Automatic verification of Finite Variant Property beyond convergent equational theories

Vincent Cheval
University of Oxford, United Kingdom

Caroline Fontaine
Université Paris-Saclay, CNRS, ENS Paris-Saclay,
Laboratoire Méthodes Formelles, France

***Index Terms*—Rewriting, Equational Theory, Finite Variant Property, Verification, Unification, Cryptographic protocols, Symbolic models**

***Abstract*—Computer-aided analysis of security protocols heavily relies on equational theories to model cryptographic primitives. Most automated verifiers for security protocols focus on equational theories that satisfy the *Finite Variant Property (FVP)*, for which solving unification is decidable. However, they either require to prove FVP by hand or at least to provide a representation as an E-convergent rewrite system, usually E being at most the equational theory for an associative and commutative function symbol (AC). The verifier ProVerif is probably the only exception amongst these tools as it automatically proves FVP without requiring a representation, but on a small class of equational theories. In this work, we propose a novel semi-decision procedure for proving FVP, without the need for a specific representation, and for a class of theories that goes beyond the ones expressed by an E-convergent rewrite system. We implemented a prototype and successfully applied it on several theories from the literature.**

I. Introduction

Cryptographic protocols are distributed programs designed to secure communications. They are used in diverse critical applications, such as electronic voting, cryptocurrencies, secure messaging, online payment, etc. Each protocol comes with its own list of security claims, some of which are common to all applications (e.g., secrecy and authentication) and others of which are tailored to specific domains (e.g., coercion resistance in electronic voting).

The design of such protocols being notoriously error prone, it is now standard practice to provide formal proofs that the target security properties are really satisfied. Over the years, there have been many success stories on the symbolic analysis of impactful real-life security protocols, for example TLS [1], [2], [3], ECH [4], Signal [5], 5G-AKA [6], Noise [7], EMV [8] and IKEv2 [9]. These successes were partly due to advances in the capabilities of automatic verifiers for cryptographic properties, as ProVerif [10], [11], Tamarin [12], [13], Maude-NPA [14], [15], DeepSec [16], [17] and AKiSs [18], [19].

All the above-mentioned tools rely on similar underlying symbolic models. In particular, messages exchanged over the network are expressed by terms constructed out of function symbols (representing cryptographic primitives), names (representing large numbers such as keys), and variables. Here, a term $\mathsf{enc}(m, k)$ represents the symmetric encryption of the plain text m with the key k using the encryption algorithm enc. The algebraic properties of these cryptographic primitives are expressed by means of an *equational theory*, comprising of equations between two terms that indicate that they correspond to the same message. In our example, an encryption algorithm comes with an associated decryption algorithm, denoted here dec. The link between enc and dec can be expressed by the equation $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$. A set E of such equations is an equational theory and induces an equivalence relation $=_E$ which represents the terms *equal modulo* E. For example, $\mathsf{dec}(\mathsf{enc}(\mathsf{h}(a), k), k) =_E \mathsf{h}(a)$ but $\mathsf{dec}(\mathsf{enc}(\mathsf{h}(a), k), k') \neq_E \mathsf{h}(a)$ as the decryption key k' is different from the encryption key k.

Checking equality of two terms modulo an equational theory E is the most basic problem that these automatic tools must decide. However, as they consider an active attacker that controls the network, they need to verify the security claims for every possible message crafted by the attacker that would be accepted by honest participants. Among other things, this entails checking whether two terms can *unify modulo the equational theory* E, that is, for terms u and v, checking whether there exists a substitution σ, i.e. a mapping from variables to terms, such that $u\sigma =_E v\sigma$. Such a σ is called a *unifier*. Not only do the tools need to decide whether such a unifier exists, they also need the ability to compute the set of *most general unifiers*. This is a computationally hard problem that heavily depends on the equations in E and that is undecidable in general.

The problem of unification modulo an equational theory has been extensively studied [20], [21], [22], [23], [24]. A very successful technique consists in finding a representation of E as a *convergent rewrite system* and applying *basic narrowing* [25], [26]. A rewrite system $\mathcal{R}$ is a set of oriented equations also called *rules*, which allows rewriting a term u to another term v through a *rewrite step*, denoted $u \to_{\mathcal{R}} v$. For example, when $\mathcal{R} = \{\mathsf{dec}(\mathsf{enc}(x, y), y) \to x\}$, we have $\mathsf{g}(\mathsf{dec}(\mathsf{enc}(\mathsf{h}(a), k), k), b) \to_{\mathcal{R}} \mathsf{g}(\mathsf{h}(a), b)$. Convergence implies that the rewrite system is *terminating*, i.e., it does not allow infinite sequences of rewrite steps, and that it is *confluent*, i.e., all sequences of rewrite steps from an initial term t end up in the same term. Basic narrowing does not always terminate, but it is a generic technique that covers a large class of cryptographic primitives. This is for instance the underlying technique used by the verifier DeepSec [16],

This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

[17]. However, basic narrowing fails when the equational theory contains function symbols that are associative and commutative (AC), such as Exclusive-Or (XOR). Not only does basic narrowing fail as such equational theory cannot be represented by a terminating rewrite system, but even basic AC-narrowing (that is basic narrowing modulo AC) was shown to be incomplete [27].

Many works have thus developed ad-hoc algorithms solving the unification problem for specific equational theories: AC [21], [28], [29], [30], [31], Abelian Groups ($\mathcal{AG}$) [32], [33], AC plus unit, idempotency (ACUI) and distributivity [34], Exclusive-Or with homomorphism (XORh) [35], $\mathcal{AG}$ with homomorphism ($\mathcal{AG}h$) [36], etc. However, in the context of the automated verification of cryptographic protocols, ad-hoc procedures are not sufficient. The ever-increasing complexity of protocols and their cryptographic primitives require tools that support user-defined equational theories and primitives.

In 2005, Comon and Delaune introduced the *finite variant property* (FVP) [27] for equational theories. They present a general technique, showing that when an equational theory E satisfies the FVP modulo some smaller equational theory E', the problem of unification modulo E can be reduced to the problem of unification modulo E', using what is now known as *E-variant narrowing*. This is why FVP is of particular interest in the case of protocol analyses, as many user-defined equational theories can be reduced in practice to unification modulo AC. Since 2005, and although the definition of FVP is quite general, most efforts in the literature for proving the FVP have focused on cases where E can be represented by a E'-convergent rewrite system either by hand [27], [37] or automatically [38], [39].

It is important to notice that using the FVP to solve the unification problem is usually less efficient than using a bespoke algorithm. This is partly due to the fact that unification modulo AC is extremely costly, even on small problems, leading to significant decreases in performance, whereas unification modulo XOR and $\mathcal{AG}$ are much more efficient. As an example, the unification of $x{+}x{+}x{+}x$ and $x_1{+}x_2{+}x_3{+}x_4$ modulo AC has more than 34 billion most general unifiers [28] whereas the unification modulo XOR has only 57. Nevertheless, using the FVP for solving unification remains in our opinion the most successful technique in practice.

The same year Comon and Delaune introduced the FVP, Blanchet, Abadi and Fournet published a paper [40] on an extension of the ProVerif tool in which they handled the equational theories by introducing the notion of *extended signature modelling an equational theory*. In [40], they described two procedures to create such extended signatures from an equational theory given as input: one for E'-convergent equational theories with $E' = \emptyset$, and the other for *linear equational theories* (i.e., those for which in all equations each variable occurs at most once in the left-hand side and at most once in the right-hand side) that cannot be expressed as a convergent rewrite system. Interestingly, their notion of extended signatures modelling an equational theory [40] can be shown to coincide with the FVP when $E' = \emptyset$.

As mentioned, most past works on the FVP have been focusing on equational theories E that can be represented by an E'-convergent rewrite system $\mathcal{R}$ where $E' = AC$. Furthermore, they also require the rewrite system $\mathcal{R}$ to be provided in order to either compute E-variants or check the FVP. This is the case for the Maude-NPA and Maude tools. The Tamarin tool is even further restrictive as it only allows four groups of built-in function symbols with associative-commutative properties: Exclusive-OR, Diffie-Hellman groups (Abelian Group with an exponentiation operator), Bilinear-pairing and multiset (simple AC function symbol). In particular, (i) the function symbols defined in these equational theories cannot be used in user-defined equational theories; (ii) user-defined primitives cannot have function symbols with associative-commutative properties.

The limitations in Tamarin partly come from the fact that creating an AC-convergent rewrite system $\mathcal{R}$ that satisfies the FVP can be very tricky. In the case of $\mathcal{AG}$, the only known representation showing the FVP is a peculiar rewrite system containing 10 rewrite rules, first proposed by Lankford [41], [27]. In contrast, we make the following observations on the algorithms proposed in [40] and currently used in ProVerif:

- they can be used to effectively compute variants of terms and most general unifiers;
- they do not require any input other than the equational theory E itself (no mandatory rewrite system requested);
- they handle some theories that are not convergent (e.g., they handle the linear theory $\exp(\exp(g,x),y) = \exp(\exp(g,y),x)$);
- they terminate only if E has the FVP modulo the empty theory. In particular, they cannot handle theory modulo AC.

Our contributions: This present paper generalises and improves upon the framework and algorithms of [40]. Our main contributions are as follows:

- we introduce a new notion of *Rewrite Theory mimicking an equational theory E* that implies the FVP modulo an equational theory beyond AC, e.g. XOR, XORh, $\mathcal{AG}$, $\mathcal{AG}$h; thereby avoiding the costly unification modulo AC;
- we show under which conditions our framework coincides with the FVP;
- we provide a new semi-decision procedure that can automatically transform an equational theory E into a rewrite theory that mimics E. We lift the convergent or linear restrictions from [40] as our procedure requires no initial condition on E;
- we consider some optimisation techniques for both scope and efficiency;
- we have implemented in a prototype our algorithm restricted to FVP modulo AC and successfully tested it on several theories with known FVP modulo AC from the literature (e.g. XOR, $\mathcal{AG}$, Diffie-Hellman bilinear pairing). We also showed that many of these equational theories augmented with a homomorphic symbol with different

group operators also satisfy the FVP. This is for example the case for XOR, $\mathcal{AG}$, ElGamal and (a)symmetric encryption. Finally, we also provide several toy examples showing that the prototype also works on equational theory without AC-convergent rewrite system representation. The code and examples are available at [42].

We believe that our work is generic enough to be ported to most automatic verifiers of security protocols, leading to a significant improvement in their scope, usability and efficiency. The full proof can be found in a technical report [43].

Outline: In Section II, we provide some preliminary definitions, including the finite variant property. Section III presents the framework with the new notion of rewrite theory and Section IV explains how it relates to the finite variant property. In Section V, we shall describe our semi-decision procedure for generating rewrite theories and present a detailed overview of its proof of correctness. Section VI focuses on three possible optimisations to our procedure. Finally, in Section VII, we present experimental results on a subset of equational theories that can be handled by our prototype and discuss some adjacent properties of our procedure and its limitations.

II. Preliminaries

A. Terms

We use classical notation and terminology from [27] on terms, unification, rewrite systems. Let $\mathcal{X}$ be an infinite set of variables and $\mathcal{N}$ be an infinite enumerable set of names (also called *free constant* in the rewriting community). The set $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ consists of all terms built over the finite ranked alphabet $\mathcal{F}$ of function symbols, variables from $\mathcal{X}$ and names from $\mathcal{N}$. A term t is *ground* when $t \in \mathcal{T}(\mathcal{F}, \mathcal{N})$. The set of positions of a term t is written $\mathcal{P}os(t)$. The subterm of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ at position $p \in \mathcal{P}os(t)$ is written $t|_p$. We denote $\mathrm{st}(t)$ (resp. $\mathrm{st}^s(t)$) the set of all (resp. strict) subterms of t. The term obtained by replacing $t|_p$ with u is denoted $t[u]_p$. The set of variables occurring in t is denoted $vars(t)$. The set of names occurring in t is denoted $names(t)$.

A context $C[\_]$ is a term where one of its subterms is replaced by a hole denoted by _. Given a term u, $C[u]$ denotes the term obtained from $C[\_]$ by replacing the hole _ with u.

A *substitution* σ is a mapping from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. The *domain* of σ, denoted $dom(\sigma)$, is the set of variables x such that $x \neq x\sigma$. The *image* of σ, denoted $img(\sigma)$, is the set of terms $x\sigma$ where $x \in dom(\sigma)$. Given $\mathsf{N}, \mathsf{N}' \subseteq \mathcal{N}$, a bijective mapping ρ from N to N' is called a *renaming*. We will denote $dom(\rho) = \mathsf{N}$ and $img(\rho) = \mathsf{N}'$.

B. Equational theories

An *equational theory* E is a set of equations (unordered pairs of terms) from $\mathcal{T}(\mathcal{F}, \mathcal{X})$, meaning that they should not contain names. It induces the relation $=_E$ which is the least congruence on $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ such that $u\sigma =_E v\sigma$ for all pairs $u = v \in E$ and substitutions σ. E is *regular* if for all equations $t_1 = t_2 \in E$, $vars(t_1) = vars(t_2)$. E is *trivial* if for all terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, $s =_E t$.

Two terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ are *E-unifiable* if there exists a substitution σ, called *E-unifier*, such that $s\sigma =_E t\sigma$. For example, using the infix notation for $+$, the set $E_\oplus$ of equations for XOR is as follows:

$$E_\oplus = \left\{ \begin{array}{c} x + y = y + x, x + (y + z) = (x + y) + z, \\ x + x = 0, x + 0 = x \end{array} \right\}$$

The equations for Abelian Groups ($\mathcal{AG}$) include $x * 1 = x$ and $x * inv(x) = 1$ with AC equations for $*$. We say that a finite set of substitutions $S = \{\sigma_1, \ldots, \sigma_n\}$ is a *complete set of E-unifiers of s, t* if for all E-unifiers σ of s, t, there exist $i \in \{1, \ldots, n\}$ and a substitution θ such that for all $x \in vars(s, t)$, $x\sigma =_E x\sigma_i\theta$. We say that S is a *complete set of most general E-unifiers of s, t*, denoted by $\mathrm{mgu}_E(s, t)$, when it is a complete set of E-unifiers and no substitution is an instance of another, i.e. for all $\sigma_1, \sigma_2 \in S$, there is no substitution α such that $\sigma_1 =_E \sigma_2\alpha$. Note that it is easy to derive an algorithm that produces finite complete sets of most-general E-unifiers from an algorithm that produces finite complete sets of E-unifiers.

C. Rewriting

A *term rewrite system* (TRS) is a finite set of *rewrite rules* $\ell \to r$ where $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. A term $s \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ rewrites to t by a TRS $\mathcal{R}$, denoted $s \to_{\mathcal{R}} t$, if there exist $\ell \to r$ in $\mathcal{R}$, $p \in \mathcal{P}os(s)$ and a substitution σ such that $s|_p = \ell\sigma$ and $t = s[r\sigma]_p$.

