Congruence Closure with Free Variables

Haniel Barbosa, Pascal Fontaine, Andrew Reynolds

Congruence Closure with Free Variables

Haniel Barbosa[1,2], Pascal Fontaine[1⋆] and Andrew Reynolds[3]

[1] LORIA–INRIA, Université de Lorraine, Nancy, France
[2] Universidade Federal do Rio Grande do Norte, Natal, RN, Brazil
[3] University of Iowa, Iowa City, USA
{Haniel.Barbosa,Pascal.Fontaine}@inria.fr, andrew.j.reynolds@gmail.com

Abstract. Many verification techniques nowadays successfully rely on SMT solvers as back-ends to automatically discharge proof obligations. These solvers generally rely on various instantiation techniques to handle quantifiers. We here show that the major instantiation techniques in SMT solving can be cast in a unifying framework for handling quantified formulas with equality and uninterpreted functions. This framework is based on the problem of E-ground (dis)unification, a variation of the classic rigid E-unification problem. We introduce a sound and complete calculus to solve this problem in practice: Congruence Closure with Free Variables (CCFV). Experimental evaluations of implementations of CCFV in the state-of-the-art solver CVC4 and in the solver veriT exhibit improvements in the former and makes the latter competitive with state-of-the-art solvers in several benchmark libraries stemming from verification efforts.

1 Introduction

SMT solvers [7] are highly efficient at handling large ground formulas with interpreted symbols, but they still struggle with quantified formulas. Pure quantified first-order logic is best handled with *resolution* and *superposition*-based theorem proving [3]. Although there are first attempts to unify such techniques with SMT [12], the main approach used in SMT is still *instantiation*: quantified formulas are reduced to ground ones and refuted with the help of decision procedures for ground formulas. The main instantiation techniques are E-matching based on triggers [11,16,25], finding conflicting instances [23] and model-based quantifier instantiation (MBQI) [18,24]. Each of these techniques contributes to the efficiency of state-of-the-art solvers, yet each one is typically implemented independently.

We introduce the E-ground (dis)unification problem as the cornerstone of a unique framework in which all these techniques can be cast. This problem relates to the classic problem of rigid E-unification and is also NP-complete. Solving

⋆ This work has been partially supported by the ANR/DFG project STU 483/2-1 SMArT ANR-13-IS02-0001 of the Agence Nationale de la Recherche, by the H2020-FETOPEN-2016-2017-CSA project SC^2 (712689), and by the European Research Council (ERC) starting grant Matryoshka (713999).

E-ground (dis)unification amounts to finding substitutions such that literals containing free variables hold in the context of currently asserted ground literals. Since the instantiation domain of those variables can be bound, a possible way of solving the problem is by first non-deterministically guessing a substitution and checking if it is a solution. The *Congruence Closure with Free Variables* algorithm (CCFV, for short) presented here is a practical decision procedure for this problem based on the classic congruence closure algorithm [20,21]. It is goal-oriented: solutions are constructed incrementally, taking into account the congruence closure of the terms defined by the equalities in the context and the possible assignments to the variables.

We then show how to build on CCFV to implement trigger-based, conflict-based and model-based instantiation. An experimental evaluation of the technique is presented, where our implementations exhibits improvements over state-of-the-art approaches.

1.1 Related work

Instantiation techniques for SMT have been studied extensively. Heuristic instantiation based on E-matching of selected triggers was introduced by Detlefs et al. [16]. A highly efficient implementation of E-matching was presented by de Moura and Bjørner [11]; it relies on elaborated indexing techniques and generation of machine code for optimizing performance. Rümmer uses triggers alongside a classic tableaux method [25]. Trigger based instantiation unfortunately produces many irrelevant instances. To tackle this issue, a goal-oriented instantiation technique producing only useful instances was introduced by Reynolds et al. [23]. CCFV shares resemblance with this algorithm, the search being based on the structure of terms and a current model coming from the ground solver. The approach here is however more powerful and more general, and somehow subsumes this previous technique. Ge and de Moura's model based quantifier instantiation (MBQI) [18] provides a complete method for first-order logic through successive derivation of conflicting instances to refine a candidate model for the whole formula, including quantifiers. Thus it also allows the solver to find finite models when they exist. Model checking is performed with a separate copy of the ground SMT solver searching for a conflicting instance. Alternative methods for model construction and checking were presented by Reynolds et al. [24]. Both these model based approaches [18,24] allow integration of theories beyond equality, while CCFV for now only handles equality and uninterpreted functions.

Backeman and Rümmer solve the related problem of rigid E-unification through encoding into SAT, using an off-the-shelf SAT solver to compute solutions [5]. Our work is more in line with goal-oriented techniques as those by Goubault [19] and Tiwari et al. [26]; congruence closure algorithms being very efficient at checking solutions, we believe they can also be the core of efficient algorithms to discover them. CCFV differs from those previous techniques notably, since it handles disequalities and since the search for solutions is pruned based on the structure of a ground model and is thus most suitable for an SMT context.

2 Notations and basic definitions

We refer to classic notions of many-sorted first-order logic (e.g. by Baader and Nipkow [1] and by Fitting [17]) as the basis for notations in this paper. Only the most relevant are mentioned.

A *first-order language* is a tuple $\mathscr{L} = \langle \mathcal{S}, \mathcal{X}, \mathcal{P}, \mathcal{F}, sort \rangle$ in which $\mathcal{S}$, $\mathcal{X}$, $\mathcal{P}$ and $\mathcal{F}$ are disjoint enumerable sets of *sort*, *variable*, *predicate* and *function symbols*, respectively, and $sort : \mathcal{X} \cup \mathcal{F} \cup \mathcal{P} \rightarrow \mathcal{S}^+$ is a function assigning sorts, according to the symbols' arities. Nullary functions and predicates are called *constants* and *propositions*, respectively. *Formulas* and *terms* are generated in a well-sorted manner by

$$t ::= x \mid f(t, \ldots, t) \qquad \varphi ::= t \simeq t \mid p(t, \ldots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x_1 \ldots x_n.\varphi$$

in which $x, x_1, \ldots, x_n \in \mathcal{X}$, $p \in \mathcal{P}$ and $f \in \mathcal{F}$. The predicate symbol $\simeq$ stands for *equality.* The terms in a formula φ are denoted by $\mathbf{T}(\varphi)$. In a function or predicate application, the symbol being applied is referred as the term's *top symbol.* The *free variables* of a formula φ are denoted by $\mathrm{FV}(\varphi)$. A formula or term is *ground* iff it contains no variables. Whenever convenient, an enumeration of symbols $s_1, \ldots, s_n$ will be represented as $\mathbf{s}$.

A *substitution* σ is a mapping from variables to terms. The application of σ to the formula φ (respectively the term t) is denoted by $\varphi\sigma$ ($t\sigma$). The *domain of* σ is the set $dom(\sigma) = \{x \mid x \in \mathcal{X} \text{ and } x\sigma \neq x\}$, while the *range of* σ is $ran(\sigma) = \{x\sigma \mid x \in dom(\sigma)\}$. A substitution σ is *ground* iff every term in $\mathrm{ran}(\sigma)$ is ground and *acyclic* iff, for any variable x, x does not occur in $x\sigma \ldots \sigma$. For an acyclic substitution, $\sigma^\star$ is the fixed point substitution of σ.

Given a set of ground terms $\mathbf{T}$ closed under the subterm relation and a congruence relation $\simeq$ on $\mathbf{T}$, a *congruence* over $\mathbf{T}$ is a subset of $\{s \simeq t \mid s, t \in \mathbf{T}\}$ closed under entailment. The *congruence closure* (CC, for short) of a set of equations E on a set of terms $\mathbf{T}$ is the least congruence on $\mathbf{T}$ containing E. Given a consistent set of equality literals E, two terms t_1, t_2 are said *congruent* iff $E \models t_1 \simeq t_2$ and *disequal* iff $E \models t_1 \not\simeq t_2$. The *congruence class* in $\mathbf{T}$ of a given term is the set of terms in $\mathbf{T}$ congruent to it. The signature of a term is the term itself for a nullary symbol, and $f(c_1, \ldots c_n)$ for a term $f(t_1, \ldots t_n)$ with c_i being the class of t_i. The *signature class* of t is a set $[t]_E$ containing one and only one term in the class of t for each signature. Notice that the signature class of two terms in the same class is the same set of terms, and is a subset of the congruence class. We drop the subscript in $[t]_E$ when E is clear from the context. The *set of signature classes* of E on a set of terms $\mathbf{T}$ is $E^{\mathrm{CC}} = \{[t] \mid t \in \mathbf{T}\}$.

