# Congruence Closure with Free Variables (Work in Progress)

Haniel Barbosa and Pascal Fontaine

LORIA–INRIA, Nancy, France
{Haniel.Barbosa, Pascal.Fontaine}@inria.fr

**Abstract.** This paper presents preliminary work on the definition of a general framework for handling quantified formulas in SMT solving. Its focus is on the derivation of instances conflicting with a ground context, redefining the approach introduced in [11]. An enhanced version of the classical congruence closure algorithm, able to handle free variables, is presented.

## 1 Introduction

SMT solvers (see [3] for a general presentation of SMT) are extremely efficient at handling large ground formulas with interpreted symbols, but they still struggle to manage quantified formulas. Quantified first-order logic is best handled with resolution-based theorem proving [10]. Although there are first attempts to unify SMT and resolution [8], the main approach used in SMT is still *instantiation*: quantified formulas are freed from quantifiers and are refuted with the help of decision procedures for ground formulas. Even though such techniques as $E$-matching [6] and model based quantifier instantiation (MBQI) [7] have been used successfully in state-of-the-art solvers, there are still far more instances produced than would actually be needed. Reynolds et al. [11] present an alternative approach: instances are generated such that they are conflicting, by construction, with the ground context produced by the solver. This provides a strong advantage due to its finer instantiation guideline: less instances are needed to prove a formula unsatisfiable, as their experimentation data indicates.

Since their method is restricted to problems in pure first-order logic with equality, it has strong reminiscence of the (non-simultaneous) rigid $E$-unification problem. Unifying two expressions with free variables modulo a set of equations is equivalent, as shown in [12], to finding instances conflicting with a ground context. In this preliminary work we try to exploit this relation while revisiting the technique: defining an enhanced version of the classic congruence closure procedure capable of handling *free variables* unification accordingly. We aim for a better integration of ground conflicting instance generation and $E$-matching techniques within core SMT algorithms (namely the congruence closure decision procedure), with MBQI being used as last resort.

## 2 Notations and basic definitions

We recall here some usual notions of first-order logic. For simplicity, we work in mono-sorted (in contrast to many-sorted) languages. A *first-order language* is a tuple $\mathscr{L} = \langle \mathcal{X}, \mathcal{P}, \mathcal{F} \rangle$ where $\mathcal{X}$, $\mathcal{P}$ and $\mathcal{F}$ are enumerable sets of *variable*, *predicate* and *function symbols*, respectively. Every function and predicate symbol has an arity. Nullary functions and predicates are called *constants* and *propositions*, respectively. *Formulas* and *terms* are generated by

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

in which $x, x_1, \ldots, x_n \in \mathcal{X}$, $p \in \mathcal{P}$ and $f \in \mathcal{F}$. The symbol $\approx$ stands for *equality*. We will mostly work in predicate-free languages. The usual conventions for *disequality*, *existential quantification* and connectives are assumed. In particular we use $\wedge$ for conjunction. The terms in a formula $\varphi$ are denoted by $\mathbf{T}^{\varphi}$. *Atoms* are formulas of the form $t \approx t$ and $p(t, \ldots, t)$. A *literal* is an atom or its negation. A subformula *appears positively (resp. negatively) in* $\varphi$ iff it is under an even (resp. odd) number of negations. A subformula $\forall x_1 \ldots x_n.\psi$ of $\varphi$ is *weakly (resp. strongly) quantified* iff it appears positively (resp. negatively) in $\varphi$. A formula is in *Skolem form* iff it has no strong quantifiers.

Whenever convenient, an enumeration of symbols $s_1, \ldots, s_n$ will be represented as $\mathbf{s}$. Analogously, an enumeration of binary operations $s_1$ *op* $t_1, \ldots, s_n$ *op* $t_n$ is represented as $\mathbf{s}$ *op* $\mathbf{t}$.

