Variant-based Equational Unification under Constructor Symbols

Damián Aparicio-Sánchez Santiago Escobar Julia Sapiña
VRAIN (Valencian Research Institute for Artificial Intelligence)
Universitat Politècnica de València
Valencia, Spain
{daapsnc,sescobar,jsapina}@upv.es

Equational unification of two terms consists of finding a substitution that, when applied to both terms, makes them equal modulo some equational properties. A narrowing-based equational unification algorithm relying on the concept of the variants of a term is available in the most recent version of Maude, version 3.0, which provides quite sophisticated unification features. A variant of a term $t$ is a pair consisting of a substitution $\sigma$ and the canonical form of $t_\sigma$. Variant-based unification is decidable when the equational theory satisfies the finite variant property. However, this unification procedure does not take into account constructor symbols and, thus, may compute many more unifiers than the necessary or may not be able to stop immediately. In this paper, we integrate the notion of constructor symbol into the variant-based unification algorithm. Our experiments on positive and negative unification problems show an impressive speedup.

1 Introduction

Equational unification of two terms is of special relevance to many areas in computer science, including logic programming, and consists of finding a substitution that, when applied to both terms, makes them equal modulo some equational properties. Several algorithms have been developed in the literature for specific equational theories, such as associative-commutative symbols, exclusive-or, Diffie-Hellman, or Abelian Groups (see [3]). Narrowing proved to be complete for unification [23] and several cases have been studied where narrowing provides a decidable unification algorithm [1, 2]. A narrowing-based equational unification algorithm relying on the concept of the variants of a term [11] has been developed in [22] and it is available in the most recent version of Maude, version 3.0, which provides quite sophisticated unification features [9, 13].

Several tools and techniques rely on Maude’s advanced unification capabilities, such as termination [14] and local confluence and coherence [15, 16] proofs, narrowing-based theorem proving [35] or testing [34], and logical model checking [20, 4]. The area of cryptographic protocol analysis has also benefited from advanced unification algorithms: Maude-NPA [19], Tamarin [12] and AKISS [5] rely on the different unification features of Maude. Furthermore, numerous decision procedures for formula satisfiability modulo equational theories also rely on unification, either based on narrowing [37] or by using variant generation in finite variant theories [32].

Constructor symbols are extensively used in computer science: for representing data instead of functions, for manipulating programs as data, or for reasoning in complex semantic structures. In an equational theory, constructors can be characterized in the “no junk, no confusion” style of Goguen and
Burtsell [7], providing the mathematical semantics of the equational theory as the initial algebra of a Maude functional module, which corresponds to the least Herbrand model in logic programming (see [9]). However, this more general notion of constructor differs from the “logic” notion of a functor and the “functional” notion of a symbol not appearing in the root position of the left-hand side of any equation. The notion of a constructor symbol has not yet been integrated into the variant-based equational unification procedure of Maude and, thus, it may compute many more unifiers than the necessary or it may not be able to stop immediately. In this paper, we integrate the notion of constructor symbol into the variant-based unification algorithm with an impressive speedup.

After some preliminaries in Section 2 we recall variant-based unification in Section 3. In Section 4, we define our new unification algorithm that reduces the total execution time. Our experiments in Section 5 show that this improved unification algorithm works well in practice. We conclude in Section 6.

2 Preliminaries

We follow the classical notation and terminology from [36] for term rewriting, from [3] for unification, and from [27] for rewriting logic and order-sorted notions.

We assume an order-sorted signature \( \Sigma = (S, \leq, \Sigma) \) with a poset of sorts \((S, \leq)\). The poset \((S, \leq)\) of sorts for \( \Sigma \) is partitioned into equivalence classes, called connected components, by the equivalence relation \((\leq \cup \geq)^+\). We assume that each connected component \([s]\) has a top element under \(\leq\), denoted \(\top_{[s]}\) and called the top sort of \([s]\). This involves no real loss of generality, since if \([s]\) lacks a top sort, it can be easily added. We also assume an \(S\)-sorted family \(\mathcal{X} = \{\mathcal{X}_{s}\}_{s \in S}\) of disjoint variable sets with each \(\mathcal{X}_s\) countably infinite. \(\mathcal{T}_\Sigma(\mathcal{X})\) is the set of terms of sort \(s\), and \(\mathcal{X}_{\Sigma s}\) is the set of ground terms of sort \(s\). We write \(\mathcal{T}_\Sigma(\mathcal{X})\) and \(\mathcal{X}_{\Sigma}\) for the corresponding order-sorted term algebras. Given a term \(t\), \(\text{Var}(t)\) denotes the set of variables in \(t\).

Positions are represented by sequences of natural numbers denoting an access path in the term when viewed as a tree. The top or root position is denoted by the empty sequence \(\lambda\). We define the relation \(p \leq q\) between positions as \(p \leq p\) for any \(p\); and \(p \leq p.q\) for any \(p\) and \(q\). Given \(U \subseteq \Sigma \cup \mathcal{X}\), \(\text{Pos}_U(t)\) denotes the set of positions of a term \(t\) that are rooted by symbols or variables in \(U\). The set of positions of a term \(t\) is written \(\text{Pos}(t)\), and the set of non-variable positions \(\text{Pos}_\Sigma(t)\). The subterm of \(t\) at position \(p\) is \(t \upharpoonright p\) and \(t \upharpoonright u\) is the term where \(t \upharpoonright p\) is replaced by \(u\).

A substitution \(\sigma \in \text{Subst}(\Sigma, \mathcal{X})\) is a sorted mapping from a finite subset of \(\mathcal{X}\) to \(\mathcal{T}_\Sigma(\mathcal{X})\). Substitutions are written as \(\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}\) where the domain of \(\sigma\) is \(\text{Dom}(\sigma) = \{X_1, \ldots, X_n\}\) and the set of variables introduced by terms \(t_1, \ldots, t_n\) is written \(\text{Ran}(\sigma)\). The identity substitution is \(id\). Substitutions are homomorphically extended to \(\mathcal{T}_\Sigma(\mathcal{X})\). The application of a substitution \(\sigma\) to a term \(t\) is denoted by \(t \sigma\) or \(\sigma(t)\). For simplicity, we assume that every substitution is idempotent, i.e., \(\sigma\) satisfies \(\text{Dom}(\sigma) \cap \text{Ran}(\sigma) = \emptyset\). The restriction of \(\sigma\) to a set of variables \(V\) is \(\sigma|_V\), i.e., \(\forall x \in V, \sigma|_V(x) = \sigma(x)\) and \(\forall x \notin V, \sigma|_V(x) = x\). Composition of two substitutions \(\sigma\) and \(\sigma'\) is denoted by \(\sigma \sigma'\). Combination of two substitutions \(\sigma\) and \(\sigma'\) such that \(\text{Dom}(\sigma) \cap \text{Dom}(\sigma') = \emptyset\) is denoted by \(\sigma \cup \sigma'\). We call a substitution \(\sigma\) a variable renaming if there is another substitution \(\sigma^{-1}\) such that \((\sigma \sigma^{-1})|_{\text{Dom}(\sigma)} = id\).

