

Most General Variant Unifiers*

Santiago Escobar

Julia Sapiña

VRAIN (Valencian Research Institute for Artificial Intelligence)
Universitat Politècnica de València
Valencia, Spain
{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. Equational unification is of special relevance to automated deduction, theorem proving, protocol analysis, partial evaluation, model checking, etc. Several algorithms have been developed in the literature for specific equational theories, such as associative-commutative symbols, exclusive-or, Diffie-Hellman, or Abelian Groups. Narrowing was proved to be complete for unification and several cases have been studied where narrowing provides a decidable unification algorithm. A new narrowing-based equational unification algorithm relying on the concept of the *variants* of a term has been developed and it is available in the most recent version of Maude, version 2.7.1, which provides quite sophisticated unification features. A variant of a term t is a pair consisting of a substitution σ and the canonical form of $t\sigma$. Variant-based unification is decidable when the equational theory satisfies the *finite variant property*. However, it may compute many more unifiers than the necessary and, in this paper, we explore how to strengthen the variant-based unification algorithm implemented in Maude to produce a minimal set of most general variant unifiers. Our experiments suggest that this new adaptation of the variant-based unification is more efficient both in execution time and in the number of computed variant unifiers than the original algorithm available in Maude.

1 Introduction

Equational unification of two terms is of special relevance to many areas in computer science 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 was proved to be complete for unification [21, 22] and several cases have been studied where narrowing provides a decidable unification algorithm [1, 2]. A new narrowing-based equational unification algorithm relying on the concept of the *variants* of a term [9] has been developed [19] and it is available in the most recent version of Maude, version 2.7.1, which provides quite sophisticated unification features [7, 31].

Several tools and techniques rely on Maude’s advanced unification capabilities, such as termination [13] and local confluence and coherence [14] proofs, narrowing-based theorem proving [34] or testing [33], and *logical model checking* [18, 4]. The area of cryptographic protocol analysis has also benefited: the Maude-NPA tool [17] is the most successful example of using variant-based equational unification in Maude and the Tamarin tool [26, 10, 11] also relies on variants. Numerous decision procedures for formula satisfiability modulo equational theories also rely on unification, either based on narrowing [36] or by using variant generation in finite variant theories [32].

*This work has been partially supported by the EU (FEDER) and the Spanish MCIU under grant RTI2018-094403-B-C32, by the Spanish Generalitat Valenciana under grants PROMETEO/2019/098 and APOSTD/2019/127, and by the US Air Force Office of Scientific Research under award number FA9550-17-1-0286.

However, variant-based unification may compute many more unifiers than the necessary. In this paper, we explore how to improve the variant-based unification algorithm implemented in Maude to produce a smaller, yet complete, set of most general variant unifiers. After some preliminaries in Section 2, we recall variant-based unification in Section 3 and propose how to compute a set of most general variant unifiers in Section 4. In Section 5, we propose a new fast algorithm that considerably reduces the number of variant unifiers by computing a complete (yet not always minimal) set of most general unifiers modulo the considered theory. Our experiments in Section 6 demonstrate that this new adaptation of the variant-based unification is more efficient both in execution time and in the number of computed variant unifiers than the original algorithm. We conclude in Section 7.

2 Preliminaries

We follow the classical notation and terminology from [35] 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 Σ 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. $\mathcal{T}_\Sigma(\mathcal{X})_s$ is the set of terms of sort s , and $\mathcal{T}_{\Sigma, s}$ is the set of ground terms of sort s . We write $\mathcal{T}_\Sigma(\mathcal{X})$ and \mathcal{T}_Σ for the corresponding order-sorted term algebras. Given a term t , $\text{Var}(t)$ denotes the set of variables in t .

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 $\sigma = \{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$ where the domain of σ is $\text{Dom}(\sigma) = \{X_1, \dots, X_n\}$ and the set of variables introduced by terms t_1, \dots, 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 σ to a term t is denoted by $t\sigma$ or $\sigma(t)$. For simplicity, we assume that every substitution is idempotent, i.e., σ satisfies $\text{Dom}(\sigma) \cap \text{Ran}(\sigma) = \emptyset$. The restriction of σ 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 σ and σ' is denoted by $\sigma\sigma'$. Combination of two substitutions σ and σ' such that $\text{Dom}(\sigma) \cap \text{Dom}(\sigma') = \emptyset$ is denoted by $\sigma \cup \sigma'$. We call an idempotent substitution σ a *variable renaming* if there is another idempotent substitution σ^{-1} such that $(\sigma\sigma^{-1})|_{\text{Dom}(\sigma)} = \text{id}$.

