Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 225 (2009) 405–420
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

EufDpll - A Tool to Check Satisfiability of Equality Logic Formulas

Olga Tveretina [1]

Institute for Computing and Information Sciences,
Radboud University,
Nijmegen, The Netherlands

Wieger Wesselink [2]

Department of Computer Science
TU Eindhoven
Eindhoven, The Netherlands

Abstract

Decision procedures for subsets of First-Order Logic form the core of many verification tools. Applications include hardware and software verification. The logic of Equality with Uninterpreted Functions (EUF) is a decidable subset of First-Order Logic. The EUF logic and its extensions have been applied for proving equivalence between systems. We present a branch and bound decision procedure for EUF logic based on the generalisation of the Davis-Putnam-Loveland-Logemann procedure (EUF-DPLL). EufDpll is a tool to check satisfiability of EUF formulas based on this procedure.

Keywords: equality logic with uninterpreted functions, satisfiability, DPLL procedure.

1 Introduction

Equality logic with uninterpreted functions (EUF) is a major decidable theory used in verification of infinite-state systems [9]. The functions are termed "uninterpreted" because the only thing we know about them is that two applications of a function to the same arguments will produce the same value. An EUF formula is a boolean formula over atoms that are equalities between terms. In this logic, formulas have truth values while terms have values from some domain. For example, the formula: $f(x) \not\approx f(z) \wedge x \approx y \wedge y \approx z$ is unsatisfiable.

[1] Email: o.tveretina@cs.ru.nl
[2] Email: j.w.wesselink@tue.nl

doi:10.1016/j.entcs.2008.12.089

The GDPLL procedure [2] is a generalisation of the well-known DPLL procedure [6] which was introduced in the early 60s as a proof procedure for first-order logic. The GDPLL procedure is defined in terms of four basic operations (REDUCE, Eligible, SatCriterion and Filter), that have to be filled in for a particular logic. An original DPLL procedure is an instance of GDPLL in case of propositional logic.

The satisfiability problem for EUF logic naturally fits into the GDPLL framework. In this paper we provide an algorithm for this logic which is an instance of GDPLL. Since the algorithm is an instance of GDPLL, we have to check its soundness and completeness by verifying the conditions mentioned in Section 3.

EufDpll is a tool to check satisfiability of formulas in the logic of equality with uninterpreted functions. It is based on the presented theoretical framework. As the programming language was used $C++$ and the ATerm library. The ATerm implementation is based on maximal subterm sharing and automatic garbage collection. Therefore, syntactical identity of terms can be checked in constant time.

1.1 Applications

Testing and verification are the bottleneck of development of complex systems. This applies, in particular, to hardware and software systems. Recently theorem provers were used to verify a pipelined microprocessor. The method proposed by Burch and Dill [5] greatly enhanced verification techniques. The state of a register at any point in the computation can be represented by a symbolic term. Uninterpreted functions can be used to abstract blocks of combinational logic, for example ALU, as black boxes. Uninterpreted functions without arguments are considered as term variable and can be used to abstract constant values that have special semantic meaning, e.g. the data value 0. So the logic of equality with uninterpreted functions provides a means of abstracting the manipulation of data by a processor when verifying the correctness of its control logic.

The behaviour of software is much more complex than that of hardware due to the potentially enormous state space of a program. The development of powerful and complex software systems requires more sophisticated methods than traditional techniques to ensure functional correctness of the code. Software systems grow in scale and functionality. As a result of increasing complexity, the likelihood of errors is much greater. Hence, formal verification has become an increasingly important technique to establish the correctness of designs. Reasoning precisely about program operators is in general undecidable. A common practice is to model $n-$ary operators as an uninterpreted function under the theory of equality. EUF-logic was used for translation validation [13,12,14,15], i.e checking the correctness of a compiler's translation by verifying the equivalence between the source and target codes.

In general, this type of logic is mainly used for proving the equivalence between systems. The method has two phases: the first phase consists of the construction of a logical formula which is valid if and only if the implementation is correct with respect to the specification. While verifying the equivalence between two formulas, it is possible to replace all functions, except the equality sign and propositional operators, with uninterpreted functions. During the second phase a decision procedure

checks validity of the formula. The validity problem for this logic is decidable.

1.2 Related work

In the past years, various procedures for checking the satisfiability of such formulas have been suggested. Barrett et al. [3] proposed a decision procedure based on computing the congruence closure in combination with case splitting.