Given a rewrite system $\mathcal{R}$ and an equational theory E, *s rewrites into t by $\mathcal{R}$ modulo E*, denoted $s \to_{\mathcal{R},E} t$, iff there exist a position $p \in \mathcal{P}os(s)$, a rule $\ell \to r \in \mathcal{R}$ and a substitution σ such that $s|_p =_E \ell\sigma$ and $t = s[r\sigma]_p$. The relation $\to_{\mathcal{R},E}$ has usually been used [44] to implement the larger relation $\to_{\mathcal{R}/E} = (=_E \circ \to_{\mathcal{R}} \circ =_E)$.

A rewrite system $\mathcal{R}$ is *E-confluent* if and only if for all $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, if $s =_{(\mathcal{R}^=) \cup E} t$ then there exist s', t' such that $s \to^*_{\mathcal{R},E} s'$, $t \to^*_{\mathcal{R},E} t'$ and $s' =_E t'$, where $\mathcal{R}^= = \{\ell = r \mid \ell \to r \in \mathcal{R}\}$ and $\to^*_{\mathcal{R},E}$ is the reflexive transitive closure of $\to_{\mathcal{R},E}$. We say that $\mathcal{R}$ is *E-terminating* when the relation $\to_{\mathcal{R}/E}$ is well founded. $\mathcal{R}$ is said to be *E-convergent* when it is both E-terminating and E-confluent. Finally, we also say that $\mathcal{R}$ is E-confluent (resp. convergent) for E' when $\mathcal{R}$ is E-confluent (resp. convergent) and $(\mathcal{R}^=) \cup E$ is the same relation as E', i.e. $s =_{(\mathcal{R}^=) \cup E} t$ iff $s =_{E'} t$.

Example 1. *Defining $\mathcal{R} = \{x + 0 \to x; x + x \to 0\}$ and $\mathcal{R}_\oplus = \mathcal{R} \cup \{x + (x + y) \to y\}$, it is well known that $\mathcal{R}_\oplus$ is AC-convergent for XOR but not $\mathcal{R}$. Indeed, the term $t_1 = (a + a) + b \to^*_{\mathcal{R},AC} b$ but the term $t_2 = a + (a + b)$ cannot be rewritten by $\to_{\mathcal{R},AC}$ since neither a nor $(a + b)$ can be AC-matched with the left hand side of a rule in $\mathcal{R}$. In other words, $t_2 \not\to_{\mathcal{R},AC}$. Therefore, $\mathcal{R}$ is not AC-confluent.*

This also illustrates the difference between $\to_{\mathcal{R}/AC}$ and $\to_{\mathcal{R},AC}$, as $t_2 \to_{\mathcal{R}/AC} 0 + b \to_{\mathcal{R}/AC} b$.

D. Ordering terms

A strict order on terms $>$ is said to be a *rewrite order* when it is closed by application of contexts (*i.e.*, $t > s$ implies

$C[t] > C[s]$) and substitutions (i.e., $t > s$ implies $t\sigma > s\sigma$). When the rewrite order is well-founded, it is called a *reduction order*. An order $>$ is *E-compatible* if $s' =_E s > t =_E t'$ implies $s' > t'$. We say that $>$ is *E-total on ground terms* (E-total for short) if for all $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{N})$, $s \neq_E t$ implies $s > t$ or $t > s$. Furthermore, $>$ is *E-compatible with a rewrite system R* when it is E-compatible and for all $\ell \to r \in \mathcal{R}$, we have $\ell > r$. It is well known that a rewrite system $\mathcal{R}$ is E-terminating if and only if there exists a reduction order E-compatible with $\mathcal{R}$. An E-compatible reduction order induces a well-defined order on the set of $=_E$-equivalence classes. Moreover, when $>$ is in addition E-total, $>$ becomes total on $\mathcal{T}(\mathcal{F}, \mathcal{N})/_{=_E}$. In such a case, given another equational theory E' such that $E \subseteq E'$, we define $\min_{E',>}(t)$ as the smallest element of $\mathcal{T}(\mathcal{F}, \mathcal{N})/_{=_E}$ by $>$ such that $t =_{E'} \min_{E',>}(t)$. In other words, $\min_{E',>}(t)$ is the smallest term modulo E by $>$ that is equal modulo E' to t.

Example 2. *Consider the order as follows: $t >_\# s$ when*

- *the number of function symbols or names in t is greater than in s*
- *for all $x \in \mathcal{X}$, the number of occurrences of x in t is greater than in s*
- *at least one of these inequalities is strict.*

The order $>_\#$ is a reduction order AC-compatible but not AC-total.

Remark. *It was already shown in [45] that the existence of a non-empty E-compatible reduction order necessarily implies E to be regular. In other words, when E is not regular, no constraint system can be E-terminating. Indeed, consider an equation $u = C[x]$ in E with x a variable not in u, and a rewrite rule $\ell \to r$ in $\mathcal{R}$. We have $C[\ell] \to_\mathcal{R} C[r] =_E C[\ell]$. Hence, $=_E \circ \to_\mathcal{R} \circ =_E$ is not well founded.*

There exist many ways to implement a reduction order, specially when $E = \emptyset$. The most commonly used in the literature are *recursive path ordering* (RPO) [46] and *lexicographic path ordering* (LPO) [47]. They enjoy nice properties such as the *subterm property* (i.e. $C[s] > s$ for all non-empty contexts $C[\_]$) and being total on ground terms. When $E \neq \emptyset$, it is more complicated to create an E-compatible and E-total reduction order but previous works have been successful when considering theories such as AC [48], [49], [50], [51] or permutations [52].

Example 3. *Consider $\mathcal{F} = \mathcal{F}_{AC} \cup \mathcal{F}_o$ where $\mathcal{F}_{AC}$ are the binary function symbols with associative and commutative properties, and $\mathcal{F}_o$ are all the other function symbols.*

To define the order $>_{AC}$ from [49], we first need to consider a strict order on the function symbols, denoted $>_\mathcal{F}$. Additionally, we need consider the terms in their flattened form*, e.g., $f(u_1, f(u_2, u_3))$ with $f \in \mathcal{F}_{AC}$ is represented as $f(u_1, u_2, u_3)$. This is a classic representation of terms when working with AC function symbols. Let us denote by $\mathtt{tf}(t)$ the term t in flattened representation. We now define some preliminary notions as follows.*

For all $s = f(s_1, \ldots, s_n)$ with $f \in \mathcal{F}_{AC}$, we define:

- $\text{EmbSmall}(s) = \{\mathtt{tf}(f(s_1, \ldots, s_{i-1}, v_j, s_{i+1}, \ldots, s_n)) \mid s_i = h(v_1, \ldots, v_r) \wedge f >_\mathcal{F} h \wedge j \in \{1, \ldots, r\}\}$;
- $\text{BigHead}(s) = \{\!\{s_i \mid i \in \{1, \ldots, n\} \wedge top(s_i) >_\mathcal{F} f\}\!\}$;
- $\text{NoSmallHead}(s) = \{\!\{s_i \mid i \in \{1, \ldots, n\} \wedge f \not>_\mathcal{F} top(s_i)\}\!\}$;
- $\#(f(s_1, \ldots, s_n)) = \#_v(s_1) + \ldots + \#_v(s_n)$ *with* $\#_v(x) = x$ *and* $\#_v(t) = 1$ *if t is not a variable;*

where $\{\!\{u_1, u_2, \ldots, u_k\}\!\}$ is a multiset.

The order $>_{AC}$ is defined as: $s = f(s_1, \ldots, s_n) >_{AC} g(t_1, \ldots, t_m)$ if and only if one of the following properties holds

1) $s_i \geq_{AC} t$ *for some* $i \in \{1, \ldots, n\}$
2) $f >_\mathcal{F} g$ *and* $s >_{AC} t_i$ *for all* $i \in \{1, \ldots, m\}$
3) $f = g \in \mathcal{F}_o$ *and* $(s_1, \ldots, s_n) >^{lex}_{AC} (t_1, \ldots, t_m)$ *and* $s >_{AC} t_i$ *for all* $i \in \{1, \ldots, m\}$
4) $f = g \in \mathcal{F}_{AC}$ *and* $\exists s' \in \text{EmbSmall}(s)$ *s.t.* $s' \geq_{AC} t$
5) $f = g \in \mathcal{F}_{AC}$ *and* $\forall t' \in \text{EmbSmall}(t)$, $s >_{AC} t'$ *and* $\text{NoSmallHead}(s) \geq^{mul}_{AC} \text{NoSmallHead}(t)$ *and either:*
   - $\text{BigHead}(s) >^{mul}_{AC} \text{BigHead}(t)$ *or*
   - $\#(s) > \#(t)$ *or*
   - $\#(s) \geq \#(t)$ *or* $\{\!\{s_1, \ldots, s_n\}\!\} >^{mul}_{AC} \{\!\{t_1, \ldots, t_m\}\!\}$

with $>^{lex}_{AC}$ the lexicographic extension of $>_{AC}$ on sequences and $>^{mul}_{AC}$ is the multiset extension of $>_{AC}$ on finite multisets.

The order $>_{AC}$ was shown in [49] to be a AC-compatible and AC-total reduction order. It is also AC-compatible with $\mathcal{R}_\oplus$ from Example 1. This is one of the two reduction orders we implemented in our prototype.

Most orders in the literature do not consider names in terms. For the need of our algorithm, we will require our reduction order to be stable by application of renamings that preserve the order. Formally, a renaming ρ *preserves* $>$ when for all $a, b \in dom(\rho)$, $a > b$ iff $a\rho > b\rho$. We say that $>$ is *stable by renaming* when for all terms s, t, for all renamings ρ preserving $>$, if $names(s, t) \subseteq dom(\rho)$ then $s > t$ iff $s\rho > t\rho$.

E. Finite variant property

We rely on the seminal notion of finite variant property which has been initially introduced by [27] adapted to our notation. Let $E \subseteq E'$ be two equational theories. Let $>$ be an E-total and E-compatible reduction order. Given two terms $t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, (t', θ) is a *E'-variant* of t when $t\theta =_{E'} t'$. A *complete set of E'-variants modulo E of t* is a set S of E'-variants of t such that for all substitutions σ closing for t (i.e. $t\sigma$ is ground), there exist $(t', \theta) \in S$ and a substitution α such that $\min_{E',>}(t\sigma) =_E t'\alpha$.

Definition 1 ([27])**.** *Given two equational theories $E \subseteq E'$, E' has the* finite variant property modulo E *w.r.t. $>$ if for all terms t, there exists a finite complete set of E'-variants modulo E of t.*

Intuitively, if S is the complete set of E'-variants modulo E of t then the smallest term by $>$ equal modulo E' to any closing instantiation of t is an instantiation (modulo E) of a term in S. To draw a parallel with the notion of most general unifiers, one could say that the terms in S of t are the *most*

general smallest E-instantiations of t by $>$. Note that even a simple term can have many variants.

Example 4. *Coming back to the theory $E_\oplus$ of XOR, the complete set of $E_\oplus$-variants modulo AC of $x_1 + x_2$ is composed of 7 variants:*

$$x_1, \{x_2 \mapsto 0\} \quad x_2, \{x_1 \mapsto 0\} \quad y, \{x_1 \mapsto y + x_2\}$$
$$y_1 + y_2, \{x_1 \mapsto y_1 + z, x_2 \mapsto y_2 + z\} \quad 0, \{x_2 \mapsto x_1\}$$
$$y, \{x_2 \mapsto x_1 + y\} \quad x_1 + x_2, \emptyset$$

For the Abelian Group theory, the term x_1+x_2 has 47 variants.

In the rest of this paper, we will say that $>$ is a *E-strong reduction order* when it is E-compatible, E-total and stable by renaming.

III. Rewrite theory

In the vein of previous works [40], [38], [27], [53], [39], a rewrite theory T can be seen as a decomposition of the main equational theory E into a rewrite system and a smaller equational theory. However, as previously mentioned, the decomposition of an equation theory E into a rewrite system that is AC-convergent does not necessarily lead to an efficient unification algorithm as the unification modulo AC is notoriously slow. On the other hand, the problem of matching modulo AC, that is, checking whether there exists σ such that $u\sigma =_{AC} v$, can be often solved efficiently in practice [54], despite still being an NP-complete problem [55]. This distinction between unification and matching modulo AC gave rise to one of the key component of our rewrite theories: instead of decomposing the main equation theory E into one rewrite system and one small equational theory, a rewrite theory T will decompose E into two sets of rewrite rules $\mathcal{R}_\downarrow$ and $\mathcal{R}$ and two equational theories $E_\downarrow$ and $E_\mathcal{A}$ such that $E_\downarrow \subseteq E_\mathcal{A} \subseteq E$. Intuitively, we will assume the existence of an efficient algorithm for solving unification modulo $E_\mathcal{A}$ (e.g. XOR, $\mathcal{AG}$) whereas we only require an efficient algorithm for solving matching modulo $E_\downarrow$ (e.g. AC). Similarly, the set of rewrite rules $\mathcal{R}$ will be used to compute the variants of terms modulo $E_\mathcal{A}$, whereas the set of rewrite rules $\mathcal{R}_\downarrow$ will be used to normalise the terms modulo $E_\downarrow$. Finally, our rewrite theory T will also include an $E_\downarrow$-strong reduction order compatible with $\mathcal{R}_\downarrow$. This order will serve several purposes: first, it allows us to ensure $E_\downarrow$-termination of $\mathcal{R}_\downarrow$ (we do not require $E_\downarrow$-convergence); second, it will be used to show that E has the finite variant property modulo $E_\mathcal{A}$; and it will play a crucial part in the proof of correctness of our main algorithm (see Section V).

Definition 2. *A* rewrite theory *T is a tuple $(>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ where:*

- ***S1*** *$E_\downarrow$ and $E_\mathcal{A}$ are equational theories with $E_\downarrow \subseteq E_\mathcal{A}$*
- ***S2*** *$>$ is a $E_\downarrow$-strong reduction order compatible with $\mathcal{R}_\downarrow$*
- ***S3*** *$\mathcal{R}$ is a rewrite system such that for all $f \in \mathcal{F}$, denoting $\ell = f(x_1, \ldots, x_n)$, we have $(\ell \to \ell) \in \mathcal{R}$, and for all $\ell' \to r' \in \mathcal{R}$, we have $vars(r') \subseteq vars(\ell')$*

In Item **S3**, the presence of rules of the form $f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)$ is used in the next section to homogenise the definition of computation of most general unifiers and variants. Intuitively, when computing the variants of a term u, we will use the rules in $\mathcal{R}$ to rewrite all symbols in u. Hence, the above rule represents the case when the symbol f is actually not rewritten.