A \(\Sigma\)-equation is an unoriented pair \(t = t'\), where \(t, t' \in \mathcal{T}_\Sigma(\mathcal{X})\) for some sort \(s \in S\). An equational theory \((\Sigma, E)\) is a pair with \(\Sigma\) an order-sorted signature and \(E\) a set of \(\Sigma\)-equations. Given \(\Sigma\) and a set \(E\) of \(\Sigma\)-equations, order-sorted equational logic induces a congruence relation \(=_{E}\) on terms \(t, t' \in \mathcal{T}_\Sigma(\mathcal{X})\) (see [28]). We say \(\sigma_1 =_E \sigma_2\) iff \(\sigma_1(x) =_E \sigma_2(x)\) for any variable \(x\). Throughout this paper we assume that \(\mathcal{T}_\Sigma(\mathcal{X}) \neq \emptyset\) for every sort \(s\), because this affords a simpler deduction system. An equational theory \((\Sigma, E)\) is regular if for each \(t = t'\) in \(E\), we have \(\text{Var}(t) = \text{Var}(t')\). An equational theory \((\Sigma, E)\) is linear if for each
$t = t'$ in $E$, each variable occurs only once in $t$ and $t'$. An equational theory $(\Sigma, E)$ is sort-preserving if for each $t = t'$ in $E$, each sort $s$, and each substitution $\sigma$, we have $t \sigma \in \mathcal{T}_E(\mathcal{X})_s$ iff $t' \sigma \in \mathcal{T}_E(\mathcal{X})_s$. An equational theory $(\Sigma, E)$ is defined using top sorts if for each equation $t = t'$ in $E$, all variables in $\text{Var}(t)$ and $\text{Var}(t')$ have a top sort. Given two terms $t$ and $t'$, we say $t$ is more general than $t'$, denoted as $t \sqsubseteq t'$, if there is a substitution $\eta$ such that $t\eta = \eta t'$. Similarly, given two substitutions $\sigma$ and $\rho$, we say $\sigma$ is more general than $\rho$ for a set $W$ of variables, denoted as $\sigma|_W \sqsubseteq \rho|_W$, if there is a substitution $\eta$ such that $(\sigma\eta)|_W = \eta \rho|_W$. The $\sqsubseteq$ relation induces an equivalence relation $\simeq_E$, i.e., $t \simeq_E t'$ iff $t \sqsubseteq t'$ and $t' \sqsubseteq t$.

An $E$-unifier for a $\Sigma$-equation $t = t'$ is a substitution $\sigma$ such that $t\sigma = t'\sigma$. For $\text{Var}(t) \cup \text{Var}(t') \subseteq W$, a set of substitutions $CSU^W_E(t = t')$ is said to be a complete set of unifiers for the equality $t = t'$ modulo $E$ away from $W$ iff: (i) each $\sigma \in CSU^W_E(t = t')$ is an $E$-unifier of $t = t'$; (ii) for any $E$-unifier $\rho$ of $t = t'$ there is a $\sigma \in CSU^W_E(t = t')$ such that $\sigma|_W \sqsubseteq_E \rho|_W$; and (iii) for all $\sigma \in CSU^W_E(t = t')$, $\text{Dom}(\sigma) \subseteq (\text{Var}(t) \cup \text{Var}(t'))$ and $\text{Ran}(\sigma) \cap W = \emptyset$. Given a conjunction $\Gamma$ of equations, a set $U$ of $E$-unifiers of $\Gamma$ is said to be minimal if it is complete and for all distinct elements $\sigma$ and $\sigma'$ in $U$, $\sigma \sqsubseteq E \sigma'$ implies $\sigma = E \sigma'$.

A unification algorithm is said to be finitary and complete if it always terminates after generating a finite and complete set of unifiers. A unification algorithm is said to be minimal and complete if it always returns a minimal and complete set of unifiers.

A rewrite rule is an oriented pair $l \rightarrow r$, where $l \not\in \mathcal{X}$ and $l, r \in \mathcal{T}_E(\mathcal{X})_s$ for some sort $s \in S$. An (unconditional) order-sorted rewrite theory is a set of $\Sigma$-equations, and $R$ a set of rewrite rules. The set $R$ of rules is sort-decreasing if for each $t \rightarrow t'$ in $R$, each $s \in S$, and each substitution $\sigma$, $\sigma t \in \mathcal{T}_E(\mathcal{X})_s$ implies $\sigma t' \in \mathcal{T}_E(\mathcal{X})_s$. The rewriting relation on $\mathcal{T}_E(\mathcal{X})$, written $t \rightarrow_R t'$ holds between $t$ and $t'$ iff there exist $p \in \text{Pos}_E(t)$, $l \rightarrow r \in R$ and a substitution $\sigma$, such that $t|_p = l\sigma$, and $t' = (r \sigma)p$. The relation $\rightarrow_{R/E}$ on $\mathcal{T}_E(\mathcal{X})$ is $t \rightarrow_{R/E} r \rightarrow_{R/E} t'$ if $t' = (r \sigma)p$ for some sort $s \in S$. The transitive (resp. transitive and reflexive) closure of $\rightarrow_{R/E}$ is denoted $\rightarrow^+_{R/E}$ (resp. $\rightarrow^*_{R/E}$).

Reducibility of $\rightarrow_{R/E}$ is undecidable in general since $E$-congruence classes can be arbitrarily large. Therefore, $R/E$-rewriting is usually implemented by $R,E$-rewriting under some conditions on $R$ and $E$ such as confluence, termination, and coherence (see [24, 30, 33]). A relation $\rightarrow_{R,E}$ on $\mathcal{T}_E(\mathcal{X})$ is defined as: $t \rightarrow_{R,E} t'$ iff there is a non-variable position $p \in \text{Pos}_E(t)$, a rule $l \rightarrow r \in R$, and a substitution $\sigma$ such that $t|_p = l\sigma$ and $t' = (r \sigma)p$. The narrowing relation $\rightarrow_{R,E}^*$ on $\mathcal{T}_E(\mathcal{X})$ is defined as: $t \rightarrow_{R,E}^* t'$ iff there is a non-variable position $p \in \text{Pos}_E(t)$, a rule $l \rightarrow r \in R$, and a substitution $\sigma$ such that $t|_p = l\sigma$ and $t' = (r \sigma)p$. We call $(\Sigma, B, E)$ a decomposition of an order-sorted equational theory $(\Sigma, E \sqsubseteq B)$ if $B$ is regular, linear, sort-preserving, defined using top sorts, and has a finitary and complete unification algorithm, and equations $E$ are oriented into rules $E$ such that they are sort-decreasing and convergent, i.e., confluent, terminating, and strictly coherent modulo $B$ [15, 26, 30]. The irreducible version of a term $t$ is denoted by $t|_{E,B}$.

Given a decomposition $(\Sigma, B, E)$ of an equational theory and a term $t$, a pair $(t', \theta)$ of a term $t'$ and a substitution $\theta$ is an $E, B$-variant (or just a variant) of $t$ if $t \theta = t'$ and $\theta|_{E,B} = \theta$ [11, 22]. A complete set of $E, B$-variants [22] (up to renaming) of a term $t$ is a subset, denoted by $[t]|_{E,B}$, of the set of all $E, B$-variants of $t$ such that, for each $E, B$-variant $(t', \sigma)$ of $t$, there is an $E, B$-variant $(t'', \theta)$ in $[t]|_{E,B}$ such that $(t'', \theta) \sqsubseteq_{E,B} (t', \sigma)$, i.e., there is a substitution $\rho$ such that $t'' = t''\rho$ and $\sigma|_{\text{Var}(t)} = \theta\rho$|_{\text{Var}(t)}. A decomposition $(\Sigma, B, E)$ has the finite variant property (FVP) [22] (also called a finite variant decomposition) iff for each $\Sigma$-term $t$, there exists a complete and finite set $[t]|_{E,B}$ of variants of $t$. Note that whether a decomposition has the finite variant property is undecidable [6], but a technique based on the dependency pair framework has been developed in [22] and a semi-decision procedure that works well in practice is available in [8].
3 Variant-based Equational Unification in Maude 3.0

Rewriting logic [27] is a flexible semantic framework within which different concurrent systems can be naturally specified (see [29]). Rewriting Logic is efficiently implemented in the high-performance system Maude [9], which has itself a formal environment of verification tools thanks to its reflective capabilities (see [10, 29]).

