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 σ 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 was 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

[*]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.

Burstall [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 $(\mathsf{S}, \leq)$ 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 $[\mathsf{s}]$ has a *top element* under $\leq$, denoted $\top_{[\mathsf{s}]}$ and called the *top sort* of $[\mathsf{s}]$. This involves no real loss of generality, since if $[\mathsf{s}]$ lacks a top sort, it can be easily added. We also assume an S-sorted family $\mathscr{X} = \{\mathscr{X}_{\mathsf{s}}\}_{\mathsf{s} \in \mathsf{S}}$ of disjoint variable sets with each $\mathscr{X}_{\mathsf{s}}$ countably infinite. $\mathscr{T}_{\Sigma}(\mathscr{X})_{\mathsf{s}}$ is the set of terms of sort s, and $\mathscr{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathscr{T}_{\Sigma}(\mathscr{X})$ and $\mathscr{T}_{\Sigma}$ for the corresponding order-sorted term algebras. Given a term t, $\mathit{Var}(t)$ 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 Λ. 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 $U \subseteq \Sigma \cup \mathscr{X}$, $\mathit{Pos}_U(t)$ 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 $\mathit{Pos}(t)$, and the set of non-variable positions $\mathit{Pos}_{\Sigma}(t)$. The subterm of t at position p is $t|_p$ and $t[u]_p$ is the term t where $t|_p$ is replaced by u.

A *substitution* $\sigma \in \mathscr{S}\mathit{ubst}(\Sigma, \mathscr{X})$ is a sorted mapping from a finite subset of $\mathscr{X}$ to $\mathscr{T}_{\Sigma}(\mathscr{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where the domain of σ is $\mathit{Dom}(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $\mathit{Ran}(\sigma)$. The identity substitution is id. Substitutions are homomorphically extended to $\mathscr{T}_{\Sigma}(\mathscr{X})$. 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 $\mathit{Dom}(\sigma) \cap \mathit{Ran}(\sigma) = \emptyset$. 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 $\mathit{Dom}(\sigma) \cap \mathit{Dom}(\sigma') = \emptyset$ is denoted by $\sigma \cup \sigma'$. We call a substitution σ a variable *renaming* if there is another substitution σ^{-1} such that $(\sigma\sigma^{-1})|_{\mathit{Dom}(\sigma)} = \mathit{id}$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathscr{T}_{\Sigma}(\mathscr{X})_{\mathsf{s}}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *equational theory* (Σ, E) is a pair with Σ an order-sorted signature and E a set of Σ-equations. Given Σ and a set E of Σ-equations, order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathscr{T}_{\Sigma}(\mathscr{X})$ (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 $\mathscr{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ 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 $\mathit{Var}(t) = \mathit{Var}(t')$. 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 $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ iff $t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$. An equational theory (Σ, E) is *defined using top sorts* if for each equation $t = t'$ in E, all variables in $Var(t)$ and $Var(t')$ 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 $(\sigma\eta)|_W =_E \rho|_W$. The $\sqsupseteq_E$ relation induces an equivalence relation $\simeq_E$, i.e., $t \simeq_E t'$ iff $t \sqsupseteq_E t'$ and $t \sqsubseteq_E t'$.

An *E-unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_E t'\sigma$. For $Var(t) \cup Var(t') \subseteq W$, 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 $\sigma \in CSU_E^W(t = t')$, $Dom(\sigma) \subseteq (Var(t) \cup Var(t'))$ and $Ran(\sigma) \cap W = \emptyset$. 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 $l \to r$, where $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple (Σ, E, R) with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. The set R of rules is *sort-decreasing* if for each $t \to t'$ in R, each $\mathsf{s} \in \mathsf{S}$, and each substitution σ, $t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ implies $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$. The rewriting relation on $\mathcal{T}_\Sigma(\mathcal{X})$, written $t \to_R t'$ holds between t and t' iff there exist $p \in Pos_\Sigma(t)$, $l \to r \in R$ and a substitution σ, such that $t|_p = l\sigma$, and $t' = t[r\sigma]_p$. The relation $\to_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $=_E; \to_R; =_E$. The transitive (resp. transitive and reflexive) closure of $\to_{R/E}$ is denoted $\to_{R/E}^+$ (resp. $\to_{R/E}^*$).