Example 5. *Coming back to the theory $E_\oplus$, we can define two rewrite theories $T_1 = (>_{AC}, \mathcal{R}_1, \mathcal{R}_{\downarrow_1}, AC, AC)$ and $T_2 = (>_{AC}, \mathcal{R}_2, \mathcal{R}_{\downarrow_2}, AC, E_\oplus)$ where $\mathcal{R}_1$ is the rewrite system that includes the following rules:*

$$x_1 + 0 \to x_1 \qquad 0 + x_2 \to x_2 \qquad (y + x_2) + x_2 \to y$$
$$x_1 + x_1 \to 0 \qquad (y_1 + z) + (y_2 + z) \to y_1 + y_2$$
$$x_1 + x_2 \to x_1 + x_2 \qquad x_1 + (x_1 + y) \to y \qquad 0 \to 0$$

and $\mathcal{R}_2 = \{x_1 + x_2 \to x_1 + x_2; 0 \to 0\}$.

Notice that $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) corresponds to the $E_\oplus$-variants modulo AC (resp. $E_\oplus$) of $x_1 + x_2$ and 0. Though we will formally define the notion of a rewrite theory mimicking an equational theory later in Definition 4, $\mathcal{R}_1$ and $\mathcal{R}_2$ containing variants will explain why both rewrite theories T_1 and T_2 mimic $E_\oplus$.

Hence, for a larger equational theory E that contains $E_\oplus$, by taking $E_\mathcal{A}$ to be $E_\oplus$ instead of AC in the rewrite theory, we also reduce the number of rewrite rules in the rewrite system $\mathcal{R}$. Consider, as toy-example, $E_{\mathsf{ed}} = E_\oplus \cup \{\mathsf{d}(\mathsf{e}(x + k_2, k_1), k_1, k_2) = x\}$. The term $\mathsf{d}(\mathsf{e}(x + r, k), k, r)$ has 8 E_{ed}-variants modulo AC whereas it only has 2 E_{ed}-variants modulo $E_\oplus$, corresponding to the following two rules: $\mathsf{d}(\mathsf{e}(x + k_2, k_1), k_1, k_2) \to x$ and $\mathsf{d}(x, k_1, k_2) \to \mathsf{d}(x, k_1, k_2)$.

The rewrite systems $\mathcal{R}_{\downarrow_1}$ and $\mathcal{R}_{\downarrow_2}$ in the rewrite theories T_1 and T_2 are only used for normalisation. As such, their content can vary and will mostly be useful for the generation of rewrite theories (see Section V). For instance, on input $E_\oplus$, our prototype will start by building a rewrite theory with $\mathcal{R}_{\downarrow_1} = \emptyset$ but gradually augment it to reach $\mathcal{R}_{\downarrow_1} = \{x+0 \to x; x+x \to 0\}$.

Although the rewrite system $\mathcal{R}_\downarrow$ in a rewrite theory T is intuitively used for normalisation, Definition 2 does not impose $E_\downarrow$-convergence of $\mathcal{R}_\downarrow$. Therefore, we rely on $>$ being a $E_\downarrow$-strong reduction order compatible with $\mathcal{R}_\downarrow$ to define a notion of normal form with respect to a rewrite theory T.

Definition 3. *Let $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ be a rewrite theory and E an equational theory. Let k be the smallest name in $\mathcal{N}$ by $>$. A set of terms $\mathcal{M}$ is* in normal form*, denoted $\mathrm{nf}_{T,E}(\mathcal{M})$, when $E_\mathcal{A} \subseteq E$ and there exists an injective substitution σ such that $dom(\sigma) = vars(\mathcal{M})$, $img(\sigma) \subseteq \mathcal{N}$ and:*

- *$\forall a \in img(\sigma)$, $a \neq k$ and $\forall b \in names(\mathcal{M})$, $a > b$*
- *$\forall t \in \mathcal{M}$, $t\sigma =_{E_\downarrow} \min_{E,>}(t\sigma)$.*

Intuitively, a term t is in normal form when there is no other term equal to t modulo E that is strictly smaller than t by $>$. Such intuition only holds on ground terms as the order $>$ is $E_\downarrow$-total. When the term t contains variables, it may not be ordered with other terms. Thus, to discuss the normal

form of a term with variables, we will consider all variables of t as names, i.e. we will close t with some substitution σ such that $img(\sigma)$ only contains names. However, the choice of names in the substitution σ may impact the minimality by $>$. In particular, to distinguish variables with names in t, it is important for σ to select names not already in t. Similarly, when we need to consider multiple terms in normal forms, e.g. in a set of terms $\mathcal{M}$, we need to select names that are not already in any of the terms in $\mathcal{M}$. This is the purpose of the first item of Definition 3 which guarantees that we do not confuse the names already in $\mathcal{M}$ with the one used to close $\mathcal{M}$, i.e. in $img(\sigma)$.

Example 6. *Consider two symbols $f/2$ and $h/1$, and an equational theory E representing that f is commutative. Take $E_\downarrow = \emptyset$. Consider a lexicographic path ordering such that on ground terms, the smallest terms are $a < b < h(a) < c < \dots$ where a, b, c are names. By definition, $f(x, h(a)) =_E f(h(a), x)$. With such order, both terms $f(x, h(a))$ and $f(h(a), x)$ individually are in normal form since $f(b, h(a)) < f(h(a), b)$ ($\sigma = \{x \to b\}$) and $f(h(a), c) < f(c, h(a))$ ($\sigma = \{x \to c\}$). In other words, there is a way to close each term t such that the resulting term $t\sigma$ is the smallest by $<$ amongst the terms equal modulo E, that is $t\sigma = \min_{E,>}(t\sigma)$. However, the set $\{f(x, h(a)), h(b)\}$ is not in normal form whereas $\{f(h(a), x), h(b)\}$ is in normal form since we cannot instantiate x by b any more as it already occurs in the sets.*

Manipulating the injective substitution σ and $\min_{E,>}$ in the definition can be quite cumbersome. Hence, we state below some interesting properties that show how we can manipulate sets of terms that are in normal form, without relying on the renaming nor $\min_{E,>}$. The proof of Lemma 1 can be found in [43, Appendix B].

Lemma 1. *Let $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ be a rewrite theory and E be a non-trivial equational theory. Assume that $E_\mathcal{A} \subseteq E$, and that for all terms t and s, $t \to_{\mathcal{R}_\downarrow, E_\downarrow} s$ implies $t =_E s$. Let $\mathcal{M}$ be a set of terms such that $\mathrm{nf}_{T,E}(\mathcal{M})$. Then the following properties hold:*

- ***N1*** *for all $t \in \mathrm{st}(\mathcal{M})$, t is irreducible by $\to_{\mathcal{R}_\downarrow, E_\downarrow}$.*
- ***N2*** *for all $s, t \in \mathrm{st}(\mathcal{M})$, if $s =_E t$ then $s =_{E_\downarrow} t$.*
- ***N3*** *for all $x \in \mathcal{X}$, $\mathrm{nf}_{T,E}(\mathcal{M} \cup \{x\})$.*
- ***N4*** *for all terms t, there exists a term s such that $t =_E s$ and $\mathrm{nf}_{T,E}(\mathcal{M} \cup \{s\})$.*

Item **N1** is particularly useful when the order $>$ is not (easily) computable as it provides a sound and easy way for checking that t is not in normal form. Additionally, since $>$ is compatible with $\mathcal{R}_\downarrow$, $t \to^*_{\mathcal{R}_\downarrow, E_\downarrow} t'$ implies $t > t'$. Hence, an irreducible term obtained by rewriting t is a potential candidate for being minimal by $>$ amongst the equivalence class of t modulo E. We can now define when a rewrite theory T mimics an equational theory E.

Definition 4. *A rewrite theory $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$* mimics an equational theory E *when:*

- ***M1*** *$E_\mathcal{A} \subseteq E$ and if $t \to_{\mathcal{R}_\downarrow, E_\downarrow} t'$ then $t =_E t'$.*
- ***M2*** *If $f(t_1, \dots, t_n) \to t$ is in $\mathcal{R}$ then $f(t_1, \dots, t_n) =_E t$.*
- ***M3*** *If $f(t_1, \dots, t_n) =_E t$ and $\mathrm{nf}_{T,E}(\{t_1, \dots, t_n, t\})$ then there exist σ and $f(s_1, \dots, s_n) \to s$ in $\mathcal{R}$ such that, $t =_{E_\mathcal{A}} s\sigma$, and for all $i \in \{1, \dots, n\}$, $t_i =_{E_\mathcal{A}} s_i\sigma$.*

*In Item **M3**, the variables in the rewrite rule $f(s_1, \dots, s_n) \to s$ are assumed to be freshly renamed, to ensure that $vars(s_1, \dots, s_n, s) \cap vars(t_1, \dots, t_n, t) = \emptyset$.*

Example 7. *T_1 and T_2 from Example 5 both mimic $E_\oplus$. The rewrite theory T_2 trivially mimics $E_\oplus$, but is useless since the equational theory $E_\mathcal{A}$ in T_2 is also $E_\oplus$. The rewrite theory $T_{\mathsf{ed}} = (>_{AC}, \mathcal{R}, \mathcal{R}_\downarrow, AC, E_\oplus)$ mimics E_{ed} with $\emptyset \subseteq \mathcal{R}_\downarrow \subseteq \mathcal{R}_\oplus$ and $\mathcal{R}$ containing the rules:*

$$x + y \to x + y \qquad 0 \to 0 \qquad \mathsf{e}(x, y) \to \mathsf{e}(x, y)$$
$$\mathsf{d}(\mathsf{e}(x + k_2, k_1), k_1, k_2) \to x \qquad \mathsf{d}(x, k_1, k_2) \to \mathsf{d}(x, k_1, k_2)$$

IV. Complete set of E-variants

To show the relation between rewrite theories mimicking equational theories and the finite variant property, we first need to define the notion of *to-evaluate symbols* and *to-evaluate terms*. For each function symbol $f \in \mathcal{F}$, we associate a function symbol $\overline{f}$ that will correspond to a symbol that needs to be evaluated. We define $\overline{\mathcal{F}} = \{\overline{f} \mid f \in \mathcal{F}\}$. The symbols in $\overline{\mathcal{F}}$ are called *to-evaluate symbols* (TE symbols for short).

Definition 5. *A* to-evaluate term *(TE-term for short) is a term $t \in \mathcal{T}(\overline{\mathcal{F}} \cup \mathcal{F}, \mathcal{X} \cup \mathcal{N})$ such that either $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, or $t = \overline{f}(t_1, \dots, t_n)$, and $t_1, \dots, t_n$ are TE-terms.*

Intuitively, in a to-evaluate term, no to-evaluate symbol in $\overline{\mathcal{F}}$ can occur "below" a standard symbol from $\mathcal{F}$; or in other words, no to-evaluate symbol can occur in a subterm rooted by a symbol from $\mathcal{F}$. Given a term t, we denote by $\overline{t}$ the TE-term obtained from t in which all function symbols f have been replaced by $\overline{f}$.

Example 8. *Consider $T = (>, \mathcal{R}, \emptyset, \emptyset, \emptyset)$ a rewrite theory mimicking standard randomized encryption $E = \{\mathrm{dec}(\mathrm{enc}(x, r, k), k) = x\}$. The set of rewrite rules $\mathcal{R}$ can be composed of the following rules:*

$$\mathrm{dec}(\mathrm{enc}(x, r, k), k) \to x \qquad \mathrm{dec}(x, k) \to \mathrm{dec}(x, k)$$
$$\mathrm{enc}(x, r, y) \to \mathrm{enc}(x, r, y)$$

We have that $\overline{\mathrm{dec}}(\mathrm{enc}(m, r, k), k')$ is a TE-term but $\mathrm{enc}(\overline{\mathrm{dec}}(x, k'), r, k')$ is not a TE-term.

We now define the *close evaluation of TE-terms*, from which we will derive the procedure to check the equality modulo E.

Definition 6. *Let $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ be a rewrite theory. We define the non-deterministic* close evaluation of a TE-term t *into a term s, denoted $t \Downarrow_T s$, as follows:*

$$t \Downarrow_T t \quad \text{if } t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$$
$$\overline{f}(t_1, \dots, t_n) \Downarrow_T r\sigma \quad \text{if } f(\ell_1, \dots, \ell_n) \to r \in \mathcal{R}, \text{ and } \forall i \in \{1, \dots, n\}, t_i \Downarrow_T s_i \text{ and } s_i =_{E_\mathcal{A}} \ell_i\sigma$$

The evaluation on TE-terms intuitively evaluates the function symbols from the bottom up.

Example 9. *Coming back to the rewrite theory T of Example 8, denoting $t = \overline{\mathrm{dec}}(\mathrm{enc}(m, r, k'), k')$ we have $t \Downarrow_T m$ and $t \Downarrow_T \mathrm{dec}(\mathrm{enc}(m, r, k'), k')$. The first evaluation corresponds to the evaluation of $\overline{\mathrm{dec}}$ with the rewrite rule $\mathrm{dec}(\mathrm{enc}(x, r, k), k) \to x$ whereas the second evaluation corresponds to the evaluation of $\overline{\mathrm{dec}}$ with the rewrite rule $\mathrm{dec}(x, k) \to \mathrm{dec}(x, k)$.*

The evaluation on TR-terms allows us to have a simple procedure to test the equality modulo E, as shown in the following result (proof in [43, Appendix C]).

Theorem 1 (Equality modulo E)**.** *Let $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ be a rewrite theory and E a non-trivial equational theory. If T mimics E then for all $t, s \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, $t =_E s$ if and only if there exist t', s' such that $\overline{t} \Downarrow_T t'$, $\overline{s} \Downarrow_T s'$ and $t' =_{E_\mathcal{A}} s'$.*

We can now define the open evaluation of a sequence of TE-terms L as a relation $L \Downarrow'_T (L', \sigma)$ where σ are the instantiations of the variables of L obtained during the evaluation of the TE-symbols in L and where L' is the result of such evaluation.

Definition 7. *Let $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ be a rewrite theory. We define the open evaluation on a sequence of TE-terms L, denoted $L \Downarrow'_T (L', \sigma)$, as follows:*

$$[] \Downarrow'_T ([], \emptyset)$$

$$[t] \Downarrow'_T ([t], \emptyset) \quad \text{if } t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$$

$$[\overline{h}(t_1, \ldots, t_n)] \Downarrow'_T ([u\sigma_u], \sigma'\sigma_u)$$
$$\text{if } [t_1; \ldots; t_n] \Downarrow'_T ([s_1; \ldots; s_n], \sigma'),\ h(u_1, \ldots, u_n) \to u \in \mathcal{R},$$
$$\text{and } \sigma_u \in \mathrm{mgu}_{E_\mathcal{A}}((s_1, \ldots, s_n), (u_1, \ldots, u_n))$$

$$t \cdot L \Downarrow'_T (s\sigma' \cdot L', \sigma\sigma') \quad \text{if } [t] \Downarrow'_T ([s], \sigma) \text{ and } L\sigma \Downarrow'_T (L', \sigma')$$

Example 10. *Coming back to the rewrite theory T of Example 8, we consider the $t = \overline{\mathrm{dec}}(x, \mathrm{dec}(y, b))$. Notice here that only the outer symbol dec is to be evaluated. In particular $[\mathrm{dec}(y, b)] \Downarrow'_T ([\mathrm{dec}(y, b)], \emptyset)$. Therefore, we have: $[t] \Downarrow'_T ([z], \{x \to \mathrm{enc}(z, r, \mathrm{dec}(y, b))\})$ and $[t] \Downarrow'_T ([\mathrm{dec}(x, \mathrm{dec}(y, b))], \emptyset)$.*

The open evaluation intuitively computes the variants of terms contained in the sequence of terms L as shown in the next result (proof in [43, Appendix C]).