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

An E -unifier for a Σ -equation $t = t'$ is a substitution σ such that $t\sigma =_E t'\sigma$. For $\text{Var}(t) \cup \text{Var}(t') \subseteq W$, a set of substitutions $CSU_E^W(t = t')$ is said to be a *complete* set of unifiers for the equality $t = t'$ modulo E away from W iff: (i) each $\sigma \in CSU_E^W(t = t')$ is an E -unifier of $t = t'$; (ii) for any E -unifier ρ of $t = t'$ there is a $\sigma \in CSU_E^W(t = t')$ such that $\sigma|_W \sqsupseteq_E \rho|_W$; and (iii) for all $\sigma \in CSU_E^W(t = t')$, $\text{Dom}(\sigma) \subseteq (\text{Var}(t) \cup \text{Var}(t'))$ and $\text{Ran}(\sigma) \cap W = \emptyset$. Given a conjunction Γ of equations, a set U of E -unifiers of Γ is said to be *minimal* if it is complete and for all distinct elements σ and σ' in U , $\sigma \sqsupseteq_E \sigma'$ 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 $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_s$ for some sort $s \in S$. An (*unconditional*) *order-sorted rewrite theory* is a triple (Σ, E, R) with Σ an order-sorted signature, E a set of Σ -equations, and R a set of rewrite rules. The set R of rules is *sort-decreasing* if for each $t \rightarrow t'$ in R , each $s \in S$, and each substitution $\sigma, t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_s$ implies $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_s$. The rewriting relation on $\mathcal{T}_\Sigma(\mathcal{X})$, written $t \rightarrow_{p,R} t'$ (or just $t \rightarrow_R t'$) holds between t and t' iff there exist $p \in \text{Pos}_\Sigma(t)$, $l \rightarrow r \in R$ and a substitution σ , such that $t|_p = l\sigma$, and $t' = t[r\sigma]_p$. The relation $\rightarrow_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $=_E; \rightarrow_R; =_E$. 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 [23, 30]). A relation $\rightarrow_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \rightarrow_{p,R,E} t'$ (or just $t \rightarrow_{R,E} t'$) iff there is a non-variable position $p \in \text{Pos}_\Sigma(t)$, a rule $l \rightarrow r$ in R , and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. The narrowing relation $\rightsquigarrow_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \overset{\sigma}{\rightsquigarrow}_{p,R,E} t'$ (or just $t \overset{\sigma}{\rightsquigarrow}_{R,E} t'$) iff there is a non-variable position $p \in \text{Pos}_\Sigma(t)$, a rule $l \rightarrow r$ in R , and a substitution σ such that $t|_p \sigma =_E l\sigma$ and $t' = (t[r]_p)\sigma$.

We call (Σ, B, E) a *decomposition* of an order-sorted equational theory $(\Sigma, E \uplus B)$ if B is regular, linear, sort-preserving, defined using top sorts, and has a finitary and complete unification algorithm, which implies that B -matching is decidable, and equations E are oriented into rules \vec{E} such that they are sort-decreasing and *convergent*, i.e., confluent, terminating, and strictly coherent modulo B [14, 25, 30]. The irreducible version of a term t is denoted by $t \downarrow_{R,E}$.

Given a decomposition (Σ, B, E) of an equational theory and a term t , a pair (t', θ) of a term t' and a substitution θ is an E, B -variant (or just a variant) of t if $t\theta \downarrow_{E,B} =_B t'$ and $\theta \downarrow_{E,B} =_B \theta$ [9, 19]. A *complete set of E, B -variants* [19] (up to renaming) of a term t is a subset, denoted by $\llbracket t \rrbracket_{E,B}$, of the set of all E, B -variants of t such that, for each E, B -variant (t', σ) of t , there is an E, B -variant $(t'', \theta) \in \llbracket t \rrbracket_{E,B}$ such that $(t'', \theta) \sqsupseteq_{E,B} (t', \sigma)$, i.e., there is a substitution ρ such that $t' =_E t''\rho$ and $\sigma|_{\text{Var}(t)} =_E (\theta\rho)|_{\text{Var}(t)}$. A decomposition (Σ, B, E) has the *finite variant property* (FVP) [19] (also called a *finite variant decomposition*) iff for each Σ -term t , there exists a complete and finite set $\llbracket t \rrbracket_{E,B}$ of variants of t . Note that whether a decomposition has the finite variant property is undecidable [5] but a technique based on the dependency pair framework has been developed in [19] and a semi-decision procedure that works well in practice is available in [6].

3 Variant-based Equational Unification in Maude 2.7.1

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 [7], which has itself a formal environment of verification tools thanks to its reflective capabilities (see [8, 29]).

Since 2007, several symbolic capabilities have been successively added to Maude (see [12, 31] and references therein). First, Maude has been endowed with unification, i.e., *order-sorted equational unification*. Second, Maude has been extended with symbolic reachability features that rely on Maude's unification, i.e., *narrowing-based reachability analysis* as well as the more general *symbolic LTL model checking of infinite-state systems* [18, 4]. However, Maude's unification features are quite general in nature: (i) they are applicable to order-sorted signatures; (ii) they work modulo any combination of the equational axioms of associativity (A), commutativity (C), and identity (U); and (iii) they work modulo a set of equations that are assumed convergent modulo axioms. The third part is supported via the concept of the *variants* of a term [9] and the *folding variant narrowing strategy* [19], which achieves termination when the equational theory has the *finite variant property* [9, 19]. All these unification capabilities are seamlessly provided by a variant-based unification command in Maude, as shown below.

Equational unification can be simply understood as variant computation in an extended equational theory.

Definition 1. [19] *Given a decomposition (Σ, B, E) with a poset of sorts (S, \leq) of an equational theory (Σ, \mathcal{E}) , we extend (Σ, B, E) and (S, \leq) to $(\widehat{\Sigma}, B, \widehat{E})$ and (\widehat{S}, \leq) as follows:*

1. *we add a new sort Truth to \widehat{S} , not related to any sort in Σ ,*
2. *we add a constant operator tt of sort Truth to $\widehat{\Sigma}$,*
3. *for each top sort of a connected component $[s]$, we add an operator $\text{eq} : [s] \times [s] \rightarrow \text{Truth}$ to $\widehat{\Sigma}$, and*
4. *for each top sort $[s]$, we add a variable $X:[s]$ and an extra rule $\text{eq}(X:[s], X:[s]) \rightarrow \text{tt}$ to \widehat{E} .*

Then, given any two Σ -terms t, t' , if θ is an equational unifier of t and t' , then the E, B -canonical forms of $t\theta$ and $t'\theta$ must be B -equal and therefore the pair (tt, θ) must be a variant of the term $\text{eq}(t, t')$. Furthermore, if the term $\text{eq}(t, t')$ has a finite set of most general variants, then we are *guaranteed* that the set of most general \mathcal{E} -unifiers of t and t' is *finite*.

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', σ) of a term t uses new freshly generated variables.