In [1] Ackermann showed that the problem of deciding the validity of an EUF formula can be reduced to checking the satisfiability of an equality formula. Many current approaches [7,4] use a transformation of an EUF formula into the equality logic formula. Then the equality logic formula can be transformed into a propositional one and a standard satisfiability checker can be applied. Goel et al. [7] and Bryant et al. [4] reduced an equality formula to a propositional one by adding transitivity constraints. In this approach it is analyzed which transitivity properties may be relevant.

A different approach is called range allocation [13,15]. In this approach a formula structure is analyzed to define a small domain for each variable. Then a standard BDD-based tool is used to check satisfiability of the formula under the domain.

Another approach is given in [8]. This approach is based on BDD computation, with some extra rules for dealing with transitivity. Unfortunately, the unicity of the reduced BDDs is lost.

An approach based on the DPLL procedure for EUF is introduced in [10,11]. The proposed DPLL procedure calls the congruence closure module for positive equations.

2 Basic definitions and preliminaries

2.1 Syntax

EUF formulas can be seen as propositional combinations of equalities between terms. The terms are defined recursively as following.

Definition 2.1 (**Terms, subterms**)

- Given a signature $\Sigma = (\mathsf{Fun}, \mathsf{ar})$, the set $\mathsf{Term}(\Sigma)$ of *terms*, or for simplicity Term, over Σ is defined recursively: for $n \geq 0$, $f(t_1, \ldots, t_n)$ is a term if $t_1, \ldots, t_n$ are terms, $f \in \mathsf{Fun}$, and $\mathsf{ar}(f) = n$.
- We use the lower case letters s, t, and u to denote terms.
- For each term $f(t_1, \ldots, t_n) \in \mathsf{Term}$, $n \geq 0$, the set $\mathsf{SubTerm}(f(t_1, \ldots, t_n))$ of *subterms* of $f(t_1, \ldots, t_n)$ is defined recursively:

$$\mathsf{SubTerm}(f(t_1, \ldots, t_n)) = \{f(t_1, \ldots, t_n)\} \cup \bigcup_{i=1}^{n} \mathsf{SubTerm}(t_i).$$

Formulas are defined recursively as follows.

Definition 2.2 An EUF formula is defined as follows:

formula:= formula $\vee$ formula | $\neg$ formula | atom

atom:= term$\approx$term

term:= variable | function (list of terms),

where variables are defined over some (possibly infinite) domain.

In the following by an EUF-CNF we mean an EUF formula in conjunctive normal form. The set of all EUF-CNFs is denoted by EUF-Cnf. An EUF-CNF is a conjuction of clauses, where each clause is a disjunction of literals. We will represent these conjunctions and disjunctions using set notation. In the remainder literals $s \approx t$ and $s \not\approx t$ are considered to be unordered, meaning that they are equivalent to $t \approx s$ and $t \not\approx s$ respectively.

2.2 Semantics

Let At be a set of atoms.

We define an *interpretation* as a function

$$\mathsf{I} : \mathsf{At} \to \{\mathsf{true}, \mathsf{false}\}.$$

A literal l is true in I iff either l is an atom a and $\mathsf{I}(a) = \mathsf{true}$ or l is a negated atom $\neg a$ and $\mathsf{I}(a) = \mathsf{false}$. We write $\mathsf{I} \models l$, if a literal l is true in I.

We define an E-interpretation as one satisfying the following conditions.

- $\mathsf{I} \models t \approx t$;
- if $\mathsf{I} \models s \approx t$ then $\mathsf{I} \models t \approx s$;
- if $\mathsf{I} \models s \approx u$ and $\mathsf{I} \models u \approx t$ then $\mathsf{I} \models s \approx t$;
- if $\mathsf{I} \models s_i \approx t_i$ for all $i \in \{1, \ldots, n\}$ then $\mathsf{I} \models f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n)$.

We write $\mathsf{I} \models \phi$ if a formula ϕ is true in I.

Definition 2.3 A formula ϕ is called satisfiable if $\mathsf{I} \models \phi$ for some E-interpretation I. Otherwise ϕ is called unsatisfiable. By definition the empty clause $\perp$ is unsatisfiable.

We will use throughout the paper the following notations. Let $t \notin \mathsf{SubTerm}(s)$. Then $\phi[t := s]$ denotes a formula ϕ that is obtained from ϕ by substituting recursively occurrences of the term t by the term s till no occurrences of t are left.