Terms and formulas without variables are denoted *ground*. *Free* and *bound* variables are defined in the usual way. A *substitution* is a function from variables to terms such that $\sigma = \{\mathbf{x} \mapsto \mathbf{t}\}$ maps each variable $x_i \in \mathbf{x}$ into the term $t_i \in \mathbf{t}$ and every other variable not in $\mathbf{x}$ to itself. $\varphi\sigma$ (resp. $t\sigma$) denotes the recursive application of $\sigma$ in the structure of the formula (resp. term) in a capture-avoiding way, while not substituting bound variables. The *domain of* $\sigma$ is the set $dom(\sigma) = \{x \mid x \in \mathcal{X} \text{ and } x\sigma \neq x\}$, while the *range of* $\sigma$ is $ran(\sigma) = \{x\sigma \mid x \in dom(\sigma)\}$. $\sigma$ is a *ground substitution* iff every term in $ran(\sigma)$ is ground. The *composition* of two substitutions $\sigma_1$ and $\sigma_2$ is defined such that $\sigma_1 \circ \sigma_2 = \{x \mapsto (x\sigma_2)\sigma_1 \mid x \in \mathcal{X}\}$. It is commutative for ground substitution, that is, everywhere in this text. A formula $\psi_1$ is an *instance* of a formula $\psi_2$ iff there is a substitution $\sigma$ such that $\psi_1 = \psi_2\sigma$.

An *interpretation* is represented as a tuple $\mathcal{M} = \langle \mathcal{D}, \mathcal{I}, \mathcal{V} \rangle$, in which $\mathcal{D}$ is a non-empty *domain*; $\mathcal{I}$ is a function mapping each function symbol $f$ to a function $f^{\mathcal{I}} : \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{D}$ and each predicate symbol $p$ to a predicate $p^{\mathcal{I}} : \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \{\top, \bot\}$; $\mathcal{V}$ is a *valuation* assigning an element of $\mathcal{D}$ to every variable. $\mathcal{M}$ assigns a value in $\mathcal{D}$ to every term $t$, denoted $[\![t]\!]^{\mathcal{M}}$, and a *truth value* $(\top, \bot)$ to every formula $\varphi$, denoted $[\![\varphi]\!]^{\mathcal{M}}$, through the usual recursive definition. $\mathcal{M}$ *satisfies* $\varphi$, written $\mathcal{M} \models \varphi$, iff $[\![\varphi]\!]^{\mathcal{M}} = \top$, in which case $\mathcal{M}$ is a *model* of $\varphi$. $\varphi$ is *satisfiable* iff it has a model. It is *unsatisfiable* otherwise. A set of formulas $\Gamma$ *entails* a set of formulas $\Delta$, written $\Gamma \models \Delta$, iff all interpretations satisfying every $\varphi \in \Gamma$ also satisfy every $\psi \in \Delta$.

An interpretation $\mathcal{M}$ *propositionally (resp. groundly) satisfies* $\varphi$, written $\mathcal{M} \models^{\mathrm{p}} \varphi$ (resp. $\mathcal{M} \models^{\mathrm{g}} \varphi$), iff it is a model of its *propositional (resp. ground)*

*abstraction*, which is a propositional (resp. ground) formula where every non-propositional atom (resp. quantified subformula) is mapped into a fresh propositional symbol. These notions carry out accordingly to other definitions.

## 3 Congruence Closure with Free Variables

Modern SMT solvers handle quantified formulas using instantiation. That is, while checking the satisfiability of a formula $\varphi$ in a theory $\mathcal{T}$, the ground abstraction of the formula is given to the ground SMT solver, which provides a groundly $\mathcal{T}$-satisfiable set of literals. These abstracted literals correspond to (concrete) ground literals $\mathcal{L}$ and quantified[1] formulas $\mathcal{Q}$, with $\mathcal{L} \cup \mathcal{Q} \models \varphi$. If $\mathcal{L} \cup Q$ is $\mathcal{T}$-satisfiable, then so is $\varphi$. This satisfiability check could be done by a model finder. As in [11] we focus here on the problem of finding instances from $\mathcal{Q}$ that groundly refute $\mathcal{L}$. Repeatedly adding such instances conjunctively to the original formula, combined with MBQI [7], provides a practical and powerful procedure to deal with unsatisfiable formulas in SMT. We believe that better integrating ground conflicting instance generation [11] and MBQI within the core SMT algorithm will generate new heuristics and ideas for a layered approach to quantifier handling.

It is assumed, for simplicity, that $\mathcal{L}$ contains only equality literals and that each formula in $\mathcal{Q}$ is of the form $\forall \mathbf{x}.\psi$, where $\psi$ is quantifier-free and consists of a single clause of non-ground equality literals. We further assume that $\mathcal{T}$ is the empty theory, that is, we work in pure first-order logic with equality. Since any unsatisfiable formula in pure first-order logic is also unsatisfiable in any theory, the techniques here can also be seen as an incomplete algorithm for SMT, whatever the background theories.