Reducibility of $\to_{R/E}$ 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 $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \to_{R,E} t'$ iff there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. The narrowing relation $\rightsquigarrow_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \overset{\sigma}{\rightsquigarrow}_{R,E} t'$ iff there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $l \to r$ 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, and equations E are oriented into rules $\overrightarrow{E}$ 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 $t{\downarrow_{E,B}}$.

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 $\theta{\downarrow_{E,B}} =_B \theta$ [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', σ) of t, there is an E,B-variant $(t'', \theta) \in [\![t]\!]_{E,B}$ such that $(t'', \theta) \sqsupseteq_{E,B} (t', \sigma)$, i.e., there is a substitution ρ such that $t' =_E t''\rho$ and $\sigma|_{Var(t)} =_E (\theta\rho)|_{Var(t)}$. A decomposition (Σ, B, E) has the *finite variant property* (FVP) [22] (also called a *finite variant decomposition*) iff for each Σ-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 unification algorithm relying on the concept of the *variants* [11] of a term is also available. A variant of a term t is a pair consisting of a substitution σ and the canonical form of $t\sigma$. Narrowing was proved to be complete for unification in [23], but variant-based unification is decidable when the equational theory satisfies the *finite variant property* [11, 22]. 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',σ) of a term t uses new freshly generated variables.

Definition 1 (Variant Intersection)**.** [22] *Given a decomposition (Σ,B,E) of an equational theory, two Σ-terms t_1 and t_2 such that $W_\cap = Var(t_1)\cap Var(t_2)$ and $W_\cup = Var(t_1)\cup Var(t_2)$, and two sets V_1 and V_2 of variants of t_1 and t_2, respectively, we define $V_1 \cap V_2 = \{(u_1\sigma, \theta_1\sigma \cup \theta_2\sigma \cup \sigma) \mid (u_1,\theta_1) \in V_1 \wedge (u_2,\theta_2) \in V_2 \wedge \exists\sigma : \sigma \in CSU_B^{W_\cup}(u_1 = u_2) \wedge (\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}\}$.*

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

Corollary 2 (Finitary $\mathscr{E}$-unification)**.** [22] *Let (Σ,B,E) be a finite variant decomposition of an equational theory. Given two terms t,t', the set $CSU_{E\cup B}^{\cap}(t = t') = \{\theta \mid (w,\theta) \in [\![t]\!]_{E,B} \cap [\![t']\!]_{E,B}\}$ 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:

```
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`. *The second equation is necessary for coherence modulo AC.*

```
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] * %3:[EXor]           ...         X --> %1:[EXor] * %3:[EXor]
Y --> %2:[EXor] * %4:[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 $X*Y \overset{?}{=} U*V$ 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.