Example 2.4 Let us consider $\phi = \{\{f(f(x)) \approx y\}, \{x \approx g(y)\}\}$.
Then $\phi[f(x) := x] = \{\{x \approx y\}, \{x \approx g(y)\}\}$.

We define $\phi|_l = \{C - \{\neg l\} |\ C \in \phi, l \notin C\}$.

Example 2.5 Let us consider $\phi = \{\{x \approx f(y), z \approx g(z)\}, \{x \not\approx f(y), y \approx g(z)\}\}$.
Then $\phi|_{x \approx f(y)} = \{\{y \approx g(z)\}\}$.

In the following by $\mathsf{Lit}(\phi)$ and by $\mathsf{Term}(\phi)$ we mean, correspondingly the set of literals contained in ϕ and the set of terms in ϕ.

3 Generalization of DPLL

The DPLL procedure, due to Davis, Putnam, Logemann, and Loveland, is the basis of some of the most successful propositional satisfiability solvers. The original DPLL procedure was developed as a proof-procedure for first-order logic. It has been used so far almost exclusively for propositional logic because of its highly inefficient treatment of quantifiers. In this chapter we sketch the basic ideas of GDPLL, the general version of the DPLL procedure. For a full description the reader is referred to [2].

The DPLL algorithm is a complete, backtracking-based algorithm for deciding the satisfiability of propositional logic formulas in conjunctive normal form. It consists of the following three rules: the unit clause rule, the splitting rule, and the pure literal rule. These rules reduce a formula according to some criteria. Therefore, a function REDUCE which performs all rules for formula reduction is assumed. Like DPLL, GDPLL has a splitting rule, which carries out a case analysis with respect to an atom a.

As it is defined in [2], we assume a function $\mathsf{REDUCE} : \mathsf{Cnf} \to \mathsf{Cnf}$ where Cnf denotes the set of formulas in Conjunctive Normal Form. We define the set $\mathsf{R} = \{\varphi \in \mathsf{REDUCE}(\mathsf{Cnf}) \mid \bot \notin \varphi\}$.

In the following we also assume functions

- $\mathsf{Eligible} : \mathsf{R} \to \mathsf{At}$,
- $\mathsf{SatCriterion} : \mathsf{R} \to \{\mathsf{true}, \mathsf{false}\}$,
- Filter, where $\mathsf{Filter}(\varphi, a)$ is defined for $\varphi \in \mathsf{R}$ and $a \in \mathsf{Eligible}(\varphi)$.

In [2] the following requirements on the above functions are introduced: for all $\psi \in \mathsf{Cnf}$, for all $\varphi \in \mathsf{R}$, and for all $a \in \mathsf{Eligible}(\varphi)$ the functions should satisfy the following properties.

(i) $\mathsf{REDUCE}(\psi)$ is satisfiable iff ψ is satisfiable,

(ii) φ is satisfiable iff at least one of $\mathsf{Filter}(\varphi, a)$ or $\mathsf{Filter}(\varphi, \neg a)$ is satisfiable,

(iii) $\mathsf{REDUCE}(\mathsf{Filter}(\varphi, a)) \prec \varphi$ and $\mathsf{REDUCE}(\mathsf{Filter}(\varphi, \neg a)) \prec \varphi$, for some well-founded strict order $\prec$ on $\mathsf{REDUCE}(\mathsf{Cnf})$.

(iv) if $\mathsf{SatCriterion}(\varphi) = \mathsf{true}$ then φ is satisfiable,

(v) if $\mathsf{SatCriterion}(\varphi) = \mathsf{false}$ then $\mathsf{Eligible}(\varphi) \neq \emptyset$.

The algorithm is represented in Figure 1. The procedure takes as an input $\varphi \in \mathsf{Cnf}$. GDPLL proceeds until either the function SatCriterion has returned true for at least one branch, or the empty clause has been derived for all branches. Respectively, either SAT or UNSAT is returned.