Definition 2 (Variant Intersection). [19] *Given a decomposition (Σ, B, E) of an equational theory, two Σ -terms t_1 and t_2 such that $W_\cap = \text{Var}(t_1) \cap \text{Var}(t_2)$ 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 define $V_1 \cap V_2 = \{(u_1\sigma, \theta_1\sigma \cup \theta_2\sigma \cup \sigma) \mid (u_1, \theta_1) \in V_1 \wedge (u_2, \theta_2) \in V_2 \wedge \exists \sigma : \sigma \in \text{CSU}_B^{W_\cup}(u_1 = u_2) \wedge (\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}\}$.*

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

Proposition 3 (Variant-based Unification). [19] *Let (Σ, B, E) be a decomposition of an equational theory. Let t_1, t_2 be two Σ -terms. Then, ρ is an unifier of t_1 and t_2 iff $\exists (t', \rho) \in \llbracket t_1 \rrbracket_{E, B} \cap \llbracket t_2 \rrbracket_{E, B}$.*

The most recent version 2.7.1 of Maude [7] incorporates variant-based unification based on the folding variant narrowing strategy [19]. First, there exists a variant generation command of the form:

```
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:

```
variant unify [ n ] in ModId : T1 =? T1' /\ ... /\ 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 . Note that the theory is not coherent modulo AC without the second equation.

```
fmod EXCLUSIVE-OR is
  sorts Elem ElemXor .
  subsort Elem < ElemXor .
  ops a b c : -> Elem .
  op mt : -> ElemXor .
  op *_ : ElemXor ElemXor -> ElemXor [assoc comm] .
  vars X Y Z U V : [ElemXor] .
  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 [9, 19]), 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
[ElemXor]: #1:[ElemXor] * #2:[ElemXor]  ...  [ElemXor]: %1:[ElemXor]
X --> #1:[ElemXor]          ...          X --> %1:[ElemXor]
Y --> #2:[ElemXor]          ...          Y --> mt
```

Note that there are two forms of fresh variables, $\#n:Sort$ and $\%n:Sort$, depending on whether they are generated by unification modulo axioms or by narrowing with the equations modulo axioms. Also note that the two forms have different counters.

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
X --> %1:[ElemXor] * %3:[ElemXor]
Y --> %2:[ElemXor] * %4:[ElemXor]
V --> %1:[ElemXor] * %2:[ElemXor]
U --> %3:[ElemXor] * %4:[ElemXor]

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

Note that this method does not provide an equational unification algorithm in general: given an equational theory (Σ, \mathcal{E}) and two terms t, t' that have a finite, minimal, and complete set of equational

unifiers modulo \mathcal{E} , the equational theory \mathcal{E} may not have a finite variant decomposition. An example is the unification under homomorphism (or one-side distributivity), where there is a finite number of unifiers of two terms but the theory does not satisfy the finite variant property (see [9, 19]).

The following result from [19] ensures a complete set of unifiers for a finite variant decomposition.

Corollary 4 (Finitary \mathcal{E} -unification). [19] *Let (Σ, B, E) be a finite variant decomposition of an equational theory. Given two terms t, t' , the set $CSU_{E \cup B}^\cap(t = t') = \{\theta \mid (w, \theta) \in \llbracket t \rrbracket_{E, B} \cap \llbracket t' \rrbracket_{E, B}\}$ is a finite and complete set of unifiers for $t = t'$.*

However, Corollary 4 does not provide a minimal set of *most general* unifiers w.r.t. the $\sqsupseteq_{E \cup B}$ relation. 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 [24]. For the unification problem $X * Y \stackrel{?}{=} U * V$ of Example 1, the most general unifier w.r.t. $\sqsupseteq_{E \cup B}$ is $\{X \mapsto Y * U * V\}$, which should be appropriately written as

$$\sigma = \{X \mapsto Y' * U' * V', Y \mapsto Y', U \mapsto U', V \mapsto V'\}.$$

Note that $\{Y \mapsto X * U * V\}$, $\{U \mapsto Y * X * V\}$, and $\{V \mapsto Y * U * X\}$ are equivalent to the former unifier w.r.t. $\sqsupseteq_{E \cup B}$ by composing σ with, respectively, $\rho_1 = \{Y' \mapsto X'' * U'' * V'', X' \mapsto X'', U' \mapsto U'', V' \mapsto V''\}$, $\rho_2 = \{U' \mapsto Y'' * X'' * V'', X' \mapsto X'', Y' \mapsto Y'', V' \mapsto V''\}$, and $\rho_3 = \{V' \mapsto Y'' * U'' * X'', X' \mapsto X'', U' \mapsto U'', Y' \mapsto Y''\}$. Similarly, $\{X \mapsto U, Y \mapsto V\}$ and $\{X \mapsto V, Y \mapsto U\}$ are equivalent to all the previous ones.

4 Computing More General Variant Unifiers

Note that when (Σ, B, E) is a finite variant decomposition and B -unification is finitary, we get an $E \cup B$ -matching algorithm as $Match_{E \cup B}(u, v) = \{\theta \mid \bar{\theta} \in CSU_{E \cup B}^\cap(u = \bar{v})\}$, where \bar{v} is obtained from v by turning its variables x_1, \dots, x_n into fresh constants $\bar{x}_1, \dots, \bar{x}_n$, and θ is obtained from $\bar{\theta}$ by, given a binding $x \mapsto \bar{t} \in \bar{\theta}$, adding the binding $x \mapsto t$ to θ ; the term t is easily obtained from \bar{t} by replacing every occurrence of a fresh constant $\bar{x}_1, \dots, \bar{x}_n$ by its original. We say $t \sqsupseteq_{E \cup B} t'$ if $Match_{E \cup B}(t, t') \neq \emptyset$, and $t \sqsubset_{E \cup B} t'$ if $t \sqsupseteq_{E \cup B} t'$ and $t \neq_{E \cup B} t'$.

It is easy to provide, at the theoretical level, a minimal set of most general variant unifiers by post-filtering the set of computed unifiers by using $\sqsupseteq_{E \cup B}$.