Theorem 2 (Complete sets of E-variants)**.** *Let $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ be a rewrite theory and E a non-trivial equational theory. If T mimics E then for all terms t, the set $\{(t', \alpha) \mid [\overline{t}] \Downarrow'_T ([t'], \alpha)\}$ is a complete set of E-variants modulo $E_\mathcal{A}$.*

Corollary 1 (Finite variant property)**.** *Let $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ be a rewrite theory and E a non-trivial equational theory. If T mimics E then E has the finite variant property modulo $E_\mathcal{A}$.*

As the open evaluation allows to compute the finite complete set of variants, it also allows to compute the complete set of most general E-unifiers, as shown below (proof in [43, Appendix C]).

Theorem 3 (Most general E-unifiers)**.** *Let $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ be a rewrite theory and E a non-trivial equational theory. If T mimics E then the set $\{\alpha\sigma_u \mid [\overline{t}, \overline{s}] \Downarrow'_T ([t', s'], \alpha) \wedge \sigma_u \in \mathrm{mgu}_{E_\mathcal{A}}(t', s')\}$ is a complete set of most general E-unifiers of t, s.*

Corollary 1 showed that a rewrite theory mimicking an equational theory implies that it has the finite variant property. In fact we can show that finite variant property and mimicking of equational theories are equivalent definitions when $E_\mathcal{A} = E_\downarrow$, provided that the set of function symbols $\mathcal{F}$ contain some free symbols. In essence, we provide a more *practical* characterisation of the finite variant property.

Theorem 4. *Let $E_\downarrow \subseteq E_\mathcal{A} \subseteq E$ be three non-trivial equational theories. Let $>$ be a $E_\downarrow$-strong reduction ordering. Assume that there exist a binary function $\mathsf{b} \in \mathcal{F}$ and a constant $\mathsf{a}_{\min} \in \mathcal{F}$ such that b does not occur in E and for all ground terms t, $\mathsf{a}_{\min} \not> t$.*

If E has the finite variant property modulo $E_\mathcal{A}$ and

- *either $E_\mathcal{A} = E_\downarrow$*
- *or for all $f/n \in \mathcal{F}$, the complete set S of E-variants modulo $E_\mathcal{A}$ of $\mathsf{b}(f(x_1, \ldots, x_n), \mathsf{b}(x_1, \mathsf{b}(x_2, \ldots \mathsf{b}(x_{n-1}, x_n))))$ satisfies for all $(t, \theta) \in S$, $vars(t_{|1}) \subseteq vars(t_{|2})$ (where $t_{|i}$ denotes the i^{th} subterm of t)*

then there exists $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ that mimics E.

V. Generating rewrite theories

We describe in this section a semi-decision procedure to generate rewrite theories that mimic an equational theory. Contrary to previous sections, where we only assumed the existence of an algorithm to compute the complete set of most general $E_\mathcal{A}$-unifiers, we will also assume in this section that we have at our disposal an algorithm for computing the most general $E_\downarrow$-unifiers. Our approach takes advantage of the following remarks.

First, by relying on most general $E_\downarrow$-unifiers only in the generation of rewrite theories, we greatly limit our reliance on it. Focusing on the automated verification of protocols use case, this means that we only need to run this semi-decision procedure once for each particular equational theory. In particular, this means that if during the protocol verification the user wants to make several small modifications to the protocol steps, but not to the equational theory, then the procedure has not to be run again. Even different protocols using the same equational theory would require the procedure to be run only once. This observation applies to several protocol verification tools such as ProVerif [11], Tamarin [56] and Maude-NPA [57].

Also, although the generic algorithm solely relies on $E_\downarrow$-most general unifiers and will typically terminate only if E has the finite variant property modulo $E_\downarrow$, we will show that our optimisations allow us to reduce the number of variants when focusing on FVP modulo $E_\mathcal{A}$.

Last, although our optimised algorithm can take as input a rewrite system $\mathcal{R}_\downarrow$, the algorithm does not really need that particular knowledge. Indeed, a user could start with an empty $\mathcal{R}_\downarrow$ and the algorithm will by itself augment $\mathcal{R}_\downarrow$ with appropriate rewrite rules. This feature is particularly useful in the context of automated verification of protocols, as users do not necessarily have the knowledge to create a decomposition of E by themselves.

A. Overlapping rewrite rules

The core of the procedure consists in taking two rewrite rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$ where r_1 and ℓ_2 overlap and producing a new rewrite rule.

Definition 8. *Let $E_\downarrow$ be an equational theory. We say that* the rewrite rule $\ell_1 \to r_1$ is $E_\downarrow$-overlapping on a position p with the rewrite rule $\ell_2 \to r_2$ *if $p \in \mathcal{P}os(r_1)$, $r_{1|_p}$ is not a variable and there exists $\sigma \in \mathrm{mgu}_{E_\downarrow}(r_{1|_p}, \ell_2)$.*

The set of rewrite rules merging the two $E_\downarrow$-overlapping rules on p, denoted $(\ell_1 \to r_1 \overset{p,E_\downarrow}{\rhd} \ell_2 \to r_2)$, is defined as:

$$\{\ell_1\sigma \to r_1\sigma[r_2\sigma]_p \mid \sigma \in \mathrm{mgu}_{E_\downarrow}(r_{1|_p}, \ell_2)\}$$

When $p \notin \mathcal{P}os(r_1)$ or $r_{1|_p} \in \mathcal{X}$, $(\ell_1 \to r_1 \overset{p,E_\downarrow}{\rhd} \ell_2 \to r_2) = \emptyset$.

Overlapping rewrite rules are closely related to the classical notion of critical pairs used for example in the Knuth-Bendix completion algorithm. In particular, the critical pairs of $\ell_1 \to r_1$ and $\ell_2 \to r_2$ are the pairs (s,t) such that $s \to t \in (r_1 \to \ell_1 \overset{p,E_\downarrow}{\rhd} \ell_2 \to r_2)$. Notice that here, the overlap is between ℓ_1 and ℓ_2. We denote by $(\ell_1 \to r_1 \overset{p,E_\downarrow}{\bowtie} \ell_2 \to r_2)$ the set of rules from $(\ell_1 \to r_1 \overset{p,E_\downarrow}{\rhd} \ell_2 \to r_2)$ and their opposite orientation rules.

B. The procedure

(NormR)
$$\mathcal{R} \cup \{s \to t\} \rightsquigarrow_{\mathcal{R}_\downarrow, E_\downarrow} \mathcal{R} \cup \{s \to t'\} \qquad \text{if } t =_{E_\downarrow} \circ \to_{\mathcal{R}_\downarrow} t'$$

(NormL)
$$\mathcal{R} \cup \{s \to t\} \rightsquigarrow_{\mathcal{R}_\downarrow, E_\downarrow} \mathcal{R} \cup \{s[s']_i \to t, t \to s[s']_i\}$$
$$\text{if } s = f(s_1, \ldots, s_n), i \in \{1, \ldots, n\}, s_i =_{E_\downarrow} \circ \to_{\mathcal{R}_\downarrow} s'$$

(Eq)
$$\mathcal{R} \cup \{s \to t\} \rightsquigarrow_{\mathcal{R}_\downarrow, E_\downarrow} \mathcal{R} \qquad \text{if } s =_{E_\downarrow} t$$

(Sub)
$$\mathcal{R} \cup \{u \to t\} \rightsquigarrow_{\mathcal{R}_\downarrow, E_\downarrow} \mathcal{R}$$
$$\text{if } (u' \to t') \in \mathcal{R}, C[u'\sigma] =^s_{E_\downarrow} u, C[t'\sigma] =_{E_\downarrow} t,$$
$$t' \not\to_{\mathcal{R}_\downarrow/E_\downarrow} \text{ and } \forall p > \varepsilon.\ u'_{|p} \not\to_{\mathcal{R}_\downarrow/E_\downarrow}$$

Fig. 1. Normalisation rules

The main procedure will consist in generating the rewrite systems $\mathcal{R}$ and $\mathcal{R}_\downarrow$ that will in the end be part of the rewrite theory $(>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$. Since $\mathcal{R}_\downarrow$ in combination with $E_\downarrow$ is used to normalise the rules, the procedure naturally relies on a subroutine that will just normalise the rewrite rules in the rewrite system $\mathcal{R}$. We define the subroutine by a set of normalisation rules on $\mathcal{R}$ displayed in Figure 1. The rule NormR takes a rule $s \to t$ from $\mathcal{R}$ and replaces t by a term t' that t rewrites into, i.e. $t =_{E_\downarrow} \circ \to_{\mathcal{R}_\downarrow} t'$. There may be different possible t', specially because $\mathcal{R}_\downarrow$ is not necessarily $E_\downarrow$-convergent. However, it suffices to take only one of the terms t rewrites into. The rule NormL similarly normalises one of the arguments of the left-hand side. However, it also produces the rule where we swap the two sides. Intuitively, when transforming a rule $s \to t$, we need to show that if an instance of the rule is *well-ordered*, that is when $s\sigma > t\sigma$ then one of the instantiated produced rules should also be well-ordered. In the case of NormR, we know that $t > t'$ as $>$ is $E_\downarrow$-compatible with $\mathcal{R}_\downarrow$. Thus, $t\sigma > t'\sigma$ which implies $s\sigma > t'\sigma$. Hence, we do not need to consider the rule $t' \to s$. In the case of NormL however, such reasoning does not hold as we might have $t\sigma > s[s']_i\sigma$. Therefore, we output both rules $s[s']_i \to t$ and $t \to s[s']_i$.

Note that in [40], when the equational theory can be modelled as a $\emptyset$-convergent rewrite system, their procedure has a similar normalisation rule but does not swap the rules as is done in NormL. Hence, when it is known that $\mathcal{R}_\downarrow$ is $E_\downarrow$-convergent, one could argue that we should add a specific rule where the swap does not occur. However, it is unnecessary thanks to the rule Eq. Assume for simplicity that t is already normalised by NormR, once the rule $t \to s[s']_i$ is added to $\mathcal{R}$, it will be simplified once again by successive applications of the rule NormR. As $\mathcal{R}_\downarrow$ is $E_\downarrow$-convergent, it will produce a rule $t \to t'$ with $t =_{E_\downarrow} t'$ which will disappear by Eq.

Finally, the rule Sub removes a rule that is already subsumed by another rule in $\mathcal{R}$. In the definition of the rule, $=^s_{E_\downarrow}$ denotes the *strict equality modulo* $E_\downarrow$, that is $t =^s_{E_\downarrow} s$ iff $t = f(t_1, \ldots, t_n)$, $s = f(s_1, \ldots, s_n)$ and for all $i \in \{1, \ldots, n\}$, $t_i =_{E_\downarrow} s_i$. Intuitively, this equality modulo $E_\downarrow$ does not affect the root symbol of t and s. Additionally, the rule Sub requires that the rule $s' \to t'$ that subsumes $s \to t$ should be irreducible by $\to_{\mathcal{R}_\downarrow/E_\downarrow}$ (strictly for s').

Definition 9. *A* normalisation step*, denoted $\mathcal{R} \rightsquigarrow_{\mathcal{R}_\downarrow, E_\downarrow} \mathcal{R}'$, is defined when $\mathcal{R}'$ is the result of the application of* NormR*,* NormL*,* Eq *or* Sub *on $\mathcal{R}$. The set $\mathcal{R}$ is said to be normalised when $\mathcal{R} \not\rightsquigarrow_{\mathcal{R}_\downarrow, E_\downarrow}$. We define the function* `normalise`$(\mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow)$ *that computes and returns a normalised $\mathcal{R}'$ such that $\mathcal{R} \rightsquigarrow^*_{\mathcal{R}_\downarrow, E_\downarrow} \mathcal{R}'$.*

Lemma 2. *Let $\mathcal{R}_\downarrow$ be a set of rewrite rules. Let $E_\downarrow$ be an equational theory. Let $>$ be a $E_\downarrow$-strong reduction ordering compatible with $\mathcal{R}_\downarrow$. For all sets of rewrite rules $\mathcal{R}$, the function* `normalise`$(\mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow)$ *terminates.*

Proof. As $>$ is $E_\downarrow$-compatible with $\mathcal{R}_\downarrow$, we have that for all terms t, t', if $t =_{E_\downarrow} \circ \to_{\mathcal{R}_\downarrow} t'$ then $t > t'$. Hence, when the rules NormL and NormR replace a rule $s \to t$, they do it by replacing one of the term (either s or t) by a new term strictly smaller by $>$. As the order $>$ is well-founded, we cannot have an infinite sequence of rules generated by normalisation rules,

and so $\texttt{normalise}(\mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow)$ terminates. □

1 **Function** $\texttt{generate\_rw\_th}(E', \mathcal{R}_\downarrow, E_\downarrow)$
2 $\quad \mathcal{R} := \mathcal{R}_\downarrow \cup \{s \to t, t \to s \mid (s = t) \in E'\}$
3 $\quad \mathcal{R} := \texttt{normalise}(\mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow)$
4 $\quad$ **repeat**
5 $\quad\quad \mathcal{R}' := \emptyset$
6 $\quad\quad \mathcal{R}_0 := \mathcal{R}$
7 $\quad\quad$ **forall** $(\ell_1 \to r_1) \in \mathcal{R} \cup E_\downarrow, (\ell_2 \to r_2) \in \mathcal{R}$, *position* p **do**
8 $\quad\quad\quad \mathcal{R}' := \mathcal{R}' \cup (r_1 \to \ell_1 \overset{p,E_\downarrow}{\bowtie} \ell_2 \to r_2)$
9 $\quad\quad\quad \mathcal{R}' := \mathcal{R}' \cup (\ell_1 \to r_1 \overset{p,E_\downarrow}{\bowtie} \ell_2 \to r_2)$
10 $\quad\quad\quad \mathcal{R}' := \mathcal{R}' \cup (r_1 \to \ell_1)$
11 $\quad\quad$ **end**
12 $\quad\quad \mathcal{R} := \texttt{normalise}(\mathcal{R}' \cup \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow)$
13 $\quad$ **until** $\mathcal{R} = \mathcal{R}_0$
14 $\quad \mathcal{R} = \mathcal{R} \cup \{f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n) \mid f/n \in \mathcal{F}\}$
15 $\quad$ **return** $\mathcal{R}$

Algorithm 1: Generic generation of rewrite theories

We can now define the main procedure displayed in Algorithm 1. The set of rewrite rules $\mathcal{R}$ gathers the rules generated throughout the algorithm. It is initialised on Line 2 with all the rules in E', in both orientations, as well as all the rules in $\mathcal{R}_\downarrow$. Note that we do not consider both orientations for the rules in $\mathcal{R}_\downarrow$. Intuitively, in the proof (see Section V-C for an overview), we will only consider applications of rewrite rules that follow the ordering $>$, i.e. if $t \to_{\mathcal{R},E_\downarrow} s$ by some rule $\ell \to r$ with substitution σ then $\ell\sigma > r\sigma$ or $\ell\sigma =_{E_\downarrow} r\sigma$. As such, since any rule $(\ell \to r) \in \mathcal{R}_\downarrow$ satisfies $\ell > r$, we will never consider an application of the rule $r \to \ell$.