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

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

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

```
  Unifier #4                    Unifier #5                    Unifier #6
  X --> #1:[EXor]                X --> #1:[EXor]                X --> #1:[EXor]
  V --> #1:[EXor] * #2:[EXor]    V --> #2:[EXor]                V --> mt
  U --> #2:[EXor]                U --> #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 $\{X \mapsto U * V\}$. 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 Σ as a disjoint union $\Sigma = \mathscr{D} \uplus \mathscr{C}$ where $\mathscr{D}$ are called *defined* symbols and $\mathscr{C}$ are called *constructor* symbols. In a decomposition (Σ, B, E), the *canonical term algebra* $Can_{\Sigma/(E,B)} = \{t{\downarrow}_{E,B} \mid t \in \mathscr{T}_\Sigma\}$ is typically made of constructor terms, but this more general notion of constructor differs from the "logic" and "functional" notions. A decomposition (Σ, B, E) *protects* a *constructor decomposition* $(\mathscr{C}, B_\mathscr{C}, E_\mathscr{C})$ iff $\mathscr{C} \subseteq \Sigma$, $B_\mathscr{C} \subseteq B$, and $E_\mathscr{C} \subseteq E$, and for all $t, t' \in \mathscr{T}_\mathscr{C}(\mathscr{X})$ we have: (i) $t =_{B_\mathscr{C}} t' \iff t =_B t'$, (ii) $t = t{\downarrow}_{E_\mathscr{C}, B_\mathscr{C}} \iff t = t{\downarrow}_{E,B}$, and (iii) $Can_{\mathscr{C}/(E_\mathscr{C}, B_\mathscr{C})} = Can_{\Sigma/(E,B)}|_\mathscr{C}$. A *constructor decomposition* $(\mathscr{C}, B_\mathscr{C}, \emptyset)$ 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 $*$ for exponents so that $(z^x)^y = (z^y)^x = z^{x*y}$. 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 : Exp ElemSet -> Exp .
  op _*_ : ElemSet ElemSet -> ElemSet [assoc comm ctor] .
  var X : Gen . vars Y Z : ElemSet .
  eq exp(exp(X,Y),Z) = exp(X,Y * 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 $U * V$, the fast unification algorithm of [21] is able to return only one unifier but still generates all the variants of term $U * V$, unnecessarily consuming resources. On the other hand, a unification problem between terms

$f(X*Y)$ and $g(U*V)$ where f and g are different constructor symbols forces the generation of all the variants of the terms $X*Y$ and $U*V$ wasting resources. Let us consider a unification problem $C_1[X] \stackrel{?}{=} C_2[t]$ where both C_1 and C_2 are made of constructor symbols and either there exists σ s.t. $C_1[\square]\sigma =_B C_2[\square]\sigma$ or there is no such σ.

Definition 3 (Constructor-root Position)**.** *Given a decomposition (Σ,B,E) protecting a free constructor decomposition $(\mathscr{C},B_{\mathscr{C}},\emptyset)$ and given a Σ-term t and a position $p \in Pos(t)$, we say p is a* constructor-root *position in t if for all $q < p$, $root(t|_q) \in \mathscr{C}$.*

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

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

Definition 5 (Constructor-root Variant Unifier)**.** *Given a decomposition (Σ,B,E) protecting a free constructor decomposition $(\mathscr{C},B_{\mathscr{C}},\emptyset)$, two Σ-terms t_1 and t_2 s.t. $W_\cap = Var(t_1)\cap Var(t_2)$, $W_\cup = Var(t_1)\cup Var(t_2)$, $(u_1,\theta_1) \in [\![t_1]\!]_{E,B}$, $(u_2,\theta_2) \in [\![t_2]\!]_{E,B}$, and $\sigma \in CSU_B^{W_\cup}(u_1 = u_2)$ s.t. $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$, 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 $\mathtt{X} \stackrel{?