Maude 3.0 offers quite sophisticated symbolic capabilities (see [31] and references therein). Among these symbolic features, equational unification [9] is a twofold achievement. On the one hand, Maude provides an order-sorted equational unification command for any combination of symbols having any combination of associativity, commutativity, and identity [13]. This is remarkable, since there is no other system with such an advanced unification algorithm. On the other hand, a narrowing-based equational combination of associativity, commutativity, and identity [13]. This is remarkable, since there is no other system with such an advanced unification algorithm. On the other hand, a narrowing-based equational unification algorithm relying on the concept of the models of 

\[
\sum_t \sigma \ni \text{then } \text{the canonical form of } t \sigma \text{ satisfies the finite variant property}\ [11, 22]. \text{ The finite variant property has become an essential property in some research areas, such as cryptographic protocol analysis, where Maude-NPA [19], Tamarin [12] and AKISS [5] rely on the different unification features of Maude.}

Let us make explicit the relation between variants and equational unification. First, we define the intersection of two sets of variants. Without loss of generality, we assume in this paper that each variant pair \( (t', \sigma) \) of a term \( t \) uses new freshly generated variables.

**Definition 1 (Variant Intersection).** [22] Given a decomposition \( (\Sigma, B, E) \) of an equational theory, two \( \Sigma \)-terms \( t_1 \) and \( t_2 \) such that \( W_1 = \text{Var}(t_1) \cap \text{Var}(t_2) \) and \( W_2 = \text{Var}(t_1) \cup \text{Var}(t_2) \), and two sets \( V_1 \) and \( V_2 \) of variants of \( t_1 \) and \( t_2 \), respectively, we define \( V_1 \cap V_2 = \{ u_1 \sigma, \theta_1 \sigma \cup \theta_2 \sigma \cup \sigma \} | (u_1, \theta_1) \in V_1 \wedge (u_2, \theta_2) \in V_2 \wedge \exists \sigma : \sigma \in CSU^u_B(u_1 = u_2) \wedge (\theta_1 \sigma)|w_1, =_B (\theta_2 \sigma)|w_2 \}.

Then, we define variant-based unification as the computation of the variants of the two terms in a unification problem and their intersection.

**Corollary 2 (Finitary \( \varepsilon \)-unification).** [22] Let \( (\Sigma, B, E) \) be a finite variant decomposition of an equational theory. Given two terms \( t, t' \), the set \( CSU^s_{E,B}(t = t') = \{ \theta | (w, \theta) \in [[t]]_{E,B} \cap [[t']]_{E,B} \} \) is a finite and complete set of unifiers for \( t = t' \).

The most recent version 3.0 of Maude [9] incorporates variant-based unification based on the folding variant narrowing strategy [22]. First, there exists a variant generation command of the form:

\[
\text{get variants [ n ] in ModId : Term}.
\]

where \( n \) is an optional argument providing a bound on the number of variants requested, so that if the cardinality of the set of variants is greater than the specified bound, the variants beyond that bound are omitted; and \( ModId \) is the identifier of the module where the command takes place. Second, there exists a variant-based unification command of the form:

\[
\text{variant unify [ n ] in ModId : T1 =? T1' /} \ldots / \text{ Tk =? Tk'}.
\]

where \( k \geq 1 \) and \( n \) is an optional argument providing a bound on the number of unifiers requested, so that if there are more unifiers, those beyond that bound are omitted; and \( ModId \) is the identifier of the module where the command takes place.

**Example 1.** Consider the following equational theory for exclusive-or that assumes three extra constants \( a, b, \) and \( c \). The second equation is necessary for coherence modulo \( AC \).
Variant-based Equational Unification under Constructor Symbols

fmod EXCLUSIVE-OR is
  sorts Elem EXor . subsort Elem < EXor .
  ops a b c : -> Elem . op mt : -> EXor . op _*_ : EXor EXor -> EXor [assoc comm] .
  vars X Y Z U V : [EXor] .
  eq [idem] : X * X = mt [variant] .
  eq [idem-Coh] : X * X * Z = Z [variant] .
  eq [id] : X * mt = X [variant] .
endfm

The attribute variant specifies that these equations will be used for variant-based unification. Since this theory has the finite variant property (see [11, 22]), given the term X * Y it is easy to verify that there are seven most general variants.

Maude> get variants in EXCLUSIVE-OR : X * Y .

Variant #1 ... Variant #7
[EXor]: #1:[EXor] * #2:[EXor] ... [EXor]: %1:[EXor]
X --> #1:[EXor] ... X --> %1:[EXor]
Y --> #2:[EXor] ... Y --> mt

Note that Maude produces fresh variables of the form #n:Sort or %n:Sort using two different counters (see [9] for details). When we consider a variant unification problem between terms X * Y and U * V, there are 57 unifiers:

Maude> variant unify in EXCLUSIVE-OR : X * Y =? U * V .

Unifier #1 ... Unifier #2
X --> %1:[EXor] * %2:[EXor] ... X --> %1:[EXor] * %3:[EXor]
Y --> %2:[EXor] ... Y --> %2:[EXor]
V --> %1:[EXor] * %2:[EXor] ... V --> %1:[EXor] * %2:[EXor]
U --> %3:[EXor] * %4:[EXor] ... U --> %3:[EXor]

However, this variant-based unification algorithm may compute many more unifiers than the necessary or may not be able to stop immediately. For instance, it is well-known that unification in the exclusive-or theory is unitary, i.e., there exists only one most general unifier modulo exclusive-or [25].

For the unification problem X * Y = U * V of Example 1, the most general unifier w.r.t. \( \equiv_{EUB} \) is \{X \mapsto Y \ast U \ast V\}, which should be appropriately written as \( \sigma = \{X \mapsto Y \ast U \ast V, Y \mapsto Y', U \mapsto U', V \mapsto V'\} \). Note that \{Y \mapsto X \ast U \ast V\}, \{U \mapsto Y \ast X \ast V\}, and \{V \mapsto Y \ast U \ast X\} are equivalent to the former unifier w.r.t. \( \equiv_{EUB} \) by composing \( \sigma \) with, respectively, \( \rho_1 = \{Y' \mapsto Y'' \ast U'' \ast V'', X' \mapsto X'', U' \mapsto U'', V' \mapsto V''\} \), \( \rho_2 = \{U' \mapsto Y'' \ast X'' \ast V'', X' \mapsto X'', Y' \mapsto Y'' \ast V'', V' \mapsto V''\} \), and \( \rho_3 = \{V' \mapsto Y'' \ast U'' \ast X'', X' \mapsto X'', U' \mapsto U'' \ast V'', V' \mapsto Y''\}\). Similarly, \{X \mapsto U, Y \mapsto V\} and \{X \mapsto V, Y \mapsto U\} are equivalent to all the previous ones.

Furthermore, since the variants of both terms are generated by Corollary 2, there may be very simple unification problems such as X =? U where the generation of the variants of t is unnecessary. For example, when unifying terms X and U \ast V, the variants of U \ast V are generated

Maude> variant unify in EXCLUSIVE-OR : X =? U \ast V .