*Problem description* Given some formula $\forall \mathbf{x}.\psi \in \mathcal{Q}$, if there exists a substitution $\sigma$ such that $\mathcal{L} \models \neg\psi\sigma$, then, by Lemma 1, there is a ground substitution $\sigma'$ such that $ran(\sigma') \subseteq \mathbf{T}^{\mathcal{L}}$. Such a substitution refutes $\mathcal{L}$ and is called *ground conflicting*.

As shown in [12], computing a ground conflicting substitution is equivalent to solving a non-simultaneous rigid $E$-unification problem. Therefore it becomes the problem of finding a conjunctive set $\Sigma$ of equalities $x \approx t$ ($x \in \mathrm{FV}(\psi)$ and $t \in \mathbf{T}^{\mathcal{L}}$) such that $\mathcal{L} \models \neg\psi \wedge \Sigma$, each variable occurring at most once in $\Sigma$. Since this is a ground problem, classical SMT solving tools, i.e., *Congruence Closure* (see e.g. [9]) can be adapted to solve it: unification of free variables must be handled, associating variables in $\mathrm{FV}(\psi)$ to ground terms already occurring in $\mathcal{L}$.

**Lemma 1.** *Consider a ground formula $\varphi$ and a formula $\psi$ with free variables. If there exists a substitution $\sigma$ such that $\varphi \models \psi\sigma$, then there is a ground substitution $\sigma'$ such that $\varphi \models \psi\sigma'$ and* $\mathrm{ran}(\sigma') \subseteq \mathbf{T}^{\varphi}$.

[1] Assuming $\varphi$ is Skolemized, we can safely assume $\mathcal{Q}$ only contains weakly quantified formulas.

*Proof.* Let $\mathcal{M}$ be model of $\varphi$ such that its domain elements are the interpretation of terms in $\mathbf{T}^{\varphi}$. Since $\varphi \models \psi\sigma$, $\mathcal{M}$ is a also a model of $\psi\sigma$. Therefore, for each variable $x \in dom(\sigma)$ there is a term $t \in \mathbf{T}^{\varphi}$ such that $[\![x\sigma]\!]^{\mathcal{M}} = [\![t]\!]^{\mathcal{M}}$. Thus, a substitution $\sigma' = \{x \mapsto t \mid x \in dom(\sigma), t \in \mathbf{T}^{\varphi}, [\![x\sigma]\!]^{\mathcal{M}} = [\![t]\!]^{\mathcal{M}}\}$ fulfills the desired condition.

**Algorithm CCFV**

The CCFV procedure shown in Figure 1 derives, if any, a ground substitution $\sigma$ such that $\mathcal{L} \wedge \psi\sigma$ is unsatisfiable. It computes a sequence of substitutions $\sigma_0, \ldots, \sigma_k$ such that, for $\neg\psi = l_1 \wedge \cdots \wedge l_k$,

$$\sigma_0 = \varnothing;\ \sigma_{i-1} \subseteq \sigma_i \text{ and } \mathcal{L} \models l_i\sigma_i$$

which guarantees that $\mathcal{L} \models \neg\psi\sigma_k$.

*Example 1.* Consider a set of literals $\mathcal{L} = \{f(c) \approx a,\ f(a) \approx b,\ f(a) \not\approx f(b)\}$ and a quantified formula $\forall x_1, x_2.\ (f(x_1) \not\approx a \vee f(x_2) \approx b)$. Successively evaluating $\neg\psi = (f(x_1) \approx a \wedge f(x_2) \not\approx b)$ yields $\sigma_1 = \{x_1 \mapsto c\}$ such that $\mathcal{L} \models (f(x_1) \approx a)\sigma_1$ and $\sigma_2 = \{x_1 \mapsto c, x_2 \mapsto b\}$ such that $\mathcal{L} \models (f(x_2) \not\approx b)\sigma_2$. Therefore $\sigma = \sigma_2$ is a ground conflicting substitution, since $\mathcal{L} \wedge \psi\sigma$ is groundly unsatisfiable.

*Preliminaries* $\mathfrak{C}$ and $\mathfrak{D}$ are initially, respectively, the set of equalities and of disequalities in $\mathcal{L}$.