3 *E*-ground (dis)unification

For simplicity, and without loss of generality, we consider formulas in Skolem form, with all quantified subformulas being quantified clauses; we also assume all atomic formulas are equalities. SMT solvers proceed by enumerating the models for the propositional abstraction of the input formula, i.e. the formula obtained by replacing every atom and quantified subformula by a proposition. Such a model of the propositional abstraction corresponds to a set $E \cup \mathcal{Q}$, in which E and $\mathcal{Q}$ are conjunctive sets of ground literals and quantified formulas, respectively. If $E \cup \mathcal{Q}$ is consistent, all of its models also satisfy the input formula; if not, a new candidate model is derived. The ground SMT solver first checks the satisfiability of E, and, if it is satisfiable, proceeds to reason on the set of quantified formulas $\mathcal{Q}$. Ground instances $\mathcal{I}$ are derived from $\mathcal{Q}$, and subsequently the satisfiability of $E \cup \mathcal{I}$ is checked. This is repeated until either a conflict is found, and a new model for the propositional abstraction must be produced, or no more instantiations are possible. Of course, the whole process might not terminate and the solver might loop indefinitely.

In this approach, a central problem is to determine which instances $\mathcal{I}$ to derive. Section 5 shows that the problem of finding instances via existing instantiation techniques can be reduced to the problem of E-ground (dis)unification.

Definition 1 (E-ground (dis)unification). *Given two finite sets of equality literals E and L, E being ground, the E-ground (dis)unification problem is that of finding substitutions σ such that $E \models L\sigma$.*

E-ground (dis)unification can be recast as the classic problem of (non-simultaneous) rigid E-unification (transformation proof in Appendix B), i.e. computing substitutions σ such that $E^{eq}\sigma \models s\sigma \simeq t\sigma$, in which E^{eq} is a set of equations and s, t are terms. Rigid E-unification has been studied extensively in the context of automated theorem proving [2,9,14]. In particular, its intrinsic relation with congruence closure has been investigated by Goubault [19] and Tiwari et al. [26], in which variations of the classic procedure are integrated with first-order rewriting techniques and the search for solutions is guided by the structure of the terms. We build on these ideas to develop our method for solving E-ground (dis)unification, as discussed in Section 4.

Example 1. Consider the sets $E = \{f(a) \simeq f(b), h(a) \simeq h(c), g(b) \not\simeq h(c)\}$ and $L = \{h(x_1) \simeq h(c), h(x_2) \not\simeq g(x_3), f(x_1) \simeq f(x_3), x_4 \simeq g(x_5)\}$. A solution for their E-ground (dis)unification problem is $\{x_1 \mapsto a, x_2 \mapsto c, x_3 \mapsto b, x_4 \mapsto g(x_5)\}$.

The above example shows that x_5 can be mapped to any term; this E-ground (dis)unification problem has infinitely many solutions. However, here, like in general,[1] the set of all solutions can be finitely represented:

Theorem 1. *Given an E-ground (dis)unification problem, if a substitution σ exists such that $E \models L\sigma$, then there is an acyclic substitution σ' such that $ran(\sigma') \subseteq \mathbf{T}(E \cup L)$, $\sigma'^{\star}$ is ground, and $E \models L\sigma'^{\star}$.*

Proof. The proof can be found in Appendix A. □

As a corollary, the problem is in NP: it suffices indeed to guess an acyclic substitution with $ran(\sigma') \subseteq \mathbf{T}(E \cup L)$, and check (polynomially) that it is a solution. The problem is also NP-hard, by reduction of 3-SAT (Appendix C). As our experiments show, however, a concrete algorithm effective in practice is possible.

[1] It is assumed, without loss of generality, that $\mathbf{T}(E \cup L)$ contains at least one ground term of each sort in $E \cup L$.

4 Congruence Closure with Free Variables

In this section we describe a calculus to find each substitution σ solving an E-ground (dis)unification problem $E \models L\sigma$. This calculus, *Congruence Closure with Free Variables* (CCFV), uses a congruence closure algorithm as a core element to guide the search and build solutions. It proceeds by building a set of equations E_σ such that $E \cup E_\sigma \models L$, in which E_σ corresponds to a solution substitution, built step by step, by decomposing L in a top-down manner into sets of simpler constraints.

Example 2. Considering again E and L as in Example 1, the calculus should find σ such that

$$\begin{aligned} & f(a) \simeq f(b), h(a) \simeq h(c), g(b) \not\simeq h(c) \\ & \qquad \models (h(x_1) \simeq h(c) \wedge h(x_2) \not\simeq g(x_3) \wedge f(x_1) \simeq f(x_3) \wedge x_4 \simeq g(x_5))\, \sigma \end{aligned}$$

For L to be entailed by $E \cup E_\sigma$, each of its literals contributes to equations in E_σ in the following manner:

- $h(x_1) \simeq h(c)$: either $x_1 \simeq c$ or $x_1 \simeq a$ belongs to E_σ;
- $h(x_2) \not\simeq g(x_3)$: either $x_2 \simeq c \wedge x_3 \simeq b$ or $x_2 \simeq a \wedge x_3 \simeq b$ belongs to E_σ;
- $f(x_1) \simeq f(x_3)$: either $x_1 \simeq x_3$ or $x_1 \simeq a \wedge x_3 \simeq b$ or $x_1 \simeq b \wedge x_3 \simeq a$ must be in E_σ;
- $x_4 \simeq g(x_5)$: the literal itself must be in E_σ.

One solution is thus $E_\sigma = \{x_1 \simeq a, x_2 \simeq a, x_3 \simeq b, x_4 \simeq g(x_5)\}$, corresponding to the acyclic substitution $\sigma = \{x_1 \mapsto a, x_2 \mapsto a, x_3 \mapsto b, x_4 \mapsto g(x_5)\}$. Notice that, for any ground term $t \in \mathbf{T}(E \cup L)$, $\sigma_g = \sigma \cup \{x_5 \mapsto t\}$ is such that $ran(\sigma_g) \subseteq \mathbf{T}(E \cup L)$, ${\sigma_g}^\star$ is ground, and $E \models L{\sigma_g}^\star$.

4.1 The calculus

Given an E-ground (dis)unification problem $E \models L\sigma$, the CCFV calculus computes the various possible E_σ corresponding to a coverage of all substitution solutions, i.e. such that $E \cup E_\sigma \models L$. We describe the calculus as a set of rules that operate on states of the form $E_\sigma \Vdash_E C$, in which C is a (disjunctive normal form) formula stemming from the decomposition of L into simpler constraints, and E_σ is a conjunctive set of equalities representing a partial solution. Starting from the initial state $\varnothing \Vdash_E L$, the right side of the state is progressively decomposed, whereas the left side is step by step augmented with new equalities building the candidate solution. Example 2 shows that, for a literal to be entailed by $E \cup E_\sigma$, sometimes several solutions E_σ exist, thus the calculus involves branching. To simplify the presentation, the rules do not apply branching directly, but build disjunctions on the right part of the state, those disjunctions later leading to branching. A branch is closed when its constraint is decomposed into either $\bot$ or $\top$. The latter are branches for which $E \cup E_\sigma \models L$ holds.

The set of CCFV derivation rules is presented in Table 1; t stands for a ground term, x, y for variables, u for non-ground terms, $u_1, \ldots, u_n$ for terms such that at least one is non-ground and $s, s_1, \ldots, s_n$ for terms in general. Rules are applied top-down, the symmetry of equality being used implicitly. Each rule simplifies the constraint of the right hand side of the state, and as a consequence any derivation strategy is terminating (Theorem 2).

When an equality is added to the left hand side of a state $E_\sigma \Vdash_E C$ (rule Assign), the constraint C is normalized with respect to congruence closure to reflect the assignments to variables. That is, all terms in C are representatives of classes in the congruence closure of $E \cup E_\sigma$. We write

$$rep(x) = \begin{cases} \text{some chosen } y \in [x]_{E_\sigma} & \text{if all terms in } [x]_{E_\sigma} \text{ are variables} \\ rep(f(\mathbf{s})) & \text{otherwise, for some } f(\mathbf{s}) \in [x]_{E_\sigma} \end{cases}$$
$$rep(f(s_1, \ldots, s_n)) = \begin{cases} f(s_1, \ldots, s_n) & \text{if } f(s_1, \ldots, s_n) \text{ is ground} \\ f(rep(s_1), \ldots, rep(s_n)) & \text{otherwise} \end{cases}$$

and write $rep(C)$ to denote the result of applying rep on both sides of each literal $s \simeq s'$ or $s \not\simeq s'$ in C. The above definition of rep leaves room for some choice of representative, but soundness and completeness are not impacted by the choice. What actually matters is whether the representative is a variable, a ground term or a non-ground function application. The Assign rule adds equations from the right side of the state into the tentative solution in the left side of the state: it extends E_σ with the mapping for a variable. Because C is replaced by $rep(C)$, one variable (either x, or s if it is a variable) disappears from the right side.