}{=} \mathtt{V}*\mathtt{U}$ above, the unifier $\{\mathtt{X} \mapsto \%1*\%2,\ \mathtt{V} \mapsto \%1,\ \mathtt{U} \mapsto \%2\}$ is constructor-root, since `X` is a constructor-root variable in the left unificand and `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 $\{\mathtt{X} \mapsto \mathtt{mt},\ \mathtt{V} \mapsto \%1,\ \mathtt{U} \mapsto \%1\}$ is not constructor-root because `V` and `U` are not constructor-root variables and for the bindings $\mathtt{V} \mapsto \%1$ and $\mathtt{U} \mapsto \%1$ 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 (Σ,B,E) protecting a free constructor decomposition $(\mathscr{C},B_{\mathscr{C}},\emptyset)$, two Σ-terms t_1 and t_2 s.t. $W_\cap = Var(t_1)\cap Var(t_2)$, $W_\cup = Var(t_1)\cup Var(t_2)$, $(u_1,\theta_1) \in [\![t_1]\!]_{E,B}$, $(u_2,\theta_2) \in [\![t_2]\!]_{E,B}$, and a* constructor-root variant unifier *$\sigma \in CSU_B^{W_\cup}(u_1 = u_2)$ s.t. $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$, then $\forall (u_1',\theta_1') \in [\![t_1]\!]_{E,B}$ s.t. $(u_1',\rho) \in [\![u_1]\!]_{E,B}$ and $\theta_1'|_{W_\cup} =_B \theta_1\rho|_{W_\cup}$, if there exists $\sigma' \in CSU_B^{W_\cup}(u_1' = u_2)$ s.t. $(\theta_1'\sigma')|_{W_\cap} =_B (\theta_2\sigma')|_{W_\cap}$, 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 $((\theta_1\cup\theta_2)\sigma)|_{W_\cup} \sqsupseteq_{E\cup B} ((\theta_1'\cup\theta_2)\sigma')|_{W_\cup}$. Similarly for any $(u_2',\theta_2') \in [\![t_2]\!]_{E,B}$.*

Proof. By contradiction. Let us assume $\exists\sigma' \in CSU_B^{W_\cup}(u_1' = u_2)$ s.t. $((\theta_1\cup\theta_2)\sigma)|_{W_\cup} \not\sqsupseteq_{E\cup B} ((\theta_1'\cup\theta_2)\sigma')|_{W_\cup}$. First, $\theta_1'|_{t_1} = \theta_1|_{t_1}\rho|_{u_1}$ and thus the difference is in $\sigma|_{u_2}$ and $\sigma'|_{u_2}$. By the constructor-root property, $\forall x \mapsto t \in \sigma|_{u_2}$ either x is a constructor-root variable in u_2 or $\forall x' \mapsto t' \in (\sigma\setminus\{x \mapsto t\})|_{u_2}$ s.t. $t' =_B C[t]$, x' is a constructor-root variable in u_2. But then $\forall y \in Var(u_2)$, there exist $y \mapsto w_1 \in \sigma|_{u_2}$ and $y \mapsto w_2 \in \sigma'|_{u_2}$ and $(w_2,\rho) \in [\![w_1]\!]_{E,B}$, i.e., $\sigma|_{u_2} \sqsupseteq_{E\cup B} \sigma'|_{u_2}$, which contradicts the assumption. □

We define the case when there is no σ s.t. $C_1[\square]\sigma =_B C_2[\square]\sigma$. 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 (Σ,B,E) protecting a free constructor decomposition $(\mathscr{C},B_{\mathscr{C}},\emptyset)$, two Σ-terms t_1 and t_2 s.t. $W_\cap = Var(t_1)\cap Var(t_2)$, $W_\cup = Var(t_1)\cup Var(t_2)$, $(u_1,\theta_1) \in [\![t_1]\!]_{E,B}$, and $(u_2,\theta_2) \in [\![t_2]\!]_{E,B}$, the pair (u_1,u_2) is a* constructor-root failure pair *if*

$CSU_B^{W_\cup}(u_1 = u_2) = \emptyset$ and there exists two constructor contexts $C_1[\square,\ldots,\square]$ and $C_2[\square,\ldots,\square]$, terms $v_1,\ldots,v_n,w_1,\ldots,w_m$, and fresh distinct variables $x_1,\ldots,x_n,y_1,\ldots,y_m$ s.t. $u_1 =_B C_1[v_1,\ldots,v_n]$, $u_2 =_B C_2[w_1,\ldots,w_m]$, and $CSU_B^{W_\cup}(C_1[x_1,\ldots,x_n] = C_2[y_1,\ldots,y_m]) = \emptyset$.

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

Lemma 8 (Constructor-root Failure Pair)**.** *Given a decomposition (Σ,B,E) protecting a free constructor decomposition $(\mathscr{C},B_{\mathscr{C}},\emptyset)$, two Σ-terms t_1 and t_2 s.t. $W_\cup = Var(t_1)\cup Var(t_2)$, $(u_1,\theta_1) \in [\![t_1]\!]_{E,B}$, $(u_2,\theta_2) \in [\![t_2]\!]_{E,B}$, and (u_1,u_2) is a constructor-root failure term, then $CSU_{E\cup B}^{W_\cup}(u_1 = u_2) = \emptyset$.*

Proof. Immediate by Definition 7. □

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* `ctor` *attribute.*

