$\langle x, \langle z, \langle 0, 0\rangle\rangle\rangle \stackrel{?}{=}_E \langle y^{-1}, \langle a^{-1}, \langle x \oplus a, y \oplus z\rangle\rangle\rangle$	53.7	1	58.0	4
14	$\langle x, \langle z, x\rangle\rangle \stackrel{?}{=}_E \langle y^{-1}, \langle a^{-1}, a \oplus y \oplus z\rangle\rangle$	0.5	2	0.7	4
15	$\langle x, a\rangle \oplus \langle y, a\rangle \stackrel{?}{=}_E \langle b, a\rangle \oplus \langle c, a\rangle$	0.9	2	0.9	2

Table 2: Runtimes and sizes of complete sets of unifiers of sample unification problems: "size" denotes the size of the returned complete set of unifiers; "vi opt" stands for "variable identification optimization"; $x_1, \ldots, x_{10}, y_1, \ldots, y_{10}$ are variables. Runtime tests obtained on a 1.5 GHz Intel Pentium M processor.

no	unification problem	with vi opt time (secs)	with vi opt size	without vi opt time (secs)	without vi opt size
1	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle$	< 0.01	0	< 0.01	0
2	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle \oplus \langle x_2, y_2\rangle$	< 0.01	1	< 0.01	1
3	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle \oplus \ldots \oplus \langle x_3, y_3\rangle$	< 0.01	0	< 0.01	0
4	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle \oplus \ldots \oplus \langle x_4, y_4\rangle$	< 0.01	3	< 0.01	4
5	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle \oplus \ldots \oplus \langle x_5, y_5\rangle$	0.01	0	0.01	0
6	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle \oplus \ldots \oplus \langle x_6, y_6\rangle$	0.04	15	0.04	31
7	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle \oplus \ldots \oplus \langle x_7, y_7\rangle$	0.23	0	0.23	0
8	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle \oplus \ldots \oplus \langle x_8, y_8\rangle$	1.74	105	1.53	379
9	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle \oplus \ldots \oplus \langle x_9, y_9\rangle$	9.57	0	9.59	0
10	$0 \stackrel{?}{=}_E \langle x_1, y_1\rangle \oplus \ldots \oplus \langle x_{10}, y_{10}\rangle$	239.71	945	70.68	6556

Problem 6 and 8. There is no difference in Problem 7 and 9 since this problem is not unifiable, and hence, the algorithm has to try all possible variable identifications. Problem 13 is, compared to the other examples, so difficult because a lot of variable identifications have to be done before the solution is found. y and z have to be identified, as well as x and a (in fact a cannot be identified with x because a is not a variable, but in the purification step a is substituted by a new variable w and then x and w are identified). Because of the use of the non-free $\cdot^{-1}$ operator the choice of the theory indices step also plays a role here and requires more computations.

The problems in Table 2 are only of theoretical interest, they typically do not occur in applications, but illustrate the limitations of optimizations. Note that the size of a minimal complete set of unifiers may be exponential in the size of the unification problem.

9 Conclusion

Motivated by the analysis of security protocols, we have presented a unification algorithm for an equational theory including ACUN. Our algorithm contains several optimizations which make use of the specific properties of the equational theories at hand and performs well on practical examples, both in terms of its runtime and the size of the complete set of unifiers returned. As such, our algorithm is well-suited as a subprocedure in constraint solving algorithms for security protocol analysis with XOR.

One enhancement that can be easily made and boosts the performance if XOR terms are rare is the following: Instead of splitting and purifying the given problem in the beginning, one could run the unification algorithm of the standard theory first. If it encounters an XOR term it can still call the combination algorithm. It also has to call the combination algorithm if it tries to assign x a term t where x occurs in t in the context of the XOR operator. Small problems like $x \stackrel{?}{=}_E \{a \oplus y\}_k$, which often occur during constraint solving, can be handled much faster since no purification has to be done. The algorithm will just assign the free variable x to the term $\{a \oplus y\}_k$. Another example where this enhancement is very useful is the following. Consider the problem $\langle t_1, t_2 \rangle \stackrel{?}{=}_E \langle t_3, t_4 \rangle$, where $t_1, \ldots, t_4$ are arbitrary terms. The standard algorithm will try to unify t_1 with t_2. If this fails there is no hope that the original problem is solvable and it is unnecessary to look at t_3 and t_4.

One future direction is to incorporate other operators and their algebraic properties into our algorithm, including important operators such as Diffie-Hellman Exponentiation and RSA encryption. In [6, 7], it was shown that fully automatic analysis of security protocols is also possible in presence of such operators.

References

[1] F. Baader and K. U. Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. *J. of Symbolic Computation*, 21:211–243, 1996.

[2] F. Baader and W. Snyder. Unification Theory. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 8, pages 447–533. Elsevier Science Publishers, Amsterdam, 2001.

[3] A. Boudet. Combining unification algorithms. *J. of Symbolic Computation*, 16(6):597–626, 1993.