The other rules can be divided into two categories. First are the branching rules (U_var through R_gen), which enumerate all possibilities for deriving the entailment of some literal from C. For example, the rule U_comp enumerates the possibilities for which a literal of the form $f(u_1, \ldots, u_n) \simeq f(s_1, \ldots, s_n)$ is entailed, which may be either due to syntactic unification, since both terms have the same top symbol, or by matching f-terms occurring in the same signature class of E^{CC}. Second are the structural rules (Split, Fail and Yield), which create or close branches. Split creates branches when there are disjunctions in the constraint. Fail closes a branch when it is no longer possible to build on the current solution to entail the remaining constraints. Yield closes a branch when all remaining constraints are already entailed by $E \cup E_\sigma$, with E_σ embodying a solution for the given E-ground (dis)unification problem. Theorems 3 and 4 state the correctness of the calculus.

If a branch is closed with Yield, the respective E_σ defines a substitution $\sigma = \{x \mapsto rep(x) \mid x \in \mathrm{FV}(L)\}$. The set $\textsc{Sols}(E_\sigma)$ of all ground solutions extractable from E_σ is composed of substitutions σ_g which extend σ by mapping all variables in $ran(\sigma^\star)$ into ground terms in $\mathbf{T}(E \cup L)$, s.t. each σ_g is acyclic, $\sigma_g^\star$ ground and $E \models L\sigma_g^\star$.

4.2 A strategy for the calculus

A possible derivation strategy for CCFV, given an initial state $\varnothing \Vdash_E L$, is to apply the sequence of steps described below at each state $E_\sigma \Vdash_E C$. Let sel be a function that selects a literal from a conjunction according to some heuristic,

$$\frac{E_\sigma \Vdash_E x \simeq s \wedge C}{E_\sigma \cup \{x \simeq s\} \Vdash_E rep(C)}\ \textsc{Assign} \qquad \text{if } x \notin \mathrm{FV}(s)$$

$$\frac{E_\sigma \Vdash_E x \simeq f(u_1, \ldots, u_n) \wedge C}{E_\sigma \Vdash_E \bigvee_{[t] \in E^{\mathrm{cc}}, f(t_1,\ldots,t_n) \in [t]} (x \simeq t \wedge u_1 \simeq t_1 \wedge \cdots \wedge u_n \simeq t_n \wedge C)}\ \textsc{U\_var}$$

if $x \in \mathrm{FV}(f(u_1, \ldots, u_n))$

$$\frac{E_\sigma \Vdash_E f(u_1, \ldots, u_n) \simeq f(s_1, \ldots, s_n) \wedge C}{E_\sigma \Vdash_E (u_1 \simeq s_1 \wedge \cdots \wedge u_n \simeq s_n \wedge C) \vee \bigvee_{\substack{[t] \in E^{\mathrm{cc}},\\ f(t_1,\ldots,t_n) \in [t], f(t'_1,\ldots,t'_n) \in [t]}} \begin{pmatrix} u_1 \simeq t_1 \wedge \cdots \wedge u_n \simeq t_n \wedge \\ s_1 \simeq t'_1 \wedge \cdots \wedge s_n \simeq t'_n \wedge C \end{pmatrix}}\ \textsc{U\_comp}$$

$$\frac{E_\sigma \Vdash_E f(u_1, \ldots, u_n) \simeq g(s_1, \ldots, s_m) \wedge C}{E_\sigma \Vdash_E \bigvee_{\substack{[t] \in E^{\mathrm{cc}},\\ f(t_1,\ldots,t_n) \in [t], g(t'_1,\ldots,t'_m) \in [t]}} \begin{pmatrix} u_1 \simeq t_1 \wedge \cdots \wedge u_n \simeq t_n \wedge \\ s_1 \simeq t'_1 \wedge \cdots \wedge s_m \simeq t'_m \wedge C \end{pmatrix}}\ \textsc{U\_gen}$$

if $f \neq g$

$$\frac{E_\sigma \Vdash_E x \not\simeq y \wedge C}{E_\sigma \Vdash_E \bigvee_{[t_1],[t_2] \in E^{\mathrm{cc}}, E \models t_1 \not\simeq t_2} (x \simeq t_1 \wedge y \simeq t_2 \wedge C)}\ \textsc{R\_var}$$

$$\frac{E_\sigma \Vdash_E x \not\simeq f(s_1, \ldots, s_n) \wedge C}{E_\sigma \Vdash_E \bigvee_{\substack{[t],[t'] \in E^{\mathrm{cc}},\\ E \models t \not\simeq t', f(t'_1,\ldots,t'_n) \in [t']}} (x \simeq t \wedge s_1 \simeq t'_1 \wedge \cdots \wedge s_n \simeq t'_n \wedge C)}\ \textsc{R\_fapp}$$

$$\frac{E_\sigma \Vdash_E f(u_1, \ldots, u_n) \not\simeq g(s_1, \ldots, s_m) \wedge C}{E_\sigma \Vdash_E \bigvee_{\substack{[t],[t'] \in E^{\mathrm{cc}}, E \models t \not\simeq t',\\ f(t_1,\ldots,t_n) \in [t], g(t'_1,\ldots,t'_m) \in [t']}} \begin{pmatrix} u_1 \simeq t_1 \wedge \cdots \wedge u_n \simeq t_n \wedge \\ s_1 \simeq t'_1 \wedge \cdots \wedge s_m \simeq t'_m \wedge C \end{pmatrix}}\ \textsc{R\_gen}$$

$$\frac{E_\sigma \Vdash_E C_1 \vee C_2}{E_\sigma \Vdash_E C_1 \qquad E_\sigma \Vdash_E C_2}\ \textsc{Split} \qquad \frac{E_\sigma \Vdash_E C}{E_\sigma \Vdash_E \top}\ \textsc{Yield} \quad \text{if } E \cup E_\sigma \models C$$

$$\frac{E_\sigma \Vdash_E C}{E_\sigma \Vdash_E \bot}\ \textsc{Fail}$$

if no other rule can be applied; or
C is a conjunction and $E \not\models \ell$, for some ground $\ell \in C$

Table 1: The CCFV calculus in equational FOL. E is fixed from a problem $E \models L\sigma$.

such as selecting first literals with less variables or literals whose top symbols have less ground signatures in E^{CC}. The result of SEL is denoted *selected literal*. Since no two rules can be applied on the same literal, the function SEL effectively enforces an order on the application of the rules.

1. *Select branch*: While C is a disjunction, apply SPLIT and consider the left-most branch, by convention.
2. *Simplify constraint*: Apply the rule for which $\text{SEL}(C)$ is amenable.
3. *Discard failure*: If FAIL was applied or a branching rule had the empty disjunction as a result, discard this branch and consider the next open branch.
4. *Mark success*: If all remaining constraints in the branch are entailed by $E \cup E_\sigma$, apply YIELD to mark the successful branch and then consider the next open branch.

A solution σ for the E-ground (dis)unification problem $E \models L\sigma$ can be extracted at each branch terminated by the YIELD rule (Corollary 1).

Example 3. Consider again E and L as in Example 1. The set of signature classes of E is

$$E^{\text{CC}} = \{[a], [b], [c], [f(a), f(b)], [h(a), h(c)], [g(b)]\}$$

Let SEL select the literal in C with the minimum number of variables. The derivation tree produced by CCFV for this problem is shown below. Selected literals are underlined. Disjunctions and the application of SPLIT are kept implicit to simplify the presentation, as is the handling of $x_4 \simeq g(x_5)$. Its entailment does not relate with the other literals in L and it can be handled by an early application of ASSIGN.