Given a term $s \in \mathbf{T}^{\mathcal{L}\cup\{\psi\}}$, $[s]$ denotes the *congruence class* of $s$ in the partition of $\mathbf{T}^{\mathcal{L}\cup\{\psi\}}$ induced by $\mathfrak{C}$, i.e., $[s] = \{t \mid t \in \mathbf{T}^{\mathcal{L}\cup\{\psi\}},\ \mathfrak{C} \models s \approx t\}$. Operations on substitutions are performed modulo the current partition, so that, e.g., $\theta \subseteq_{\mathfrak{C}} \sigma$ iff, for every $x \in dom(\theta)$, $\mathfrak{C} \models x\sigma \approx x\theta$, rather than requiring $x\sigma = x\theta$.

$\Delta_{\mathbf{x}}$ denotes the set of *unfeasible substitutions*, i.e., $\delta \in_{\mathfrak{C}} \Delta_{\mathbf{x}}$ iff there is no $\sigma$ such that $\delta \subseteq_{\mathfrak{C}} \sigma$ and $\mathcal{L} \models \neg\psi\sigma$. It is initially empty.

SEL denotes a function mapping variables to themselves or ground terms, such that $\text{SEL}(x) = x$ iff $[x]$ contains no ground terms, otherwise $\text{SEL}(x)$ is some ground term $t \in [x]$.

*Algorithm* The *current instantiation* in $\mathfrak{C}$ is obtained depending on the congruence classes of $\mathbf{x}$ containing ground terms or not. Thus $\{x \mapsto \text{SEL}(x) \mid x \in \mathbf{x}\}$ denotes the current instantiation in $\mathfrak{C}$.

For each $l \in \neg\psi$, the procedure HANDLE checks its consistency w.r.t. $\mathfrak{C} \cup \mathfrak{D}$. If they are incompatible then the current instantiation is unfeasible, which triggers its addition to $\Delta_{\mathbf{x}}$ and backtracking. Otherwise it tries to extend the current instantiation to some $\sigma$, not in $\Delta_{\mathbf{x}}$, for which $\mathcal{L} \models l\sigma$. If it fails to do so, then again the current instantiation is unfeasible. If every literal in $\neg\psi$ is asserted successfully, the current instantiation represents a conflicting substitution and is outputed by CCFV. On the other hand, if $\varnothing$ is added to $\Delta_{\mathbf{x}}$, there is no $\sigma$ such that $\mathcal{L} \models \neg\psi\sigma$. So the procedure terminates without producing a ground conflicting substitution.

```
   proc CCFV(L, ψ)
1  |  C ← {s ≈ t | s ≈ t ∈ L};   D ← {s ≉ t | s ≉ t ∈ L};   Δ_x ← ∅      // Init
2  |  foreach l ∈ ¬ψ do
3  |  |  if not(HANDLE(C, D, Δ_x, l)) then
4  |  |  |  Δ_x ← Δ_x ∪ {{x ↦ SEL(x) | x ∈ x}}
5  |  |  |  if ∅ ∈ Δ_x then return ∅                  // No σ s.t. L ⊨ ¬ψσ
6  |  |  |  RESET(C, D, ¬ψ)                              // Backtracking
7  |  return {x ↦ SEL(x) | x ∈ x}                        // L ⊨ ¬ψσ

   proc HANDLE(C, D, Δ_x, l)
8  |  match l :
9  |  |  u ≈ v :
10 |  |  |  if C ∪ D ⊨ u ≉ v then return ⊥          // Checks consistency
11 |  |  |  C ← C ∪ {u ≈ v}                            // Updates C ∪ D
12 |  |  u ≉ v :
13 |  |  |  if C ⊨ u ≈ v then return ⊥
14 |  |  |  D ← D ∪ {u ≉ v}
15 |  δ ← {x ↦ SEL(x) | x ∈ X}                       // Current instantiation
16 |  Λ ← (UNIFY δ l) \_C Δ_x                  // L ⊨ lσ, for every σ ∈ Λ
17 |  if Λ ≠ ∅ then
18 |  |  let σ ∈ Λ in
19 |  |  |  C ← C ∪ ⋃_{x∈dom(σ)} {x ≈ xσ}
20 |  |  return ⊤
21 |  return ⊥
```

**Fig. 1:** Congruence Closure with Free Variables.

Backtracking is handled by the procedure RESET, which simply resets $\mathfrak{C}$ and $\mathfrak{D}$ to their initial states, while the literals in $\neg\psi$ are marked to be reevaluated by the loop. This is a naïve approach, for simplicity. Smarter backtracks are achievable through careful analysis of the dependencies for the inconsistency found.

