Unification Modulo Builtins

Ștefan Ciobâcă, Andrei Arusoaie and Dorel Lucanu

Alexandru Ioan Cuza University, Iași
{stefan.ciobaca,arusoaie.andrei,dlucanu}@info.uaic.ro

Abstract. Combining rewriting modulo an equational theory and SMT solving introduces new challenges in the area of term rewriting. One such challenge is unification of terms in the presence of equations and of uninterpreted and interpreted function symbols. The interpreted function symbols are part of a builtin model which can be reasoned about using an SMT solver. In this article, we formalize this problem, that we call unification modulo builtins. We show that under reasonable assumptions, complete sets of unifiers for unification modulo builtins problems can be effectively computed by reduction to usual E-unification problems and by relying on an oracle for SMT solving.

Keywords: unification, constrained term rewriting systems, satisfiability modulo theories

1 Introduction

A recent line of work [6,11,17,20,1,5,16,3,7] aims at combining rewriting modulo E with SMT solving. The goal is to enable the modelling and analysis of systems beyond what is possible by rewriting modulo E or by SMT solving alone. The unifying idea among the approaches above is that rewriting is constrained by some logical formula and that it also involves elements like integers, reals, bitvectors, arrays, etc. that are handled by the SMT solver. As an example, consider the following constrained rewrite system computing the Collatz sequence:

$$\begin{array}{lll}
\mathsf{n} \mapsto N & \Rightarrow \mathsf{cnt} \mapsto 0, \mathsf{n} \mapsto N & \text{if } \top, \\
\mathsf{n} \mapsto 2 \times N + 1, \mathsf{cnt} \mapsto C & \Rightarrow \mathsf{cnt} \mapsto C + 1, \mathsf{n} \mapsto 6 \times N + 4 & \text{if } N > 0, \\
\mathsf{n} \mapsto 2 \times N, \mathsf{cnt} \mapsto C & \Rightarrow \mathsf{cnt} \mapsto C + 1, \mathsf{n} \mapsto N & \text{if } N > 0, \\
\mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto C & \Rightarrow \mathsf{result} \mapsto C & \text{if } \top.
\end{array}$$

The rewrite system above reduces a state of the form $\mathsf{n} \mapsto N$, where N is a natural number, to a final state of the form $\mathsf{result} \mapsto C$, where C is the number of steps taken by N to reach 1 in the Collatz transformation. If the Collatz conjecture is false, then $\mathsf{n} \mapsto N$ does not necessarily terminate.

The symbols appearing in the rewrite system are the following: 1. $\_ \mapsto \_$ (the underscore denotes the place of an argument) is a two-argument free symbol, and n, cnt,result are free constants; 2. $\_,\_$ is an ACI symbol (making, e.g., $\mathsf{cnt} \mapsto 0, \mathsf{n} \mapsto N$ the same as $\mathsf{n} \mapsto N, \mathsf{cnt} \mapsto 0$); 3. N, C are integer variables and the symbols $\times, +, >, 0, 1, 2, \ldots$ have their usual mathematical interpretation.

Unification modulo builtins is motivated by the computation of the possible successors of a term with variables in a constrained rewrite system such as the one above. Assuming that we are in a symbolic state of the form $\mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$, which of the four constrained rewrite rules above could be applied to this state? To answer this question, we should first solve the following equations:

$$\begin{aligned}
\mathsf{n} \mapsto N &= \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3, \\
\mathsf{n} \mapsto 2 \times N + 1, \mathsf{cnt} \mapsto C &= \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3, \\
\mathsf{n} \mapsto 2 \times N, \mathsf{cnt} \mapsto C &= \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3, \\
\mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto C &= \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3.
\end{aligned}$$

Solving such an equation is *E-unification modulo builtins*. Whereas usual (E-)unification is solving an equation of the form $t_1 = t_2$ in the algebra of terms (or in the quotient algebra of terms in the case of E-unification), E-unification modulo builtins is solving an equation of the form $t_1 = t_2$ in an algebra combining three types of symbols: 1. free symbols (such as $\_ \mapsto \_$), 2. symbols satisfying an equational theory E (such as $\_, \_$, satisfying ACI) and 3. builtin symbols such as integers, bitvectors, arrays or others (handled by an SMT solver).

Unlike regular syntactic unification problems (or E-unification problems), where the solution to a unification problem is a unifier (or complete set of unifiers), the solution to an E-unification modulo builtins problem is a logical constraint. We provide an algorithm that reduces E-unification modulo builtins to usual E-unification. As an example, consider the third equation above:

$$\mathsf{n} \mapsto 2 \times N, \mathsf{cnt} \mapsto C = \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3.$$

As the symbol $\_, \_$ is commutative, the equation reduces to solving the builtin constraint $2 \times N = N' + 3 \wedge C = C' + N'$ (for example, by relying on the SMT solver). We show that *any* E-unification modulo builtins problem reduces to a set of logical constraints involving only builtin symbols, plus some substitutions in the range of which there are only non-builtin symbols. Our approach works for *any set of builtins*, not just integers.

Contributions 1. We formalize the problem of *E-unification modulo builtins*, which appears naturally in the context of combining rewriting and SMT solving; 2. We define the notions of *E-unifier modulo builtins*, which generalizes the usual notion of E-unifier, and of *complete set of E-unifiers*; 3. We propose an algorithm for the problem of E-unification modulo builtins, which works by reduction to regular E-unification problems, given an oracle for SMT solving; 4. The algorithm not only decides E-unification modulo builtins, but it can also construct a complete set of E-unifiers modulo builtins; 5. We prove that the algorithm is correct and we also implement the algorithm as a Maude prototype.

Related Work Our algorithm relies on abstractions of terms, various forms of which have been known for a long time and used, for example, in algorithms for

combining decision procedures for theories in [15]. The abstractions of terms used here were defined and used for the first time in [2] for automatically obtaining a symbolic execution framework for a given program language definition. In parallel, abstractions were used in [17] for rewriting modulo SMT, where a builtin equational theory is used instead of the data sub-signature and its model. Both approaches use an SMT solver to check satisfiability of the constraints. In [1], Aguirre et al. introduce narrowing for proving reachability in rewriting logic enriched with SMT capabilities. In [20], Skeirik et al. extend the idea from [14] and show how Reachability Logic (defined in [8]) can be generalized to rewrite theories with SMT solving and how safety properties can be encoded as reachability properties. In [17], Rocha et al. were the first to combine rewriting and SMT solving in order to model and analyze an open system; they solve a particular case of the problem of E-unification modulo builtins that can be reduced to matching. In [5], Bae and Rocha introduce guarded terms, which generalize constrained terms. Logically constrained term rewriting systems (LCTRSs), which combine term rewriting and SMT constraints are introduced in [13,11]. LCTRSs generalize previous formalisms like TRSs enriched with numbers and Presburger constraints (e.g., as in [10]) by allowing arbitrary theories that can be handled by SMT solvers. Early ideas on adding constraints to deduction rules in general and unification in particular date back to the 1990s, with articles such as [12] and [9]. Mixed terms, defined in [9], are similar to our terms that mix free symbols with builtins, except that in [9] builtins are treated as constants. Constrained unification, introduced in [9] as a particular type of constrained deduction, can be seen as a sound and complete (but not necessarily terminating) proof system for a special case of E-unification modulo builtins, where the equational theory E is empty. Additionally, in both [9] and [12], it is assumed that constraints always have a solved form, assumption that is critical in order to present the deduction rules. As we do not make this assumption, and only rely on off-the-shelf SMT solvers, our approach is more general from this point of view. Our work can be seen in the abstract as a combination of E-unification and SMT solving, similar to combinations of unification algorithms for (disjoint) theories (e.g., as in [19,4]).