$$\dfrac{\varnothing \Vdash_E \underline{h(x_1) \simeq h(c)}, h(x_2) \not\simeq g(x_3), f(x_1) \simeq f(x_3)}{\mathcal{A} \qquad\qquad\qquad \mathcal{B}}\ \text{U\_COMP}$$

with $\mathcal{A}$ being

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varnothing \Vdash_E \underline{x_1 \simeq c}, h(x_2) \not\simeq g(x_3),\ f(x_1) \simeq f(x_3)}{\{x_1 \simeq c\} \Vdash_E h(x_2) \not\simeq g(x_3), \underline{f(c) \simeq f(x_3)}}\ \text{ASSIGN}}{\{x_1 \simeq c\} \Vdash_E h(x_2) \not\simeq g(x_3), \underline{x_3 \simeq c}}\ \text{U\_COMP}}{\{x_1 \simeq c, x_3 \simeq c\} \Vdash_E h(x_2) \not\simeq g(c)}\ \text{ASSIGN}}{\{x_1 \simeq c, x_3 \simeq c\} \Vdash_E \bot}\ \text{R\_GEN}}{\{x_1 \simeq c, x_3 \simeq c\} \Vdash_E \bot}\ \text{FAIL}$$

and $\mathcal{B}$:

$$\dfrac{\dfrac{\varnothing \Vdash_E \underline{x_1 \simeq a}, h(x_2) \not\simeq g(x_3), f(x_1) \simeq f(x_3)}{\{x_1 \simeq a\} \Vdash_E h(x_2) \not\simeq g(x_3), \underline{f(a) \simeq f(x_3)}}\ \text{ASSIGN}}{\dfrac{\dfrac{\dfrac{\{x_1 \simeq a\} \Vdash_E h(x_2) \not\simeq g(x_3), \underline{x_3 \simeq a}}{\{x_1 \simeq a, x_3 \simeq a\} \Vdash_E h(x_2) \not\simeq g(a)}\ \text{ASSIGN}}{\{x_1 \simeq a, x_3 \simeq a\} \Vdash_E \bot}\ \text{R\_GEN}}{\{x_1 \simeq a, x_3 \simeq a\} \Vdash_E \bot}\ \text{FAIL} \quad \dfrac{\dfrac{\dfrac{\dfrac{\{x_1 \simeq a\} \Vdash_E h(x_2) \not\simeq g(x_3), \underline{x_3 \simeq b}}{\{x_1 \simeq a, x_3 \simeq b\} \Vdash_E h(x_2) \not\simeq g(b)}\ \text{ASSIGN}}{\{x_1 \simeq a, x_3 \simeq b\} \Vdash_E x_2 \simeq a}\ \text{R\_GEN}}{\{x_1 \simeq a, x_2 \simeq a, x_3 \simeq b\} \Vdash_E \top}\ \text{ASSIGN}}{\{x_1 \simeq a, x_2 \simeq a, x_3 \simeq b\} \Vdash_E \top}\ \text{YIELD}}\ \text{U\_COMP}$$

A solution is produced by the rightmost branch of $\mathcal{B}$.

4.3 Correctness of CCFV

Theorem 2 (Termination). *All derivations in* CCFV *are finite.*

Proof (sketch). The width of any split rule is always finite. It then suffices to show that the depth of the tree is bounded. For simplicity, but without any fundamental effect on the proof, let us assume that all rules but SPLIT apply on conjunctions. Let $d(C)$ be the sum of the depths of all occurrences of variables in the literals of the conjunction C. The ASSIGN rule decreases the number of variables of C. The FAIL and YIELD rules close a branch. All remaining rules from $E_\sigma \Vdash_E C$ to $E'_\sigma \Vdash_E C'_1 \vee \cdots \vee C'_n$ decrease d, i.e. $d(C) > d(C'_1), \ldots, d(C) > d(C'_n)$. At each node, $d(C)$ or the number of variables in C are decreasing, except at the SPLIT steps. Since no branch can contain infinite sequences of SPLIT applications, the depth is always finite. □

Lemma 1. *Given a computed solution E_σ for an E-ground (dis)unification problem $E \models L\sigma$, each $\sigma_g \in$ SOLS(E_σ) is an acyclic substitution such that $ran(\sigma_g) \subseteq \mathbf{T}(E \cup L)$ and $\sigma_g^\star$ is ground.*

Proof (sketch). The proof can be found in Appendix D. □

Lemma 2 (Rules capture entailment conditions). *For each rule*

$$\frac{E_\sigma \Vdash_E C}{E'_\sigma \Vdash_E C'}\ \mathrm{R}$$

and any ground substitution σ, $E \models (\{C\} \cup E_\sigma)\sigma$ iff $E \models (\{C'\} \cup E'_\sigma)\sigma$.

Proof (sketch). The proof can be found in Appendix D. □

Theorem 3 (Soundness). *Whenever a branch is closed with* YIELD, *every $\sigma_g \in$ SOLS(E_σ) is s.t. $E \models L\sigma_g^\star$.*

Proof (sketch). Consider an arbitrary substitution $\sigma_g \in$ SOLS(E_σ) at the application of YIELD. Lemma 1 ensures that $\sigma_g^\star$ is ground. Thanks to the side condition of the YIELD rule and of the construction of $\sigma_g^\star$, $E \models (\{C\} \cup E_\sigma)\sigma_g^\star$ at the leaf. Then, thanks to Lemma 2, $E \models (\{C\} \cup E_\sigma)\sigma_g^\star$ also holds at the root, in which $C = L$ and $E_\sigma = \emptyset$. Thus $E \models L\sigma_g^\star$. □

Theorem 4 (Completeness). *Let σ be a solution for an E-ground (dis)unification problem $E \models L\sigma$. Then there exists a derivation tree starting on $\varnothing \Vdash_E L$ with at least one branch closed with* YIELD *s.t. $\sigma_g \in$ SOLS(E_σ) and $E \models L\sigma_g^\star$.*

Proof (sketch). By Theorem 1, there is an acyclic substitution σ_g corresponding to σ such that $ran(\sigma_g) \subseteq \mathbf{T}(E \cup L)$, $\sigma_g^\star$ is ground and $E \models L\sigma_g^\star$. Lemma 2 ensures that all rules in CCFV preserve the entailment conditions according to ground substitutions, therefore there is a branch in the derivation tree starting from $\varnothing \Vdash_E L$ whose leaf is $E_\sigma \Vdash_E \top$ and $\sigma_g \in$ SOLS(E_σ). □

Corollary 1 (CCFV decides E-ground (dis)unification). *Any derivation strategy based on the* CCFV *calculus is a decision procedure to find all solutions σ for the E-ground (dis)unification problem $E \models L\sigma$.*

5 Relation to instantiation techniques

Here we discuss how different instantiation techniques for evaluating a candidate model $E \cup \mathcal{Q}$ can be related with E-ground (dis)unification and thus integrated with CCFV.

5.1 Trigger based instantiation

The most common instantiation technique in SMT solving is a heuristic one: its search is based solely on E-matching of selected triggers [11,16,25], without further semantic criteria. A *trigger* T for a quantified formula $\forall \mathbf{x}.\psi \in \mathcal{Q}$ is a set of terms $f_1(\mathbf{s}_1), \dots, f_n(\mathbf{s}_n) \in \mathbf{T}(\psi)$ s.t. $\{\mathbf{x}\} \subseteq \mathrm{FV}(f_1(\mathbf{s}_1)) \cup \cdots \cup \mathrm{FV}(f_n(\mathbf{s}_n))$. Instantiations are determined by E-matching all terms in T with terms in $\mathbf{T}(E)$, such that resulting substitutions allow instantiating $\forall \mathbf{x}.\psi$ into ground formulas. Computing such substitutions amounts to solving the E-ground (dis)unification problem

$$E \models (f_1(\mathbf{s}_1) \simeq y_1 \wedge \cdots \wedge f_n(\mathbf{s}_n) \simeq y_n)\, \sigma$$

with the further restriction that σ is acyclic, $ran(\sigma) \subseteq \mathbf{T}(E \cup L)$ and σ is ground. This forces each y_i to be grounded into a term in $\mathbf{T}(E)$, thus enumerating all possibilities for E-matching $f_i(\mathbf{s}_i)$.[2] The desired instantiations are obtained by restricting the found solutions to $\mathbf{x}$.

Example 4. Consider the sets $E = \{f(a) \simeq g(b),\ h(a) \simeq b, f(a) \simeq f(c)\}$ and $\mathcal{Q} = \{\forall x.\ f(x) \not\simeq g(h(x))\}$. Triggers from $\mathcal{Q}$ are $T_1 = \{f(x)\}$, $T_2 = \{h(x)\}$, $T_3 = \{f(x), g(h(x))\}$ and so on. The instantiations from those triggers are derived from the solutions yielded by CCFV for the respective problems:

- $E \models (f(x) \simeq y)\sigma$, solved by substitutions $\sigma_1 = \{y \mapsto f(a), x \mapsto a\}$ and $\sigma_2 = \{y \mapsto f(c), x \mapsto c\}$
- $E \models (h(x) \simeq y)\sigma$, solved by $\sigma = \{y \mapsto h(a), x \mapsto a\}$
- $E \models (f(x) \simeq y_1 \wedge g(h(x)) \simeq y_2)\sigma$, by $\sigma = \{y_1 \mapsto f(a), y_2 \mapsto g(b), x \mapsto a\}$

Discarding entailed instances Trigger-based instantiation may produce instances which are already entailed by the ground model. Such instances most probably will not contribute to the solving, so they should be discarded. Checking this, however, is not straightforward with pre-processing techniques. CCFV, on the other hand, allows it by simply checking, given an instantiation σ for a quantified formula $\forall \mathbf{x}.\psi$, whether there is a literal $\ell \in \psi$ s.t. $E \cup E_\sigma \models \ell$, with $E_\sigma = \{x \simeq x\sigma \mid x \in dom(\sigma)\}$.