Proposition 5 (Post-filtered Variant-based Unification). *Let (Σ, B, E) be a finite variant decomposition of an equational theory. Given two terms t, t' , the set $CSU_{E \cup B}^{\cap, \sqsupseteq}(t = t') = \{\theta \mid \theta \in CSU_{E \cup B}^\cap(t = t') \wedge \nexists \theta' \in CSU_{E \cup B}^\cap(t = t') \setminus \{\theta\} : \theta' \sqsubset_{E \cup B} \theta\}$ is a finite and complete set of unifiers for $t = t'$. Even more, the quotient $CSU_{E \cup B}^{\cap, \sqsupseteq}(t = t') / \simeq_{E \cup B}$ w.r.t. the equivalence relation $\simeq_{E \cup B}$ induced from $\sqsupseteq_{E \cup B}$ is a finite, minimal, and complete set of unifiers for $t = t'$.*

We have implemented both post-filtering stages $CSU_{E \cup B}^{\cap, \sqsupseteq}(t = t')$ and $CSU_{E \cup B}^{\cap, \sqsupseteq}(t = t') / \simeq_{E \cup B}$ in an extended version of Full Maude version 27g [20] available at <http://safe-tools.dsic.upv.es/mgvu>. The new command implementing the algorithm $CSU_{E \cup B}^{\cap, \sqsupseteq}(t = t')$ is as follows:

```
(post variant unify [ n ] in ModId : T1 =? T1' /\ ... /\ 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.

When we consider the previous variant unification problem between terms $X * Y$ and $U * V$, now we get just 7 unifiers from the 57 unifiers above.

```

Maude> (post variant unify in EXCLUSIVE-OR : X * Y =? U * V .)
Unifier #1
X --> %1:[ElemXor] * %3:[ElemXor]
Y --> %2:[ElemXor] * %4:[ElemXor]
V --> %1:[ElemXor] * %2:[ElemXor]
U --> %3:[ElemXor] * %4:[ElemXor]
...
Unifier #7
X --> %2:[ElemXor]
Y --> %1:[ElemXor]
V --> %1:[ElemXor]
U --> %2:[ElemXor]

```

The new command reporting the quotient $CSU_{EUB}^{\square, \square}(t = t') /_{\simeq_{EUB}}$ is as follows:

```
(post quotient variant unify [ n ] in ModId : T1 =? T1' /\ ... /\ 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.

When we consider the previous variant unification problem between terms $X * Y$ and $U * V$, now we get just one unifier, since all the seven unifiers reported before are equivalent modulo exclusive-or.

```

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

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

```

5 Fast Computation of More General Variant Unifiers

The computation of both $CSU_{EUB}^{\square, \square}(t = t')$ and $CSU_{EUB}^{\square, \square}(t = t') /_{\simeq_{EUB}}$ is extremely expensive (see Section 6 below), both in execution time and memory usage, because we must use the same variant-based unification command in Maude for obtaining the variant unifiers and then for filtering them. In this section, we provide the main contribution of this paper on improving the computation of a set of most general variant unifiers. Let us motivate our main results with an example.

When we consider a variant unification problem between terms X and $U * V$, we get an explosion of all the variants of $U * V$.

```

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

Unifier #1
X --> %1:[ElemXor] * %2:[ElemXor]
V --> %1:[ElemXor]
U --> %2:[ElemXor]

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

Unifier #3
X --> #2:[ElemXor] * #3:[ElemXor]
V --> #1:[ElemXor] * #2:[ElemXor]
U --> #1:[ElemXor] * #3:[ElemXor]

```

```

Unifier #4
X --> #1:[ElemXor]
V --> #1:[ElemXor] * #2:[ElemXor]
U --> #2:[ElemXor]

```

```

Unifier #5
X --> #1:[ElemXor]
V --> #2:[ElemXor]
U --> #1:[ElemXor] * #2:[ElemXor]

```

```

Unifier #6
X --> #1:[ElemXor]
V --> mt
U --> #1:[ElemXor]

```

```

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

```

but it is clear that the simplest, most general unifier is $\{X \mapsto U * V\}$

```
Maude> (post quotient variant unify in EXCLUSIVE-OR : X =? U * V .)
```

```

Unifier #1
X --> %1:[ElemXor] * %2:[ElemXor]
V --> %1:[ElemXor]
U --> %2:[ElemXor]

```

The main idea here, common to any unification algorithm (see [3]), is that when a variable is found, i.e., $X \stackrel{?}{=} t$, there is no need to search for further unifiers, since any other unifier will be an instance of $X \mapsto t$. We have formalized this idea but extended it to the case of having any context $C[X] \stackrel{?}{=} C[t]$. Indeed, we have formalized it for the very general case of having any context modulo B , i.e., $C_1[X] \stackrel{?}{=} C_2[t]$ s.t. $C_1[\square] =_B C_2[\square]$. The following auxiliary result stating that it is possible that any narrowing step from t does not interfere with C_1 , C_2 and X is essential.

Lemma 6. *Given a decomposition (Σ, B, E) of an equational theory, two Σ -terms t_1 and t_2 s.t. $W_\cap = \text{Var}(t_1) \cap \text{Var}(t_2)$ and $W_\cup = \text{Var}(t_1) \cup \text{Var}(t_2)$, $(u_1, \theta_1) \in \llbracket t_1 \rrbracket_{E,B}$, $(u_2, \theta_2) \in \llbracket t_2 \rrbracket_{E,B}$, $\sigma \in \text{CSU}_B^{W_\cup}(u_1 = u_2)$ s.t. $(\theta_1 \sigma)|_{W_\cap} =_B (\theta_2 \sigma)|_{W_\cap}$, $(u'_1, \theta'_1) \in \llbracket t_1 \rrbracket_{E,B}$ s.t. $(u'_1, \rho) \in \llbracket u_1 \rrbracket_{E,B}$ and $\theta'_1|_{W_\cup} =_B \theta_1 \rho|_{W_\cup}$, $\sigma' \in \text{CSU}_B^{W_\cup}(u'_1 = u_2)$ s.t. $(\theta'_1 \sigma)|_{W_\cap} =_B (\theta_2 \sigma)|_{W_\cap}$, and $\text{Dom}(\sigma) \cap \text{Dom}(\rho) = \emptyset$, then $((\theta_1 \cup \theta_2) \sigma)|_{W_\cup}$ and $((\theta'_1 \cup \theta_2) \sigma')|_{W_\cup}$ are both equational unifiers of t_1 and t_2 but $((\theta_1 \cup \theta_2) \sigma)|_{W_\cup} \not\supseteq_{E \cup B} ((\theta'_1 \cup \theta_2) \sigma')|_{W_\cup}$.*