After a first normalisation on Line 3, the algorithm enters the main loop which merges $E_\downarrow$-overlapping rewrite rules from $\mathcal{R}$, and then normalises these newly generated rewrite rules with the current rules in $\mathcal{R}$. The process repeats until we reach a fixpoint on $\mathcal{R}$. On Line 7, for sake of readability, we write $\mathcal{R} \cup E_\downarrow$ for the set $\mathcal{R} \cup \{\ell \to r, r \to \ell \mid (\ell = r) \in E_\downarrow\}$. When looking at the two rewrite rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$, we consider the two cases, that are when ℓ_1 overlaps with ℓ_2 (Line 8) and when r_1 overlaps with ℓ_2 (Line 9).

Notice that Line 10 also adds all the rules in $\mathcal{R}$ with opposite orientation. Indeed, similarly to the initialisation, the algorithm aims to maintain that rules with both orientations should occur in $\mathcal{R}$. But this may be disrupted by normalising with a non-$E_\downarrow$-convergent $\mathcal{R}_\downarrow$. Hence, Line 10 ensures the invariant. Finally, upon exiting the loop, the algorithm adds to $\mathcal{R}$ on Line 14 the rule $f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)$ to satisfy Item **S3** of Definition 2.

Before stating the theorem indicating the correctness of Algorithm 1, we need to introduce a final assumption on $E_\downarrow$: we will require *$E_\downarrow$ to have finite equivalence classes*, that is for all terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, the set $\{t' \mid t =_{E_\downarrow} t'\}$ is finite. This is a strong assumption that is satisfied, amongst others, by AC, C (commutative) and A (associative) equational theories, hence allowing us to cover all the relevant equational theories. For instance, in our implemented prototype (see Section VII), we have focused on $E_\downarrow$ being AC. Nevertheless, showing the correctness of the algorithm without this assumption remains open.

Theorem 5. *Let $E_\downarrow, E'$ be two equational theories such that $E_\downarrow$ has finite equivalence classes. Let $\mathcal{R}_\downarrow$ be a set of rewrite rules. Let $>$ be a $E_\downarrow$-strong reduction ordering compatible with $\mathcal{R}_\downarrow$. Let $E = E' \cup \mathcal{R}_\downarrow^{=} \cup E_\downarrow$.*

If E is not trivial and $\texttt{generate\_rw\_th}(E', \mathcal{R}_\downarrow, E_\downarrow)$ terminates and returns $\mathcal{R}$ such that for all $(\ell \to r) \in \mathcal{R}$, $vars(r) \subseteq vars(l)$, then $(>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\downarrow)$ is a rewrite theory that mimics E.

Remark that when $\mathcal{R}_\downarrow = \emptyset$, the normalisation rules do not affect the rewrite system $\mathcal{R}$ given as input, that is $\texttt{normalise}(\mathcal{R}, \emptyset, E_\downarrow) = \mathcal{R}$. Hence, for the algorithm to effectively work, it is preferable to provide a rewrite system $\mathcal{R}_\downarrow$ as large as possible (e.g. one that is $E_\downarrow$-convergent with E). However, to avoid for users the requirement to manually provide this rewrite system, we present in Section VI some optimisations that allow the prototype to start with $\mathcal{R}_\downarrow = \emptyset$ and to gradually augment it during the execution of the procedure.

Comparison with the procedures in [40]: Our algorithm is a direct generalization of the two algorithms presented in [40]: when E is oriented as a convergent rewrite system $\mathcal{R}_\downarrow$, Algorithm 1 of [40] actually computes $\texttt{generate\_rw\_th}(\emptyset, \mathcal{R}_\downarrow, \emptyset)$. When E is a linear equational theory, Algorithm 2 of [40] actually computes $\texttt{generate\_rw\_th}(E, \emptyset, \emptyset)$. A direct consequence of Theorem 5 is that Algorithm 2 presented in [40] for linear equational theories was in fact sound for any equational theory.

C. Overview of the proof of Theorem 5

The complete proof can be found in [43, Appendix D]. We place ourselves within the hypotheses of Theorem 5, which are: $E_\mathcal{A} = E_\downarrow$, and $>$ is a $E_\downarrow$-strong reduction order compatible with $\mathcal{R}_\downarrow$, and $\texttt{generate\_rw\_th}(E', \mathcal{R}_\downarrow, E_\downarrow)$ terminates and returns $\mathcal{R}$ such that for all $(\ell \to r) \in \mathcal{R}$, $vars(r) \subseteq vars(\ell)$. Let us denote $E = E' \cup \mathcal{R}_\downarrow^{=} \cup E_\downarrow$ and $T = (>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\downarrow)$.

1) T is a rewrite theory: Showing that T is a rewrite theory is a simple matter as Items **S1** and **S2** are given as assumptions. Moreover, Item **S3** is guaranteed by Line 14 of Algorithm 1 and by our assumption that for all $(\ell \to r) \in \mathcal{R}$, $vars(r) \subseteq vars(\ell)$.

2) Towards T mimics E: Amongst the three properties required to show that T mimics E, only the last one, i.e. Item **M3**, is difficult. Item **M1** is directly obtained, since $E_\downarrow = E_\mathcal{A}$ and $E_\downarrow \cup \mathcal{R}_\downarrow^{=} \cup E' = E$. The proof of Item **M2** is mostly given by the following lemma.

Lemma 3. *Let $\ell_1 \to r_1$ and $\ell_2 \to r_2$ be two rewrite rules such that $\ell_1 =_E r_1$ and $\ell_2 =_E r_2$. For all positions p, for all $\ell_3 \to r_3 \in (\ell_1 \to r_1 \overset{p,E_\downarrow}{\rhd} \ell_2 \to r_2)$, $\ell_3 =_E r_3$.*

Proof. If $\ell_3 \to r_3 \in (\ell_1 \to r_1 \overset{p,E_\downarrow}{\rhd} \ell_2 \to r_2)$ then $p \in \mathcal{P}os(r_1)$ and there exists $\sigma \in \mathrm{mgu}_{E_\downarrow}(r_{1|p}, \ell_2)$ such that $\ell_3 = \ell_1\sigma$ and $r_3 = r_1\sigma[r_2\sigma]_p$. $\ell_2 =_E r_2$ implies $\ell_2\sigma =_E r_2\sigma$. Similarly, $\ell_1\sigma =_E r_1\sigma$. Since σ is a $E_\downarrow$-unifier of $r_{1|p}$ and ℓ_2, and since $E_\downarrow \subseteq E$, we have $r_{1|p}\sigma =_E \ell_2\sigma =_E r_2\sigma$. This implies that $r_1\sigma = r_1\sigma[r_{1|p}\sigma]_p =_E r_1\sigma[r_2\sigma]_p$. Hence $\ell_3 = \ell_1\sigma =_E r_1\sigma =_E r_1\sigma[r_2\sigma]_p = r_3$. □

Notice that all rules $\ell \to r$ in the initial value $\mathcal{R}$ on Line 2 of Algorithm 1 satisfy $\ell =_E r$. By applying Lemma 3 and noticing that the normalisation rules NORML and NORMR preserve this invariant, since $\mathcal{R}_\downarrow{}^{=} \cup E_\downarrow \subseteq E$, we obtain that $\mathcal{R}$ satisfies Item **M2**.

3) A mountainous landscape of equality modulo E: Consider two terms t and s such that $t =_E s$. The definition of $=_E$ is given by being the least congruence such that $u\sigma =_E v\sigma$ for all equations $u = v \in E$. Another way of viewing this definition is that there exists a finite sequence $t = t_0 \to_{\mathcal{R}_E} t_1 \to_{\mathcal{R}_E} \ldots \to_{\mathcal{R}_E} t_{n-1} \to_{\mathcal{R}_E} t_n = s$ where $\mathcal{R}_E$ are the rules $\ell \to r$ such that $(\ell = r)$ or $(r = \ell)$ is in E.

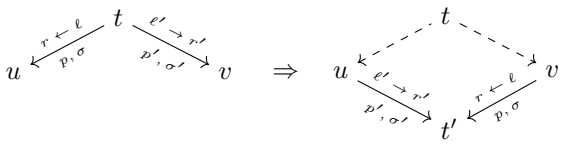
Let us assume for the moment that all $t_0, \ldots, t_n$ are ground. If p_i, σ_i and $\ell_i \to r_i$ are respectively the position, the substitution and the rewrite rule used in the rewrite step $t_{i-1} \to_{\mathcal{R}_E} t_i$ then, as $>$ is a $E_\downarrow$-total, we know that either $\ell_i\sigma_i > r_i\sigma_i$ or $\ell_i\sigma_i =_{E_\downarrow} r_i\sigma_i$ or $\ell_i\sigma_i < r_i\sigma_i$. The main idea behind the proof is to only consider rewrite steps that *follow the order* $>$. Formally, we write $t \xrightarrow[p,\sigma]{\ell \to r} s$ when $t_{|p} = \ell\sigma$ and $s = t[r\sigma]_p$ and ($\ell\sigma > r\sigma$ or $\ell\sigma =_{E_\downarrow} r\sigma$). Similarly, we write $s \xleftarrow[p,\sigma]{r \leftarrow \ell} t$ when $t \xrightarrow[p,\sigma]{\ell \to r} s$.

Of course, when a rule $\ell \to r$ is not already ordered by $>$, i.e. $\ell > r$, there may be some substitutions σ for which $\ell\sigma > r\sigma$ and some substitutions σ' for which $\ell\sigma' < r\sigma'$. This explains why in the initial set $\mathcal{R}$ defined in Line 2 of Algorithm 1, when $\ell = r \in E'$, both $\ell \to r$ and $r \to \ell$ are in $\mathcal{R}$, as well as why only $\mathcal{R}_\downarrow \subseteq \mathcal{R}$ and not the rules in $\mathcal{R}_\downarrow$ with opposite orientation.

Using this ordered rewrite step, we can graphically represent the equality modulo E, $t =_E s$ as a mountainous landscape with peaks, plateaus and valleys (see Figure 2) where each increase or decrease of altitude is due to an ordered rewrite step from a rule in $\mathcal{R}$. The first part of the proof intuitively consists in transforming the mountain into a single valley. To do so, we will apply successive transformations that will replace local peaks into local valleys, until no peak remains.

4) Transforming peaks into valleys: In this section, we present a subset of the transformations needed to reshape the mountainous landscape. In particular, we first start by looking at peaks of the form $u \xleftarrow[p,\sigma]{r \leftarrow \ell} t \xrightarrow[p',\sigma']{\ell' \to r'} v$.

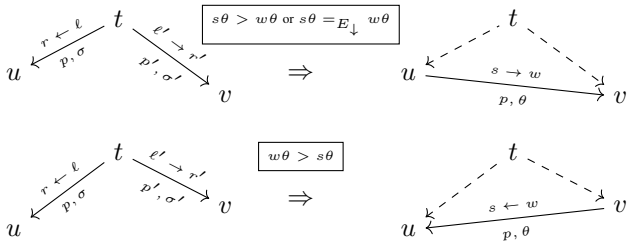
a) Peak with parallel positions.: Assume that $p \parallel p'$, which means that p and p' are not prefix of each other. We know that $\ell\sigma = t_{|p}$ and $\ell'\sigma' = t_{|p'}$. Since $p \parallel p'$, t is in fact of the form $t = C[\ell\sigma, \ell'\sigma']$ with $C[\_,\_]$ a term context. Hence, taking $t' = C[r\sigma, r'\sigma']$, we have $u \xrightarrow[p',\sigma']{\ell' \to r'} t'$ and $t' \xleftarrow[p,\sigma]{r \leftarrow \ell} v$. Graphically, we thus applied the following transformation:



Since $>$ is an $E_\downarrow$-compatible reduction order, $\ell\sigma > r\sigma$ implies $t > u$ and $v > t'$. Similarly, $\ell'\sigma'$ implies $t > v$ and $u > t'$. Therefore $t > t'$. Notice that the peak, whose highest altitude was represented by t, becomes a local valley whose highest altitude is either u or v.

b) Peak with overlapping positions.: When the positions p and p' are not parallel, we cannot apply the previous transformation. However, we can rely on the rules obtained by merging $E_\downarrow$-overlapping rules. Assume that $p' = p \cdot q$, i.e. p is a prefix of p', and $q \in \mathcal{P}os(\ell)$ and $\ell_{|q} \notin \mathcal{X}$. In such a case, $\ell_{|q}\sigma = \ell'\sigma'$. W.l.o.g., we assume that distinct rules in the mountainous landscape have distinct variables. We can therefore define $\gamma = \sigma \cup \sigma'$ yielding $\ell_{|q}\gamma = \ell'\gamma$ and so $\ell_{|q}$ and ℓ' being unifiable. Hence, not only there exist $\alpha \in \mathrm{mgu}_{E_\downarrow}(\ell_{|q}, \ell')$ and θ such that $\gamma =_{E_\downarrow} (\alpha\theta)_{|dom(\gamma)}$; but the rule $r \to \ell$ is also $E_\downarrow$-overlapping on q with $\ell' \to r'$. Hence, we can find $(s \to w) \in (r \to \ell \overset{q}{\rhd} \ell' \to r')$ such that $s\theta =_{E_\downarrow} r\sigma$ and $w\theta =_{E_\downarrow} \ell\sigma[r'\sigma']_q$.

Note that we cannot deduce how $s\theta$ and $w\theta$ are ordered, i.e. whether $s\theta > w\theta$ or $s\theta =_{E_\downarrow} w\theta$ or $w\theta > s\theta$. We thus consider two cases when transforming the peak. When $E_\downarrow = \emptyset$, the two transformations are graphically represented below.



These two cases explain why in Line 7 of Algorithm 1, we augment $\mathcal{R}'$ with $r \to \ell \overset{q}{\bowtie} \ell' \to r'$.

When $E_\downarrow \neq \emptyset$, we cannot apply exactly the same transformation since, in the mountainous landscape, all rewrite steps are syntactic and not modulo $E_\downarrow$. However, $s\theta =_{E_\downarrow} r\sigma$ implies that there exists a sequence of rewrite steps from $s\theta$ to $r\sigma$ using only rewrite rules in $E_\downarrow$. To discuss about such sequences more easily, we write $u \overset{\mathsf{tr}}{\Longleftrightarrow} v$ when tr is a *rewrite trace*, *i.e.* a sequence of rewrite step arguments $(p_1, \sigma_1 : \ell_1 \sim_1 r_1) \ldots (p_n, \sigma_n : \ell_n \sim_n r_n)$ with $\sim_1, \ldots, \sim_n \in \{\leftarrow, \to\}$ and when there exist terms $t_0, \ldots, t_n$ such that $t_0 = u$, $t_n = v$ and for all $i \in \{1, \ldots, n\}$,

$t_{i-1} \xrightarrow[p_i,\sigma_i]{\ell_i \to r_i} t_i$ when $\sim_i = \to$ and $t_{i-1} \xleftarrow[p_i,\sigma_i]{\ell_i \leftarrow r_i} t_i$ otherwise.