---

[2] For CCFV to generate such solutions it is sufficient to add the side condition to Assign that s is a variable or a ground term and to remove the side condition of U_var. This will lead to the application of U_var in each $f_i(\mathbf{s}_1) \simeq y_i$.

5.2 Conflict based instantiation

A goal-oriented instantiation technique was introduced by Reynolds et al. [23] to provide fewer and more meaningful instances. Quantified formulas are evaluated, independently, in search for *conflicting instances*: for each quantified formula $\forall \mathbf{x}.\psi \in \mathcal{Q}$, only instances $\psi\sigma$ for which $E \cup \psi\sigma$ is unsatisfiable are derived. Such instances force the derivation of a new candidate model $E \cup \mathcal{Q}$ for the formula. Finding a conflicting instance amounts to solving the E-ground (dis)unification problem

$$E \models \neg\psi\sigma, \text{ for some } \forall \mathbf{x}.\psi \in \mathcal{Q}$$

since $\neg\psi$ is a conjunction of equality literals. Differently from the algorithm shown in [23], CCFV finds all conflicting instantiations for a given quantified formula.

Example 5. Let E and $\mathcal{Q}$ be as in Example 4. Applying CCFV in the problem

$$E \models (f(x) \simeq g(h(x)))\,\sigma$$

leads to the sole conflicting instantiation $\sigma = \{x \mapsto a\}$.

Propagating equalities As discussed in [23], even when the search for conflicting instances fails it is still possible to "propagate" equalities. Given some $\neg\psi = \ell_1 \wedge \cdots \wedge \ell_n$, let σ be a ground substitution s.t. $E \models \ell_1\sigma \wedge \cdots \wedge \ell_{k-1}\sigma$ and all remaining literals $\ell_k\sigma, \ldots, \ell_n\sigma$ not entailed are ground disequalities with $(\mathbf{T}(\ell_k) \cup \cdots \cup \mathbf{T}(\ell_n)) \subseteq \mathbf{T}(E)$. The instantiation $\forall \mathbf{x}.\psi \rightarrow \psi\sigma$ introduces a disjunction of equalities constraining $\mathbf{T}(E)$. CCFV can generate such propagating substitutions if the side conditions of Fail and Yield are relaxed w.r.t. ground disequalities whose terms occur in $\mathbf{T}(E)$ and originally had variables: the former is not applied based on them and the latter is if all other literals are entailed.

Example 6. Consider $E = \{f(a) \simeq t, t' \simeq g(a)\}$ and $\forall x.\ f(x) \not\simeq t \vee f(x) \simeq g(x)$. When applying CCFV in the problem

$$E \models (f(x) \simeq t \wedge f(x) \not\simeq g(x))\,\sigma$$

to entail the first literal a candidate solution $E_\sigma = \{x \simeq a\}$ is produced. The second literal would then be normalized to $f(a) \not\simeq g(a)$, which would lead to the application of Fail, since it is not entailed by E. However, as it is a disequality whose terms are in $\mathbf{T}(E)$ and originally had variables, the rule applied is Yield instead. The resulting substitution $\sigma = \{x \mapsto a\}$ leads to propagating the equality $f(a) \simeq g(a)$, which merges two classes previously different in E^{CC}.

5.3 Model based instantiation (MBQI)

A complete instantiation technique was introduced by Ge and de Moura [18]. The set E is extended into a total model, each quantified formula is evaluated in this total model, and conflicting instances are generated. The successive rounds of instantiation either lead to unsatisfiability or, when no conflicting instance is generated, to satisfiability with a concrete model. Here we follow the model construction guidelines by Reynolds et al. [24].

A distinguished term e^τ is associated to each sort $\tau \in \mathcal{S}$. For each $f \in \mathcal{F}$ with sort $\langle \tau_1, \ldots, \tau_n, \tau \rangle$ a *default value* ξ_f is defined such that

$$\xi_f = \begin{cases} f(t_1, \ldots, t_n) \in \mathbf{T}(E) & \text{if } [t_1] = [e^{\tau_1}], \ldots, [t_n] = [e^{\tau_n}] \\ \text{some } t \in \mathbf{T}(E) & \text{otherwise} \end{cases}$$

The extension E_{TOT} is built s.t. all fresh ground terms which might be considered when evaluating $\mathcal{Q}$ are in its congruence closure, according to the respective default values; and all terms in $\mathbf{T}(E)$ not asserted equal are explicitly asserted disequal, i.e.

$$\begin{aligned} E_{\text{TOT}} = E \cup & \bigcup\nolimits_{t_1, t_2 \in \mathbf{T}(E)} \{ t_1 \not\simeq t_2 \mid E \not\models t_1 \simeq t_2 \} \\ & \bigcup\nolimits_{\forall \mathbf{x}. \psi \in \mathcal{Q}, \mathbf{t} \in \mathbf{T}(E)} \left\{ f(\mathbf{s})\sigma \simeq \xi_f \;\middle|\; \begin{array}{l} \sigma = \{\mathbf{x} \mapsto \mathbf{t}\}, f(\mathbf{s}) \in \mathbf{T}(\psi) \text{ and} \\ f(\mathbf{s})\sigma \text{ is not in the CC of } E. \end{array} \right\} \end{aligned}$$

As before, finding conflicting instances amounts to solving the E-ground (dis)unification problem

$$E_{\text{TOT}} \models \neg\psi\sigma, \text{ for some } \forall \mathbf{x}.\psi \in \mathcal{Q}$$

Example 7. Let $E = \{f(a) \simeq g(b),\ h(a) \simeq b\}$, $\mathcal{Q} = \{\forall x.\ f(x) \not\simeq g(x),\ \forall xy.\ \psi\}$ and $e = a$, with all terms having the same sort. The computed default values of the function symbols are $\xi_f = f(a), \xi_g = a, \xi_h = h(a)$. For simplicity, the extension E_{TOT} is shown explicitly only for $\forall x.\ f(x) \not\simeq g(x)$,

$$\begin{aligned} E_{\text{TOT}} = E & \cup \{a \not\simeq b, a \not\simeq f(a), b \not\simeq f(a)\} \\ & \cup \{f(b) \simeq f(a), f(f(a)) \simeq f(a), g(a) \simeq a, g(f(a)) \simeq a\} \cup \{\ldots\} \end{aligned}$$

Applying CCFV in

$$\{\ldots, f(a) \simeq g(b), f(b) \simeq f(a), \ldots\} \models f(x) \simeq g(x)\sigma$$

leads to a conflicting instance with $\sigma = \{x \mapsto b\}$. Notice that it is not necessary to explicitly build E_{TOT}, which can be quite large. Terms can be defined lazily as they are required by CCFV for building potential solutions.

6 Implementation and Experiments

CCFV has been implemented in the veriT [10] and CVC4 [6] solvers. As is common in SMT solvers, they make use of an E-graph to represent the set of signature classes E^{CC} and efficiently check ground entailment.[3] Indexing techniques

[3] Currently the ground congruence closure procedures are not closed under entailment w.r.t. disequalities. E.g. $g(f(a), h(b)) \not\simeq g(f(b), h(a)) \in E$ does not lead to the addition of $a \not\simeq b$ to the data structure. A complete implementation of CCFV requires the ground congruence closure to entail all entailed disequalities.

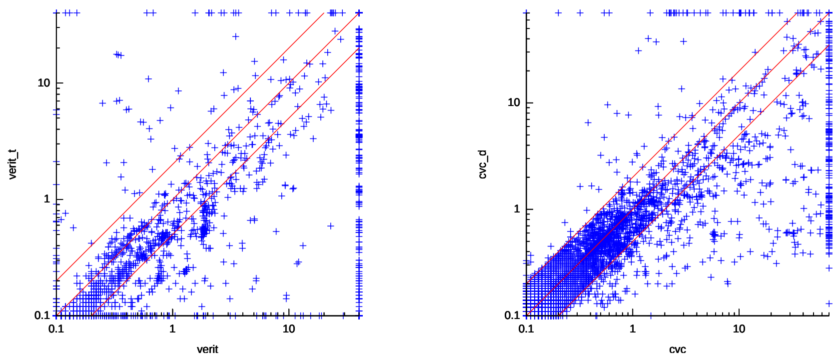
for fast retrieval of candidates are paramount for a practical procedure, so E^{CC} is indexed by top symbols. Each function symbol points to all their related signatures. They are kept sorted by congruence classes to allow binary search when retrieving all signatures with a given top symbol congruent to a given term. To quickly discard classes without signatures with a given top symbol, bit masks are associated to congruence classes: each symbol is assigned an arbitrary bit, and the mask for the class is the set of all bits of the top symbols. Another important optimization is to minimize E, since the candidate model $E \cup \mathcal{Q}$ produced by the SAT solver and guiding the instantiation is generally not minimal. A minimal partial model (a *prime implicant*) for the CNF is computed in linear time [15], and this model is further reduced to circumvent the effect of the CNF transformation, using a process similar to the one described by de Moura and Bjørner [11] for *relevancy*.