Proof. The statement of the Lemma is depicted in Figure 1. The proof is done by realizing that $\text{Dom}(\sigma) \cap \text{Dom}(\rho) = \emptyset$ implies that $((\theta_1 \cup \theta_2)(\sigma \cup \rho))|_{W_\cup}$ is also a unifier of t_1 and t_2 , and then

$$(\theta_1 \cup \theta_2) \sigma|_{W_\cup} \supseteq_{E \cup B} (\theta_1 \cup \theta_2)(\sigma \cup \rho)|_{W_\cup} \supseteq_{E \cup B} (\theta_1 \cup \theta_2) \sigma \rho \sigma'|_{W_\cup} =_B (\theta'_1 \cup \theta_2) \sigma'|_{W_\cup}$$

□

We redefine the intersection of two sets of variants. Note that this definition does not prevent the generation of the variants of both terms in an unification problem; techniques for avoiding the generation of variants are outside the scope of this paper.

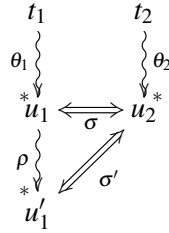


Figure 1: Sketch of the proof of Lemma 6

Definition 7 (Fast Variant Intersection). *Given a decomposition (Σ, B, E) of an equational theory, two Σ -terms t_1 and t_2 such that $W_\cap = \text{Var}(t_1) \cap \text{Var}(t_2)$ 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 define*

$$\begin{aligned}
V_1 \mathbin{\frown} V_2 = & \{ (u_1 \sigma, \theta_1 \sigma \cup \theta_2 \sigma \cup \sigma) \mid (u_1, \theta_1) \in V_1 \wedge (u_2, \theta_2) \in V_2 \wedge \\
& \exists \sigma : \sigma \in \text{CSU}_B^{W_\cup}(u_1 = u_2) \wedge (\theta_1 \sigma)|_{W_\cap} =_B (\theta_2 \sigma)|_{W_\cap} \wedge \\
& (\sharp(u'_1, \theta'_1) \in V_1, \sharp \rho : (u_1, \rho) \in \llbracket u'_1 \rrbracket_{E,B} \wedge \\
& \sharp \sigma' : \sigma' \in \text{CSU}_B^{W_\cup}(u'_1 = u_2) \wedge (\theta'_1 \sigma')|_{W_\cap} =_B (\theta_2 \sigma')|_{W_\cap} \wedge \text{Dom}(\sigma') \cap \text{Dom}(\rho) = \emptyset) \wedge \\
& (\sharp(u'_2, \theta'_2) \in V_2, \sharp \rho : (u_2, \rho) \in \llbracket u'_2 \rrbracket_{E,B} \wedge \\
& \sharp \sigma' : \sigma' \in \text{CSU}_B^{W_\cup}(u_1 = u'_2) \wedge (\theta_1 \sigma')|_{W_\cap} =_B (\theta'_2 \sigma')|_{W_\cap} \wedge \text{Dom}(\sigma') \cap \text{Dom}(\rho) = \emptyset) \}
\end{aligned}$$

Then, we define variant-based unification as the computation of the variants of the two terms in a unification problem and their *minimal* intersection; its proof is immediate by Lemma 6.

Proposition 8 (Fast Variant-based Unification). *Let (Σ, B, E) be a finite variant decomposition of an equational theory. Given two terms t, t' , on the one hand, the set $\text{CSU}_{EUB}^{\mathbin{\frown}}(t = t') = \{ \theta \mid (w, \theta) \in \llbracket t \rrbracket_{E,B} \mathbin{\frown} \llbracket t' \rrbracket_{E,B} \}$ is a finite and complete set of unifiers for $t = t'$ and the quotient $\text{CSU}_{EUB}^{\mathbin{\frown}}(t = t') / \simeq_{EUB}$ is also a (generally smaller) finite and complete set of unifiers for $t = t'$. On the other hand, the set $\text{CSU}_{EUB}^{\mathbin{\frown}, \sqsupset}(t = t') = \{ \theta \mid \theta \in \text{CSU}_{EUB}^{\mathbin{\frown}}(t = t') \wedge \sharp \theta' \in \text{CSU}_{EUB}^{\mathbin{\frown}}(t = t') \setminus \{ \theta \} : \theta' \sqsupset_{EUB} \theta \}$ is a finite and complete set of unifiers for $t = t'$. Furthermore, the quotient $\text{CSU}_{EUB}^{\mathbin{\frown}, \sqsupset}(t = t') / \simeq_{EUB}$ is a finite, minimal, and complete set of unifiers for $t = t'$.*

We have implemented these four fast unification methods in an extended version of Full Maude version 27g [20], which is available at <http://safe-tools.dsic.upv.es/mgvu>:

- The new command implementing the algorithm $\text{CSU}_{EUB}^{\mathbin{\frown}}(t = t')$ is

```
(fast variant unify [ n ] in ModId : T1 =? T1' /\ ... /\ Tk =? Tk' .)
```
- The new command implementing the algorithm $\text{CSU}_{EUB}^{\mathbin{\frown}}(t = t') / \simeq_{EUB}$ is

```
(fast quotient variant unify [ n ] in ModId : T1 =? T1' /\ ... /\ Tk =? Tk' .)
```
- The new command implementing the algorithm $\text{CSU}_{EUB}^{\mathbin{\frown}, \sqsupset}(t = t')$ is

```
(fast post variant unify [ n ] in ModId : T1 =? T1' /\ ... /\ Tk =? Tk' .)
```
- And the new command implementing the algorithm $\text{CSU}_{EUB}^{\mathbin{\frown}, \sqsupset}(t = t') / \simeq_{EUB}$ is

```
(fast post quotient variant unify [ n ] in ModId : T1 =? T1' /\ ... /\ Tk =? Tk' .)
```

For the unification problem $X * Y$ and $U * V$, the `fast` command delivers 8 unifiers instead of the 57 unifiers for standard variant unification. However, 7 of those 8 unifiers are equivalent, thus the `fast quotient` command delivers only 2 unifiers. Likewise, the `fast post` command returns the same 7 unifiers as the `post` command, and the `fast post quotient` command gets the same (most general) unifier as the `post quotient` command above. Note that the `fast unification` command and the `fast quotient unification` command compute these unifiers in a fraction of time compared to the `post unification` command and the `post quotient unification` command (see unification problem P_6 in Section 6).

When we consider the previous variant unification problem between terms X and $U * V$, now we get just one unifier as desired, and again in a fraction of time compared to $CSU_{EUB}^{\cap, \sqsupset}(t = t')$ (see unification problem P_1 in Section 6).

```
Maude> (fast variant unify in EXCLUSIVE-OR : X =? U * V .)
```

```
Unifier #1
X --> %1:[ElemXor] * %2:[ElemXor]
V --> %1:[ElemXor]
U --> %2:[ElemXor]
```

Note that, in this case, clearly the `fast post` and `fast post quotient` unification commands do not improve over the `fast unification` command.

6 Experimental Evaluation

To evaluate the performance of both the post-filtering and the fast unification techniques, we have conducted a series of benchmarks available at <http://safe-tools.dsic.upv.es/mgvu>.

All the experiments were conducted on a PC with a 3.3GHz Intel Xeon E5-1660 and 64GB RAM. First, we created a battery of 20 different unification problems for both the *exclusive-or* and the *abelian group* theories. For each problem and theory, we computed: (i) the unifiers by using the standard `variant unify` command provided by the C++ core system of Maude; (ii) the unifiers by using the `post quotient variant unify` command implemented at the metalevel of Maude; (iii) the unifiers by using the `fast quotient variant unify` command implemented at the metalevel of Maude; and (iv) the unifiers by using the `fast post quotient variant unify` command, also implemented at the metalevel of Maude. We measured both the number of computed unifiers and the time required for their computation.

Since it is unfair to compare the performance between compiled code and interpreted code, i.e., the C++ core system of Maude and a Maude program using Maude's metalevel, we have reimplemented the `variant unify` command at the metalevel and applied the post-filtering and the fast variant intersection to the output returned by this reimplementation.

Table 1 (resp. Table 2) shows the results obtained for the *exclusive-or* (resp. *abelian group*) theory. *T/O* indicates that a generous 24 hours *timeout* was reached without any response. The first column describes the unification problem, while the following $\#_{maude}$, $\#_{post}$, $\#_{fast}$, and $\#_{fast,post}$ columns show the number of computed unifiers for Maude's unification command, the post-filtering technique producing the quotient w.r.t. \sqsupset_{EUB} , the fast unification technique, and the combination of fast and the post-filtering, respectively. The \mathcal{T}_{maude} column measures the time (in milliseconds) required to execute the `variant unify` command for the given input problem, the \mathcal{T}_{post} column measures the time required by the reimplementa- tion of the `variant unify` command together with the post-filtering technique, the \mathcal{T}_{fast} column measures the time required by the reimplementa- tion of the `variant unify` command together with

	Unification problem	#maude	\mathcal{T}_{maude}	#post	\mathcal{T}_{post}	#fast	\mathcal{T}_{fast}	#fast.post	$\mathcal{T}_{fast.post}$
P_1	$V_1 \stackrel{?}{=} V_2 * V_3$	7	0	1	13	1	4	1	4
P_2	$V_1 \stackrel{?}{=} V_2 * V_3 * V_4$	57	49	1	6545	1	1080	1	1168
P_3	$V_1 \stackrel{?}{=} f_1(V_2 * V_3 * f_1(V_4))$	21	3	1	199	1	47	1	47
P_4	$V_1 \stackrel{?}{=} f_2(V_2 * V_3, f_1(V_2 * V_4))$	61	98	1	18895	1	1463	1	1470
P_5	$V_1 \stackrel{?}{=} f_3(V_2 * V_3, f_1(V_3 * V_4), f_2(V_2, f_1(V_4)))$	61	193	1	20949	1	1958	1	1966
P_6	$V_1 * V_2 \stackrel{?}{=} V_3 * V_4$	57	10	1	12240890	2	72	1	10005912
P_7	$V_1 * V_2 \stackrel{?}{=} f_1(V_3 * V_4)$	28	8	1	697	4	17	1	41
P_8	$V_1 * V_2 \stackrel{?}{=} f_1(V_3 * V_3 * f_1(V_4))$	4	0	1	6	4	3	1	5
P_9	$V_1 * V_2 \stackrel{?}{=} f_2(V_3 * V_4, f_1(V_3 * V_5))$	244	741	1	30490862	4	2193	1	14836
P_{10}	$V_1 * V_2 \stackrel{?}{=} f_3(V_3 * V_4, f_1(V_4 * V_5), f_2(V_3, f_1(V_5)))$	244	1277	1	30423527	4	2868	1	14802
P_{11}	$f_1(V_1) \stackrel{?}{=} f_1(V_2 * V_3)$	7	0	1	13	1	4	1	4
P_{12}	$f_1(V_1) * f_1(V_2) \stackrel{?}{=} f_1(V_3) * f_1(V_3 * V_4)$	13	3	2	118	2	8	2	9
P_{13}	$f_1(V_1 * V_2) \stackrel{?}{=} f_1(V_3 * V_4 * V_5)$	973	857	-	T/O	8	15539	-	T/O
P_{14}	$f_2(V_1 * V_2, V_2 * V_3) \stackrel{?}{=} f_2(V_4, V_5)$	61	97	1	32836	1	1471	1	1473
P_{15}	$f_3(V_1 * V_2, V_3 * V_4, V_5 * V_6) \stackrel{?}{=} f_3(V_7, V_8, V_9)$	343	173	1	165260	1	20608	1	20634
P_{16}	$V_1 \stackrel{?}{=} a * b * V_2$	8	0	1	11	1	2	1	2
P_{17}	$V_1 * V_2 \stackrel{?}{=} a * b * V_3$	69	9	1	2259	5	74	1	183
P_{18}	$V_1 * a \stackrel{?}{=} V_2 * b$	8	0	1	11	4	2	1	4
P_{19}	$f_1(a) * f_1(V_1) \stackrel{?}{=} f_1(V_2 * b) * f_1(V_3 * c)$	16	3	3	104	10	13	3	47
P_{20}	$f_2(a, V_1) \stackrel{?}{=} f_2(V_2 * V_3, f_1(a * b))$	4	0	1	9	4	4	1	5