Organization In Section 2 we formalize constrained terms, which occur naturally in rewriting modulo builtins. Section 3 introduces the concept of E-unification modulo builtins and the notion of (complete set of) E-unifiers. In Section 4 we present an algorithm for solving E-unification modulo builtins by reduction to usual E-unification. Section 5 concludes the paper and provides possible directions for future work.

2 Constrained Terms

We first formalize *builtins*, which represent the parts of the model handled by an SMT solver. This section extends our formalism introduced in [6] by the presence of an an equational theory E and by the introduction of *defined operations*.

Definition 1 (Builtin Signature). *A* builtin signature $\Sigma^{\mathsf{b}} \triangleq (S^{\mathsf{b}}, F^{\mathsf{b}})$ *is any many-sorted signature that includes the following distinguished objects: 1. a sort Bool, together with two constants* $\top$ *and* $\bot$ *of sort Bool; 2. the propositional operation symbols* $\neg : Bool \to Bool$, $\wedge, \vee, \to \; : Bool \times Bool \to Bool$ *and 3. an equality predicate symbol* $= : s \times s \to Bool$ *for each sort* $s \in S^{\mathsf{b}}$.

Example 1. We consider the builtin signature $\Sigma^{INT} = (S^{INT}, F^{INT})$, where $S^{INT} = \{Bool, Int, Arr, Id\}$ and F^{INT} includes, in addition to the required symbols (boolean connectives $\neg, \wedge, \vee$, boolean constants $\top$ and $\bot$ and the equality predicates for *Bool* and *Int*), the following function symbols:

$$
\begin{array}{ll}
\mathsf{cnt}, \mathsf{result}, \mathsf{n}, \ldots : \to Id & + : Int \times Int \to Int; \\
\times : Int \times Int \to Int; & mod : Int \times Int \to Int; \\
\leq : Int \times Int \to Bool; & get : Arr \times Int \to Int; \\
put : Arr \times Int \times Int \to Arr; & 0, 1, 2, \ldots : \to Int.
\end{array}
$$

When working over this builtin signature, we take the liberty to write terms using infix notation for function symbols, so that $x + y$ is a term of sort *Int* and $x + y \leq z$ is a term of sort *Bool* whenever x, y and z are of sort *Int*. We also use infix notation for the boolean operations: $\neg b$, $a \wedge b$, $a \vee b$, $a \to b$.

Definition 2 (Builtin Model). *A* builtin model M^{b} *is a model of a builtin signature* Σ^{b}*, where the interpretation of the distinguished objects of the builtin signature is fixed as follows:* $M^{\mathsf{b}}_{Bool} = \{\top, \bot\}$, $M^{\mathsf{b}}_{\top} = \top$, $M^{\mathsf{b}}_{\bot} = \bot$, $M^{\mathsf{b}}_{=}(a, b) = \top$ *iff* $a = b$, $M^{\mathsf{b}}_{\neg}(\top) = \bot$, $M^{\mathsf{b}}_{\neg}(\bot) = \top$, $M^{\mathsf{b}}_{\wedge}(\top, b) = M^{\mathsf{b}}_{\wedge}(b, \top) = b$, $M^{\mathsf{b}}_{\wedge}(\bot, b) = M^{\mathsf{b}}_{\wedge}(b, \bot) = \bot$*, and so on.*

Example 2. Continuing Example 1, we consider the builtin model M^{INT} that interprets *Bool* as required in Definition 2, the sort *Int* as the set of integers: $M^{INT}_{Int} = \mathbb{Z}$, the sort *Id* as the set of identifiers (strings) and the sort *Arr* as the set of arrays, where both the indices and the values are integers. The builtin model M^{INT} also interprets $+$, $\times$, *mod* and $\leq$ as expected: integer addition, integer multiplication, remainder (defined arbitrarily when the divisor is 0) and respectively the less-than-or-equal relation on integers. The symbol *get* is interpreted as the selection of an array element from a given index, and *put* is interpreted as the update of an array on a given index with a new given element. First-order logical constraints over this model can be solved by an SMT solver implementing the theories of integers, booleans and arrays.

Next, we introduce a formalization of terms extended with builtins.

Definition 3 (Signature Modulo a Builtin Model). *A* signature modulo a builtin model *is a tuple* $\Sigma \triangleq (S, \leq, F, M^{\mathsf{b}})$ *consisting of 1. an order-sorted signature* $(S, \leq, F)$*, and 2. a builtin* Σ^{b}*-model* M^{b}*, where* $\Sigma^{\mathsf{b}} \triangleq (S^{\mathsf{b}}, F^{\mathsf{b}})$ *is a builtin subsignature of* $(S, \leq, F)$*, and 3. the set* $F \setminus F^{\mathsf{b}}$ *is partitioned into two subsignatures:* constructors F^{c} *and* defined operations F^{d} *such that* $F^{\mathsf{c}}_{w,s} = \emptyset$ *for each* $s \in S^{\mathsf{b}}$.

We further assume that the only builtin constant symbols in Σ are the elements of the builtin model, i.e., $F^{\mathsf{b}}_{\varepsilon,s} = M^{\mathsf{b}}_s$. Σ^{b} is called the builtin subsignature *of Σ and $\Sigma^{\mathsf{c}} = (S, \leq, F^{\mathsf{c}} \cup \bigcup_{s \in S^{\mathsf{b}}} F^{\mathsf{b}}_{\varepsilon,s})$ the* constructor subsignature *of Σ.*

Terms over only one of the builtin subsignature and respectively constructor subsignature are *pure*. Terms mixing both constructors and builtins have the builtins as *alien* terms. Due to the restrictions on builtins, we cannot arbitrary nest of builtins and constructors, as is the case when combining arbitrary disjoint signatures (in our case, constructors cannot occur under builtins).

Example 3. We consider the signature $\Sigma = (S, \leq, F, M^{INT})$, where the set of sorts $S = \{Id, Int, Bool, Arr, Val, State\}$ consists of the four builtin sorts *Int*, *Bool*, *Id* and *Arr*, together with the additional sorts *State* and *Val*, where the subsorting relation $\leq = \{Int \leq Val, Arr \leq Val\} \subseteq S \times S$, and where the set of function symbols F includes, in addition to the builtin symbols in F^{INT}, the following function symbols:

$$\mathtt{emp} : \rightarrow State, \qquad \_\mapsto\_ : Id \times Val \rightarrow State,$$
$$\_,\_ : State \times State \rightarrow State, \quad keyOcc : Id \times State \rightarrow Int.$$

The set F^{c} of constructor symbols consists of the symbols defined by the first three declarations above and the set of defined symbols is $F^{\mathsf{d}} = \{keyOcc\}$.