During rule application, matching a term $f(\mathbf{u})$ with a ground term $f(\mathbf{t})$ fails unless all the ground arguments are pairwise congruent. Thus after an assignment, if an argument of a term $f(\mathbf{u})$ in a branching constraint becomes ground, it can be checked whether there is a ground term $f(\mathbf{t}) \in \mathbf{T}(E)$ s.t., for every ground argument u_i, $E \models u_i \simeq t_i$. If no such term exists and $f(\mathbf{u})$ is not in a literal amenable for U_COMP, the branch can be *eagerly discarded*. For this technique, a dedicated index for each function symbol f maps tuples of pairs, with a ground term and a position, $\langle (t_1, i_1), \ldots, (t_k, i_k) \rangle$ to all signatures $f(t'_1, \ldots, t'_n)$ in E^{CC} s.t. $E \models t_1 \simeq t'_{i_1}, \ldots, E \models t_k \simeq t'_{i_k}$, i.e. all signatures whose arguments, in the respective positions, are congruent with the given ground terms.

Experiments Here we evaluate the impact of optimizations and instantiation techniques based on CCFV over previous versions and compare them against the state-of-the-art instantiation based solver Z3 [13]. Different configurations are identified in this section according to which techniques and algorithms they have activated:

t : trigger instantiation through CCFV;
c : conflict based instantiation through CCFV;
e : optimization for eagerly discarding branches with unmatchable applications;
d : discards already entailed trigger based instances (as in 5.1)

The configuration **verit** refers to the previous version of veriT, which only offered support for quantified formulas through naïve trigger instantiation, without further optimizations. The configuration **cvc** refers to version 1.5 of CVC4, which applies **t** and **c** by default, as well as propagation of equalities. Both implementations of CCFV include efficient term indexing and apply a simple selection heuristic, checking ground and reflexive literals first but otherwise considering the conjunction of constraints as a queue. The evaluation was made on the UF, UFLIA, UFLRA and UFIDL categories of SMT-LIB [8], with 10 495 benchmarks annotated as *unsatisfiable*, mostly stemming for verification and ITP platforms. The categories with bit vectors and non-linear arithmetic are currently not supported by veriT and in those in which uninterpreted functions



(a) **verit+t** vs. **verit** (b) **cvc+d** vs. **cvc**

Fig. 1: Improvements in veriT and CVC4

are not predominant the techniques shown here are not as effective. Our experiments were conducted using machines with 2 CPUs Intel Xeon E5-2630 v3, 8 cores/CPU, 126GB RAM, 2x558GB HDD. The timeout was set for 30 seconds, since our goal is evaluating SMT solvers as back-ends of verification and ITP platforms, which require fast answers.

Figure 1 exhibits an important impact of CCFV and the techniques and optimizations built on top of it. **verit+t** performs much better than **verit**, solely due to CCFV. **cvc+d** improves significantly over **cvc**, exhibiting the advantage of techniques based on the entailment checking features of CCFV. The comparison between the different configurations of veriT and CVC4 with the SMT solver Z3 (version 4.4.2) is summarized in Table 2, excluding categories whose problems are trivially solved by all systems, which leaves 8 701 problems for consideration. **verit+tc** shows further improvements, solving approximately the same number of problems as Z3, although mostly because of the better performance on the *sledgehammer* benchmarks, containing less theory symbols. It also performs best in the *grasshopper* families, stemming from the heap verification tool GRASShopper [22]. Considering the overall performance, both **cvc+d** and **cvc+e** solve significantly more problems than **cvc**, specially in benchmarks from verification platforms, approaching the performance of Z3 in these families. Both these techniques, as well as the propagation of equalities, are fairly important points in the performance of CVC4, so their implementation is a clear direction for improvements in veriT.

7 Conclusion and Future Work

We have introduced CCFV, a decision procedure for E-ground (dis)unification, and shown how the main instantiation techniques of SMT solving may be based on it. Our experimental evaluation shows that CCFV leads to significant improvements in the solvers CVC4 and veriT, making the former surpass the state-

Logic	Class	Z3	**cvc+d**	**cvc+e**	**cvc**	**verit+tc**	**verit+t**	**verit**
UF	grasshopper	418	411	420	415	**430**	418	413
	sledgehammer	1249	1438	**1456**	1428	1265	1134	1066
UFIDL	all	**62**	**62**	**62**	**62**	58	58	58
UFLIA	boogie	**852**	844	834	801	705	660	661
	sexpr	**26**	12	11	11	7	5	5
	grasshopper	341	322	326	319	**357**	340	335
	sledgehammer	1581	1944	**1953**	1929	1783	1620	1569
	simplify	**831**	766	706	705	803	735	690
	simplify2	**2337**	2330	2292	2286	2304	2291	2177
	Total	7697	**8129**	8060	7956	7712	7261	6916

Table 2: Instantiation based SMT solvers on SMT-LIB benchmarks

of-the-art in instantiation based SMT solving and the latter competitive in several benchmark libraries. The calculus presented is very general, allowing for different strategies and optimizations, as discussed in previous sections.

A direction for improvement is to use *lemma learning* in CCFV, in a similar manner as SAT solvers do. When a branch fails to produce a solution and is discarded, analyzing the literals which led to the conflict can allow *backjump* rather than simple backtracking, thus further reducing the solution search space. The *Complementary Congruence Closure* introduced by Backeman and Rümmer [4] could be extended to perform such an analysis.

Like other main instantiation techniques in SMT, the framework here focuses on the theory of equality only. Extensions to first-order theories such as arithmetic are left for future work. The implementation of MBQI based on CCFV, whose theoretical suitability we outlined, is left for future work as well. Another possible extension of CCFV is to handle rigid E-unification, so it could be applied in techniques such as BREU [5]. This amounts to have non-ground equalities in E, so it is not trivial. It would, however, allow integrating an efficient goal-oriented procedure into E-unification based calculi.

Acknowledgments We are grateful to David Déharbe for his help with the implementation of CCFV and to Jasmin Blanchette for suggesting textual improvements. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several universities as well as other organizations (`https://www.grid5000.fr`).

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998.
2. F. Baader and W. Snyder. Unification Theory. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.
3. L. Bachmair and H. Ganzinger. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
4. P. Backeman and P. Rümmer. Efficient Algorithms for Bounded Rigid E-unification. In H. de Nivelle, editor, *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 9323 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 2015.
5. P. Backeman and P. Rümmer. Theorem Proving with Bounded Rigid E-Unification. In A. Felty and A. Middeldorp, editors, *Proc. Conference on Automated Deduction (CADE)*, volume 9195 of *Lecture Notes in Computer Science*. Springer, 2015.
6. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification (CAV)*, pages 171–177. Springer, 2011.
7. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, 2009.
8. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *International Workshop on Satisfiability Modulo Theories (SMT)*, 2010.
9. B. Beckert. Ridig E-Unification. In W. Bibel and P. H. Schimidt, editors, *Automated Deduction: A Basis for Applications. Foundations: Calculi and Methods*, volume 1. Bluwer Academic Publishers, 1998.
10. T. Bouton, D. C. B. de Oliveira, D. Déharbe, and P. Fontaine. veriT: An Open, Trustable and Efficient SMT-Solver. In R. A. Schmidt, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156. Springer, 2009.
11. L. de Moura and N. Bjørner. Efficient E-Matching for SMT Solvers. In F. Pfenning, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 4603 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2007.
12. L. de Moura and N. Bjørner. Engineering DPLL(T) + Saturation. In A. Armando, P. Baumgartner, and G. Dowek, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Computer Science*, pages 475–490. Springer, 2008.
13. L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.
14. A. Degtyarev and A. Voronkov. Equality Reasoning in Sequent-Based Calculi. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 611–706. Elsevier, 2001.