Table 1: Experimental evaluation (exclusive-or)

the fast unification technique, and the $\mathcal{T}_{fast,post}$ column measures the time required of all three combined, the reimplementaion, the fast technique, and the post-filtering.

Table 1 shows that, for the *exclusive-or* theory, the fast post quotient unification command almost replicates the results obtained by using the post quotient unification command, but in a fraction of time, as in unification problems P_9 and P_{10} . For the number of unifiers, Maude reported 973 unifiers for the unification problem P_{13} , and the fast unification technique delivers just 8 unifiers, whereas applying the post-filtering technique to either standard or fast unification is hopeless. For the execution time, the unification problem P_6 reports only 10 milliseconds for \mathcal{T}_{maude} , 72 milliseconds for \mathcal{T}_{fast} , 12240890 milliseconds (3,4 hours) for \mathcal{T}_{post} , and 10005912 milliseconds (2,7 hours) for $\mathcal{T}_{fast,post}$, demonstrating that the post-filtering technique is expensive in any case.

Table 2 shows the experimental results for the *abelian group* theory. Since this theory is far more complex than the *exclusive-or* theory, the execution time and the number of unifiers are bigger than those in Table 1. For the unification problem P_{27} , Maude reported 376 unifiers and the fast unification technique reported just 8 unifiers. The post-filtering technique delivers only one most general unifier, but it takes 22207559 milliseconds (6,2 hours) to compute it from the 376 unifiers and only 27870 milliseconds (less than 28 seconds) to compute it from the 8 unifiers, demonstrating that applying the fast unification technique is advantageous in any case.

7 Conclusion and Future Work

The variant-based equational unification algorithm implemented in the most recent version of Maude, version 2.7.1, may compute many more unifiers than the necessary and, in this paper, we have explored how to strengthen such an algorithm to produce a smaller set of variant unifiers. Our experiments suggest

Unification problem		#maude	\mathcal{T}_{maude}	#post	\mathcal{T}_{post}	#fast	\mathcal{T}_{fast}	#fast.post	$\mathcal{T}_{fast.post}$
P_{21}	$V_1 \stackrel{?}{=} V_2 + V_3$	47	68	1	6185	1	778	1	806
P_{22}	$V_1 \stackrel{?}{=} f_1(V_2 + V_3)$	47	68	1	6117	1	796	1	808
P_{23}	$V_1 \stackrel{?}{=} f_1(V_2 + V_2 + f_1(V_3))$	8	13	1	125	1	43	1	43
P_{24}	$V_1 \stackrel{?}{=} f_2(V_2 + V_3 + f_1(V_3), V_4)$	103	371	1	55662	1	10696	1	10696
P_{25}	$V_1 \stackrel{?}{=} f_3(V_2, f_1(V_3 + V_4), f_2(V_3, V_5))$	6	2	1	30	1	7	1	7
P_{26}	$V_1 + V_2 \stackrel{?}{=} V_3 + V_4$	3611	21663	-	T/O	167	439304	-	T/O
P_{27}	$V_1 + V_2 \stackrel{?}{=} f_1(V_3 + V_4)$	376	13864	1	22207559	8	3830	1	27870
P_{28}	$V_1 + V_2 \stackrel{?}{=} f_1(V_3 + V_3 + f_1(V_4))$	64	1239	1	82170	8	904	1	3382
P_{29}	$V_1 + V_2 \stackrel{?}{=} f_2(V_3 + V_4, f_1(V_5))$	376	13373	1	19468887	8	4059	1	30537
P_{30}	$V_1 + V_2 \stackrel{?}{=} f_3(V_3 + V_3, V_4, V_5)$	32	466	1	4743	8	836	1	1194
P_{31}	$f_1(V_1) \stackrel{?}{=} f_1(V_2 + V_3)$	47	71	1	9985	1	842	1	849
P_{32}	$f_1(V_1) + f_1(V_2) \stackrel{?}{=} f_1(V_3) + f_1(V_3 + V_4)$	93	150	1	699872	1	1417	1	1449
P_{33}	$f_1(V_1 + V_2) \stackrel{?}{=} f_1(V_3 + -V_4)$	3702	25277	-	T/O	109	283851	1	48028877
P_{34}	$f_2(V_1 + V_2, V_2 + V_3) \stackrel{?}{=} f_2(V_4, -V_5)$	188	356	1	154443	1	2384	1	2409
P_{35}	$f_3(V_1 + V_2, f_1(V_3), -V_4) \stackrel{?}{=} f_3(V_5, -V_6, V_6)$	47	1812	1	35992	1	25889	1	29674
P_{36}	$V_1 \stackrel{?}{=} a + -b + V_2$	14	5	1	117	1	20	1	29
P_{37}	$V_1 + V_2 \stackrel{?}{=} a + b + V_3$	510	1411	1	1366009	107	5557	1	288552
P_{38}	$V_1 + a \stackrel{?}{=} V_2 + b$	14	9	1	107	8	8	1	63
P_{39}	$f_1(a) + f_1(V_1) \stackrel{?}{=} f_1(V_2 + -b) + f_1(V_3 + c)$	12	17	2	277	2	150	2	142
P_{40}	$f_2(a, V_1) \stackrel{?}{=} f_2(V_2 + V_3, f_1(a + b))$	8	79	2	831	8	764	1	920