```
fmod FASTvsCR is sort S .
  ops a b c : -> S [ctor] . op s : S -> S [ctor] .
  op g : S 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) .

  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
```

$$\begin{tikzcd}
 & \mathtt{f(X,Y,Z)} \arrow[dl, squiggly, "\{\mathtt{X}\mapsto\mathtt{a}\}"'] \arrow[dr, squiggly, "\{\mathtt{X}\mapsto\mathtt{b}\}"] & \overset{?}{=} & \mathtt{s(W)} \\
\mathtt{s(Z)} & & \mathtt{g(Y,Z)} \arrow[d, squiggly, "\{\mathtt{Y}\mapsto\mathtt{c}\}"'] & \\
 & & \mathtt{s(Z)} &
\end{tikzcd}$$

Let us assume we have an expression ♣ with a considerably large narrowing tree and two new unification problems (b) $f(X,Y,♣) = s(W)$ and (c) $f(X,♣,Z) = s(W)$. Note that the unifiers of (a) are still valid for (b), whereas only the first unifier of (a) is valid for (c), assuming ♣ never narrows into c. Both the variant-based unification command of Maude and the fast command of [21] cannot avoid the computation of ♣ in both unification problems (b) and (c). However, the technique described below is able to avoid the full computation of ♣ in (b), since the two unifiers are constructor-root, although it cannot avoid the full computation of ♣ 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 (Σ,B,E) protecting a free constructor decomposition $(\mathscr{C},B_{\mathscr{C}},\emptyset)$, two Σ-terms t_1 and t_2 such that $W_\cap = Var(t_1)\cap Var(t_2)$ and $W_\cup = Var(t_1)\cup Var(t_2)$, and two sets V_1 and V_2 of variants of t_1 and t_2, respectively, we say that an intersection*

$V_1 \cap V_2$ is constructor-root *if for each leaf $(u_1,\theta_1) \in V_1$ (resp. $(u_2,\theta_2) \in V_2$), and for each leaf $(u_2,\theta_2) \in V_2$ (resp. $(u_1,\theta_1) \in V_1$) such that $\sigma \in CSU_B^{W_\cup}(u_1 = u_2)$ and $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$, we have $(\theta_1 \cup \theta_2)\sigma$ is* constructor-root.

Definition 10 (Failure Intersection)**.** *Given a decomposition (Σ,B,E) protecting a free constructor decomposition $(\mathscr{C},B_{\mathscr{C}},\emptyset)$ two Σ-terms t_1 and t_2 such that $W_\cap = Var(t_1) \cap Var(t_2)$ and $W_\cup = Var(t_1) \cup Var(t_2)$, and two sets V_1 and V_2 of variants of t_1 and t_2, respectively, we say that an intersection $V_1 \cap V_2$ is a* failure intersection *if for each leaf $(u_1,\theta_1) \in V_1$ (resp. $(u_2,\theta_2) \in V_2$), and for each leaf $(u_2,\theta_2) \in V_2$ (resp. $(u_1,\theta_1) \in V_1$) such that $\sigma \in CSU_B^{W_\cup}(u_1 = u_2)$ and $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$, we have the pair (u_1,u_2) is a constructor-root failure pair.*

The following example shows that the folding variant narrowing trees of both terms t_1,t_2 of a unification problem $t_1 = t_2$ must be unfolded down to a frontier where all leaves of t_1 are tested for unification with all the leaves of t_2.

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

```
variant unify in FG : f(X, Y, Z) =? f(U, V, W)  .