Note that a ground term t defined on the constructor subsignature has the property that any of its builtin subterms is an element of $\Sigma^{\mathsf{b}}_{\varepsilon,s} = M^{\mathsf{b}}_s$. In the rest of this section, $\Sigma = (S, \leq, F, M^{\mathsf{b}})$ denotes a signature modulo a builtin model.

Definition 4 (Model M^{Σ} Generated by a Signature Modulo a Builtin Model). *Let $\Sigma \triangleq (S, \leq, F, M^{\mathsf{b}})$ be a signature modulo a builtin model. The model M^{b} is extended to a $(S, \leq, F)$-model M^{Σ}, defined as follows: 1. $M^{\Sigma}_s = T_{\Sigma^{\mathsf{c}},s}$ for each $s \in S \setminus S^{\mathsf{b}}$, i.e. M^{Σ}_s includes the constructor terms; 2. $M^{\Sigma}_f = M^{\mathsf{b}}_f$ for each $f \in F^{\mathsf{b}}$; 3. M^{Σ}_f is the term constructor $M^{\Sigma}_f(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for each $f \in F^{\mathsf{c}}$; 4. M^{Σ}_f is a function $M^{\Sigma}_f : M^{\Sigma}_{s_1} \times \cdots \times M^{\Sigma}_{s_n} \rightarrow M^{\Sigma}_s$ for each $f \in F^{\mathsf{d}}_{s_1 \ldots s_n, s}$.*

Note that elements of the carrier sets of M^{Σ} are ground terms of the appropriate sort over the signature Σ (as mentioned earlier, builtin elements are constants in Σ). We make the standard assumption that $M_s \neq \emptyset$ for any $s \in S$. Since the defined function symbols can be interpreted in various ways, it follows that M^{Σ} is not uniquely defined, but its carrier sets are uniquely defined by the builtin model and the constructors.

Example 4. Continuing Example 3, we consider the model M^{Σ} obtained by extending M^{INT}. We have that $M^{\Sigma}_{Val} = M^{\Sigma}_{Int} \cup M^{\Sigma}_{Arr}$, M_{State} is the set of finite sets of the form $\{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$ with $n > 0$, $x_i \in M^{\Sigma}_{Id}$ and $v_i \in M^{\Sigma}_{Val}$ for $i = 1, \ldots, n$, $M^{\Sigma}_{\_\mapsto\_}(X, Y)$ is the singleton set $\{X \mapsto Y\}$, and $M^{\Sigma}_{\_,\_}(S_1, S_2)$ is the union of the sets S_1 and S_2.

The defined function M^{Σ}_{keyOcc} is recursively defined as follows:
$M^{\Sigma}_{keyOcc}(X, \mathtt{emp}) = 0, \qquad M^{\Sigma}_{keyOcc}(X, (Y \mapsto V)) = \delta_{X,Y},$
$M^{\Sigma}_{keyOcc}(X, (S_1, S_2)) = M^{\Sigma}_{+}(M^{\Sigma}_{keyOcc}(X, S_1), M^{\Sigma}_{keyOcc}(X, S_2)),$

where δ is the Kronecker delta. We consider a set E of identities such as associativity, commutativity or idempotence for certain function symbols in Σ. We let $\cong$ to be the equivalence relation induced by E on M^Σ. We make the assumption that $\cong$ is a congruence on M^Σ, i.e., that it is compatible with the functions, including the defined ones. We define $M^\Sigma_\cong \triangleq M^\Sigma/\cong$ to be the quotient algebra induced by the congruence $\cong$ on M^Σ.

Example 5. Continuing Example 4, we consider the model M^Σ. Let E consist of the ACI identities for the operation $\_,\_$. We have that the equivalence relation $\cong$ induced by E on M^Σ is a congruence.

Definition 5 (Constraint Formulas). *The set* $\mathrm{CF}(\Sigma, \mathcal{X})$ *of* constraint formulas *over variables* $\mathcal{X}$ *is inductively defined as follows:*

$$\phi ::= b \mid t_1 = t_2 \mid \exists x.\phi' \mid \neg\phi' \mid \phi_1 \wedge \phi_2,$$

where b *ranges over* $T_{\Sigma,Bool}(\mathcal{X})$, t_i *over* $T_{\Sigma,s_i}(\mathcal{X})$.

That is, the constraints are the usual formulas in first-order logic with equality. The only non-standard feature, which does not restrict generality, is that we use terms of sort *Bool* as atomic formulas. The role of predicates is played by functions returning *Bool*. As usual, we may also use the following formulas defined as sugar syntax: 1. $t_1 \neq t_2$ for $\neg(t_1 = t_2)$, 2. $\forall x.\phi$ for $\neg\exists x.\neg\phi$, 3. $\forall X.\phi$ for $\forall x_1.\ldots.\forall x_n.\phi$, where $X = \{x_1, \ldots, x_n\}$, 4. $\phi_1 \vee \phi_2$ for $\neg(\neg\phi_1 \wedge \neg\phi_2)$, 5. $\phi_1 \rightarrow \phi_2$ for $\neg\phi_1 \vee \phi_2$. We denote by $var(\phi)$ the set of variables freely occurring in ϕ.

Example 6. The following formulas are in $\mathrm{CF}(\Sigma, \mathcal{X})$:

$$\begin{aligned}
\phi_1 &\triangleq \forall I.0 \leq I \wedge I < N \rightarrow get(A, I) \geq 0,\\
\phi_2 &\triangleq \forall X.keyOcc(X, S) \leq 1,\\
\phi_3 &\triangleq \exists U.(U > 1 \wedge U < N \wedge mod(N, U) = 0).
\end{aligned}$$

We have $var(\phi_1) = \{N, A\}$, $var(\phi_2) = \{S\}$ and $var(\phi_3) = \{N\}$, assuming that $\{I, U, V, N\} \subseteq \mathcal{X}_{Int}$, $A \in \mathcal{X}_{Arr}$, and $X \in \mathcal{X}_{Id}$.

Definition 6 (Semantics of Constraint Formulas). *The* satisfaction relation $\vDash$ *is inductively defined over the model* $M^\Sigma_\cong$, *valuations* $\alpha : \mathcal{X} \rightarrow M^\Sigma_\cong$, *and formulas* $\phi \in \mathrm{CF}(\Sigma, \mathcal{X})$, *as follows:*

1. $M^\Sigma_\cong, \alpha \vDash b$ *iff* $\alpha(b) = \top$, *where* $b \in T_{\Sigma,Bool}(\mathcal{X})$;
2. $M^\Sigma_\cong, \alpha \vDash t_1 = t_2$ *iff* $\alpha(t_1) = \alpha(t_2)$;
3. $M^\Sigma_\cong, \alpha \vDash \exists x.\phi$ *iff* $\exists a \in M_s$ *(where* $x \in \mathcal{X}_s$*) such that* $M^\Sigma_\cong, \alpha[x \mapsto a] \vDash \phi$;
4. $M^\Sigma_\cong, \alpha \vDash \neg\phi$ *iff* $M^\Sigma_\cong, \alpha \nvDash \phi$;
5. $M^\Sigma_\cong, \alpha \vDash \phi_1 \wedge \phi_2$ *iff* $M^\Sigma_\cong, \alpha \vDash \phi_1$ *and* $M^\Sigma_\cong, \alpha \vDash \phi_2$,

where $\alpha[x \mapsto a]$ *denotes the valuation* α' *defined by* $\alpha'(y) = \alpha(y)$, *for all* $y \neq x$, *and* $\alpha'(x) = a$.