We call each $(p_i, \sigma_i : \ell_i \sim_i r_i)$ a *rewrite label*. When tr contains only right (resp. left) oriented rewrite rules, we say that it is a *right (resp. left) rewrite trace*, and we denote $u \overset{\mathsf{tr}}{\Longrightarrow} v$ (resp. $u \overset{\mathsf{tr}}{\Longleftarrow} v$).

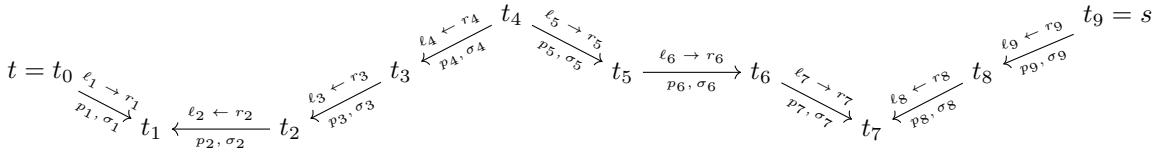
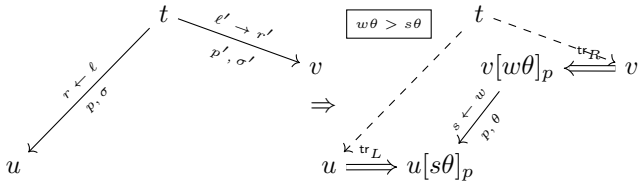


Fig. 2. Mountainous landscape of $t =_E s$

Coming back to the peak of our landscape, $s\theta =_{E_\downarrow} r\sigma$ and $w\theta =_{E_\downarrow} \ell\sigma[r'\sigma']_q$ imply $u =_{E_\downarrow} u[s\theta]_p$ and $v =_{E_\downarrow} v[w\theta]_p$, which in turn imply that there exist two rewrite traces tr_L and tr_R with rules only in $E_\downarrow$ such that $u \xRightarrow{\mathsf{tr}_L} u[s\theta]_p$ and $v[w\theta]_p \xLeftarrow{\mathsf{tr}_R} v$. We can therefore amend our transformations to add these rewrite traces. For example, when $w\theta > s\theta$, the transformation can be graphically represented as follows:

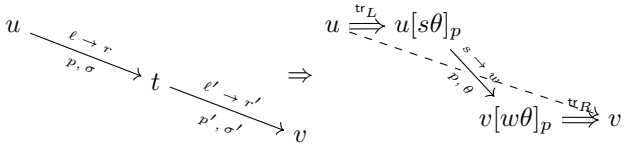


Although the *altitude* of the mountainous landscape decreases, the presence of tr_L and tr_R may increase its *length*, i.e. the number of rewrite steps. This will be taken into account when showing that repeated applications of transformations necessarily terminate.

For brevity, we omit here the other transformations used to remove peaks, for instance when $p' = p \cdot q$ and $q \notin \mathcal{P}os(\ell)$ or $\ell_{|q} \in \mathcal{X}$. They can however be found in [43, Appendix D].

5) Ordering slopes by decreasing position: In addition to removing peaks of the mountainous landscape, we also order the rewrite steps on a slope by decreasing position. Intuitively, our aim is to ensure that if the rewrite steps $u \xrightarrow[p,\sigma]{\ell \to r} t \xRightarrow{\mathsf{tr}} w \xrightarrow[p',\sigma']{\ell' \to r'} v$ are part of our mountainous landscape then either $p \mid\mid p'$ or $p' < p$. In other words, if $u \xRightarrow{\mathsf{tr}} v$ then any rule that affects the root symbol of u should be the last rule applied in tr. Once again, we can achieve this by transforming the landscape.

Consider for example the slope $u \xrightarrow[p,\sigma]{\ell \to r} t \xrightarrow[p',\sigma']{\ell' \to r'} v$ where $p' = p \cdot q$, i.e. p is a prefix of p', and $q \in \mathcal{P}os(\ell)$ and $\ell_{|q} \notin \mathcal{X}$ with $\ell'\sigma' > r'\sigma'$. These two rewrite steps are not ordered by decreasing position. Similarly to our transformation that removes peaks with overlapping positions, there exist $(s \to w) \in (\ell \to r \overset{q}{\triangleright} \ell' \to r')$ and a substitution θ such that $s\theta =_{E_\downarrow} \ell\sigma$ and $w\theta =_{E_\downarrow} r\sigma[r'\sigma']_q$. In this case, one can show that we necessarily have $s\theta > w\theta$ as $>$ is a $E_\downarrow$-strong reduction ordering and $\ell'\sigma' > r'\sigma'$. Once again, $s\theta =_{E_\downarrow} \ell\sigma$ and $w\theta =_{E_\downarrow} r\sigma[r'\sigma']_q$ imply that there exist two rewrite traces tr_L and tr_R with rules only in $E_\downarrow$ such that $u \xRightarrow{\mathsf{tr}_L} u[s\theta]_p$ and $v[w\theta]_p \xRightarrow{\mathsf{tr}_R} v$. The transformation is graphically represented below.



When $E_\downarrow = \emptyset$, as $u = u[s\theta]_p$ and $v[w\theta]_p = v$, the sequences tr_L and tr_R would be empty hence the number of wrongly ordered rewrite steps would decrease. However, when $E_\downarrow \neq \emptyset$, tr_L and tr_R may introduce wrongly ordered rewrite steps. Therefore, we relax the notion of rewrite steps ordered by decreasing position by only looking at cases where $\ell'\sigma' > r'\sigma'$, thus excluding rules in $E_\downarrow$.

Definition 10. *A right rewrite trace* tr *is* ordered by decreasing position *when for all sub-traces of* tr *of the form* $(p, \sigma : \ell \to r)\mathsf{tr}'(p', \sigma' : \ell' \to r')$*, if* $\ell'\sigma' > r'\sigma'$ *then* $p \mid\mid p'$ *or* $p' < p$*. Similarly, a left rewrite trace* tr *is ordered by decreasing position when for all sub-traces of* tr *of the form* $(p, \sigma : r \leftarrow \ell)\mathsf{tr}'(p', \sigma' : r' \leftarrow \ell')$*, if* $\ell\sigma > r\sigma$ *then* $p \mid\mid p'$ *or* $p < p'$*.*

With this new definition, as tr_R only contains rules from $E_\downarrow$, the rewrite trace $(p, \theta : s \to w)\mathsf{tr}_R$ is naturally ordered by decreasing position. To handle tr_L, one can notice that in the equality $s\theta =_{E_\downarrow} \ell\sigma$, we in fact have $s = \ell\alpha$ with $(\alpha\theta)_{|dom(\sigma)} =_{E_\downarrow} \sigma$. Thus, after showing that ℓ cannot be a variable as $E_\downarrow$ has finite equivalence classes and $>$ has the subterm property on ground terms, we thus obtain $\ell\sigma \xRightarrow{\mathsf{tr}'_L} s\theta$ for some tr'_L where the positions in tr'_L are all different from the root position ε, i.e. they do not affect the root of ℓ. Therefore, we deduce that there exists $u \xRightarrow{\mathsf{tr}_L} u[s\theta]_p$ where p is a strict prefix of all positions of tr_L.

The proof of Theorem 5 and in particular the proof of Item **M3** from Definition 4 is completed by showing that when no more landscape transformation rule is applicable, the rewrite trace is a valley whose slopes are ordered by decreasing position. In such a case, as $f(t_1, \ldots, t_n) =_E t$, $\mathcal{M} = \{t_1, \ldots, t_n, t\}$ and $\mathrm{nf}_{T,E}(\mathcal{M})$ holds, we can show that the ordered slopes is in fact only composed of a single rule, that is the one used to prove Item **M3**.

VI. Optimisations

The procedure presented above has two main flaws: it may not terminate and it may produce rewrite rules where variables of the right hand side are not all included in the left hand side, the latter thus possibly violating the requirement from Item **S3** of Definition 2. Although we will never be able to guarantee termination (otherwise all equational theories would satisfy the

finite variant property, which is not the case), we suggest in this section some optimisations that will help the procedure terminate more often. Moreover, we will also propose an additional optimisation that will ensure that all rewrite rules generated have variables in their right hand side occurring in their left-hand side.

A. Dealing with right-hand side variables

Recall that $>$ is a $E_\downarrow$-strong reduction order and $E_\downarrow$ has finite equivalence classes. Without restriction, the smallest ground term by $>$ could be either a name or a constant. It cannot be any other term as $>$ satisfies the subterm property on ground term, i.e. any strict subterm is strictly smaller than the term itself. To apply our optimisation, we will assume that the smallest ground term by $>$, denoted $\mathsf{a}_{\min}$, is a constant that does not occur in $E_\downarrow$. In such a way, $\mathsf{a}_{\min}$ will be able to appear within our rewrite rules and not be affected by $E_\downarrow$. In practice, we can always augment $\mathcal{F}$ with a new constant disjoint from $\mathcal{F}$, and define a new reduction from $>$ where this constant is minimal.

The main idea of the rule comes from the following simple observation. Assume that $s =_E t$ and $x \in vars(t) \setminus vars(s)$. Thus, for all terms u_1, u_2, denoting $\sigma_1 = \{x \mapsto u_1\}$ and $\sigma_2 = \{x \mapsto u_2\}$, we have $t\sigma_1 =_E s =_E t\sigma_2$. In other words, the value of x does not really matter. Thus, when $s \to t$ occurs in $\mathcal{R}$, we can replace x by the minimal term $\mathsf{a}_{\min}$. In order to achieve correctness, we also introduce the rewrite rule $t \to t\{x \mapsto \mathsf{a}_{\min}\}$. Therefore, we augment the set of normalisation rules with the following rule VAR.

$$\begin{array}{l}(\textsc{Var})\\ \mathcal{R} \cup \{s \to t\} \rightsquigarrow_{\mathcal{R}_\downarrow, E_\downarrow} \mathcal{R} \cup \{s \to t\rho, t \to t\rho\}\\ \quad \text{if } \rho : V \to \{\mathsf{a}_{\min}\} \text{ and } V = (vars(t) \setminus vars(s)) \neq \emptyset\end{array}$$

For example, this optimisation allows our prototype to generate a rewrite theory for the equational theory $\{g(x,y) = f(x,z); h(x) = a; f(a,z) = p(x,z)\}$.

B. Checking the order of rules

So far, we assumed the existence of an $E_\downarrow$-strong reduction order but it is only used in the proof and not in the algorithm itself. However, if we have an effective way of testing whether $t > s$ then we can remove any rule $s \to t$ from $\mathcal{R}$ such that $t > s$ since they will never be used in a rewrite trace between two terms. Indeed, $u \xrightarrow[p,\sigma]{s \to t} v$ and $u \xleftarrow[p,\sigma]{t \leftarrow s} v$ requires that either $s\sigma =_{E_\downarrow} t\sigma$ or $s\sigma > t\sigma$; and $t > s$ implies $t\sigma > s\sigma$ for all σ. In a similar fashion, as $>$ satisfies the subterm property on ground terms, s cannot be a strict subterm of t as it would imply $t\sigma > s\sigma$. We can therefore augment the set of normalisation rules with the following rule ORD.

$$\begin{array}{l}(\textsc{Ord})\\ \mathcal{R} \cup \{s \to t\} \rightsquigarrow_{\mathcal{R}_\downarrow, E_\downarrow} \mathcal{R} \quad \text{if } t > s \text{ or } s \in \mathrm{st}^s(t)\end{array}$$

On the other hand, when a rule $s \to t$ already satisfies $s > t$, we also have the opportunity to add it to $\mathcal{R}_\downarrow$ as the only requirement on $\mathcal{R}_\downarrow$ is to be $E_\downarrow$-terminating. When it occurs, we restart the algorithm by re-initialising $\mathcal{R}$ to its initial value with the augmented $\mathcal{R}_\downarrow$. This optimisation allows the prototype to terminate on the equational theory $\{\exp(\mathrm{g}, \mathrm{one}) = g; \exp(\exp(\mathrm{g}, x), y) = \exp(\exp(\mathrm{g}, y), x)\}$.

C. Subsumed rules modulo $E_\mathcal{A}$

Theorem 5 generates a rewrite theory mimicking E, of the form $(>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\downarrow)$, where the equational theory $E_\downarrow$ used for normalisation is also used for the computation of most general unifiers. As mentioned at the beginning of the paper, we aim for rewrite theories of the form $(>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$. Of course, when $E_\downarrow \subseteq E_\mathcal{A}$, it is easy to see that if $(>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\downarrow)$ mimics E then so does $(>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$. However, many of the rules in $\mathcal{R}$ would become superfluous: typically, when two rules are equal modulo $E_\mathcal{A}$, only one of them is needed. We therefore consider a function that will remove these superfluous rules (Algorithm 2).

Function `cleanup`$(\mathcal{R}, E_\mathcal{A})$
  while $\mathcal{R} = \mathcal{R}' \cup \{s \to r, s' \to r'\}$ *and* $\exists \alpha$ *s.t.* $s\alpha =^s_{E_\mathcal{A}} s'$ *and* $r\alpha =_{E_\mathcal{A}} r'$ **do** $\mathcal{R} := \mathcal{R}' \cup \{s \to r\}$
  return $\mathcal{R}$

Algorithm 2: Removing superfluous rewrite rules

Correctness of this function is given below (proof in [43, Appendix E]).

Lemma 4. *If $(>, \mathcal{R}, \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ is a rewrite theory mimicking an equational theory E and $\mathcal{R}' =$ `cleanup`$(\mathcal{R}, E_\mathcal{A})$ then $(>, \mathcal{R}', \mathcal{R}_\downarrow, E_\downarrow, E_\mathcal{A})$ is a rewrite theory mimicking E.*

VII. Experimentation Results and Discussion

We developed a prototype implementing our semi-decision procedure in OCaml and used Maude version 3.4 [59] as backend. Our source code and examples are available in [42]. The prototype currently natively only support matching and unification modulo AC, meaning that given an input equational theory E, and an optional rewrite system $\mathcal{R}_\downarrow$, the prototype will execute `generate_rw_th`$(E, \mathcal{R}_\downarrow, AC)$, returning a rewrite system $\mathcal{R}$. In Table I, we provide an extract of our experimental results on different equational theories, indicating the equations we consider, the computation time on a MacBook Pro M2 8-core with 24GB of memory, the number of rewrite rules generated by the algorithm (# rules), the size of $\mathcal{R}$ ($|\mathcal{R}|$) and whether our prototype was able to determine that $\mathcal{R}$ is convergent (Conv). Finally, we also indicate whether a rewrite system $\mathcal{R}_\downarrow$ was initially provided to the prototype ($\mathcal{R}_\downarrow$). We explain below how our prototype check convergence.