15. D. Déharbe, P. Fontaine, D. Le Berre, and B. Mazure. Computing Prime Implicants. In *Formal Methods In Computer-Aided Design (FMCAD)*, pages 46–52. IEEE, 2013.
16. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A Theorem Prover for Program Checking. *J. ACM*, 52(3):365–473, 2005.
17. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, 1990.
18. Y. Ge and L. de Moura. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In A. Bouajjani and O. Maler, editors, *Computer Aided Verification (CAV)*, volume 5643 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2009.
19. J. Goubault. A rule-based algorithm for rigid E-unification. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Computational Logic and Proof Theory: Third Kurt Gödel Colloquium, KGC'93 Brno, Czech Republic, August 24–27, 1993 Proceedings*, pages 202–210. Springer, 1993.
20. G. Nelson and D. C. Oppen. Fast Decision Procedures Based on Congruence Closure. *J. ACM*, 27(2):356–364, 1980.
21. R. Nieuwenhuis and A. Oliveras. Fast congruence closure and extensions. *Information and Computation*, 205(4):557 – 580, 2007. Special Issue: 16th International Conference on Rewriting Techniques and Applications.
22. R. Piskac, T. Wies, and D. Zufferey. GRASShopper - Complete Heap Verification with Mixed Specifications. In E. Ábrahám and K. Havelund, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 8413 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2014.
23. A. Reynolds, C. Tinelli, and L. de Moura. Finding Conflicting Instances of Quantified Formulas in SMT. In *Formal Methods In Computer-Aided Design (FMCAD)*, pages 195–202. FMCAD Inc, 2014.
24. A. Reynolds, C. Tinelli, A. Goel, S. Krstić, M. Deters, and C. Barrett. Quantifier Instantiation Techniques for Finite Model Finding in SMT. In M. Bonacina, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 7898 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2013.
25. P. Rümmer. E-Matching with Free Variables. In N. Bjørner and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 7180 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2012.
26. A. Tiwari, L. Bachmair, and H. Ruess. Rigid E-Unification Revisited. In D. McAllester, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 1831 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2000.

Appendix A Proof of Theorem 1

Theorem. Given an E-ground (dis)unification problem, if a substitution σ exists s.t. $E \models L\sigma$, then there is an acyclic substitution σ' such that $ran(\sigma') \subseteq \mathbf{T}(E \cup L)$, $\sigma'^{\star}$ is ground, and $E \models L\sigma'^{\star}$.

Proof. First, we can assume that σ is ground. Indeed, if a non-ground substitution σ is s.t. $E \models \ell\sigma$, then $E \models \ell\sigma\sigma_g$ holds for any ground substitution σ_g. For convenience and without loss of generality, assume all terms in L are flat. We introduce the notations $E_\sigma = \{x \simeq x\sigma \mid x \in dom(\sigma)\}$ and $S_t = \{t' \mid t' \in \mathbf{T}(E \cup L \cup E_\sigma) \text{ and } E \cup E_\sigma \models t \simeq t'\}$, in which σ is a substitution and t is a term. Note that $E \models L\sigma^\star$ holds if and only if $E \cup E_\sigma \models L$ also does.

To compute the congruence closure of a set of equations E on a set of terms $\mathbf{T}$, it suffices to compute a congruence graph on $\mathbf{T}$. Two terms t, t' are equal according to E if and only if there is a path between them in the graph. There is a full edge in the graph between two terms in any equation of E, and there is a congruence edge between two terms such that all arguments are pairwise equal (i.e., there is a path between them). Some (congruence or full) redundant edges can be omitted if the extremities are already connected. The congruence graph is a least fixed point, so the explanation for the existence of a congruence edge can always be traced back to full edges only. Congruence edges can be ordered by their dependency.

Consider a congruence graph G induced by E on terms $\mathbf{T}(E \cup L \cup E_\sigma)$, and a congruence graph G' induced by furthermore adding edges for σ. Besides the edges corresponding to σ, that is, edges with a variable, the only additional edges are congruence edges; we assume no redundant edge is added to G'. Variables are at most linked to one term, and by a full edge. Other non-variable free term can only be linked either by a full edge to a variable, or to a term by a congruence edge.

All edges in $G' \setminus G$ involve at least one term u that is not (directly or indirectly) linked to any ground term in G. This is obviously true for full edges, since variables are not linked to any term in G. Consider the earliest (according to the dependency order of congruence edges) congruence edge between $f(t_1, \ldots, t_n)$ and $f(t'_1, \ldots, t'_n)$ contradicting this hypothesis. Then there is a path from $f(t_1, \ldots, t_n)$ to a ground term; this path cannot contain full edges, because this would involve two edges with a variable. Hence the path only contain congruence edges and $t_1, \ldots t_n$ are all equal to ground terms. The same holds for $f(t'_1, \ldots, t'_n)$, then t_i and t'_i are both linked to ground terms, for $i \in \{1, \ldots n\}$. One new earlier edge contradicts the hypothesis too.

As a corollary of the previous paragraph, given two terms t, t', if both terms are ground, $E \cup E_\sigma \models t \simeq t'$ if and only if $E \models t \simeq t'$.

In the following, we build an acyclic substitution σ' such that $ran(\sigma') \subseteq \mathbf{T}(E \cup L)$, and for two terms $t_1, t_2 \in \mathbf{T}(E \cup L)$, $E \cup E_\sigma \models t_1 \simeq t_2$ if and only if $E \cup E_{\sigma'} \models t_1 \simeq t_2$. Grounding σ' is again trivial. For each variable x in $dom(\sigma)$, if S_x contains a ground term t in $\mathbf{T}(E \cup L)$, then $x\sigma' = t$. Otherwise if S_x contains a term in $u \in \mathbf{T}(L)$, for all variables y in S_x, $y\sigma' = u$. If S_x does not contain any term in $\mathbf{T}(E \cup L)$, a variable z in S_x is chosen, and for all variables y in S_x, $y\sigma' = z$. Trivially, $ran(\sigma') \subseteq \mathbf{T}(E \cup L)$.

Now we prove that $E \cup E_{\sigma'} \models L$ if and only if $E \cup E_\sigma \models L$, or further, that for any two terms t, t' in $\mathbf{T}(E \cup L)$, $E \cup E_{\sigma'} \models t \simeq t'$ if and only if $E \cup E_\sigma \models t \simeq t'$. Since $E \cup E_\sigma \models E \cup E_{\sigma'}$, the only non trivial direction is that, if $E \cup E_\sigma \models t \simeq t'$, $E \cup E_{\sigma'} \models t \simeq t'$. This has already be proven in the case of two ground terms. By construction σ and σ' induce the same partition on the variables, so this is true also if t and t' are two variables. If there is a path between t and t' with only congruence edges in the graph for $E \cup E_\sigma$, t and t' are still congruent according to $E \cup E_{\sigma'}$, thanks to the fact that equality between variables and equality between ground terms is preserved. As a corollary, any equality between terms in $\mathbf{T}(E \cup L)$ is preserved.

The substitution σ' is acyclic. Otherwise for some n,

$$E \cup E_\sigma \models x_i \simeq f_i(\ldots, x_{(i+1) \bmod n}, \ldots)$$

($i \in \{1, \ldots n\}$) and also, since σ is ground,

$$E \cup E_\sigma \models x_1 \simeq f_1(\ldots, x_{(2 \bmod n)}, \ldots) \land x_1 \simeq t$$

for some ground term t. Thus $f_1(\ldots, x_{(2 \bmod n)}, \ldots)$ and $t = f_j(\ldots, t', \ldots)$ are congruent, therefore $j = 1$ and $x_{(2 \bmod n)}$ and t' should be equal. By successive application of the same reasoning step, there exists some x_i and a constant a such that $E \cup E_\sigma \models x_i \simeq a \simeq f_i(\ldots, x_{(i+1) \bmod n}, \ldots)$, which contradicts the fact that only one equality contains x_i in E_σ. □

Appendix B Reduction to rigid *E*-unification

Lemma 3 (Reduction to Ridig *E*-unification). *Finding solutions σ for an E-ground (dis)unification problem $E \models L\sigma$ can be reduced to finding substitutions σ s.t. $E^{eq}\sigma \models s\sigma \simeq t\sigma$, in which E^{eq} contains only equations and s, t are terms.*

Proof. First, we can assume that E is closed under entailment w.r.t. disequalities. Indeed, for any set of ground equality literals E, retrieving all the disequalities that it entails can be done polynomially. Let $\top$ be a constant and $f_{\not\simeq}$ a binary function symbol, both not appearing in $E \cup L$. Each disequality $s \not\simeq t$ in $E \cup L$ is replaced with $f_{\not\simeq}(s, t) \simeq \top \land f_{\not\simeq}(t, s) \simeq \top$.