Table 2: Experimental evaluation (abelian group)

that this new adaptation of the variant-based unification is more efficient both in execution time and in the number of computed variant unifiers than the original algorithm.

As far as we know, this is the first work to reduce the number of variant unifiers. The closest work are methods to combine standard unification algorithms with variant-based unification, such as [16, 15]. 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.

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:10.1007/978-3-540-70590-1_1.
- [3] F. Baader & W. Snyder (2001): *Unification Theory*. In J. A. Robinson & A. Voronkov, editors: *Handbook of Automated Reasoning*, I, 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: *Proc. 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] C. Bouchard, K. A. Gero, C. Lynch & P. Narendran (2013): *On Forward Closure and the Finite Variant Property*. In: *Proc. 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.

- [6] 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>.
- [7] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer & C. Talcott (2016): *Maude Manual (Version 2.7.1)*. Technical Report, SRI International Computer Science Laboratory. Available at: <http://maude.cs.uiuc.edu/maude2-manual/>.
- [8] 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.
- [9] H. Comon-Lundh & S. Delaune (2005): *The Finite Variant Property: How to Get Rid of Some Algebraic Properties*. In: *Proc. 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.
- [10] J. Dreier, C. Duménil, S. Kremer & R. Sasse (2017): *Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols*. In: *Proc. of the 6th International Symposium on Principles of Security and Trust (POST 2017)*, *Lecture Notes in Computer Science* 10204, Springer, pp. 117–140, doi:10.1007/978-3-662-54455-6_6.
- [11] J. Dreier, L. Hirschi, S. Radomirovic & R. Sasse (2018): *Automated Unbounded Verification of Stateful Cryptographic Protocols with Exclusive OR*. In: *Proc. of the 31st International Symposium on Computer Security Foundations (CSF 2015)*, IEEE Computer Society Press, pp. 359–373, doi:10.1109/CSF.2018.00033.
- [12] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer & C. Talcott (2018): *Associative Unification and Symbolic Reasoning Modulo Associativity in Maude*. In: *Proc. of the 12th International Workshop on Rewriting Logic and its Applications (WRLA 2018)*, *Lecture Notes in Computer Science* 11152, Springer, pp. 98–114, doi:10.1016/j.scico.2014.02.005.
- [13] F. Durán, S. Lucas & J. Meseguer (2009): *Termination Modulo Combinations of Equational Theories*. In: *Proc. 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.
- [14] F. Durán & J. Meseguer (2012): *On the Church-Rosser and Coherence Properties of Conditional Ordered Rewrite Theories*. *The Journal of Logic and Algebraic Programming* 81(7–8), pp. 816–850, doi:10.1016/j.jlap.2011.12.004.
- [15] A. K. Eeralla, S. Erbatur, A. M. Marshal & C. Ringeissen (2019): *Rule-based Unification in Combined Theories and the Finite Variant Property*. In: *Proc. 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.
- [16] S. Erbatur, D. Kapur, A. M. Marshall, P. Narendran & C. Ringeissen (2015): *Unification and Matching in Hierarchical Combinations of Syntactic Theories*. In: *Proc. 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.
- [17] 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.
- [18] S. Escobar & J. Meseguer (2007): *Symbolic Model Checking of Infinite-State Systems Using Narrowing*. In: *Proc. 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.
- [19] 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.
- [20] (2019): *Full Maude Website*. Available at: <https://github.com/maude-team/full-maude>.
- [21] J. M. Hullot (1980): *Compilation de Formes Canoniques dans les Théories Equationnelles*. Ph.D. thesis, Université de Paris-Sud.

- [22] J. P. Jouannaud, C. Kirchner & H. Kirchner (1983): *Incremental Construction of Unification Algorithms in Equational Theories*. In: *Proc. 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.
- [23] J. P. Jouannaud & H. Kirchner (1986): *Completion of a Set of Rules Modulo a Set of Equations*. *SIAM Journal on Computing* 15(4), pp. 1155–1194, doi:10.1137/0215084.
- [24] D. Kapur & P. Narendran (1987): *Matching, Unification and Complexity*. *ACM SIGSAM Bulletin* 21(4), pp. 6–9, doi:10.1145/36330.36332.
- [25] 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.
- [26] S. Meier, B. Schmidt, C. Cremers & D. A. Basin (2013): *The TAMARIN Prover for the Symbolic Analysis of Security Protocols*. In: *Proc. of the 25th International Conference on Computer Aided Verification (CAV 2013)*, *Lecture Notes in Computer Science* 8044, Springer, pp. 696–701, doi:10.1007/978-3-642-39799-8_48.
- [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: *Proc. 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: *Proc. 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] A. Riesco (2014): *Using Big-Step and Small-Step Semantics in Maude to Perform Declarative Debugging*. In: *Proc. 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.
- [34] V. Rusu (2010): *Combining Theorem Proving and Narrowing for Rewriting-Logic Specifications*. In: *Proc. 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.
- [35] TeReSe (2003): *Term Rewriting Systems*. Cambridge University Press, doi:10.1017/S095679680400526X.
- [36] 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.