  Unifier #1            Unifier #2            Unifier #3            Unifier #4
  X --> %1:S            X --> a               X --> a               X --> b
  Y --> %2:S            Y --> %2:S            Y --> #2:S            Y --> c
  Z --> %3:S            Z --> %1:S            Z --> #1:S            Z --> #1:S
  U --> %1:S            U --> a               U --> b               U --> a
  V --> %2:S            V --> %3:S            V --> c               V --> #2:S
  W --> %3:S            W --> %1:S            W --> #1:S            W --> #1:S
```

$$\begin{tikzcd}
& \mathtt{f(X,Y,Z)} \arrow[dl, squiggly, "\{\mathtt{X}\mapsto\mathtt{a}\}"'] \arrow[dr, squiggly, "\{\mathtt{X}\mapsto\mathtt{b}\}"] & & \overset{?}{=} & & \mathtt{f(U,V,W)} \arrow[dl, squiggly, "\{\mathtt{U}\mapsto\mathtt{a}\}"'] \arrow[dr, squiggly, "\{\mathtt{X}\mapsto\mathtt{b}\}"] & \\
\mathtt{s(Z)} & & \mathtt{g(Y,Z)} \arrow[d, squiggly, "\{\mathtt{V}\mapsto\mathtt{c}\}"'] & & \mathtt{s(W)} & & \mathtt{g(V,W)} \arrow[d, squiggly, "\{\mathtt{V}\mapsto\mathtt{c}\}"'] \\
& & \mathtt{s(Z)} & & & & \mathtt{s(W)}
\end{tikzcd}$$

The terms at the top position of both narrowing trees clearly unify, but the unifier is not constructor-root, so we must continue expanding both narrowing trees. The condition that all leaves of t_1 are unifiable with all the leaves of t_2 is reached only at depth 2. Indeed, if we expand the left unificand completely but the right unificand only down to the leftmost branch, then the two leaves of the narrowing tree of the left unificand unify with the leftmost leaf of the narrowing tree of the right unificand, but we may miss the last two unifiers reported above if we stop here.

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 $\mathbb{P}([\![t]\!]_{E,B})$ for the powerset of all the subsets of $[\![t]\!]_{E,B}$ such that each $V \in \mathbb{P}([\![t]\!]_{E,B})$ 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 $CSU_{E\cup B}^{\cap,V_1,V_2}(t = t')$ for a version of the unification algorithm of Corollary 2 that uses sets V_1 and V_2 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 $(\mathscr{C},B_{\mathscr{C}},\emptyset)$. Given two terms t,t' and two sets of variants $V_1 \in \mathbb{P}([\![t]\!]_{E,B})$, $V_2 \in \mathbb{P}([\![t']\!]_{E,B})$, the constructor-root variant unifiers are*

$$CSU^{\overline{\cap}}_{E\cup B}(t=t') = \begin{cases} \emptyset & \textit{if } \exists V_1 \in \mathbb{P}([\![t]\!]_{E,B}), V_2 \in \mathbb{P}([\![t']\!]_{E,B}), \\ & \textit{and they are the smallest sets s.t} \\ & V_1 \cap V_2 \textit{ is a failure intersection} \\ CSU^{\cap,V_1,V_2}_{E\cup B}(t=t') & \textit{if } \exists V_1 \in \mathbb{P}([\![t]\!]_{E,B}), V_2 \in \mathbb{P}([\![t']\!]_{E,B}), \\ & \textit{and they are the smallest sets s.t} \\ & V_1 \cap V_2 \textit{ is constructor-root} \\ CSU^{\cap,[\![t]\!]_{E,B},[\![t']\!]_{E,B}}_{E\cup B}(t=t') & \textit{otherwise} \end{cases}$$

Proposition 12 (Constructor-Root Variant-based Unification)**.** *Let* (Σ,B,E) *be a finite variant decomposition of an equational theory protecting a free constructor decomposition* $(\mathscr{C},B_{\mathscr{C}},\emptyset)$*. Given two terms* t,t'*, the set* $CSU^{\overline{\cap}}_{E\cup B}(t=t')$ *is a* finite and complete *set of unifiers for* $t=t'$.