[4] Y. Chevalier. A Simple Constraint-solving Decision Procedure for Protocols with Exclusive or. In *UNIF 2004*, 2004.

[5] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *LICS 2003*, pages 261–270. IEEE, Computer Society Press, 2003.

[6] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *FSTTCS 2003*, volume 2914 of *LNCS*, pages 124–135. Springer, 2003.

[7] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Commuting Public Key Encryption. *Electronic Notes in Theoretical Computer Science*, 125(1):55–66, 2005.

[8] Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *ASE 2001*, pages 373–376. IEEE CS Press, 2001.

[9] H. Comon-Lundh and V. Shmatikov. Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive or. In *LICS 2003*, pages 271–280. IEEE, Computer Society Press, 2003.

[10] Q. Guo, P. Narendran, and D. A. Wolfram. Unification and Matching Modulo Nilpotence. In *CADE 1996*, pages 261–274, 1996.

[11] S. Kepser and J. Richts. Optimisation Techniques for Combining Constraint Solvers. In *Frontiers of Combining Systems 2, Papers presented at FroCoS'98*, pages 193–210. Research Studies Press/Wiley, 1999.

[12] X. Leroy et al. *The Objective Caml system release 3.08*. Institut National de Recherche en Informatique et en Automatique, 2004.

[13] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS 2001*, pages 166–175. ACM Press, 2001.

[14] P. Y. A. Ryan and S. A. Schneider. An Attack on a Recursive Authentication Protocol: A Cautionary Tale. In *Information Processing Letters*, volume 65, pages 7–10, 1998.

A An ACUN-Unification Algorithm

The algorithm presented here can be used as the ACUN-unification algorithm A_{ACUN} in Section 6. It basically uses Gaussian elimination in the field GF(2), i.e. modulo 2.

Algorithm (`unify_ACUN`).

Input:
- a set of ACUN-equations $\Gamma_{\mathrm{ACUN}} = \{0 \stackrel{?}{=}_E t_1, \ldots, 0 \stackrel{?}{=}_E t_n\}$,
- a partial ordering $<_{po}$ and
- a set of variables *const*, which have to be treated as constants.

Output: "No Solution" iff the unification problem $(\Gamma_{\mathrm{ACUN}}, <, \mathit{const})$ is not unifiable for any linear ordering $<$ which extends $<_{po}$. Otherwise an mgu of $(\Gamma_{\mathrm{ACUN}}, <, \mathit{const})$ for some linear ordering $<$ which extends $<_{po}$ and where $(\Gamma_{\mathrm{ACUN}}, <, \mathit{const})$ is unifiable.

Step 1: Let $V(\Gamma_{\mathrm{ACUN}})$ be the set of variables that appear in Γ_{ACUN}. Assume that $V(\Gamma_{\mathrm{ACUN}}) \setminus \mathit{const}$ is $\{x_1, \ldots, x_m\}$ and that $V(\Gamma_{\mathrm{ACUN}}) \cap \mathit{const}$ is $\{a_1, \ldots, a_l\}$.

Γ_{ACUN} is transformed into two matrices A and B. A and B have both n rows, A has m and B has l columns. Each row represents an equation in Γ_{ACUN}. If the variable x_j occurs in t_i then $A_{i,j} := 1$, otherwise $A_{i,j} := 0$. Analogue, if the variable a_j occurs in t_i then $B_{i,j} := 1$, otherwise $B_{i,j} := 0$.

For example

$$\begin{aligned}\Gamma_{\mathrm{ACUN}} = \{0 &\stackrel{?}{=}_E x_1 \oplus x_2 \oplus a_3,\\ 0 &\stackrel{?}{=}_E x_1 \oplus x_3 \oplus a_1 \oplus a_3,\\ 0 &\stackrel{?}{=}_E x_2 \oplus x_3 \oplus a_1 \oplus a_2\}\end{aligned}$$

is transformed into

$$A = \begin{bmatrix}1 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 1\end{bmatrix} \text{ and } B = \begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{bmatrix}.$$

Step 2: The system $(A|B)$ is transformed with Gaussian elimination into an equivalent system $(T|Q)$ where T is a triangular matrix. All numbers are computed modulo 2, since the XOR operator is nilpotent.

Step 3: Normally the system $(T|Q)$ is now in solved form and a solution can easily be computed. The case here is a little more difficult since we have to respect the partial ordering $<_{po}$.

If T contains an 0-row and the same row in Q is not 0, this system has no solution and we return "No Solution". As in the case without a $<_{po}$ we start with the last row of T and assign a value to a variable s.t. the $<_{po}$ is not violated. Then we proceed upwards and look at the next row. If it is not possible to make an assignment to a variable due to the restrictions in $<_{po}$, we have to use backtracking and try other assignments in the lower rows. If we have finally succeeded and reached row one, we have found a unifier and return it. If we have exhausted all possible assignments and could still not find a valid unifier we return "No Solution".