Example 7. Continuing the previous example, the formula ϕ_3 is satisfied by the model $M^{\Sigma}_{\cong}$ defined in Example 5 and any valuation α such that $\alpha(N)$ is a composite number.

Definition 7 (Builtin Constraint Formulas). *The set* $\mathrm{CF}^b(\Sigma, \mathcal{X})$ *of* builtin constraint formulas *over the variables* $\mathcal{X}$ *is the subset of* $\mathrm{CF}(\Sigma, \mathcal{X})$ *defined inductively as follows:*

$$\phi ::= b \mid t_1 = t_2 \mid \exists x.\phi' \mid \neg\phi' \mid \phi_1 \wedge \phi_2,$$

where b *ranges over* $T_{\Sigma^b, Bool}(\mathcal{X})$, t_i *over* $T_{\Sigma,s}(\mathcal{X})$ *such that* s *is a builtin sort, and* x *ranges over all variables of builtin sort.*

Note that in builtin constraint formulas, no symbol that is not builtin is allowed. The constraint formulas ϕ_1, ϕ_3 in the previous example are builtin.

Definition 8 (Constrained Terms). *A* constrained term φ *of sort* $s \in S$ *is a pair* $\langle t \,|\, \phi \rangle$ *with* $t \in T_{\Sigma,s}(\mathcal{X})$ *and* $\phi \in \mathrm{CF}(\Sigma, \mathcal{X})$.

Let $\mathrm{CT}(\Sigma, \mathcal{X})$ denote the set of constrained terms defined over the signature Σ and the variables $\mathcal{X}$.

Example 8. In the context of the previous examples, we have the following constrained terms in $\mathrm{CT}(\Sigma, \mathcal{X})$: 1. arrays with N nonnegative values: $\langle A \,|\, \phi_1 \rangle$ 2. legal states, where an *Id* is bound to at most one value: $\langle S \,|\, \phi_2 \rangle$ 3. states where the *Id* n is bound to a composite integer: $\langle \mathsf{n} \mapsto N, S \,|\, \phi_3 \rangle$.

Definition 9 (Valuation Semantics of Constraints). *The* valuation semantics *of a constraint* ϕ *is the set of valuations* $[\![\phi]\!] \triangleq \{\alpha : \mathcal{X} \to M^{\Sigma}_{\cong} \mid M^{\Sigma}_{\cong}, \alpha \vDash \phi\}$.

Definition 10 (Semantics of Constrained Terms). *The* semantics *of a constrained term* $\langle t \,|\, \phi \rangle$ *is defined by*

$$[\![\langle t \,|\, \phi \rangle]\!] \triangleq \{\alpha(t) \mid \alpha \in [\![\phi]\!]\}.$$

Note that the semantics of constrained terms cannot distinguish between constrained terms with different sets of free variables.

3 *E*-Unification Modulo Builtins

We discuss as an example the four E-unification modulo builtin problems in the Introduction:

1. $\mathsf{n} \mapsto N = \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$
   Even if the $\_,\_$ symbol is ACI, there are no values of N, N', C' that make the lhs equal to the rhs (even if $C' + N' = N' + 3$, the atoms $\mathsf{cnt} \mapsto C' + N'$ and respectively $\mathsf{n} \mapsto N' + 3$ cannot be identified by the idempotence axiom because n and cnt are two different *Id*entifiers). Therefore, the solution to this E-unification problem is $\bot$ (false), i.e., there is no solution.

2. $\mathsf{n} \mapsto 2 \times N + 1, \mathsf{cnt} \mapsto C = \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$
   There are values for N, C, N', C' that make the terms above equal. In particular, any values that satisfy $C' + N' = C \wedge N' + 3 = 2 \times N + 1$ make the terms equal. Therefore, the constraint $C' + N' = C \wedge N' + 3 = 2 \times N + 1$ is the solution of the E-unification problem.
3. $\mathsf{n} \mapsto 2 \times N, \mathsf{cnt} \mapsto C = \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$
   Similar to the case above, the solution is $C' + N' = C \wedge N' + 3 = 2 \times N$.
4. $\mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto C = \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$
   The constraint $N' + 3 = 1 \wedge C' + N' = C$ makes the two terms equal and is the solution to the E-unification problem.

It is helpful to split the solution to a E-unification modulo builtins problem into two parts: a substitution and a logical constraint:

Definition 11 (E-Unifiers Modulo Builtins). *An* E-unifier modulo builtins *(E-umb) of two terms t_1, t_2 is a pair $u = (\sigma, \phi)$, where σ is a substitution and ϕ is a builtin constraint, such that*

$$M_{\cong}^{\Sigma} \vDash \phi \to \sigma(t_1) = \sigma(t_2).$$

Note that we require ϕ to be a *builtin* constraint (see Definition 7), not just a constraint. In particular, ϕ is not allowed to contain any non-builtin symbols. This requirement allows to handle ϕ by using an SMT solver. If ϕ is unsatisfiable, then (σ, ϕ) is vacuously an E-unifier of any two terms.

Definition 12 (Complete Set of E-Unifiers Modulo Builtins). *A set C of pairs of substitutions and builtin logical constraints is called a* complete set of E-unifiers *of t_1 and t_2 if:*

1. *each pair $(\sigma, \phi) \in C$ is an E-umb of t_1 and t_2: $M_{\cong}^{\Sigma} \vDash \phi \to \sigma(t_1) = \sigma(t_2)$;*
2. *for any partial valuation $\alpha : var(t_1, t_2) \to M_{\cong}^{\Sigma}$ such that $\alpha(t_1) = \alpha(t_2)$, there is an E-unifier $(\sigma, \phi) \in C$ and a valuation α^r such that $M_{\cong}^{\Sigma}, \alpha^r \vDash \phi$ and $\alpha = (\alpha^r \circ \sigma)|_{var(t_1,t_2)}$.*

Example 9. Consider the E-unification modulo builtins problem $\mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto C = \mathsf{cnt} \mapsto C', \mathsf{n} \mapsto N'$. A complete set of E-unifiers modulo builtins is the singleton set $\{(id, C = C' \wedge 1 = N')\}$, where id denotes the identity substitution.