Unifier #1 ... Unifier #2
X --> %1:[EXor] * %2:[EXor] ... X --> mt
V --> %4:[EXor] ... V --> #2:[EXor]
U --> %3:[EXor] ... U --> %3:[EXor]
Unifier #4
X --> #1:[EXor]
V --> #1:[EXor] * #2:[EXor]
U --> #2:[EXor]

Unifier #5
X --> #1:[EXor]
V --> #2:[EXor]
U --> mt

Unifier #6
X --> #1:[EXor]
V --> #1:[EXor] * #2:[EXor]
U --> #1:[EXor]

Unifier #7
X --> #1:[EXor]
V --> #1:[EXor]
U --> mt

but it is clear that the simplest, most general unifier is \{X \mapsto U \ast V\}. In [21], a new procedure to reduce the number of variant unifiers in situations like this was developed. We showed that this new procedure pays off in practice using both the exclusive-or and the abelian group equational theories.

4 Constructor-Root Variant-based Unification

Both the “logic” notion of a functor and the “functional” notion of a constructor refer to a symbol not appearing in the root position of the left-hand side of any predicate or equation. This notion of constructor allows to split a signature \(\Sigma\) as a disjoint union \(\Sigma = D \uplus C\) where \(D\) are called defined symbols and \(C\) are called constructor symbols. In a decomposition \((\Sigma, B, E)\), the canonical term algebra \(Can_{\Sigma/(E,B)} = \{t_{E,B} \mid t \in T_{\Sigma}\}\) is typically made of constructor terms, but this more general notion of constructor differs from the “logic” and “functional” notions. A decomposition \((\Sigma, B, E)\) protects a constructor decomposition \((C, B_C, E_C)\) iff \(C \subseteq \Sigma, B_C \subseteq B,\) and \(E_C \subseteq E,\) and for all \(t, t' \in T_{C}(\mathcal{X})\) we have:

(i) \(t =_{B_C} t' \iff t =_{B} t',\) (ii) \(t =_{E_C,B_C} t' \iff t =_{E,B} t',\) and (iii) \(Can_{C/(E_C,B_C)} = Can_{\Sigma/(E,B)|C}.\) A constructor decomposition \((C, B_C, \emptyset)\) is called free. For instance, the modular exponentiation property typical of Diffie-Hellman protocols is defined using two versions of the exponentiation operator and an auxiliary associative-commutative symbol \(\ast\) for exponents so that \((z^x)^y = (z^y)^x = z^{xy}\). Note that, in the lefthand side of the equation, the outermost exponentiation operator is defined, whereas the innermost exponentiation operator is constructor.

fmod DH-CFVP is

- sorts Exp Elem ElemSet Gen .
- subsort Elem < ElemSet .
- ops a b c : -> Elem [ctor] .
- op exp : Gen ElemSet -> Exp [ctor] .
- op exp : ExpElemSet -> Exp .
- op _*_ : ElemSet ElemSet -> ElemSet [assoc comm ctor] .
- var X : Gen . vars Y Z : ElemSet .
- eq exp(exp(X,Y),Z) = exp(X,Y \ast Z) [variant] .
endfm

Note that it may not always be possible to provide a (free) constructor decomposition, such as Example 1 where the exclusive-or symbol works both as defined and constructor (see [9] for a detailed discussion). However, it is common to combine an equational theory with many different additional constructor symbols, as shown in Section 5.

The notion of a constructor symbol has not yet been integrated into the variant-based equational unification procedure of Maude. An integration of the notion of constructor involves two challenges. On the one hand, when we consider the variant unification problem above between terms \(X\) and \(U \ast V\), the fast unification algorithm of [21] is able to return only one unifier but still generates all the variants of term \(U \ast V\), unnecessarily consuming resources. On the other hand, a unification problem between terms
Given a decomposition $(\Sigma, B, E)$ protecting a free constructor decomposition $(\mathcal{C}', B_C, \emptyset)$ and given a $\Sigma$-term $t$ and a position $p \in \text{Pos}(t)$, we say $p$ is a constructor-root position in $t$ if for all $q < p$, root$(t_q) \in \mathcal{C}'$.

**Definition 4** (Constructor-root Variable). Given a $\Sigma$-term $t$ and a variable $x$, we say $x$ is a constructor-root variable in $t$ if for all $p \in \text{Pos}(t)$, $p$ is constructor-root in $t$.

First, we define the case when there exists $\sigma$ s.t. $C_1[\Box] \sigma =_B C_2[\Box] \sigma$. Intuitively, a variant unifier $\sigma$ of $t_1$ and $t_2$ is constructor-root if each variable in $\text{Ran}(\sigma)$ is under a constructor-root variable of $t_1$ and $t_2$.

**Definition 5** (Constructor-root Variant Unifier). Given a decomposition $(\Sigma, B, E)$ protecting a free constructor decomposition $(\mathcal{C}', B_C, \emptyset)$, two $\Sigma$-terms $t_1$ and $t_2$ s.t. $W_R = \text{Var}(t_1) \cap \text{Var}(t_2)$, $W_U = \text{Var}(t_1) \cup \text{Var}(t_2)$, $(u_1, \theta_1) \in [t_1]_{E,B}$, $(u_2, \theta_2) \in [t_2]_{E,B}$, and $\sigma \in CSU_W^B(u_1 = u_2)$ s.t. $(\theta_1|\sigma)|_{W_U} = \sigma =_B (\theta_2|\sigma)|_{W_U}$, the unifier $(\theta_1 \cup \theta_2) \sigma$ is called constructor-root if for each $x \mapsto t \in \sigma$, either (i) $x \mapsto t$ is a variable renaming, (ii) $x$ is a constructor-root variable in $u_1$ and $u_2$, or (iii) for each $x' \mapsto t' \in \sigma \setminus \{x \mapsto t\}$ (and there exists at least one such binding) s.t. $t' =_B C[t]$, then $x'$ is a constructor-root variable in $u_1$ and $u_2$.

Let us motivate the usefulness of a constructor-root unifier. Given the unification problem $X \equiv_2 V + U$ above, the unifier $\{X \mapsto %1 + %2, V \mapsto %1, U \mapsto %2\}$ is constructor-root, since $X$ is a constructor-root variable in the left unificand $V$ and $U$ are not constructor-root variables but the variables $%1$ and $%2$ used in the bindings of $V$ and $U$ appear in the binding of $X$. Hence, we can safely avoid the generation of the variants of $V + U$. Note that the unifier $\{X \mapsto \mathfrak{e}, V \mapsto %1, U \mapsto %1\}$ is not constructor-root because $V$ and $U$ are not constructor-root variables and for the bindings $V \mapsto %1$ and $U \mapsto %1$ there is no other binding $X' \mapsto t'$ such that $%1$ is a subterm of $t'$ and $X'$ is a constructor-root variable.

**Lemma 6** (Constructor-root Variant Unifier). Given a decomposition $(\Sigma, B, E)$ protecting a free constructor decomposition $(\mathcal{C}', B_C, \emptyset)$, two $\Sigma$-terms $t_1$ and $t_2$ s.t. $W_R = \text{Var}(t_1) \cap \text{Var}(t_2)$, $W_U = \text{Var}(t_1) \cup \text{Var}(t_2)$, $(u_1, \theta_1) \in [t_1]_{E,B}$, $(u_2, \theta_2) \in [t_2]_{E,B}$, and a constructor-root variant unifier $\sigma \in CSU_W^B(u_1 = u_2)$ s.t. $(\theta_1|\sigma)|_{W_U} = \sigma =_B (\theta_2|\sigma)|_{W_U}$, then $\forall (u'_1, \rho_1) \in [u_1]_{E,B}$ and $\forall (u'_2, \rho_2) \in [u_2]_{E,B}$, if there exists $\sigma' \in CSU_{W'}^B(u'_1 = u'_2)$ s.t. $(\theta_1|\sigma')|_{W_U} = \sigma =_B (\theta_2|\sigma')|_{W_U}$, then $(\theta_1 \cup \theta_2|\sigma')|_{W_U} = (\theta_2 \cup \theta_2|\sigma')|_{W_U}$, are both equational unifiers of $t_1$ and $t_2$ but $((\theta_1 \cup \theta_2|\sigma')|_{W_U} \not\subseteq_{E,B} ((\theta'_1 \cup \theta_2|\sigma')|_{W_U}$). Similarly for any $(u'_2, \rho'_2) \in [t_2]_{E,B}$.