In Table I, we denote by AC$(+)$ the equations for associativity and commutativity of the symbol $+$. In [27], it was shown that adding to XOR an homomorphic symbol h with the equation $h(x + y) = h(x) + h(y)$ yields an equational theory that does not have the FVP modulo AC. However, we retrieve the FVP modulo AC when considering different operators $+_{\text{in}}$ and $+_{\text{out}}$ with the equation $\{h(x +_{\text{in}} y) =$

Acronym	Equations	time	# rules	$\lvert\mathcal{R}\rvert$	Conv	$\mathcal{R}_{\downarrow}$
ACI	$\mathrm{AC}(+) \cup \{x + x = x\}$	5s	54k	6	yes	no
ACN	$\mathrm{AC}(+) \cup \{x + x = 0\}$	35s	399k	14	yes	no
XOR	$\mathrm{AC}(+) \cup \{x + 0 = x, x + x = 0\}$	1s	38k	8	yes	no
XORed	$\mathrm{XOR} \cup \{\mathrm{d}(\mathrm{e}(x + k_2, k_1), k_1, k_2) = x\}$	3s	56k	17	yes	no
$\mathrm{H}(+_{\mathrm{in}}, +_{\mathrm{out}})$	$\{h(x +_{\mathrm{in}} y) = h(x) +_{\mathrm{out}} h(y)\}$	1s	10	5	yes	no
$\mathrm{XOR}h_{\neq}$	$\mathrm{XOR} \cup \mathrm{H}(+, +_{\mathrm{out}})$	2s	40k	17	yes	no
$\mathcal{AG}$	$\mathrm{AC}(*) \cup \{x * 1 = x, x * x^{-1} = 1\}$	7min 26s	530k	52	yes	yes
DH	$\mathcal{AG} \cup \{x \hat{} 1 = x, (x \hat{} y) \hat{} z = x \hat{} (y * z)\}$	17min	828k	99	yes	yes
Bilinear Pairing	$\mathrm{DH} \cup \{(x \cdot y) \cdot z = x \cdot (y * z), x \cdot 1 = x,$ $\mathrm{em}(x \cdot y, z) = \mathrm{em}(x, z) * y, \mathrm{em}(z, x \cdot y) = \mathrm{em}(z, x) * y\}$	1h 14min	1402k	194	no	yes
OldDHWeak	$\{\mathrm{g}_w \hat{} x = x, 1 \hat{} x = 1, (\mathrm{g} \hat{} y) \hat{} z = (\mathrm{g} \hat{} z) \hat{} y\}$	1s	32	10	no	no
$\mathcal{AG}h_{\neq}$	$\mathcal{AG} \cup H(*, *_{\mathrm{out}})$	16min 42s	815k	101	yes	yes
EG-Mixnet [58]	$\mathrm{AC}(*) \cup \{(x \hat{} y) \hat{} z = x \hat{} (y * z), \mathrm{dec}(\mathrm{enc}(m, x, x \hat{} y, r), x, y) = m,$ $\mathrm{check}(\mathrm{sign}(m, x, s), x, x \hat{} s) = m, \mathrm{get}(\mathrm{sign}(m, x, s)) = m\}$	1s	276	15	yes	no
EG-Renc [58]	EG-Mixnet $\cup$ AC$(+)$ $\cup \{renc(enc(m, x, x \hat{} y, r), r', x, x \hat{} y) = enc(m, x, x \hat{} y, r + r')\}$	1s	428	19	yes	no
$\mathrm{Enc}h_{\neq}$	$\mathrm{AC}(+) \cup \{\mathrm{enc}(x, z) +_{\mathrm{out}} \mathrm{enc}(y, z) = \mathrm{enc}(x + y, z), \mathrm{dec}(\mathrm{enc}(x, y), y) = x\}$	1s	24	8	yes	no
$\mathrm{AEnc}h_{\neq}$	$\mathrm{AC}(+) \cup \{\mathrm{enc}(x, \mathrm{pk}(z)) +_{\mathrm{out}} \mathrm{enc}(y, \mathrm{pk}(z)) = \mathrm{enc}(x + y, \mathrm{pk}(z)), \mathrm{dec}(\mathrm{enc}(x, \mathrm{pk}(y)), y) = x\}$	1s	24	7	yes	no
Blind Signature	$\{\mathrm{unblind}(\mathrm{blind}(x, y), y) = x, \mathrm{get}(\mathrm{sign}(x, y)) = x,$ $\mathrm{unblind}(\mathrm{sign}(\mathrm{blind}(x, y), z), y) = \mathrm{sign}(x, z), \mathrm{verify}(\mathrm{sign}(x, y), \mathrm{vk}(y)) = \mathrm{ok}\}$	1s	72	11	yes	no
Toy example 1	$\{g(x, y) = f(x, z), h(x) = 0, f(0, z) = p(x, z)\}$	1s	218	14	no	no
Toy example 2	$\{g(x, y) = f(x, x, z), h(x, x) = 0, f(0, x, z) = p(x, z)\}$	1s	159	12	no	no
Toy example 3	$\{\mathrm{dh}(x, \mathrm{pk}(y)) = \mathrm{dh}(y, \mathrm{pk}(x)), \mathrm{dh}(x, \mathrm{invalid}) = \mathrm{invalid}\}$	1s	21	5	no	no

TABLE I
EXTRACT OF EQUATIONAL THEORIES HANDLED BY OUR PROTOTYPE.

$h(x)+_{\mathrm{out}}h(y)\}$. In the case of XOR for example, the equations $\mathrm{XOR} \cup H(+, +_{\mathrm{out}})$ ensure that $+_{\mathrm{out}}$ is also AC, nilpotent with a unit element but only over the outputs of the h function (not over all terms). The same technique applies to Abelian groups with an homomorphic symbol and (a)symmetric homomorphic encryption/decryption scheme.

We also consider more complex equations used in the literature. For instance, we consider a model of ElGamal asymmetric encryption used to model Exponentiation Mix-Nets in [58]. We also consider the Bilinear Pairing equations used in Tamarin, which is basically Diffie Hellman (DH) augmented with a bilinear map em and a scalar multiplier $\cdot$. We relied on the generic definition where the function em is not commutative, corresponding to the case where the groups used on the left and right arguments of em may be different. Interestingly, when adding the commutative property to em, our algorithm seems *stuck* in the computation of an AC-unification in Maude. However, when considering em to be both associative and commutative, our algorithm concludes and generates the same number of rewrite rules as the non-commutative case. This can be explained by the fact that our prototype currently only implements natively AC unification and not commutative-only unification. Therefore the algorithm needs to take care of the commutative property which seems to lead to some very costly unification. In a future version of our prototype with commutative unification implemented, we expect the bilinear pairing with a commutative em to conclude without problem.

Additional examples, including some toys examples, can be found in [42].

We now discuss some additional observations on our procedure and its implementation.

1) Detecting convergent equational theories: Upon closer examination, our algorithm shares several similarities with the Knuth-Bendix algorithm used to show that an equational theory is convergent. The generation of $(r_1 \to \ell_1 \overset{p, E_{\downarrow}}{\bowtie} \ell_2 \to r_2)$ and some of our normalisation rules intuitively correspond to the rules in the Knuth-Bendix algorithm. As such, it is hardly surprising that our algorithm also allows us to determine whether an equational theory can be represented by a rewrite system $\mathcal{R}$ convergent modulo $E_{\downarrow}$. In particular, if our algorithms with optimisations returns the set $\mathcal{R}$ then it suffices to check whether (almost) all rules in $\mathcal{R}$ are ordered by $>$. To be more specific, removing from $\mathcal{R}$ the rules of the form $\ell \to \ell$ added on Line 14 yields a rewrite system $\mathcal{R}'$. It then suffices to check that $>$ is compatible with $\mathcal{R}'$ to show that $\mathcal{R}'$ is $E_{\downarrow}$-convergent for E.

Termination is directly guaranteed. Confluence is given by Theorem 1. Indeed, denoting $T = (>, \mathcal{R}, \mathcal{R}_{\downarrow}, E_{\downarrow}, E_{\downarrow})$, with a simple induction one can show that $\bar{t} \Downarrow_T t'$ implies that $t \to^*_{\mathcal{R}', E_{\downarrow}} t'$. Hence, the $E_{\downarrow}$-confluence of $\mathcal{R}'$ is given by Theorem 1, since $t =_E s$ implies that there exist t', s' such that $\bar{t} \Downarrow_T t'$, $\bar{s} \Downarrow_T s'$, and $t' =_{E_{\downarrow}} s'$. This leads to an interesting observation:

Lemma 5. *If E has the finite variant property modulo $E_{\downarrow}$, then there exists a finite rewrite system $\mathcal{R}$ that is $E_{\downarrow}$-confluent for E.*

2) Existence of an AC-strong reduction order compatible with a rewrite system: Our procedure relies at minimum on the existence of an $E_{\downarrow}$-strong reduction order which, as previously mentioned, can be quite challenging to prove. In our prototype, we rely on the AC-compatible and AC-total reduction order of [49]. It is fully syntactic and RPO based, meaning that it is efficient and allows us to apply the optimisation described in Section VI-B. Notice that for this order to be AC-strong, it additionally requires to be stable by renaming. Being RPO based, this can easily be achieved by encoding names into ground terms $\mathsf{N} = \{\mathsf{a}, \mathsf{sc}(\mathsf{a}), \mathsf{sc}(\mathsf{sc}(\mathsf{a})), \ldots\}$.

Although our experiments showed that using $>_{AC}$ works

well in practice, it can be enhanced by providing a rewrite system to the algorithm instead of letting the algorithm try to create one on its own. This is for example the case with the Abelian Group ($\mathcal{AG}$) equational theory. In [27], $\mathcal{AG}$ was shown to satisfy the FVP modulo AC using an AC-convergent rewrite system first proposed by Lankford [41]. Denoting this rewrite system $\mathcal{R}_{\mathcal{AG}}$, we call our prototype with $\mathcal{R}_{\mathcal{AG}}$ given as input. However, this is only correct if we can show the existence of an AC-strong reduction order compatible with $\mathcal{R}_{\mathcal{AG}}$. It is well known that from an E-terminating rewrite system, one can build an E-compatible reduction order, but obtaining totality is not always possible. For example, consider the rewrite system $\mathcal{R} = \{f(x,x) \to k, k \to f(k_1,k_2)\}$ with k, k_1, k_2 constants. $\mathcal{R}$ is convergent but there exists no reduction order $\emptyset$-total and $\emptyset$-compatible with $\mathcal{R}$. Indeed, totality implies $f(k_1,k_2) > f(a,a)$ for some minimal ground term a; and compatibility with $\mathcal{R}$ entails that $f(a,a) > k > f(k_1,k_2)$, leading to a contradiction.

Nevertheless, we show the existence of an AC-strong reduction order compatible with $\mathcal{R}_{\mathcal{AG}}$ by composing $>_{AC}$ with the order that was used to show termination of $\mathcal{R}_{\mathcal{AG}}$ [41]. The composition of orders is given by the following result (proof in [43, Appendix F]).

Lemma 6. *Let E be an equational theory. Let $\equiv$ be an equivalence relation on terms closed by application of contexts, substitutions and renaming, and such that $u =_E t$ implies $u \equiv t$. Let $>_1$ be an E-strong reduction order. Let $\mathcal{R}$ be a set of rewrite rules and $>$ be a reduction order stable by renaming such that:*

- *if $s \equiv u > v \equiv t$ then $s > t$ ($\equiv$-compatible);*
- *for all ground terms u, v, either $u > v$ or $v > u$ or $u \equiv v$ ($\equiv$-total);*
- *for all $(\ell \to r) \in \mathcal{R}$, $\ell > r$;*
- *for all $a, b \in \mathcal{N}$, $a > b$ implies $a >_1 b$.*

Then there exists an E-strong reduction order $>_2$ compatible with $\mathcal{R}$.

Example 11. *Consider the reduction order $>_\#$ from Example 2 and the order $>_{AC}$ from Example 3. Define $\mathcal{R} = \{s \to t \mid s >_\# t\}$ and $\equiv$ the smallest equivalence relation such that $s \equiv t$ implies either $s =_{AC} t$ or there exist a, b ground terms and a term context $C[\_]$ such that $s = C[a]$ and $C[b] = t$ and $\#(a) = \#(b)$. Applying Lemma 6 with $>$ being $>_\#$ and $>_1$ being $>_{AC}$, we deduce the existence of an AC-strong reduction order $>_2$ compatible with $\mathcal{R}$. More precisely, the proof of Lemma 6 defines $>_2$ as the transitive closure of $>_2'$ where $s >_2' t$ iff:*

- *either $s >_\# t$*
- *or $s \equiv t$ and there exists u, v ground terms and a term context $C[\_]$ such that $s =_{AC} C[u]$, and $t =_{AC} C[v]$, and $u >_{AC} v$*

In our prototype, we have also implemented the reduction order $>_2$. Note that increasing the number of reduction orders handled by our prototype also increases its chance to generate a rewrite theory mimicking the target equational theory. Indeed, changing the reduction order impacts the termination of the algorithm. It is however difficult to estimate beforehand which order is more suited to a given equational theory.

3) Limitations and future work: Although the optimised algorithm allows to transform more equational theories than Algorithm 1, both are still confined to the realm of theories having the FVP modulo $E_\downarrow$. The function `cleanup`$(\mathcal{R}, E_\mathcal{A})$ takes $E_\mathcal{A}$ into account by removing superfluous rules but does not change the fact that $\mathcal{R}$ was built with $E_\downarrow$, i.e. the equational theory has the FVP modulo $E_\downarrow$. This is a limitation of our algorithm. Indeed, even if an equational theory E does not have the FVP modulo E (e.g. XOR with homomorphic symbol), E trivially has the FVP modulo E (itself), and so we could expect other interesting equational theories that include E to have the FVP modulo E. We conjecture that modifying the normalisation rule EQ and SUB to consider $=_{E_\mathcal{A}}$ instead of $=_{E_\downarrow}$ should be a step in the right direction, possibly with the hypothesis that there exists a rewrite system $\mathcal{R}_\mathcal{A}$ that is $E_\downarrow$-convergent for $E_\mathcal{A}$. However, the proof we present in this paper cannot be as simply adapted. Indeed, by going from $E_\downarrow$ to $E_\mathcal{A}$, we are losing very critical properties ($>$ is not $E_\mathcal{A}$-compatible and $E_\mathcal{A}$ does not have finite equivalence classes) for the termination of our transformations in the proof. As future work, we plan to address these challenges.