Example 10. Consider the E-unification modulo builtins problem

$$(\mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto C) = (I \mapsto C', J \mapsto N').$$

A complete set of E-unifiers modulo builtins is

$$\{(id, I = \mathsf{cnt} \wedge J = \mathsf{n} \wedge C = C' \wedge 1 = N'),$$
$$(id, I = \mathsf{n} \wedge J = \mathsf{cnt} \wedge 1 = C' \wedge C = N')\}.$$

Example 11. Consider the E-unification modulo builtins problem

$$\mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto 42 = M, \mathsf{n} \mapsto 1.$$

A complete set of E-unifiers modulo builtins is the singleton set

$$\{(\sigma, I = \mathsf{cnt} \wedge N = 42\},$$

where $dom(\sigma) = \{M\}$ and $\sigma(M) = I \mapsto N$.

Algorithm 1 Algorithm for E-Unification Modulo Builtins

1: **function** UNIFICATION(t_1, t_2)
2: ▷ returns: a complete set of E-unifiers modulo builtins of t_1 and t_2
3: compute $\langle s_1 \mid \phi^{\sigma_1} \rangle$, an abstraction of t_1
4: compute $\langle s_2 \mid \phi^{\sigma_2} \rangle$, an abstraction of t_2
5: compute $\{\tau_1, \ldots, \tau_n\}$, a complete set of E-unifiers of s_1 and s_2
6: **for** $i \in \{1, \ldots, n\}$ **do**
7: $\tau_i' \leftarrow \tau_i|_{\mathcal{X} \setminus \mathcal{X}^{\mathsf{b}}}$
8: $\phi_i' \leftarrow \phi^{\sigma_1} \wedge \phi^{\sigma_2} \wedge \bigwedge_{x \in dom(\tau_i) \cap \mathcal{X}^{\mathsf{b}}} \tau_i(x) = x$
9: **return** $\{(\tau_1', \phi_1'), \ldots, (\tau_n', \phi_n')\}$

4 An Algorithm for *E*-Unification Modulo Builtins

We propose an algorithm that computes, given two terms t_1 and t_2 without any defined operation symbols, a finite and complete set of E-unifiers modulo builtins of t_1 and t_2, assuming that a finitary E-unification algorithm exists. The algorithm critically relies on the notion of abstraction. In order to formally introduce abstractions, we first require a technical helper definition:

Definition 13 (Substitutions as Formulas). *Each substitution* $\sigma : \mathcal{X} \to T_\Sigma(\mathcal{X})$ *defines a constraint formula* $\phi^\sigma = \bigwedge_{x \in dom(\sigma)} x = \sigma(x)$.

Intuitively, an abstraction of a term t is a pair (s, σ) such that $t = \sigma(s)$, where all subterms of builtin sorts have been "moved" from s into the substitution σ, and where the domain of σ consists of fresh variables. Formally:

Definition 14 (Abstractions). *An* abstraction *of a term* $t \in T_{\Sigma \setminus \Sigma^{\mathsf{d}}}(\mathcal{X})$ *w.r.t.* Y, *where* $var(t) \subseteq Y$, *is a constrained term* $\langle t^\circ \mid \phi^{\sigma^\circ} \rangle$, *where the pair* (t°, σ°) *is inductively defined as follows:* – *if* $t \in T_{\Sigma^{\mathsf{b}}(\mathcal{X})}$ *(i.e.,* t *is a builtin subterm), then* t° *is a fresh variable w.r.t.* Y *and* $\sigma^\circ(t^\circ) = t$; – *if* $t = f(t_1, \ldots, t_n)$ *and* $f \in \Sigma \setminus \Sigma^{\mathsf{b}}$, *then* $t^\circ = f(t_1^\circ, \ldots, t_n^\circ)$ *and* $\sigma^\circ = \sigma_1^\circ \uplus \cdots \uplus \sigma_n^\circ$, *where* $\langle t_i^\circ \mid \phi^{\sigma_i^\circ} \rangle$ *is the abstraction of* t_i *w.r.t.* $Y \cup \bigcup_{j<i} var(\left\langle t_j^\circ \;\middle|\; \phi^{\sigma_j^\circ} \right\rangle)$ *(i.e. each argument* t_i *has its own fresh variables).*

In Definition 14, we assume for simplicity that any occurrence of a builtin term is replaced by a fresh variable. Therefore, two different occurrences of the same builtin term are abstracted by two different variables. However, this is not critical to the soundness of our approach: all results in the paper hold, even if the same abstracting variable is used for several occurrences of the same builtin term.

Algorithm 1 computes a complete set of E-unifiers. Note that, if $x \in \mathcal{X}^{\mathsf{b}}$ is a builtin variable, then $\tau_i(x)$ can only be a builtin variable (by the construction of the abstraction). Due to our assumptions on the builtin sorts, any builtin variable can only be unified with another builtin variable. Therefore builtin variables are treated by the E-unification algorithm used on Line 5 of Algorithm 1 as real variables, and not as free constants. As an optimization of the algorithm, any pair (τ_i', ϕ_i') can be dropped (without losing completeness) if ϕ_i' is unsatisfiable.

Algorithm 1 produces a complete set of E-unifiers modulo builtins. By Definition 11, any satisfiable instance $\alpha(\phi'_i)$ of a constraint ϕ'_i induces the concrete E-umb $\alpha(\tau'_i)$ of t_1 and t_2. Therefore, two terms are E-unifiable modulo builtins iff at least one of the constraints ϕ'_i in the result of Algorithm 1 is satisfiable.

Example 12. Consider the E-unification modulo builtins problem

$$(\mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto 42) = (Z, \mathsf{n} \mapsto 1).$$

Let $t_1 = \mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto 42$ and $t_2 = Z, \mathsf{n} \mapsto 1$. Let (s_1, σ_1) be an abstraction of t_1 defined as follows: $s_1 = I \mapsto N, J \mapsto M$ and $\sigma_1(I) = \mathsf{n}, \sigma_1(N) = 1, \sigma_1(J) = \mathsf{cnt}, \sigma_1(M) = 42$. Let (s_2, σ_2) be an abstraction of t_2 defined as follows: $s_2 = Z, K \mapsto L$ and $\sigma_2(K) = \mathsf{n}$ and $\sigma_2(L) = 1$. A complete set of ACI-unifiers of s_1 and s_2 is the set $\{\tau_1, \tau_2\}$, where:

1. $dom(\tau_1) = \{Z, K, L\}$ and $\tau_1(Z) = I \mapsto N$, $\tau_1(K) = J$, $\tau_1(L) = M$ (the first mapping in s_1 is identified to the first mapping in s_2 and the second mapping in s_1 to the second mapping in s_2);
2. $dom(\tau_2) = \{Z, K, L\}$ and $\tau_2(Z) = J \mapsto M$, $\tau_2(K) = I$, $\tau_2(L) = N$ (the first mapping in s_1 is identified to the second mapping in s_2 and the second mapping in s_1 to the first mapping in s_2);

The case where all four mappings are identified is subsumed by any of the two cases. For this example, the algorithm computes (τ'_i, ϕ'_i) as follows:

1. $dom(\tau'_1) = \{Z\}, \tau'_1(Z) = I \mapsto N$ and ϕ'_1 is
$\underbrace{I = \mathsf{n} \wedge N = 1 \wedge J = \mathsf{cnt} \wedge M = 42}_{\phi^{\sigma_1}} \wedge \underbrace{K = \mathsf{n} \wedge L = 1}_{\phi^{\sigma_2}} \wedge \underbrace{K = J \wedge L = M}_{\text{builtins from } \tau_1}$;
2. $dom(\tau'_2) = \{Z\}, \tau'_2(Z) = J \mapsto M$ and ϕ'_2 is
$\underbrace{I = \mathsf{n} \wedge N = 1 \wedge J = \mathsf{cnt} \wedge M = 42}_{\phi^{\sigma_1}} \wedge \underbrace{K = \mathsf{n} \wedge L = 1}_{\phi^{\sigma_2}} \wedge \underbrace{K = I \wedge L = N}_{\text{builtins from } \tau_2}$;

The algorithm returns the complete set $\{(\tau'_1, \phi'_1), (\tau'_2, \phi'_2)\}$ of E-unifiers modulo builtins of the terms $t_1 = \mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto 42$ and $t_2 = Z, \mathsf{n} \mapsto 1$. As ϕ'_1 is unsatisfiable, the first unifier can be pruned away and therefore $\{(\tau'_2, \phi'_2)\}$ is also a complete set of unifiers of $t_1 = \mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto 42$ and $t_2 = Z, \mathsf{n} \mapsto 1$.

The next result shows that Algorithm 1 is correct:

Theorem 1. *The set $\{(\tau'_1, \phi'_1), \ldots, (\tau'_n, \phi'_n)\}$ computed by Algorithm 1 is a complete set of E-unifiers modulo builtins of t_1 and t_2.*

Proof (Sketch). Let t_1 and t_2 be two terms without defined function symbols. Let $\langle s_1 \,|\, \phi^{\sigma_1} \rangle$ be an abstraction of t_1. Let $\langle s_2 \,|\, \phi^{\sigma_2} \rangle$ be an abstraction of t_2. Let $\{\tau_1, \ldots, \tau_n\}$ be a complete set of E-unifiers of s_1 and s_2. Let $\tau'_i = \tau_i|_{\mathcal{X} \setminus X^{\mathsf{b}}}$. Let $\phi'_i = \phi^{\sigma_1} \wedge \phi^{\sigma_2} \wedge \bigwedge_{x \in dom(\tau_i) \cap \mathcal{X}^{\mathsf{b}}} \tau_i(x) = x$.

We show that the set $\{(\tau'_1, \phi'_1), \ldots, (\tau'_n, \phi'_n)\}$ is a complete set of unifiers modulo builtins of t_1 and t_2. Firstly, we have to show soundness: that each pair (τ'_1, ϕ'_1) is indeed a unifier of t_1 and t_2 (easy).

Secondly, we show completeness. Let $\alpha : var(t_1, t_2) \to M^{\Sigma}_{\cong}$ be a partial valuation such that $\alpha(t_1) = \alpha(t_2)$. We will work with the analogous approach, with $\alpha : var(t_1, t_2) \to M^{\Sigma}$ such that $\alpha(t_1) \cong \alpha(t_2)$. Note that, by our definition of the model generated by a signature modulo builtins, $M^{\Sigma} = \mathcal{T}_{\Sigma^{\mathsf{c}}}(\emptyset)$, and therefore valuations such as α can also be seen as substitutions.

Let $\alpha' : var(s_1, s_2) \to \mathcal{T}_{\Sigma^{\mathsf{c}}}(\emptyset)$ be defined as follows:

$$\alpha'(x) = \begin{cases} \alpha(x) & \text{if } x \in var(t_1, t_2) \setminus (dom(\sigma_1) \cup dom(\sigma_2)); \\ \alpha(\sigma_1(x)) & \text{if } x \in dom(\sigma_1); \\ \alpha(\sigma_2(x)) & \text{if } x \in dom(\sigma_2). \end{cases}$$

We have $\alpha'(s_1) \cong \alpha'(s_2)$. By construction, we also have: 1. $\alpha'|_{var(s_1)} = \alpha \circ \sigma_1$; 2. $\alpha'|_{var(s_2)} = \alpha \circ \sigma_2$. As α' is a substitution unifying s_1 and s_2 modulo E (recall that $\cong$ is generated by E), it follows that there exists a unifier $\tau_i \in \{\tau_1, \ldots, \tau_n\}$ of s_1 and s_2 and a substitution α^c such that: 1. $\tau_i(s_1) \cong \tau_i(s_2)$; 2. $\alpha' = (\alpha^c \circ \tau_i)|_{var(t_1,t_2)}$. Note that, as $\alpha'|_{var(s_j)} = \alpha \circ \sigma_j$ $(1 \leq j \leq 2)$, we also have that $(\alpha^c \circ \tau_i)|_{var(s_1)} = \alpha \circ \sigma_1$ and $(\alpha^c \circ \tau_i)|_{var(s_2)} = \alpha \circ \sigma_2$.

Let $\tau'_i = \tau_i|_{\mathcal{X} \setminus X^{\mathsf{b}}}$. Let $\alpha^r : var(t_1, t_2, \phi) \to \mathcal{T}_{\Sigma^{\mathsf{c}}}(\emptyset)$ be defined as follows:

$$\alpha^r(x) = \begin{cases} \alpha'(x) \text{ if } x \in var(s_1, s_2) \\ \alpha(x) \text{ if } x \in var(t_1, t_2) \setminus var(s_1, s_2) \\ \alpha^c(y) \text{ otherwise, where } y \text{ is such that } \tau_i(x) = y \end{cases}$$

The substitution α^r defined above satisfies all conditions in Definition 12:

1. $\alpha = (\alpha^r \circ \tau'_i)|_{var(t_1,t_2)}$:
   Let $x \in var(t_1, t_2)$. We distinguish two cases: – if $x \in X^{\mathsf{b}}$, then we have $\alpha^r(\tau'_i(x)) = \alpha^r(x) = \alpha(x)$; – if $x \notin X^{\mathsf{b}}$, then $\alpha^r(\tau'_i(x)) = \alpha^r(\tau_i(x)) = \alpha^c(\tau_i(x)) = \alpha'(x) = \alpha(x)$;
2. $M^{\Sigma}_{\cong}, \alpha^r \vDash \phi'_i$:
   (a) $M^{\Sigma}_{\cong}, \alpha^r \vDash \phi^{\sigma_1}$, since: – if $x \notin dom(\sigma_1)$, then $\sigma_1(x) = x$ and therefore trivially $\alpha^r(\sigma_1(x)) = \alpha^r(x)$; – if $x \in dom(\sigma_1)$, then $\alpha^r(\sigma_1(x)) = \alpha(\sigma_1(x)) = \alpha'(x) = \alpha^r(x)$;
   (b) $M^{\Sigma}_{\cong}, \alpha^r \vDash \phi^{\sigma_2}$, analogously;
   (c) $M^{\Sigma}_{\cong}, \alpha^r \vDash \tau_i(x) = x$ for all $x \in dom(\tau_i) \cap X^{\mathsf{b}}$ trivially, by the third branch in the definition of α^r.