The function UNIFY in Figure 2 takes a set of literals $\mathcal{L}$, a substitution $\theta$ and a literal $l\theta$ as input, computing the set of substitutions $\sigma$ such that $\theta \subseteq_{\mathfrak{C}} \sigma$ and $\mathcal{L} \models l\sigma$. It is invoked with the current instantiation and the literal being asserted. If the resulting set is empty, a failure is reported, showing the unfeasibility of the current instantiation. Otherwise one of its elements is chosen and the current instantiation is updated accordingly.

*Computing feasible instantiations* Adapting the *recursive descent* E-*unification* algorithm in [1], the function UNIFY computes the set of $(\mathfrak{C}, \mathfrak{D})$-unifiers of given equality literals $u \approx v$ and $u \not\approx v$, respectively, extending an initial substitution $\sigma$. The resulting unifiers solve the $E$-unification problem for $\mathcal{L}$ and the given literal. In the presentation, $u$ and $f(\mathbf{u})$ represent non-ground terms, $v$ and $f(\mathbf{v})$ terms

that may or may not be ground and $t$ and $f(\mathbf{t})$ ground terms, with subscripts or not.

The *merging* of two ground substitutions $\sigma_1$ and $\sigma$ is defined by

$$\sigma_1 \oplus \sigma_2 = \begin{cases} \sigma_1 \circ \sigma_2 & \textbf{if } x \in (dom(\sigma_1) \cap dom(\sigma_2)) \text{ only if } \mathfrak{C} \models x\sigma_1 \approx x\sigma_2 \\ \varnothing & \textbf{otherwise} \end{cases}$$

The merging of two sets of ground substitutions is the result of pairwisely merging their members. If either set is empty, so is their merging.

*Example 2.* Given substitutions $\sigma_1 = \{x \mapsto a\}$ and $\sigma_2 = \{x \mapsto b\}$, their merging is the empty set unless $\mathfrak{C} \models a \approx b$.

$$\begin{aligned}
&(1)\ \textsc{Unify}\ \mathcal{L}\ \sigma\ t_1 \approx t_2 &&= \begin{cases} \{\sigma\} & \textbf{if } \mathfrak{C} \models t_1 \approx t_2 \\ \varnothing & \textbf{otherwise} \end{cases} \\
&(2)\ \textsc{Unify}\ \mathcal{L}\ \sigma\ x \approx t &&= \{\{x \mapsto t\} \circ \sigma\} \\
&(3)\ \textsc{Unify}\ \mathcal{L}\ \sigma\ f(\mathbf{u}) \approx t &&= \bigcup\nolimits_{f(\mathbf{t}) \in [t]} (\textsc{Unify}\ \mathcal{L}\ \sigma\ u_1 \approx t_1) \oplus \cdots \oplus (\textsc{Unify}\ \mathcal{L}\ \sigma\ u_n \approx t_n) \\
&(4)\ \textsc{Unify}\ \mathcal{L}\ \sigma\ u \approx v &&= \bigcup\nolimits_{t \in \mathbf{T}^{\mathcal{L}}} \bigcup\nolimits_{\theta \in \textsc{Unify}\ \mathcal{L}\ \varnothing\ v \approx t} \textsc{Unify}\ \mathcal{L}\ \sigma\theta\ u\theta \approx v\theta \\
&(5)\ \textsc{Unify}\ \mathcal{L}\ \sigma\ u \not\approx t &&= \bigcup\nolimits_{t_1 \not\approx t_2 \in \mathfrak{D},\, t_1 \in [t],\, t' \in [t_2]} \textsc{Unify}\ \mathcal{L}\ \sigma\ u \approx t' \\
&(6)\ \textsc{Unify}\ \mathcal{L}\ \sigma\ u \not\approx v &&= \bigcup\nolimits_{t \in \mathbf{T}^{\mathcal{L}}} \bigcup\nolimits_{\theta \in \textsc{Unify}\ \mathcal{L}\ \varnothing\ u \approx t} \textsc{Unify}\ \mathcal{L}\ \sigma\theta\ u\theta \not\approx v\theta
\end{aligned}$$

**Fig. 2.** UNIFY function

Rules are applied through pattern matching on the given literal, following the presented order. Cases (5) and (6) handle the unification with the disequalities in $\mathfrak{D}$, which ultimately also rely on the partition induced by $\mathfrak{C}$. Cases (1-4) make the computation of the feasible substitution by recursively going through the terms in the equality, eventually matching modulo $\mathfrak{C}$. The crucial test occurs in (1), when the substitution computed stands only if the equality over ground terms holds in $\mathfrak{C}$. There is no need to check if $x \in dom(\sigma)$ in (2), since every free variable in the given literals is not in the domain of the given substitution.