**Proof.** By contradiction. Let us assume $\exists \sigma' \in CSU_{W'}^B(u'_1 = u'_2) s.t. ((\theta_1 \cup \theta_2|\sigma')|_{W_U} \not\subseteq_{E,B} ((\theta'_1 \cup \theta_2|\sigma')|_{W_U}$.

First, $\theta'_1|_{w_1} = \theta_1|_{w_1}$ and thus the difference is in $\sigma|_{w_2}$ and $\sigma'|_{w_2}$. By the constructor-root property, $\forall x \mapsto t \in \sigma|_{w_2}$ either $x$ is a constructor-root variable in $u_2$ or $\forall x' \mapsto t' \in \sigma \setminus \{x \mapsto t\}$ s.t. $t' =_B C[t]$, $x'$ is a constructor-root variable in $u_2$. But then $\forall y \in \text{Var}(u_2)$, there exist $y \mapsto w_1 \in \sigma|_{w_2}$ and $y \mapsto w_2 \in \sigma'|_{w_2}$, which contradicts the assumption.

We define the case when there is no $\sigma$ s.t. $C_1[\Box] \sigma =_B C_2[\Box] \sigma$. Intuitively, two terms that form a constructor-root failure pair will never unify despite any further variant computation.

**Definition 7** (Constructor-root Failure Pair). Given a decomposition $(\Sigma, B, E)$ protecting a free constructor decomposition $(\mathcal{C}', B_C, \emptyset)$, two $\Sigma$-terms $t_1$ and $t_2$ s.t. $W_R = \text{Var}(t_1) \cap \text{Var}(t_2)$, $W_U = \text{Var}(t_1) \cup \text{Var}(t_2)$, $(u_1, \theta_1) \in [t_1]_{E,B}$, and $(u_2, \theta_2) \in [t_2]_{E,B}$, the pair $(u_1, u_2)$ is a constructor-root failure pair if
CSU\textsuperscript{W}_{B}(u_1 = u_2) = \emptyset \text{ and there exists two constructor contexts } C_1[\emptyset, \ldots, \emptyset] \text{ and } C_2[\emptyset, \ldots, \emptyset], \text{ terms } v_1, \ldots, v_n, w_1, \ldots, w_m, \text{ and fresh distinct variables } x_1, \ldots, x_n, y_1, \ldots, y_m \text{ s.t. } u_1 =_B C_1[v_1, \ldots, v_n], \text{ } u_2 =_B C_2[w_1, \ldots, w_m], \text{ and } CSU\textsuperscript{W}_{B}(C_1[x_1, \ldots, x_n] = C_2[y_1, \ldots, y_m]) = \emptyset.

Let us motivate the usefulness of a constructor-root failure pair. Given the unification problem \( f(X \ast Y) = g(V \ast U) \) above where \( f \) and \( g \) are different constructor symbols without axioms, the two terms do not unify modulo the axioms of \( \ast \) but neither \( f(W) \) and \( g(W') \) do. Hence, we can safely avoid the generation of the variants of \( V \ast U \) and \( X \ast Y \).

**Lemma 8** (Constructor-root Failure Pair). Given a decomposition \((\Sigma, B, E)\) protecting a free constructor decomposition \((\mathcal{C}', B_{\mathcal{C}'}, \emptyset)\), two \( \Sigma \)-terms \( t_1 \) and \( t_2 \) s.t. \( W_u = \text{Var}(t_1) \cup \text{Var}(t_2) \), \((u_1, \theta_1) \in [[t_1]]_{E,B}, (u_2, \theta_2) \in [[t_2]]_{E,B}, \) and \((u_1, u_2)\) is a constructor-root failure term, then \( CSU\textsuperscript{W}_{E \cup B}(u_1 = u_2) = \emptyset \).

**Proof.** Immediate by Definition[2] \( \square \)

These positive and negative stopping criteria, however, become useful only if we do not generate all the variants a priori, as it is done in Corollary[2] as well as the fast unification technique of [21]. Some incremental generation of variants is required.

**Example 2.** Consider the following theory where constructors have the \texttt{ctor} attribute.

```plaintext
fmod FASTvsCR is sort S .
  ops a b c : - > S [ctor] . op s : S - > S [ctor] .
  op g : S - > S . op f : S S S - > S . vars X Y Z W : S .
  eq f(a,X,Y) = s(Y) [variant] .
  eq f(b,X,Y) = g(X,Y) [variant] .
  eq g(c,Y) = s(Y) [variant] .
endfm
```

Consider the unification problem \((a) f(X,Y,Z) = s(W)\) with only two unifiers and its variant generation.

```
variant unify in FG : f(X,Y,Z) =? s(W) .
          \begin{array}{cc}
          Unifier \#1 & Unifier \#2 \\
          X --> a & X --> b \\
          Y --> #2:S & Y --> c \\
          Z --> #1:S & Z --> #1:S \\
          W --> #1:S & W --> #1:S
          \end{array}
```

\[ f(X,Y,Z) \uparrow \hspace{1cm} s(W) \]

\[ \begin{array}{c}
\{X\rightarrow a\} \\
\{X\rightarrow b\}
\end{array} \]

\[ \begin{array}{c}
\{Y\rightarrow c\}
\end{array} \]

\[ \begin{array}{c}
s(Z) \\
s(Y,Z) \\
s(Z)
\end{array} \]

Let us assume we have an expression \( \clubsuit \) with a considerably large narrowing tree and two new unification problems \((b) f(X,Y,\clubsuit) = s(W)\) and \((c) f(X,\clubsuit,Z) = s(W)\). Note that the unifiers of \((a)\) are still valid for \((b)\), whereas only the first unifier of \((a)\) is valid for \((c)\), assuming \( \clubsuit \) never narrows into \( c \). Both the variant-based unification command of Maude and the fast command of [21] cannot avoid the computation of \( \clubsuit \) in both unification problems \((b)\) and \((c)\). However, the technique described below is able to avoid the full computation of \( \clubsuit \) in \((b)\), since the two unifiers are constructor-root, although it cannot avoid the full computation of \( \clubsuit \) in \((c)\).

We extend the notions of constructor-root unifier and constructor-root failure pair to the pairwise combination of all the variants of a unification problem.

**Definition 9** (Constructor-Root Intersection). Given a decomposition \((\Sigma, B, E)\) protecting a free constructor decomposition \((\mathcal{C}', B_{\mathcal{C}'}, \emptyset)\), two \( \Sigma \)-terms \( t_1 \) and \( t_2 \) such that \( W_1 = \text{Var}(t_1) \cap \text{Var}(t_2) \) and \( W_\cup = \text{Var}(t_1) \cup \text{Var}(t_2) \), and two sets \( V_1 \) and \( V_2 \) of variants of \( t_1 \) and \( t_2 \), respectively, we say that an intersection
V₁ ∩ V₂ is constructor-root if for each leaf (u₁, θ₁) ∈ V₁ (resp. (u₂, θ₂) ∈ V₂), and for each leaf (u₂, θ₂) ∈ V₂ (resp. (u₁, θ₁) ∈ V₁) such that σ ∈ CSU^{W,θ}_{B}(u₁ = u₂) and (θ₁σ)|_{W,ι} = (θ₂σ)|_{W,ι}, we have (θ₁ ∪ θ₂)σ is constructor-root.