We have shown that for any α that unifies modulo builtins t_1 and t_2, there is an E-umb (τ'_i, ϕ'_i) and a substitution α^r with the properties required in Definition 12, and therefore this proves the completeness of the algorithm.

Complexity. Algorithm 1, which reduces E-unification modulo builtins to E-unification, has a linear-time overhead and therefore the running time is dominated by the algorithm for E-unification and, optionally, by some calls to the SMT solver. Lines 3 and 4 are linear in the size of the input. The running time of Line 5 is determined by the E-unification algorithm. The postprocessing step in lines $6 - 8$ is linear in the size n of the *output* of the E-unification algorithm.

Additionally, if we want to check the satisfiability of the constraints ϕ'_i on Line 9, there will be n calls to the SMT solver. In summary, the running time of the algorithm is dominated by one call to the E-unification algorithm, and, optionally, n calls to the SMT solver, where n is the number of unifiers returned by the E-unification algorithm.

5 Conclusion and Future Work

We introduced the problem of *E-unification modulo builtins*. While regular (E)-unification is about solving an equation in the algebra of terms (or the algebra of terms modulo E), E-unification modulo builtins is solving an equation in an algebra combining terms (modulo E) with builtin elements such as booleans, integers, arrays, or any other elements that can be handled by an SMT solver.

Our main contribution is to formalize the problem of E-unification modulo builtins and to provide an algorithm for it that is based on the notion of abstraction of a term. The algorithm reduces E-unification modulo builtins to regular E-unification. This allows to lift all existing E-unification algorithms to E-unification modulo builtins.

Unlike regular E-unification algorithms, which produce (sets of) substitution(s) as their output, algorithms for E-unification modulo builtins produce sets of pairs of substitutions and logical constraints. The equation being solved holds when instantiated by one of the substitutions, in the cases where the attached logical constraint holds. We show how to produce logical constraints that only contain builtins, which means that they can be fully handled by an SMT solver. We implement our approach as a Maude prototype[1] at the meta-level. Given two terms, the prototype computes a complete set of E-unifiers modulo builtins as described in Algorithm 1. As an optimization, it filters out the E-unifiers modulo builtins that have an unsatisfiable constraint, by using the integrated SMT solver. We describe the prototype in Appendix A. Our result answers in part an open question in [18], namely of finding elements that match two matching logic patterns, which is essential in developing a matching-logic prover.

The main question that needs to be answered in future work is what kind of new applications are enabled by the combination of rewrite and SMT solving. On the theoretical side, all known results in rewriting (confluence, etc.) also need to be developed in the new framework. Another open question is how to perform E-unification modulo builtins in the presence of defined operations.

Acknowledgments. We thank the anonymous reviewers for their valuable suggestions. This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI - UEFISICDI, project number PN-III-P2-2.1-BG-2016-0394, within PNCDI III.

[1] The prototype is available at:
`https://github.com/andreiarusoaie/unification-modulo-builtins`

References

1. Luis Aguirre, Narciso Martí-Oliet, Miguel Palomino, and Isabel Pita. Conditional Narrowing Modulo SMT and Axioms. In *PPDP*, pages 17–28, 2017.
2. Andrei Arusoaie, Dorel Lucanu, and Vlad Rusu. A Generic Framework for Symbolic Execution. In *SLE*, volume 8225 of *LNCS*, pages 281–301, 2013.
3. Andrei Arusoaie, Dorel Lucanu, and Vlad Rusu. Symbolic execution based on language transformation. *Comp. Lang., Systems & Structures*, 44:48–71, 2015.
4. Franz Baader and Klaus U Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *JSC*, 21(2):211–243, 1996.
5. Kyungmin Bae and Camilo Rocha. Guarded Terms for Rewriting Modulo SMT. In *FACS*, pages 78–97, 2017.
6. Ştefan Ciobâcă and Dorel Lucanu. A Coinductive Approach to Proving Reachability Properties in Logically Constrained Term Rewriting Systems. In *IJCAR 2018*. (to appear).
7. Ştefan Ciobâcă and Dorel Lucanu. RMT: Proving Reachability Properties in Constrained Term Rewriting Systems Modulo Theories. Technical Report TR 16-01, Alexandru Ioan Cuza University, Faculty of Computer Science, 2016.
8. Andrei Ştefănescu, Ştefan Ciobâcă, Radu Mereuţă, Brandon M. Moore, Traian Florin Şerbănuţă, and Grigore Roşu. All-Path Reachability Logic. In *RTA-TLCA*, volume 8560 of *LNCS*, pages 425–440, 2014.
9. John Darlington and Yike Guo. Constrained equational deduction. In *Conditional and Typed Rewriting Systems*, pages 424–435, 1991.
10. Stephan Falke and Deepak Kapur. Dependency Pairs for Rewriting with Built-In Numbers and Semantic Data Structures. In *RTA*, pages 94–109, 2008.
11. Fuhs, Carsten and Kop, Cynthia and Nishida, Naoki. Verifying Procedural Programs via Constrained Rewriting Induction. *ACM TOCL*, 18(2):14:1–14:50, 2017.
12. Claude Kirchner, Hélène Kirchner, and Michaël Rusinowitch. Deduction with symbolic constraints. Research Report RR-1358, INRIA, 1990. Projet EURECA.
13. Cynthia Kop and Naoki Nishida. Term Rewriting with Logical Constraints. In *FroCoS*, pages 343–358, 2013.
14. Dorel Lucanu, Vlad Rusu, Andrei Arusoaie, and David Nowak. Verifying reachability-logic properties on rewriting-logic specifications. In *Logic, Rewriting, and Concurrency*, volume 9200 of *LNCS*, pages 451–474, 2015.
15. Greg Nelson and Derek C Oppen. Simplification by cooperating decision procedures. *ACM TOPLAS*, 1(2):245–257, 1979.
16. Vivek Nigam, Carolyn Talcott, and Abraão Aires Urquiza. Towards the automated verification of cyber-physical security protocols: Bounding the number of timed intruders. In *ESORICS*, pages 450–470, 2016.
17. Camilo Rocha, José Meseguer, and César Muñoz. Rewriting modulo SMT and open system analysis. *JLAMP*, 86(1):269–297, 2017.
18. Grigore Roşu. Matching logic. *LMCS*, 13(4):1–61, 2017.
19. Manfred Schmidt-Schauss. Unification in a combination of arbitrary disjoint equational theories. *JSC*, 8(1):51 – 99, 1989.
20. Stephen Skeirik, Andrei Ştefănescu, and José Meseguer. A constructor-based reachability logic for rewrite theories. *CoRR*, abs/1709.05045, 2017.

A Maude Prototype

Since the algorithm needs to manipulate terms (for instance, to compute the abstractions) we use the metalevel capabilities of Maude to implement the following functionalities:

- For a given meta-representation of a term, `getAbstraction` returns the abstraction of a term w.r.t. the set of builtins sorts, which need to be provided explicitly. When computing abstractions, fresh variables are generated;
- The E-unifiers of the abstractions are computed by `unifyAbstractions`;
- The unsatisfiable formulas are filtered out by `filterUnsatUMBs`;
- The E-unification modulo builtins algorithm that we propose is implemented by `completeSetOfUMBs`, which gets as input the meta-representations of two terms and returns the unifiers modulo builtins in two steps: it first generates the substitution-formula pairs with `completeSetOfUMBsUnfiltered`, and then it eliminates the unsatisfiable solutions using the aforementioned `filterUnsatUMBs`.