Our prototype showed that for some examples relying on AC-unification, the computation time could go from several minutes up to more than one hour. Although our prototype was developed as a proof of concept, we used the tool Maude [59] to efficiently compute the most general unifiers modulo AC. However, most generated rules are duplicates of other generated rules. For example, applying our algorithm on ACN generates around 400k rules despite finally keeping only 14. We plan to identify new invariants in the proof of Theorem 5 that would reduce the number of generated rules, hence increasing our algorithm's efficiency.

Finally, the genericity of our prototype makes it an ideal candidate to be integrated to cryptographic verifiers such as Tamarin and ProVerif. We plan to first tackle its integration to ProVerif as the rewrite theory introduced in this paper is a generalisation of the framework used in ProVerif. Nevertheless, for both tools, it may require a major overhaul of the theory behind them. But this is, in our opinion, anyway the next logical step for automatic verifiers to handle more cryptographic primitives.

REFERENCES

[1] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy, SP 2017, May 22-26, 2017*. San Jose, CA, USA: IEEE Computer Society, 2017, pp. 483–502.

[2] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe, "Automated analysis and verification of TLS 1.3: 0-rtt, resumption and delayed authentication," in *IEEE Symposium on Security and Privacy, SP 2016, May 22-26, 2016*. San Jose, CA, USA: IEEE Computer Society, 2016, pp. 470–485.

[3] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of TLS 1.3," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, October 30 - November 03, 2017*, B. Thuraisingham,

D. Evans, T. Malkin, and D. Xu, Eds. Dallas, TX, USA: ACM, 2017, pp. 1773–1788.

[4] K. Bhargavan, V. Cheval, and C. A. Wood, "A symbolic analysis of privacy for TLS 1.3 with encrypted client hello," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, November 7-11, 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. Los Angeles, CA, USA: ACM, 2022, pp. 365–379.

[5] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach," in *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, April 26-28, 2017*. Paris, France: IEEE, 2017, pp. 435–450.

[6] D. A. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, "A formal analysis of 5g authentication," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. Toronto, ON, Canada: ACM, 2018, pp. 1383–1396.

[7] N. Kobeissi, G. Nicolas, and K. Bhargavan, "Noise explorer: Fully automated modeling and verification for arbitrary noise protocols," in *IEEE European Symposium on Security and Privacy, EuroS&P 2019, June 17-19, 2019*. Stockholm, Sweden: IEEE, 2019, pp. 356–370.

[8] D. A. Basin, R. Sasse, and J. Toro-Pozo, "The EMV standard: Break, fix, verify," in *42nd IEEE Symposium on Security and Privacy, SP 2021, 24-27 May 2021*. San Francisco, CA, USA: IEEE, 2021, pp. 1766–1781. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00037

[9] S. Gazdag, S. Grundner-Culemann, T. Guggemos, T. Heider, and D. Loebenberger, "A formal analysis of ikev2's post-quantum extension," in *ACSAC '21: Annual Computer Security Applications Conference, December 6 - 10, 2021*. Virtual Event, USA: ACM, 2021, pp. 91–105.

[10] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, *Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, available at https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf, 2020.

[11] B. Blanchet, V. Cheval, and V. Cortier, "Proverif with lemmas, induction, fast subsumption, and much more," in *43rd IEEE Symposium on Security and Privacy, SP 2022, May 22-26, 2022*. San Francisco, CA, USA: IEEE, 2022, pp. 69–86.

[12] D. Basin, C. Cremers, J. Dreier, S. Meier, R. Sasse, and B. Schmidt, *Tamarin prover manual*, available at https://tamarin-prover.github.io/, 2019.

[13] D. A. Basin, C. Cremers, J. Dreier, and R. Sasse, "Tamarin: Verification of large-scale, real-world, cryptographic protocols," *IEEE Secur. Priv.*, vol. 20, no. 3, pp. 24–32, 2022.

[14] S. Escovar, C. Meadows, and J. Meseguer, *Maude-NPA manual, Version 3.1*, available at https://maude.cs.illinois.edu/w/images/9/90/Maude-NPA_manual_v3_1.pdf, 2017.

[15] S. Escobar, C. Meadows, and J. Meseguer, "A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties," *Theor. Comput. Sci.*, vol. 367, no. 1-2, pp. 162–202, 2006.

[16] V. Cheval, S. Kremer, and I. Rakotonirina, "DEEPSEC: deciding equivalence properties in security protocols theory and practice," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018*. San Francisco, California, USA: IEEE Computer Society, 2018, pp. 529–546.

[17] ——, "The DEEPSEC prover," in *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, July 14-17, 2018, Proceedings, Part II*, ser. Lecture Notes in Computer Science, H. Chockler and G. Weissenbacher, Eds., vol. 10982. Oxford, UK: Springer, 2018, pp. 28–36.

[18] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," *ACM Trans. Comput. Log.*, vol. 17, no. 4, p. 23, 2016.

[19] D. Baelde, S. Delaune, I. Gazeau, and S. Kremer, "Symbolic verification of privacy-type properties for security protocols with XOR," in *30th IEEE Computer Security Foundations Symposium, CSF 2017, CA, USA, August 21-25, 2017*. Santa Barbara: IEEE Computer Society, 2017, pp. 234–248.

[20] F. Baader and W. Snyder, "Unification theory," in *Handbook of Automated Reasoning (in 2 volumes)*, J. A. Robinson and A. Voronkov, Eds. Elsevier and MIT Press, 2001, pp. 445–532.

[21] A. Boudet and E. Contejean, ""syntactic" ac-unification," in *Constraints in Computational Logics, First International Conference, CCL'94, September 7-9, 1994*, ser. Lecture Notes in Computer Science, J. Jouannaud, Ed., vol. 845. Munich, Germany: Springer, 1994, pp. 136–151.

[22] C. Kirchner, "6 - from unification in combination of equational theories to a new ac-unification algorithm," in *Rewriting Techniques*, H. Aït-Kaci and M. Nivat, Eds. Academic Press, 1989, pp. 171–210.

[23] M. Ayala-Rincón, M. Fernández, G. F. Silva, and D. N. Sobrinho, "A certified algorithm for ac-unification," in *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022*, ser. LIPIcs, A. P. Felty, Ed., vol. 228. Haifa, Israel: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 8:1–8:21.

[24] F. Baader and K. U. Schulz, "Unification in the union of disjoint equational theories: Combining decision procedures," *J. Symb. Comput.*, vol. 21, no. 2, pp. 211–243, 1996. [Online]. Available: https://doi.org/10.1006/jsco.1996.0009

[25] P. Réty, C. Kirchner, H. Kirchner, and P. Lescanne, "NARROWER: A new algorithm for unification and its application to logic programming," in *Rewriting Techniques and Applications, First International Conference, RTA-85, May 20-22, 1985, Proceedings*, ser. Lecture Notes in Computer Science, J. Jouannaud, Ed., vol. 202. Dijon, France: Springer, 1985, pp. 141–157.

[26] W. Nutt, P. Réty, and G. Smolka, "Basic narrowing revisited," *J. Symb. Comput.*, vol. 7, no. 3/4, pp. 295–317, 1989.

[27] H. Comon-Lundh and S. Delaune, "The Finite Variant Property: How to get rid of some algebraic properties," in *Term Rewriting and Applications, 16th International Conference, RTA 2005, April 19-21, 2005, Proceedings*, ser. Lecture Notes in Computer Science, J. Giesl, Ed., vol. 3467. Nara, Japan: Springer, 2005, pp. 294–307.

[28] A. Boudet, "Competing for the ac-unification race," *J. Autom. Reason.*, vol. 11, no. 2, pp. 185–212, 1993. [Online]. Available: https://doi.org/10.1007/BF00881905

[29] F. Fages, "Associative-commutative unification," *J. Symb. Comput.*, vol. 3, no. 3, pp. 257–275, 1987.

[30] M. E. Stickel, "A unification algorithm for associative-commutative functions," *J. ACM*, vol. 28, no. 3, pp. 423–434, 1981. [Online]. Available: https://doi.org/10.1145/322261.322262

[31] A. Boudet, E. Contejean, and H. Devie, "A new AC unification algorithm with an algorithm for solving systems of diophantine equations," in *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), June 4-7, 1990*. Philadelphia, Pennsylvania, USA: IEEE Computer Society, 1990, pp. 289–299.

[32] D. Kapur, P. Narendran, and L. Wang, "An e-unification algorithm for analyzing protocols that use modular exponentiation," in *Rewriting Techniques and Applications*, R. Nieuwenhuis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 165–179.

[33] D. Lankford, G. Butler, and B. Brady, "Abelian group unification algorithms for elementary terms," in *Automated theorem proving (1983)*, ser. Contemp. Math. Denver, Col.: Amer. Math. Soc., Providence, RI, 1984, vol. 29, pp. 193–199.

[34] S. Anantharaman, P. Narendran, and M. Rusinowitch, "Unification modulo *ACUI* plus distributivity axioms," *J. Autom. Reason.*, vol. 33, no. 1, pp. 1–28, 2004.

[35] Z. Liu and C. Lynch, "Efficient general unification for XOR with homomorphism," in *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, July 31 - August 5, 2011. Proceedings*, ser. Lecture Notes in Computer Science, N. S. Bjørner and V. Sofronie-Stokkermans, Eds., vol. 6803. Wroclaw, Poland: Springer, 2011, pp. 407–421.

[36] ——, "Efficient general agh-unification," *Inf. Comput.*, vol. 238, pp. 128–156, 2014.

[37] J. Meseguer, "Variants in the infinitary unification wonderland," in *Rewriting Logic and Its Applications - 13th International Workshop, WRLA 2020, October 20-22, 2020, Revised Selected Papers*, ser. Lecture Notes in Computer Science, S. Escobar and N. Martí-Oliet, Eds., vol. 12328. Virtual Event: Springer, 2020, pp. 75–95.

[38] S. Escobar, J. Meseguer, and R. Sasse, "Effectively checking the Finite Variant Property," in *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, July 15-17, 2008, Proceedings*, ser. Lecture Notes in Computer Science, A. Voronkov, Ed., vol. 5117. Hagenberg, Austria: Springer, 2008, pp. 79–93.

[39] A. Cholewa, J. Meseguer, and S. Escobar, "Variants of variants and the finite variant property," Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, Tech. Rep., 2014. [Online]. Available: https://core.ac.uk/reader/19530230

[40] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *J. Log. Algebraic Methods Program.*, vol. 75, no. 1, pp. 3–51, 2008.

[41] J.-M. Hullot, "A catalogue of canonical term rewriting systems," *Report CSL-113, SRI International*, no. ADA087641, 1980.

[42] V. Cheval, "Source code and examples for the prototype generating rewrite theory and checking Finite Variant Property," https://github.com/VincentCheval/fvpgen, 2024.

[43] V. Cheval and C. Fontaine, "Automatic verification of Finite Variant Property beyond convergent equational theories," Extended version of the present paper, available on arXiv, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2410.15289

[44] J. Jouannaud, C. Kirchner, and H. Kirchner, "Incremental construction of unification algorithms in equational theories," in *Automata, Languages and Programming, 10th Colloquium, July 18-22, 1983, Proceedings*, ser. Lecture Notes in Computer Science, J. Díaz, Ed., vol. 154. Barcelona, Spain: Springer, 1983, pp. 361–373.

[45] F. Baader, "Combination of compatible reduction orderings that are total on ground terms," in *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, June 29 - July 2, 1997*. Warsaw, Poland: IEEE Computer Society, 1997, pp. 2–13.

[46] J. Jouannaud and P. Lescanne, "On multiset orderings," *Inf. Process. Lett.*, vol. 15, no. 2, pp. 57–63, 1982. [Online]. Available: https://doi.org/10.1016/0020-0190(82)90107-7

[47] P. Lescanne, "On the recursive decomposition ordering with lexicographical status and other related orderings," *J. Autom. Reason.*, vol. 6, no. 1, pp. 39–49, 1990. [Online]. Available: https://doi.org/10.1007/BF00302640

[48] A. Rubio and R. Nieuwenhuis, "A precedence-based total ac-compatible ordering," in *Rewriting Techniques and Applications, 5th International Conference, RTA-93, June 16-18, 1993, Proceedings*, ser. Lecture Notes in Computer Science, C. Kirchner, Ed., vol. 690. Montreal, Canada: Springer, 1993, pp. 374–388.

[49] A. Rubio, "A fully syntactic AC-RPO," in *Rewriting Techniques and Applications, 10th International Conference, RTA-99, July 2-4, 1999, Proceedings*, ser. Lecture Notes in Computer Science, P. Narendran and M. Rusinowitch, Eds., vol. 1631. Trento, Italy: Springer, 1999, pp. 133–147. [Online]. Available: https://doi.org/10.1007/3-540-48685-2_11

[50] A. Rubio and R. Nieuwenhuis, "A total ac-compatible ordering based on RPO," *Theor. Comput. Sci.*, vol. 142, no. 2, pp. 209–227, 1995.

[51] A. Yamada, S. Winkler, N. Hirokawa, and A. Middeldorp, "AC-KBO revisited," *Theory Pract. Log. Program.*, vol. 16, no. 2, pp. 163–188, 2016.

[52] D. Kim and C. Lynch, "An rpo-based ordering modulo permutation equations and its applications to rewrite systems," in *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021*, ser. LIPIcs, N. Kobayashi, Ed., vol. 195. Buenos Aires, Argentina (Virtual Conference): Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 19:1–19:17.

[53] S. Escobar, J. Meseguer, and R. Sasse, "Variant narrowing and equational unification," in *Proceedings of the Seventh International Workshop on Rewriting Logic and its Applications, WRLA 2008, March 29-30, 2008*, ser. Electronic Notes in Theoretical Computer Science, G. Rosu, Ed., vol. 238, no. 3. Budapest, Hungary: Elsevier, 2008, pp. 103–119.

[54] S. Eker, "Associative-commutative rewriting on large terms," in *Rewriting Techniques and Applications*, R. Nieuwenhuis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 14–29.

[55] D. Benanav, D. Kapur, and P. Narendran, "Complexity of matching problems," *Journal of Symbolic Computation*, vol. 3, no. 1, pp. 203–216, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747717187800275

[56] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *Computer Aided Verification - 25th International Conference, CAV 2013, July 13-19, 2013. Proceedings*, ser. Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds., vol. 8044. Saint Petersburg, Russia: Springer, 2013, pp. 696–701.

[57] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago, "Symbolic protocol analysis with disequality constraints modulo equational theories," in *Programming Languages with Applications to Biology and Security - Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday*, ser. Lecture Notes in Computer Science, C. Bodei, G. Ferrari, and C. Priami, Eds., vol. 9465. Pisa, Italy: Springer, 2015, pp. 238–261.

[58] J. Dreier, P. Lafourcade, and D. Mahmoud, "Shaken, not stirred - automated discovery of subtle attacks on protocols using mix-nets," in *The 33rd USENIX Security Symposium (Usenix) 2024*, 2024.

[59] M. Chavel, F. Duran, S. Escobar, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, P. C. Ölveczky, R. Rubio, and C. L. Talcott, "The maude system," https://maude.cs.illinois.edu/w/index.php/The_Maude_System, 2014.