Definition 10 (Failure Intersection). Given a decomposition (Σ, B, E) protecting a free constructor decomposition (C*, B^*, θ) two Σ-terms t₁ and t₂ such that W₁ = Var(t₁) ∩ Var(t₂) and W₂ = Var(t₁) ∪ Var(t₂), and two sets V₁ and V₂ of variants of t₁ and t₂, respectively, we say that an intersection V₁ ∩ V₂ is a failure intersection if for each leaf (u₁, θ₁) ∈ V₁ (resp. (u₂, θ₂) ∈ V₂), and for each leaf (u₂, θ₂) ∈ V₂ (resp. (u₁, θ₁) ∈ V₁) such that σ ∈ CSU^{W,θ}_{B}(u₁ = u₂) and (θ₁σ)|_{W,ι} = (θ₂σ)|_{W,ι}, we have the pair (u₁, u₂) is a constructor-root failure pair.

The following example shows that the folding variant narrowing trees of both terms t₁, t₂ of a unification problem t₁ = t₂ must be unfolded down to a frontier where all leaves of t₁ are tested for unification with all the leaves of t₂.

Example 3. Let us consider Example 2 and the unification problem f(X, Y, Z) = f(U, V, W).

We define variant-based unification as the computation of the variants of the two terms in a unification problem. We abuse the notation and write P([[t]]_{E, B}) for the powerset of all the subsets of [[t]]_{E, B} such that each V ∈ P([[t]]_{E, B}) corresponds to the variants of a term t associated to a particular narrowing tree produced by the folding variant narrowing strategy from term t. We also write CSU^{C, W V}_{E, B}(t = t') for a version of the unification algorithm of Corollary 2 that uses sets V₁ and V₂ of variants of t and t’, respectively, instead of generating all the variants.

Definition 11 (Constructor-Root Variant-based Unification). Let (Σ, B, E) be a finite variant decomposition of an equational theory protecting a free constructor decomposition (C*, B^*, θ). Given two terms t, t’ and two sets of variants V₁ ∈ P([[t]]_{E, B}), V₂ ∈ P([[t’]]_{E, B}), the constructor-root variant unifiers are
We measured both the number of computed unifiers and the time required for their computation. Maude indicates that a generous 4 hours timeout was reached without any response. The first column describes the unification problem, while the following #maude, #fast, #n and #er-fast columns show the

\[ CSU_{\text{EUB}}^{t \mapsto t'}(t = t') = \begin{cases} \emptyset & \text{if } \exists V_1 \in \mathbb{P}(\tilde{t}_{E,B}), V_2 \in \mathbb{P}(\tilde{t'}_{E,B}), \text{and they are the smallest sets s.t } V_1 \cap V_2 \text{ is a failure intersection} \\ CSU_{\text{EUB}}^{\cap V_1, V_2}(t = t') & \text{if } \exists V_1 \in \mathbb{P}(\tilde{t}_{E,B}), V_2 \in \mathbb{P}(\tilde{t'}_{E,B}), \text{and they are the smallest sets s.t } V_1 \cap V_2 \text{ is constructor-root} \\ CSU_{\text{EUB}}^{\cap [t_{E,B}, \tilde{t'}_{E,B}]}(t = t') & \text{otherwise} \end{cases} \]

Proposition 12 (Constructor-Root Variant-based Unification). Let \((\Sigma, B, E)\) be a finite variant decomposition of an equational theory protecting a free constructor decomposition \((\psi, B_\psi, \emptyset)\). Given two terms \(t, t'\), the set \(CSU_{\text{EUB}}^{t \mapsto t'}(t = t')\) is a finite and complete set of unifiers for \(t = t'\).

Proof. By contradiction. Let us assume that \(CSU_{\text{EUB}}^{t \mapsto t'}(t = t')\) is not a complete set of unifiers of \(t\) and \(t'\). That is, there exists a unifier \(\rho' \in CSU_{\text{EUB}}^{t \mapsto t'}(t = t')\) and there is no unifier \(\rho \in CSU_{\text{EUB}}^{t \mapsto t'}(t = t')\) s.t. \(\rho \not\subseteq \rho'\). By definition, there exist smallest sets \(V_1 \in \mathbb{P}(\tilde{t}_{E,B}), V_2 \in \mathbb{P}(\tilde{t'}_{E,B})\) s.t. \(V_1 \cap V_2\) is constructor-root or a failure pair. The case of a failure pair is immediate by Lemma 8. Since \(\rho' \in CSU_{\text{EUB}}^{t \mapsto t'}(t = t')\), we have that there exists \(u_1, u_2, \theta_1, \theta_2, \sigma\) s.t. \(\rho' = \theta_1 \sigma \cup \theta_2 \sigma \cup \sigma\), \((u_1, \theta_1) \in \tilde{t}_{E,B}, (u_2, \theta_2) \in \tilde{t'}_{E,B}\), \(\sigma \in CSU_{\text{EUB}}^{W_1}(u_1 = u_2)\), and \((\theta_1 \sigma)|_{W_1} = \bot (\theta_2 \sigma)|_{W_1}\). Since \(V_1 \cap V_2\) is constructor-root, there must be two leaves \((v_1, \tau_1) \in V_1, (v_2, \tau_2) \in V_2\) and a substitution \(\tau_3\) s.t. \(\tau_3 \in CSU_{\text{EUB}}^{W_1}(v_1 = v_2)\), \((v_1, \tau_3)|_{W_1} = \bot (\tau_2 \tau_3)|_{W_1}\), and \((\tau_1 \cup \tau_2) \tau_3\) is constructor-root. Furthermore, the variant \((u_1, \theta_1)\) (resp. \((u_2, \theta_2)\)) is obtained by further narrowing of \(v_1\) (resp. \(v_2\)), i.e., \((u_1, \tau_1') \in \tilde{v_1}_{E,B}\) and \((u_2, \tau_2') \in \tilde{v_2}_{E,B}\). But then the conclusion follows, since the statement is \((\tau_1 \cup \tau_2) \tau_3|_{W_1} \not\subseteq \text{EUB} ((\theta_1 \cup \theta_2) \sigma)|_{W_1}\). □

5 Experimental Evaluation

We have performed some experiments with the constructor-root variant-based unification, which are available at [http://safe-tools.dsic.upv.es/cr-mgvu](http://safe-tools.dsic.upv.es/cr-mgvu).

All the experiments were conducted on a PC with a 3.3GHz Intel Xeon E5-1660 and 64GB RAM. We created a battery of 15 different unification problems for both the exclusive-or and the abelian group theories. These are among the most complicated cryptographic theories in protocol analysis that Maude-NPA [19], Tamarin [12] and AKISS [5] can hardly handle. Indeed, the exclusive-or and the abelian group theories cannot be specified in Maude using constructor symbols and we introduce arbitrary constructors \(f_1, f_2, f_3, f_4, f_5\), where the subindex indicates the number of arguments. This is a common situation in crypto protocol analysis where the cryptographic properties are combined with many different additional constructor symbols. Experiments using other cryptographic theories, such as Diffie-Hellmann exponentiation, or more traditional programs, such as manipulating complex data structures, could also have been included but were discarded because the improvement is less remarkable.