4 The reduction rules

The function REDUCE is defined by means of a set of reduction rules, that can be applied in any order. In this section we define reduction rules for EUF-CNFs.

```
GDPLL(φ) : {SAT, UNSAT} =
    begin
      φ := REDUCE(φ);
      if (⊥ ∈ φ) then return UNSAT;
      if (SatCriterion(φ)) then return SAT;
      choose a ∈ Eligible(φ);
      if GDPLL(Filter(φ, a)) = SAT then return SAT;
      if GDPLL(Filter(φ, ¬a)) = SAT then return SAT;
      return UNSAT;
    end;
```

Fig. 1. The GDPLL procedure

Starting with an arbitrary CNF, we can simplify all clauses contained in it.

Definition 4.1 (**Simplified clauses**)

- Suppose C is a clause. By $C\downarrow$ we mean the normal form obtained from C after applying the following simplification rule.
$$C \to C\backslash\{t \not\approx t\} \text{ if for some term } t,\ (t \not\approx t) \in C.$$
- A clause C is called simplified if $C = C\downarrow$.

Given a CNF, we can simplify it by repeatedly applying the following simplification rules.

Definition 4.2 (**Simplified CNFs**)

- Suppose φ is a CNF. By $\varphi\downarrow$ we mean the normal form of φ after applying the following rules.
  · $C \to C\downarrow$ for some clause $C \in \varphi$.
  · $\varphi \to \varphi\backslash\{C\}$ for a clause $C \in \varphi$ if for some term t, $(t \approx t) \in C$.
- A CNF φ is called simplified if $\varphi = \varphi\downarrow$.

Now we can define a system of reduction rules. Starting with an arbitrary CNF, we can transform it by repeatedly applying the reduction rules.

We will use the notation $\uplus$ for *disjoint union*, i.e. when we write $\varphi \uplus \psi$ we are referring to the union $\varphi \cup \psi$ assuming that $\varphi \cap \psi = \emptyset$.

Definition 4.3 (**Reduction rules on CNFs**) We define a reduction system for EUF-CNFs EUF-REDUCE as follows.

(i) $\{\{s \approx t\}\} \uplus \varphi \to \{\{s \approx t\}\} \uplus \varphi[s := t]$ if $s, t \in \mathsf{SubTerm}(\varphi)$ and $s \notin \mathsf{SubTerm}(t)$.

(ii) $\varphi \to \varphi\downarrow$.

Rule i of the reduction system EUF-REDUCE allows to substitute equals for equals. Recall that we consider literals to be unordered, so this rule applies to $t \approx s$ as well. Rule ii simplifies a CNF by removing all equalities of the form $t \approx t$ from a formula. The transformation by the reduction rules yields a logically equivalent CNF.

Definition 4.4 (**Reduced EUF-CNFs**) We define a reduced EUF-CNF to be a normal form with respect to the reduction system EUF-REDUCE.

By the following corollary we show which shape a reduced EUF-CNF may have.

Proposition 4.5 *If φ is a reduced formula then the following holds.*

(i) *If $\varphi = \{\{s \approx t\}\} \uplus \varphi'$ then either $s \notin \mathsf{SubTerm}(\varphi')$ or $t \notin \mathsf{SubTerm}(\varphi')$.*

(ii) *φ does not contain equalities of the form $t \approx t$.*

Proof. If φ does not satisfy i then Rule i of the reduction system EUF-REDUCE can be applied. If φ does not satisfy ii then Rule ii of the reduction system EUF-REDUCE can be applied. □

We prove in Section 4.1 that the set of reduction rules is terminating, so at least one normal form exists. Unfortunately, the rules are not confluent as it is shown by the following example. So the normal form is not unique.

Example 4.6 Consider $\varphi = \{\{x \approx f(y)\}, \{x \approx g(z)\}, \{f(y) \approx h(x,z)\}\}$.

(i) Applying Rule i of the reduction system EUF-REDUCE on $x \approx f(y)$, we can replace x with $f(y)$. Therefore, the reduced formula is

$$\varphi' = \{\{x \approx f(y)\}, \{f(y) \approx g(z)\}, \{f(y) \approx h(f(y), z)\}\}.$$

(ii) We can also replace $f(y)$ with x. The result is the reduced formula

$$\varphi'' = \{\{x \approx f(y)\}, \{x \approx g(z)\}, \{x \approx h(x,z)\}\}.$$

4.1 Termination

Now we prove termination of the reduction system and of the corresponding GDPLL procedure.

In the following we use a definition of *non-propagated unit clauses*, i.e. unit clauses to which Rule i of the reduction system can be applied.