We show how to use our prototype to find the E-unifiers modulo builtins of the E-unification modulo builtins problem $\mathtt{n} \mapsto 1, \mathtt{cnt} \mapsto 42 = Z, \mathtt{n} \mapsto 1$, introduced in Example 12. First, Maude finds a complete set of ACI-unifiers for the abstractions of the two terms. Then, from these unifiers we generate the pairs $\{(\tau_1', \phi_1'), (\tau_2', \phi_2')\}$, as shown in Example 12):

```
reduce in UNIFICATION-MODULO-BUILTINS :
completeSetOfUMBsUnfiltered(upTerm(n |-> 1,count |-> 42),
                            upTerm(Z,n |-> 1), 'STATE) .
rewrites: 2023 in 5ms cpu (6ms real) (338464 rewrites/second)
result UnificationResults:
[
'Z --> %1 |-> %2 |
abs0 #== n /\ (abs1 #== 1 /\ (abs2 #== count /\ abs3 #== 42)) /\
(abs4 #== n /\ abs5 #== 1) /\ (abs0 #== %1 /\ (abs1 #== %2 /\
  (abs2 #== %3 /\ (abs3 #== %4 /\ (abs4 #== %3 /\ abs5 #== %4)))))
],,
[
'Z --> \%1 |-> \%2 |
abs0 #== n /\ (abs1 #== 1 /\ (abs2 #== count /\ abs3 #== 42)) /\
(abs4 #== n /\ abs5 #== 1) /\ (abs0 #== %3 /\ (abs1 #== %4 /\
  (abs2 #== %1 /\ (abs3 #== %2 /\ (abs4 #== %3 /\ abs5 #== %4)))))
]
```

The variables `abs0`, `abs1`, ..., are generated during the abstraction process, while `%1`, `%2`, ... are generated by the Maude's variant unifier.

Finally, `completeSetOfUMBs` – the main function in our prototype – filters out the first unifier, which has an unsatisfiable constraint. Because the interaction between Maude and the SMT solver only supports integers and booleans, we

have encoded identifiers (of sort Id) into integers before sending the formula to the SMT solver. The solution is:

```
reduce in UNIFICATION-MODULO-BUILTINS :
completeSetOfUMBs(upTerm(n |-> 1,count |-> 42),
                  upTerm(Z,n |-> 1), 'STATE) .
rewrites: 3883 in 17ms cpu (18ms real) (227355 rewrites/second)
result UnificationResults:
[
'Z --> %1 |-> %2 |
abs0 #== n /\ (abs1 #== 1 /\ (abs2 #== count /\ abs3 #== 42)) /\
(abs4 #== n /\ abs5 #== 1.Integer) /\ (abs0 #== %3 /\ (abs1 #== %4
/\ (abs2 #== %1 /\ (abs3 #== %2 /\ (abs4 #== %3 /\ abs5 #== %4)))))
]
```

The result is essentially that the variable Z must be $\mathsf{cnt} \mapsto 42$, since $\mathtt{\%1} = \mathtt{abs2{:}Id} = \mathtt{count}$ and $\mathtt{\%2} = \mathtt{abs3{:}Id} = 42$.

We now show how our prototype solves the four E-unification modulo builtins problems discussed in the Introduction:

1. $\mathsf{n} \mapsto N = \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$
2. $\mathsf{n} \mapsto 2 \times N + 1, \mathsf{cnt} \mapsto C = \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$
3. $\mathsf{n} \mapsto 2 \times N, \mathsf{cnt} \mapsto C = \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$
4. $\mathsf{n} \mapsto 1, \mathsf{cnt} \mapsto C = \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$

The set of unifiers modulo builtins is computed by our Maude prototype for each case as shown below:

```
reduce in UNIFICATION-MODULO-BUILTINS :
completeSetOfUMBs(upTerm(n |-> N),
              upTerm(n |-> N' #+ 3,count |-> N' #+ C'), 'STATE) .
rewrites: 571 in 3ms cpu (3ms real) (148119 rewrites/second)
result UnificationResults: noUMBResults
==========================================
reduce in UNIFICATION-MODULO-BUILTINS :
completeSetOfUMBs(upTerm(n |-> 2 #* N #+ 1,count |-> C),
              upTerm(n |-> N' #+ 3,count |-> N' #+ C'), 'STATE) .
rewrites: 4574 in 15ms cpu (15ms real) (302413 rewrites/second)
result UnificationResults:
[
identity |
abs0 #== n /\ (abs1 #== 2 #* N #+ 1 /\ abs2 #== count) /\
(abs3 #== n /\ (abs4 #== N' #+ 3 /\ (abs5 #== count /\
abs6 #== N' #+ C'))) /\ (C #== %4 /\ (abs0 #== %1 /\
(abs1 #== %2 /\ (abs2 #== %3 /\ (abs3 #== %1 /\
(abs4 #== %2 /\ (abs5 #== %3 /\ abs6 #== %4)))))))
]
```

```
==========================================
reduce in UNIFICATION-MODULO-BUILTINS :
completeSetOfUMBs(upTerm(n |-> 2 #* N, count |-> C),
              upTerm(n |-> N' #+ 3,count |-> N' #+ C'), 'STATE) .
rewrites: 4528 in 15ms cpu (16ms real) (298385 rewrites/second)
result UnificationResults:
[
identity |
abs0 #== n /\ (abs1 #== 2 #* N /\ abs2 #== count) /\
(abs3 #== n /\ (abs4 #== N' #+ 3 /\ (abs5 #== count /\
abs6 #== N' #+ C'))) /\ (C #== %4 /\ (abs0 #== %1 /\
(abs1 #== %2 /\ (abs2 #== %3 /\ (abs3 #== %1 /\
(abs4 #== %2 /\ (abs5 #== %3 /\ abs6 #== %4)))))))
]
==========================================
reduce in UNIFICATION-MODULO-BUILTINS :
completeSetOfUMBs(upTerm(n |-> 1,count |-> C),
              upTerm(n |-> N' #+ 3,count |-> N' #+ C'), 'STATE) .
rewrites: 4625 in 15ms cpu (16ms real) (297141 rewrites/second)
result UnificationResults:
[
identity |
abs0 #== n /\ (abs1 #== 1 /\ abs2 #== count) /\
(abs3 #== n /\ (abs4 #== N' #+ 3 /\ (abs5 #== count /\
abs6 #== N' #+ C'))) /\ (C #== %4 /\ (abs0 #== %1 /\
(abs1 #== %2 /\ (abs2 #== %3 /\ (abs3 #==%1 /\
(abs4 #== %2 /\ (abs5 #== %3 /\ abs6 #== %4)))))))
]
```

For the first E-umb problem, the tool returns `noUMBResults`, which means that it does not have any solution. For the other examples, the prototype finds the unifiers, as expected.