For each problem and theory, we computed: (i) the unifiers using the standard variant unify command provided by the C++ core system of Maude; (ii) the unifiers using the algorithm \(CSU_{\text{EUB}}^{t \mapsto t'}(t = t')\) of [21]; (iii) the unifiers using the algorithm \(CSU_{\text{EUB}}^{t \mapsto t'}(t = t')\) of Definition 11 and (iv) the unifiers using the algorithm \(CSU_{\text{EUB}}^{t \mapsto t'}(t = t')\) obtained from \(CSU_{\text{EUB}}^{t \mapsto t'}(t = t')\) by replacing \(CSU_{\text{EUB}}^{\cap V_1, V_2}(t = t')\) with \(CSU_{\text{EUB}}^{t_1, t_2'}(t = t')\) from [21]. Note that (ii), (iii), and (iv) are implemented at the metalevel of Maude. We measured the number of computed unifiers and the time required for their computation.

Table 1 (resp. Table 2) shows the results obtained for the exclusive-or (resp. abelian group) theory. T/O indicates that a generous 4 hours timeout was reached without any response. The first column describes the unification problem, while the following #maude, #fast, #n and #er-fast columns show the
number of computed unifiers for all four unification algorithms (i), (ii), (iii), (iv) described above and the columns \( T_{\text{maude}}, T_{\text{fast}}, T_{\text{cr}} \) and \( T_{\text{cr+fast}} \) show the time (in milliseconds) required to execute the unification command. Note that it is unfair to compare the performance between compiled code \( (T_{\text{maude}} \) column) and interpreted code \( (T_{\text{fast}}, T_{\text{cr}} \) and \( T_{\text{cr+fast}} \) columns), i.e., the C++ core system of Maude and a Maude program using Maude’s metalevel. However, our constructor-root unification algorithm is able to beat the compiled code in almost all the unification problems.

Tables 1 and 2 show that the \( cr+fast \) combination is the best choice, since it combines the benefits of both the \( fast \) unification algorithm of [21] and the new constructor-root unification algorithm \( cr \). For the number of unifiers, \( cr \) always reported less unifiers than Maude except for problem \( P_{20} \), where both report the same number. However, both the \( cr \) and the \( fast \) algorithm are incomparable and \( cr \) reported less unifiers than \( fast \) in the unification problems \( P_6 \) and \( P_{10} \), whereas \( fast \) reported less unifiers than \( cr \) in the unification problems \( P_3, P_5, P_{18}, P_{19}, P_{20} \). As for the execution time, \( cr \) can beat both Maude and the \( fast \) algorithm for almost all the unification problems. Indeed, unification in the abelian group is so complex that neither Maude nor \( fast \) can terminate in most of the unification problems (e.g., \( P_{22}, P_{23}, P_{24}, P_{25}, \) and more), whereas \( cr \) did.

Our best contribution are the non-unifiable problems in the third block of Tables 1 and 2. Our new constructor-root unification algorithm immediately terminates, whereas neither Maude nor \( fast \) could, as shown in the unification problems \( P_{11}, P_{12}, P_{13}, P_{14}, P_{15}, P_{26}, P_{27}, P_{28}, P_{29}, P_{30} \).

### 6 Conclusion and Future Work

The variant-based equational unification algorithm implemented in the most recent version of Maude, version 3.0, may compute many more unifiers than the necessary or may not be able to stop immediately. Constructor symbols are extensively used in computer science, but they have not been integrated into the variant-based equational unification procedure of Maude. In this paper, we have redefined the
variant-based unification algorithm and our experiments on some unification problems show an impressive speedup. Especially for non-unifiable problems, where many resources are wasted.

As far as we know, this is the only research line to reduce the number of variant unifiers. The closest work is to combine standard unification algorithms with variant-based unification, such as [18, 17]. Note that the constructor variant unification of [32] is not connected to our work, since it is based on a new notion of constructor variant. This is just a step forward on developing new techniques for improving variant-based unification and we plan to reduce even more the number of variant unifiers.

Table 2: Experimental evaluation (abelian group)

| Unification problem | #instade | #fast | #ce | #ce+fast | #instade | #fast | #ce | #ce+fast |
|---------------------|----------|-------|-----|----------|----------|-------|-----|----------|
| $P_1$               | 3702     | 1     | 1   | 1        | 4344002  | 5034046| 1   | 1        |
| $P_2$               | 8153     | 2     | 2   | 2        | 5413107  | 5413107| 2   | 2        |
| $P_3$               | 3615     | 2     | 2   | 2        | 65746    | 65746  | 2   | 2        |
| $P_4$               | 376      | 2     | 2   | 2        | 26425    | 26425  | 2   | 2        |
| $P_5$               | 316      | 2     | 2   | 2        | 10202    | 10202  | 2   | 2        |
| $P_6$               | 158      | 1     | 1   | 1        | 426      | 426    | 1   | 1        |
| $P_7$               | -        | -     | -   | -        | T/O      | T/O    | -   | -        |
| $P_8$               | -        | -     | -   | -        | T/O      | T/O    | -   | -        |
| $P_9$               | -        | -     | -   | -        | T/O      | T/O    | -   | -        |
| $P_{10}$            | -        | -     | -   | -        | T/O      | T/O    | -   | -        |

References

[1] M. Alpuente, S. Escobar & J. Iborra (2009): Termination of Narrowing Revisited. Theoretical Computer Science 410(46), pp. 4608–4625, doi:10.1016/j.tcs.2009.07.037.

[2] M. Alpuente, S. Escobar & J. Iborra (2011): Modular Termination of Basic Narrowing and Equational Unification. Logic Journal of the IGPL 19(6), pp. 731–762, doi:10.1007/s1078-3-540-70590-1-1.

[3] F. Baader & W. Snyder (2001): Unification Theory. In J. A. Robinson & A. Voronkov, editors: Handbook of Automated Reasoning, 1, Elsevier Science, pp. 447–533, doi:10.1016/B978-044450813-3/50010-2.

[4] K. Bae, S. Escobar & J. Meseguer (2013): Abstract Logical Model Checking of Infinite-State Systems Using Narrowing. In: Proceedings of the 24th International Conference on Rewriting Techniques and Applications (RTA 2013). LIPIcs 21, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 81–96, doi:10.4230/LIPIcs.RTA.2013.81.

[5] D. Baelde, S. Delaune, I. Gazeau & S. Kremer (2017): Symbolic Verification of Privacy-Type Properties for Security Protocols with XOR. In: Proceedings of the 30th International Symposium on Computer Security Foundations (CSF 2017), IEEE Computer Society Press, pp. 234–248, doi:10.1109/CSF.2017.22.
[6] C. Bouchard, K. A. Gero, C. Lynch & P. Narendran (2013): *On Forward Closure and the Finite Variant Property*. In: *Proceedings of the 9th International Symposium on Frontiers of Combining Systems* (FroCos 2013), Lecture Notes in Computer Science 8152, Springer, pp. 327–342, doi:10.1007/978-3-642-40885-4_23

[7] R. M. Burstall & J. A. Goguen (1982): *Algebras, Theories and Freeness: An Introduction for Computer Scientists*. In M. Broy & G. Schmidt, editors: *Theoretical Foundations of Programming Methodology*, NATO Science Series 91, Springer, pp. 329–349, doi:10.1007/978-3-642-94009-3_5

[8] A. Cholewa, J. Meseguer & S. Escobar (2014): *Variants of Variants and the Finite Variant Property*. Technical Report, University of Illinois at Urbana-Champaign. Available at [http://hdl.handle.net/2142/47117](http://hdl.handle.net/2142/47117)

[9] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio & C. Talcott (2020): *Maude Manual (Version 3.0)*. Technical Report, SRI International Computer Science Laboratory. Available at [http://maude.cs.uiuc.edu](http://maude.cs.uiuc.edu)

[10] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer & C. Talcott (2007): *All About Maude: A High-Performance Logical Framework*. Springer, doi:10.1007/978-3-540-71999-1