For the transformation to preserve the entailment relation it is necessary to show that the transformed disequalities in L are still entailed by the transformed disequalities in E modulo its equalities. Assume that there is a disequality $s \not\simeq t$ in L s.t. $E \models s\sigma \not\simeq t\sigma$. Since E is closed under entailment w.r.t. disequalities, there is a disequality $s' \not\simeq t'$ or $t' \not\simeq s'$ in E s.t. $E \models s' \simeq s\sigma \land t' \simeq t\sigma$. Considering the transformation of $E \cup L$ to remove disequalities, it is straightforward to check that the entailment

$$E \cup \{f_{\not\simeq}(s', t') \simeq \top, f_{\not\simeq}(t', s') \simeq \top\} \models \{f_{\not\simeq}(s\sigma, t\sigma) \simeq \top, f_{\not\simeq}(t\sigma, s\sigma) \simeq \top\} \cup L$$

continues to hold, as well as that applying the same process to all other disequalities in $E \models L\sigma$ preserves the entailment into the resulting

$$E^{eq} \models s_1\sigma \simeq t_1\sigma \wedge \cdots s_n\sigma \simeq t_n\sigma$$

which, by taking a fresh n-ary function symbol f, can then be transformed into the equivalent $E^{eq} \models f(s_1, \ldots, s_n) \simeq f(t_1, \ldots, t_n)$. Since E^{eq} contains only equations, there is a single equation in the conclusion and the removal of disequalities preserves the entailment relation, E-ground (dis)unification is shown to be an instance of rigid E-unification. $\square$

Appendix C Complexity of *E*-ground (dis)unification

Theorem 5 (E-ground (dis)unification is NP-complete). *Finding solutions for E-ground (dis)unification is NP-complete.*

Proof. E-ground (dis)unification is NP since it can be verified, in polynomial time, with a classic congruence closure procedure, for instance, that a substitution σ solves the entailment problem. The proof of its NP-hardness is done through an encoding of 3-SAT into the entailment.

Let $\mathcal{C}$ be a set of 3-clauses containing literals over a set of propositions $\mathcal{P}$. Let $\top, \bot$ be constants and $\textsc{p}$ a unary operator such that, for any proposition, $\textsc{p}(p) = \top$ and $\textsc{p}(\neg p) = \bot$. For each clause $C \in \mathcal{C}$, let f_C be a ternary function. For each proposition $p \in \mathcal{P}$, let x_p be a variable. The set of equality literals $E \cup L$ is built such that

$$E = \bigcup_{C=\ell_1\vee\ell_2\vee\ell_3\in\mathcal{C}} \left\{ f_C(\textsc{p}(\ell_1), \textsc{p}(\ell_2), \textsc{p}(\ell_3)) \simeq \top \;\middle|\; \begin{array}{l} \textsc{p}(\ell_1) = \top \text{ or } \textsc{p}(\ell_2) = \top \\ \text{or } \textsc{p}(\ell_3) = \top \end{array} \right\}$$

$$L = \bigcup_{C=\ell_1\vee\ell_2\vee\ell_3\in\mathcal{C}} \left\{ f_C(x_{p_1}, x_{p_2}, x_{p_3}) \simeq \top \;\middle|\; \ell_i = p_i \text{ or } \ell_i = \neg p_i,\, i \in \{1..3\} \right\}$$

With this encoding the search for a substitution σ s.t. $E \models L\sigma$ can be seen as determining the satisfiability of $\mathcal{C}$. $\square$

Appendix D Lemmas for correctness

Lemma. Given a computed solution E_σ for an E-ground (dis)unification problem $E \models L\sigma$, each $\sigma_g \in \textsc{Sols}(E_\sigma)$ is an acyclic substitution s.t. $\sigma_g^\star$ is ground and $\mathit{ran}(\sigma_g) \subseteq \mathbf{T}(E \cup L)$.

Proof. The acyclicity of the substitution $\sigma = \{x \mapsto \mathit{rep}(x) \mid x \in \mathrm{FV}(L)\}$ built from E_σ is guaranteed by the occurs check in $\textsc{Assign}$ and the application of *rep* on the remaining branch constraints after each assignment. Thus, by construction, each σ_g is also acyclic and $\sigma_g^\star$ is ground. Case analysis of each rule and the construction of σ_g trivially ensure that $\mathit{ran}(\sigma_g) \subseteq \mathbf{T}(E \cup L)$. $\square$

Lemma (Rules capture entailment conditions). For each rule

$$\frac{E_\sigma \Vdash_E C}{E'_\sigma \Vdash_E C'}\ \mathrm{R}$$

and any ground substitution σ, $E \models (\{C\} \cup E_\sigma)\sigma$ iff $E \models (\{C'\} \cup E'_\sigma)\sigma$.

Proof (sketch).
Direction "if $E \models (\{C\} \cup E_\sigma)\sigma$ then $E \models (\{C'\} \cup E'_\sigma)\sigma$":

Assume $E \models (\{C\} \cup E_\sigma)\sigma$. It has to be shown that, for any rule, its application ensures that $E \models E'_\sigma\sigma$ and $E \models C'\sigma$.

The first condition holds trivially for all rules in the calculus but Assign. Considering the latter, let $C\sigma = C_1\sigma \cup \{x\sigma \simeq s\sigma\}$. Since $E \models (\{C\} \cup E_\sigma)\sigma$ and $E'_\sigma\sigma = E_\sigma\sigma \cup \{x\sigma \simeq s\sigma\}$, then $E \models E'_\sigma\sigma$.

The second condition holds for Split due to entailing a disjunction guaranteeing the entailment of at least one of the disjuncts and for Yield due to $\top$ being trivially always entailed. For Assign, let again $C\sigma = C_1\sigma \cup \{x\sigma \simeq s\sigma\}$. Since $E \models \{x\sigma \simeq s\sigma\}$ and C' is the result of replacing x, and s if it is also a variable, in C_1 by its representative, which is either s or a variable from $[x]_{E'_\sigma}$, then $E \models C'\sigma$.

Considering the rule U_var, by congruence and due to x occurring in $f(\mathbf{u})$, $E \models x\sigma \simeq f(\mathbf{u})\sigma$ implies that there are terms $t, f(\mathbf{t}') \in \mathbf{T}(E)$ s.t. $x\sigma = t$, $f(\mathbf{u})\sigma = f(\mathbf{t})$, $E \models t \simeq f(\mathbf{t})$ and t occurs in $f(\mathbf{t})$ modulo E. Thus in $C'\sigma$ there is one disjunct s.t.

$$E \models x\sigma \simeq t\sigma \wedge u_1\sigma \simeq t_1\sigma \wedge \cdots u_n\sigma \simeq t_n\sigma \wedge C\sigma$$

which is built according to the class $[x\sigma]_E$ and is trivially entailed by E, since $x\sigma = t$, $u_1\sigma = t_1, \ldots, u_n\sigma = t_n$ and $E \models C\sigma$. Proceeding analogously the condition can be shown to also hold for the remaining branching rules.

Direction "if $E \models (\{C'\} \cup E'_\sigma)\sigma$ then $E \models (\{C\} \cup E_\sigma)\sigma$":

Assume $E \models (\{C'\} \cup E'_\sigma)\sigma$. It has to be shown that, for any rule, its application ensures that $E \models E_\sigma\sigma$ and $E \models C\sigma$. Since for all rules $E_\sigma \subseteq E'_\sigma$, the first condition holds trivially.

The second condition holds for Split by the properties of disjunction and for Yield due to its side condition. For Assign, let $C\sigma = C_1\sigma \cup \{x\sigma \simeq s\sigma\}$. C' is the result of replacing in C_1 all occurrences of x, as well as of s if it is also a variable, by its representative according to E_σ. If s is not a variable, this representative is s itself, otherwise is some chosen variable from $[x]_{E'_\sigma}$. Since $\{x\sigma \simeq s\sigma\} \subseteq E'_\sigma\sigma$, whatever ground term σ maps x and s to is congruent to the representative they were replaced by in $C_1\sigma$. Therefore, since $E\sigma \models C'\sigma$, then $E \models C_1\sigma$. Moreover, trivially $E \models \{x\sigma \simeq s\sigma\}$. Thus $E \models C\sigma$.

Considering the rule U_var, there is at least one disjunct in $C'\sigma$ such that, for some $[t] \in E^{\mathrm{CC}}$ and $f(t_1, \ldots, t_n) \in [t]$,

$$E \models x\sigma \simeq t\sigma \wedge u_1\sigma \simeq t_1\sigma \wedge \cdots \wedge u_n\sigma \simeq t_n\sigma$$

By congruence, $E \models f(\mathbf{u})\sigma \simeq f(\mathbf{t})\sigma$. Since $E \models t \simeq f(\mathbf{t})$, $E \models C'\sigma$ and $C\sigma = C'\sigma \cup \{x \simeq f(\mathbf{u})\}\sigma$, it holds that $E \models C\sigma$. Proceeding analogously the condition can be shown to also hold for the remaining branching rules. $\square$