Many optimizations may be devised to improve UNIFY, (if $u$ is a function term $f(\mathbf{u})$, one does not need to go through every term in $\mathbf{T}^{\mathcal{L}}$, it is sufficient to check any terms of the form $f(\mathbf{t})$; and so on) but the above definition is sufficient for generating every unifier solving the given $E$-unification problem, as established in Lemma 2.

**Lemma 2.** *Consider a set of equality literals $\mathcal{L}$ and a substitution $\theta$. Let $\mathfrak{C}$ and $\mathfrak{D}$ be the sets of equalities and disequalities in $\mathcal{L}$, respectively, the former inducing*

*a partition of* $\mathbf{T}^{\mathcal{L}}$ *in congruence classes. Then, given literals* $u \approx v$, $u \not\approx v$,

$$\textsc{Unify}\ \mathcal{L}\ \theta\ u\theta \approx v\theta = \{\sigma \mid \theta \subseteq \sigma, \mathfrak{C} \models u\sigma \approx v\sigma\}$$
$$\textsc{Unify}\ \mathcal{L}\ \theta\ u\theta \not\approx v\theta = \{\sigma \mid \theta \subseteq \sigma, \mathfrak{C} \cup \mathfrak{D} \models u\sigma \not\approx v\sigma\}$$

*Proof (sketch).* Induction on the structure of the literals (rules 1-4 for $\mathfrak{C}$-unifiers; 5-6 for $\mathfrak{D}$-unifiers).

*Example 3.* Let CCFV be applied on the following input:

$$\mathfrak{C} = \{a \approx f(c), b \approx f(a)\} \qquad \Delta_{\mathbf{x}} = \varnothing$$
$$\mathfrak{D} = \{f(a) \not\approx f(b)\} \qquad \neg\psi = f(x_1) \approx a \wedge f(x_2) \not\approx b.$$

For $l = f(x_1) \approx a$, $\textsc{Handle}(\mathfrak{C}, \mathfrak{D}, \Delta_{\mathbf{x}}, l)$ adds $l$ to $\mathfrak{C}$, since it is consistent with $\mathfrak{C} \cup \mathfrak{D}$. The current instantiation is $\varnothing$, so $\textsc{Unify}\ \mathcal{L}\ \varnothing\ f(x_1) \approx a$ is invoked. According to its definition,

$$\begin{aligned}
\textsc{Unify}\ \mathcal{L}\ \varnothing\ f(x_1) \approx a &= \textsc{Unify}\ \mathcal{L}\ \varnothing\ f(x_1) \approx f(c) \\
&= \textsc{Unify}\ \mathcal{L}\ \varnothing\ x_1 \approx c && \text{[Rule 3]} \\
&= \{\{x_1 \mapsto c\}\} && \text{[Rule 2]}
\end{aligned}$$

since $f(c)$ is the only term in $[a]$ unifiable with $f(x_1)$: for every other $t \in [a]$, $\textsc{Unify}\ \mathcal{L}\ \varnothing\ f(x_1) \approx t = \varnothing$. Since $\{x_1 \mapsto c\}$ is not in $\Delta_{\mathbf{x}}$, it is a feasible substitution, triggering the addition of $\{x_1 \approx c\}$ to $\mathfrak{C}$ and the successful termination of this invocation of Handle.

For $l = f(x_2) \not\approx b$, Handle adds $l$ to $\mathfrak{D}$ (no inconsistency) and invokes $\textsc{Unify}\ \mathcal{L}\ \delta\ f(x_2) \not\approx b$, with $\delta = \{x_1 \mapsto c\}$ as the current instantiation. Thus,

$$\begin{aligned}
\textsc{Unify}\ \mathcal{L}\ \delta\ f(x_2) \not\approx b &= \textsc{Unify}\ \mathcal{L}\ \delta\ f(x_2) \approx f(b) && \text{[Rule 5]} \\
&= \textsc{Unify}\ \mathcal{L}\ \delta\ x_2 \approx b && \text{[Rule 3]} \\
&= \{\{x_2 \mapsto b\} \circ \delta\} && \text{[Rule 2]} \\
&= \{\{x_1 \mapsto c, x_2 \mapsto b\}\} && \text{[Composition]}
\end{aligned}$$