Proof. By contradiction. Let us assume that $CSU^{\overline{\cap}}_{E\cup B}(t=t')$ is not a complete set of unifiers of t and t'. That is, there exists a unifier $\rho' \in CSU^{\cap}_{E\cup B}(t=t')$ and there is no unifier $\rho \in CSU^{\overline{\cap}}_{E\cup B}(t=t')$ s.t. $\rho \sqsupseteq_{E\cup B} \rho'$. By definition, there exist smallest sets $V_1 \in \mathbb{P}([\![t]\!]_{E,B})$, $V_2 \in \mathbb{P}([\![t']\!]_{E,B})$ 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 $\rho' \in CSU^{\cap}_{E\cup B}(t=t')$, we have that there exists $u_1,u_2,\theta_1,\theta_2,\sigma$ s.t. $\rho' = \theta_1\sigma \cup \theta_2\sigma \cup \sigma$, $(u_1,\theta_1) \in [\![t]\!]_{E,B}$, $(u_2,\theta_2) \in [\![t']\!]_{E,B}$, $\sigma \in CSU^{W_\cup}_{B}(u_1=u_2)$, and $(\theta_1\sigma)|_{W_\cap} =_B (\theta_2\sigma)|_{W_\cap}$. 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 τ_3 s.t. $\tau_3 \in CSU^{W_\cup}_{B}(v_1=v_2)$, $(\tau_1\tau_3)|_{W_\cap} =_B (\tau_2\tau_3)|_{W_\cap}$, and $(\tau_1 \cup \tau_2)\tau_3$ is *constructor-root*. Furthermore, the variant (u_1,θ_1) (resp. (u_2,θ_2)) is obtained by further narrowing of v_1 (resp. v_2), i.e., $(u_1,\tau'_1) \in [\![v_1]\!]_{E,B}$ and $(u_2,\tau'_2) \in [\![v_2]\!]_{E,B}$. But then the conclusion follows, since the statement is $((\tau_1 \cup \tau_2)\tau_3)|_{W_\cup} \sqsupseteq_{E\cup B} ((\theta_1 \cup \theta_2)\sigma)|_{W_\cup}$. $\square$

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`.

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 $CSU^{\Cap}_{E\cup B}(t=t')$ of [21]; (iii) the unifiers using the algorithm $CSU^{\overline{\cap}}_{E\cup B}(t=t')$ of Definition 11, and (iv) the unifiers using the algorithm $CSU^{\overline{\Cap}}_{E\cup B}(t=t')$ obtained from $CSU^{\overline{\cap}}_{E\cup B}(t=t')$ by replacing $CSU^{\cap,V_1,V_2}_{E\cup B}(t=t')$ with $CSU^{\Cap,V_1,V_2}_{E\cup B}(t=t')$ from [21]. Note that (ii), (iii), and (iv) are implemented at the metalevel of Maude. We measured both 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 $\#_{maude}$, $\#_{fast}$, $\#_{cr}$ and $\#_{cr+fast}$ columns show the

	Unification problem	$\#_{maude}$	$\#_{fast}$	$\#_{cr}$	$\#_{cr+fast}$	$\mathscr{T}_{maude}$	$\mathscr{T}_{fast}$	$\mathscr{T}_{cr}$	$\mathscr{T}_{cr+fast}$
P_1	$V_1 \stackrel{?}{=} V_2 * V_3 * V_4$	57	1	1	1	50	95	1	1
P_2	$V_1 \stackrel{?}{=} f_3(V_2 * V_3, f_1(V_3 * V_4), f_2(V_2, f_1(V_4)))$	61	1	1	1	172	243	2	2
P_3	$V_1 * V_2 \stackrel{?}{=} V_3 * V_4$	57	8	41	8	9	89	83	131
P_4	$V_1 * V_2 \stackrel{?}{=} f_2(V_3, f_1(V_4 * V_5))$	28	4	4	4	12	18	8	10
P_5	$f_1(a) * f_1(V_1 * V_2) \stackrel{?}{=} f_1(b * V_3) * f_1(c * V_4)$	74	54	74	54	53	112	161	268
P_6	$f_1(V_1) \stackrel{?}{=} f_1(V_2 * V_3 * f_2(V_4, V_5))$	21	1	1	1	4	17	1	1
P_7	$f_2(V_1, V_2 * V_3 * V_4) \stackrel{?}{=} f_2(V_5 * f_1(V_6 * V_7), V_8)$	1596	1	1	1	3473	41592	9	9
P_8	$f_3(V_1, V_2, V_3) \stackrel{?}{=} f_3(f_1(V_4 * V_5), f_1(V_6 * V_7 * V_8), f_1(f_1(V_9)))$	399	1	1	1	507	3289	8	8
P_9	$f_4(V_1, V_2 * V_3, f_1(V_2 * V_4 * V_5), V_3) \stackrel{?}{=}$ $f_4(f_2(V_6, V_7) * V_6, V_8, V_9, f_1(f_1(V_{10})))$	492	14	1	1	122544	61184	14	14
P_{10}	$f_5(V_1, V_2 * V_3 * V_4, f_2(V_5, f_1(V_3 * V_4)), V_4, f_1(V_6 * V_7)) \stackrel{?}{=}$ $f_5(f_2(V_8, V_9), V_{10}, V_{11}, f_1(f_1(V_8)), V_{12})$	161	11	1	1	6780	9249	16	16
P_{11}	$f_1(V_1 * V_2) \stackrel{?}{=} f_2(V_3 * V_4 * V_5, f_2(V_4, V_5))$	0	0	0	0	985	125	1	1
P_{12}	$f_2(V_1, V_2 * V_3 * V_4) \stackrel{?}{=} f_3(V_5 * f_1(V_6 * V_7), V_8, V_9)$	0	0	0	0	2987	57	1	1
P_{13}	$f_3(V_1, V_2, V_3 * V_4) \stackrel{?}{=} f_2(f_1(V_5 * V_6 * V_7), f_1(f_1(V_8)))$	0	0	0	0	468	48	1	1