Definition 4.7 (**Non-propagated unit clauses**)

- A unit clause $\{s \approx t\}$ is called non-propagated in a CNF φ if
  - $\{s \approx t\} \in \varphi$,
  - $s, t \in \mathsf{SubTerm}(\varphi \backslash \{\{s \approx t\}\})$.
- The set of all non-propagated unit clauses in φ is denoted by $\mathsf{NPCls}(\varphi)$.

In Definition 4.3(i) the $s \approx t$ is non-propagated in $\{\{s \approx t\}\} \cup \varphi$, whereas the $s \approx t$ is propagated in $\{\{s \approx t\}\} \cup \varphi[s := t]$. Note that $s \notin \mathsf{SubTerm}(\varphi[s := t])$.

We start with a technical lemma which is used in Lemma 4.9 and in Theorem 4.11. Lemma 4.9 is also a technical lemma.

Lemma 4.8 *Suppose $s, t \in \mathsf{Term}$, where $s \notin \mathsf{SubTerm}(t)$, and T is a set of terms. Then*

$$|T \cup \{s,t\}| \geq |T[s := t] \cup \{s,t\}|.$$

Proof. We can observe that for an arbitrary set of terms T' and arbitrary terms s', t' the following holds.

- $|T'| = |T'[s' := t']|$ if for all distinct $u, v \in T'$, $u[s' := t'] \neq v[s' := t']$,
- $|T'| > |T'[s' := t]'|$ if there are distinct $u, v \in T$ such that $u[s' := t'] = v[s' := t']$.

Hence,

$$\begin{aligned} |T \cup \{s,t\}| &= |(T \backslash \{s,t\}) \cup \{s,t\}| \\ &\geq |(T \backslash \{s,t\})[s := t] \cup \{s,t\}| \\ &= |T[s := t] \cup \{s,t\}| \end{aligned}$$

□

Lemma 4.9 *Suppose* $\varphi, \varphi', \psi \in \mathsf{Cnf}$ *and* $\varphi = \{\{s \approx t\}\} \uplus \varphi'$, *where* $s, t \in \mathsf{SubTerm}(\varphi')$ *and* $s \notin \mathsf{SubTerm}(t)$, *and* $\psi = \{\{s \approx t\}\} \cup \varphi'[s := t]$. *Then one of the following holds.*

(i) $|\mathsf{SubTerm}(\psi)| < |\mathsf{SubTerm}(\varphi)|$ *or*

(ii) $|\mathsf{SubTerm}(\psi)| \leq |\mathsf{SubTerm}(\varphi)|$ *and* $|\mathsf{NPCls}(\psi)| < |\mathsf{NPCls}(\varphi)|$.

Proof. By Lemma 4.8, $|\mathsf{SubTerm}(\psi)| \leq |\mathsf{SubTerm}(\varphi)|$. Hence, it is sufficient to show that from $|\mathsf{NPCls}(\psi)| \geq |\mathsf{NPCls}(\varphi)|$ it follows that $|\mathsf{SubTerm}(\psi)| < |\mathsf{SubTerm}(\varphi)|$.

Suppose $|\mathsf{NPCls}(\psi)| \geq |\mathsf{NPCls}(\varphi)|$. Then there is some $C \in \varphi$ such that $C \notin \mathsf{NPCls}(\varphi)$ and $C[s := t] \in \mathsf{NPCls}(\psi)$.

Suppose C has n literals, with $n \geq 2$. These literals must contain at least $n+1$ different atoms in order for them to be different. The substitution $[s := t]$ maps these $n+1$ atoms to the 2 atoms of $C[s := t]$. Therefore at least two different atoms of C are mapped to the same atom of $C[s := t]$. From this we can conclude that $|\mathsf{SubTerm}(\varphi)| > |\mathsf{SubTerm}(\psi)|$.

What is left is the case that C has only one literal. Suppose $C = \{u \approx v\}$, where u, v are terms. Without loss of generality we can assume that $u \notin \mathsf{SubTerm}(\varphi \backslash \{C\})$. Since $C[s := t] \in \mathsf{NPCls}(\psi)$ there must be a clause $C' \in \psi, C' \neq C[s := t]$ that contains $u[s := t]$. Let $C'' \in \varphi$, $C'' \neq C$ be the clause of φ that is mapped to C' and let w be the subterm in clause C'' that is mapped to $u[s := t]$. Again we have found two different terms $u, w \in \mathsf{SubTerm}(\varphi)$ that are mapped to the same term $u[s := t] \in \mathsf{SubTerm}(\psi)$. Hence we can conclude that $|\mathsf{SubTerm}(\varphi)| > |\mathsf{SubTerm}(\psi)|$.