since $f(a) \not\approx f(b)$, the sole disequality in $\mathfrak{D}$, is such that $f(a) \in [b]$ and $f(b)$ is unifiable with $f(x_2)$, deriving $\{x_2 \mapsto b\} \circ \delta$. Since $\{x_1 \mapsto c, x_2 \mapsto b\}$ is not in $\Delta_{\mathbf{x}}$, $\{x_2 \approx b\}$ is added to $\mathfrak{C}$.

With no more literals to process in $\neg\psi$, the current instantiation $\sigma = \{x_1 \mapsto c, x_2 \mapsto b\}$ is outputed. It is a ground conflicting substitution for $\mathcal{L} \wedge \psi$.

The correctness of CCFV is established in Theorem 1.

**Theorem 1.** *Consider a ground formula* $\mathcal{L}$ *and a formula* $\psi$ *with free variables. If there are ground conflicting substitutions for* $\mathcal{L}$ *and* $\psi$*, then there exists a ground substitution* $\sigma$ *such that* $\sigma = \mathrm{CCFV}(\mathcal{L}, \psi)$, $\mathrm{ran}(\sigma) \subseteq \mathbf{T}^{\mathcal{L}}$ *and* $\mathcal{L} \models \neg\psi\sigma$.

*Proof (sketch).* Relying on Lemma 2, the computation of ground conflicting substitutions by CCFV can be proven through induction on the structure of $\psi$.

## 4 Future work

Since completeness is frequently lost for theories more expressive than the empty one, handling quantifiers is essentially an effort of having efficient but incomplete techniques that, for some fragments, allow decision procedures, but often not.

We want to define a framework encompassing not only the derivation of ground conflicting but of *model conflicting instances*, that is, instances refuting a ground model extended into a candidate model for the original formula. This would amount to extend CCFV to handle MBQI, an effective approach for quantifiers in SMT. Furthermore, it is essential to include theory reasoning in these processes.

We are starting to integrate the CCFV algorithm within the SMT solver veriT [5], in order to evaluate and improve our approach of ground conflicting instance generation [11]. We believe CCFV is also strongly related to approaches like bounded simultaneous rigid $E$-unification [2]. It is however not clear how to reconcile both approaches since CCFV considers only one quantified formula at a time.

While extending CCFV for deriving substitutions conflicting modulo theories, the foremost theory to consider will be *LIA*. We will investigate techniques such as *Hierarchic Theorem Proving* [4], which obtains an effective procedure for handling quantified formulas with interpreted terms. To do so they combine ground and resolution-based reasoning, besides a model refinement technique with similarities to MBQI.

## References


1. F. Baader and J. H. Siekmann. Unification theory. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume2, Deduction Methodologies*, pages 41–126. 1994.
2. P. Backeman and P. Rümmer. Theorem proving with bounded rigid E-unification. (To appear).
3. C. Barrett, R. Sebastiani, S. A. 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, Feb. 2009.
4. P. Baumgartner, J. Bax, and U. Waldmann. Finite Quantification in Hierarchic Theorem Proving. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Automated Reasoning*, volume 8562 of *Lecture Notes in Computer Science*, pages 152–167. Springer International Publishing, 2014.
5. 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-22)*, 2009.
6. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Laboratories Palo Alto, 2003.
7. Y. Ge and L. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In A. Bouajjani and O. Maler, editors, *CAV 2009*, volume 5643 of *LNCS*, pages 306–320. Springer, 2009.

8. L. Moura and N. Bjørner. Engineering DPLL(T) + saturation. In *IJCAR '08: Proceedings of the 4th international joint conference on Automated Reasoning*, pages 475–490, Berlin, Heidelberg, 2008. Springer-Verlag.
9. 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.
10. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science B.V., 2001.
11. A. Reynolds, C. Tinelli, and L. M. de Moura. Finding conflicting instances of quantified formulas in SMT. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, pages 195–202, 2014.
12. A. Tiwari, L. Bachmair, and H. Rueß. Rigid e-unification revisited. In D. McAllester, editor, *Automated Deduction CADE-17*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 220–234, Pittsburgh, PA, jun 2000. Springer-Verlag.