[11] H. Comon-Lundh & S. Delaune (2005): *The Finite Variant Property: How to Get Rid of Some Algebraic Properties*. In: *Proceedings of the 16th International Conference on Rewriting Techniques and Applications* (RTA 2005), Lecture Notes in Computer Science 3467, Springer, pp. 294–307, doi:10.1007/978-3-540-32033-3_22

[12] J. Dreier, L. Hirschi, S. Radomirovic & R. Sasse (2018): *Automated Unbounded Verification of Stateless Cryptographic Protocols with Exclusive OR*. In: *Proceedings of the 31st International Symposium on Computer Security Foundations* (CSF 2018), IEEE Computer Society Press, pp. 359–373, doi:10.1109/CSF.2018.00033

[13] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio & C. Talcott (2020): *Programming and Symbolic Computation in Maude*. Journal of Logical and Algebraic Methods in Programming 110, doi:10.1016/j.jlamp.2019.100497

[14] F. Durán, S. Lucas & J. Meseguer (2009): *Termination Modulo Combinations of Equational Theories*. In: *Proceedings of the 7th International Symposium on Frontiers of Combining Systems* (FroCos 2009), Lecture Notes in Computer Science 5749, Springer, pp. 246–262, doi:10.1007/978-3-642-04222-5_15

[15] F. Durán & J. Meseguer (2012): *On the Church-Rosser and Coherence Properties of Conditional Order-sorted Rewrite Theories*. The Journal of Logic and Algebraic Programming 81(7–8), pp. 816–850, doi:10.1016/j.jlap.2011.12.004

[16] F. Durán, J. Meseguer & C. Rocha (2020): *Ground Confluence of Order-Sorted Conditional Specifications Modulo Axioms*. Journal of Logical and Algebraic Methods in Programming 111, p. 100513, doi:10.1016/j.jlamp.2019.100513

[17] A. K. Eeralla, S. Erbatur, A. M. Marshal & C. Ringeissen (2019): *Rule-based Unification in Combined Theories and the Finite Variant Property*. In: *Proceedings of the 13th International Conference on Language and Automata Theory and Applications* (LATA 2019), Lecture Notes in Computer Science 11417, Springer, pp. 356–367, doi:10.1007/978-3-030-13435-8_26

[18] S. Erbatur, D. Kapur, A. M. Marshall, P. Narendran & C. Ringeissen (2015): *Unification and Matching in Hierarchical Combinations of Syntactic Theories*. In: *Proceedings of the 10th International Symposium on Frontiers of Combining Systems* (FroCos 2015), Lecture Notes in Computer Science 9322, Springer, pp. 291–306, doi:10.1007/978-3-319-24246-0_18

[19] S. Escobar, C. Meadows & J. Meseguer (2009): *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties*. In: *Foundations of Security Analysis and Design V* (FOSAD 2007/2008/2009 Tutorial Lectures), Lecture Notes in Computer Science 5705, Springer, pp. 1–50, doi:10.1007/978-3-642-03829-7_1
D. Aparicio-Sánchez, S. Escobar, and J. Sapiña

[20] S. Escobar & J. Meseguer (2007): Symbolic Model Checking of Infinite-State Systems Using Narrowing. In: Proceedings of the 18th International Conference on Term Rewriting and Applications (RTA 2007), Lecture Notes in Computer Science 4533, Springer, pp. 153–168, doi:10.1007/978-3-540-73449-9_13

[21] S. Escobar & J. Sapiña (2019): Most General Variant Unifiers. In: Proceedings of the 35th International Conference on Logic Programming (ICLP 2019) - Technical Communications, Electronic Proceedings in Theoretical Computer Science 306, Open Publishing Association, pp. 154–167, doi:10.4204/EPTCS.306.21

[22] S. Escobar, R. Sasse & J. Meseguer (2012): Folding Variant Narrowing and Optimal Variant Termination. The Journal of Logic and Algebraic Programming 81(7–8), pp. 898–928, doi:10.1016/j.jlap.2012.01.002

[23] J. P. Jouannaud, C. Kirchner & H. Kirchner (1983): Incremental Construction of Unification Algorithms in Equational Theories. In: Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 1990), Lecture Notes in Computer Science 154, Springer, pp. 361–373, doi:10.1007/BFb0036921.

[24] J. P. Jouannaud & H. Kirchner (1986): Completion of a Set of Rules Modulo a Set of Equations. SIAM Journal on Computing 15(4), pp. 6–9, doi:10.1137/0215084.

[25] D. Kapur & P. Narendran (1987): Matching, Unification and Complexity. ACM SIGSAM Bulletin 21(4), pp. 6–9, doi:10.1145/36330.36332.

[26] S. Lucas & J. Meseguer (2016): Normal Forms and Normal Theories in Conditional Rewriting. Journal of Logical and Algebraic Methods in Programming 85, pp. 67–97, doi:10.1016/j.jlamp.2015.06.001.

[27] J. Meseguer (1992): Conditional Rewriting Logic as a United Model of Concurrency. Theoretical Computer Science 96(1), pp. 73–155, doi:10.1016/0304-3975(92)90182-F.

[28] J. Meseguer (1997): Membership Algebra as a Logical Framework for Equational Specification. In: Proceedings of the 12th International Workshop on Algebraic Development Techniques (WADT 1997), Lecture Notes in Computer Science 1376, Springer, pp. 18–61, doi:10.1007/3-540-64299-4_26.

[29] J. Meseguer (2012): Twenty Years of Rewriting Logic. The Journal of Logic and Algebraic Programming 81(7–8), pp. 721–781, doi:10.1016/j.jlap.2012.06.003.

[30] J. Meseguer (2017): Strict Coherence of Conditional Rewriting Modulo Axioms. Theoretical Computer Science 672, pp. 1–35, doi:10.1016/j.tcs.2016.12.026.

[31] J. Meseguer (2018): Symbolic Reasoning Methods in Rewriting Logic and Maude. In: Proceedings of the 25th International Workshop on Logic, Language, Information, and Computation (WoLLIC 2018), Lecture Notes in Computer Science 10944, Springer, pp. 25–60, doi:10.1007/978-3-662-57669-4_2.

[32] J. Meseguer (2018): Variant-based Satisfiability in Initial Algebras. Science of Computer Programming 154, pp. 3–41, doi:10.1016/j.scico.2017.09.001.

[33] J. Meseguer (2020): Generalized Rewrite Theories, Coherence Completion, and Symbolic Methods. Journal of Logical and Algebraic Methods in Programming 110, doi:10.1016/j.jlamp.2019.100483.

[34] A. Riesco (2014): Using Big-Step and Small-Step Semantics in Maude to Perform Declarative Debugging. In: Proceedings of the 12th International Symposium on Functional and Logic Programming (FLOPS 2014), Lecture Notes in Computer Science 8475, Springer, pp. 52–68, doi:10.1007/978-3-319-07151-0_4.

[35] V. Rusu (2010): Combining Theorem Proving and Narrowing for Rewriting-Logic Specifications. In: Proceedings of the 4th International Conference on Tests and Proofs (TAP 2010), Lecture Notes in Computer Science 6143, Springer, pp. 135–150, doi:10.1007/978-3-642-13977-2_12.

[36] TeReSe (2003): Term Rewriting Systems. Cambridge University Press, doi:10.1017/S095679680400526X.

[37] E. Tushkanova, A. Giorgetti, C. Ringeissen & O. Kouchnarenko (2015): A Rule-based System for Automatic Decidability and Combinability. Science of Computer Programming 99, pp. 3–23, doi:10.1016/j.scico.2014.02.005.