P_{14}	$f_4(V_1, V_2 * V_3, f_1(V_2 * V_4 * V_5), V_3) \stackrel{?}{=}$ $f_3(f_2(V_6, V_7) * V_6, V_8, f_1(f_1(V_9)))$	0	0	0	0	118028	53653	1	1
P_{15}	$f_5(V_1, V_2 * V_3 * V_4, f_2(V_5, f_1(V_3 * V_4)), V_6, f_1(V_7 * V_8)) \stackrel{?}{=}$ $f_4(f_2(V_9, V_{10}), V_{11}, f_1(f_1(V_9)), V_{12})$	0	0	0	0	6968	7033	1	1

Table 1: Experimental evaluation (exclusive-or)

number of computed unifiers for all four unification algorithms (i), (ii), (iii), (iv) described above and the columns $\mathscr{T}_{maude}$, $\mathscr{T}_{fast}$, $\mathscr{T}_{cr}$ and $\mathscr{T}_{cr+fast}$ show the time (in milliseconds) required to execute the unification command. Note that it is unfair to compare the performance between compiled code ($\mathscr{T}_{maude}$ column) and interpreted code ($\mathscr{T}_{fast}$, $\mathscr{T}_{cr}$ and $\mathscr{T}_{cr+fast}$ 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_9 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

	Unification problem	$\#_{maude}$	$\#_{fast}$	$\#_{cr}$	$\#_{cr+fast}$	$\mathscr{T}_{maude}$	$\mathscr{T}_{fast}$	$\mathscr{T}_{cr}$	$\mathscr{T}_{cr+fast}$
P_{16}	$V_1 \overset{?}{=} V_2+V_3+V_4$	3702	1	1	1	4344602	5034046	1	1
P_{17}	$V_1 \overset{?}{=} f_3(V_2+V_3, f_1(V_3+V_4), f_2(V_2, f_1(V_4)))$	3789	1	1	1	6956340	5413107	2	2
P_{18}	$V_1+V_2 \overset{?}{=} V_3+V_4$	3611	664	3313	664	36258	547115	253078	657746
P_{19}	$V_1+V_2 \overset{?}{=} f_2(V_3, f_1(V_4+V_5))$	376	8	52	8	26425	5083	366	2000
P_{20}	$f_1(a)+f_1(V_1+V_2) \overset{?}{=} f_1(b+V_3)+f_1(c+V_4)$	316	193	316	193	10202	4175	3161	6976
P_{21}	$f_1(V_1) \overset{?}{=} f_1(V_2+V_3+f_2(V_4,V_5))$	158	1	1	1	426	1410	2	2
P_{22}	$f_2(V_1, V_2+V_3+V_4) \overset{?}{=} f_2(V_5+f_1(V_6+V_7), V_8)$	-	-	1	1	T/O	T/O	11	11
P_{23}	$f_3(V_1,V_2,V_3) \overset{?}{=}$ $f_3(f_1(V_4+V_5), f_1(V_6+V_7+V_8), f_1(f_1(V_9)))$	-	-	1	1	T/O	T/O	11	13
P_{24}	$f_4(V_1, V_2+V_3, f_1(V_2+V_4+V_5), V_3) \overset{?}{=}$ $f_4(f_2(V_6,V_7)*V_6, V_8, V_9, f_1(f_1(V_{10})))$	-	-	1	1	T/O	T/O	19	19
P_{25}	$f_5(V_1, V_2+V_3+V_4, f_2(V_5, f_1(V_3+V_4)), V_4, f_1(V_6+$ $V_7)) \overset{?}{=} f_5(f_2(V_8,V_9), V_{10}, V_{11}, f_1(f_1(V_8)), V_{12})$	-	-	1	1	T/O	T/O	24	24
P_{26}	$f_1(V_1+V_2) \overset{?}{=} f_2(V_3+V_4+V_5, f_2(V_4,V_5))$	-	0	0	0	T/O	5594580	1	1
P_{27}	$f_2(V_1, V_2+V_3+V_4) \overset{?}{=} f_3(V_5+f_1(V_6+V_7), V_8, V_9)$	-	0	0	0	T/O	4399334	1	1
P_{28}	$f_3(V_1, V_2, V_3+V_4) \overset{?}{=} f_2(f_1(V_5+V_6+V_7), f_1(f_1(V_8)))$	-	0	0	0	T/O	3757585	1	1
P_{29}	$f_4(V_1, V_2+V_3, f_1(V_2+V_4+V_5), V_3) \overset{?}{=}$ $f_3(f_2(V_6,V_7)*V_6, V_8, f_1(f_1(V_9)))$	-	-	0	0	T/O	T/O	1	1
P_{30}	$f_5(V_1, V_2+V_3+V_4, f_2(V_5, f_1(V_3+V_4)), V_6, f_1(V_7+$ $V_8)) \overset{?}{=} f_4(f_2(V_9,V_{10}), V_{11}, f_1(f_1(V_9)), V_{12})$	-	-	0	0	T/O	T/O	1	1

Table 2: Experimental evaluation (abelian group)

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.

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: *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:10.1007/978-94-009-7893-5_11.

[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`.

[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`.

[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 Stateful 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/jj.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.

[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. 1155–1194, 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.