□

Definition 4.10 To each $\varphi \in \mathsf{EUF\text{-}Cnf}$ we relate a pair of numbers, using $\mathsf{norm}(\varphi)$ as below:

$$\mathsf{norm}(\varphi) = (|\mathsf{SubTerm}(\varphi)| + |\mathsf{Lit}(\varphi)|, |\mathsf{NPCls}(\varphi)|).$$

Theorem 4.11 *The reduction system* EUF-REDUCE *is terminating.*

Proof. We prove termination of the reduction system EUF-REDUCE by showing that after applying each step of the reduction system on a formula, $\mathsf{norm}(\varphi)$ decreases with respect to the lexicographic order $\prec_{lex}$ on pairs. Suppose $\varphi \to \psi$, with $\varphi \neq \psi$.

(i) Suppose $\varphi = \{\{s \approx t\}\} \uplus \varphi'$, where $s, t \in \mathsf{SubTerm}(\varphi')$ and $s \notin \mathsf{SubTerm}(t)$, and $\psi = \{\{s \approx t\}\} \uplus \varphi'[s := t]$.

Taking into account Lemma 4.8, we conclude

$$\begin{aligned}|\mathsf{Lit}(\psi)| &= |\mathsf{Lit}(\varphi'[s := t] \cup \{\{s \approx t\}\})| \\ &\leq |\mathsf{Lit}(\varphi' \cup \{\{s \approx t\}\})| \\ &= |\mathsf{Lit}(\varphi)|\end{aligned}$$

By Lemma 4.9, either $|\mathsf{SubTerm}(\psi)| < |\mathsf{SubTerm}(\varphi)|$ or $|\mathsf{SubTerm}(\psi)| = |\mathsf{SubTerm}(\varphi)|$ and $|\mathsf{NPCls}(\psi)| < |\mathsf{NPCls}(\varphi)|$.

We obtain, that either $|\mathsf{SubTerm}(\psi)| + |\mathsf{Lit}(\psi)| < |\mathsf{SubTerm}(\varphi)| + |\mathsf{Lit}(\varphi)|$ or $|\mathsf{SubTerm}(\psi)| + |\mathsf{Lit}(\psi)| = |\mathsf{SubTerm}(\varphi)| + |\mathsf{Lit}(\varphi)|$ and $|\mathsf{NPCls}(\psi)| < |\mathsf{NPCls}(\varphi)|$.

(ii) Suppose $\psi = \varphi \downarrow$.

Then

$$\begin{aligned}|\mathsf{Lit}(\psi)| &\leq |\mathsf{Lit}(\varphi)| - |\{t \approx t \mid (t \approx t) \in \mathsf{Lit}(\varphi)\}| - |\{t \not\approx t \mid (t \not\approx t) \in \mathsf{Lit}(\varphi)\}| \\ &< |\mathsf{Lit}(\varphi)|\end{aligned}$$

Obviously, $|\mathsf{SubTerm}(\psi)| \leq |\mathsf{SubTerm}(\varphi)|$. We conclude, $|\mathsf{SubTerm}(\psi)| + |\mathsf{Lit}(\psi)| < |\mathsf{SubTerm}(\varphi)| + |\mathsf{Lit}(\varphi)|$.

Hence, $\psi \prec_{lex} \varphi$. □

We have proved that the reduction system EUF-REDUCE is terminating.

5 Satisfiability criterion

In this section we will consider conditions under which an EUF-CNF is satisfiable. These conditions will form a basis for the function SatCriterion.

Definition 5.1 (**The core of a EUF-CNF**) Let $\varphi \in \mathsf{Cnf}$. Then the set of positive clauses (i.e. clauses consisting entirely of positive literals $s \approx t$) of length at least two contained in φ is called the *core* of φ and denoted by $\mathsf{Core}(\varphi)$.

Let φ be a reduced EUF-CNF not containing the empty clause, such that $\mathsf{Core}(\varphi) = \emptyset$. We will give a proof that such a EUF-CNF φ is satisfiable. Without loss of generality we can restrict ourselves to the case when a EUF-CNF contains only unit clauses. It follows from the fact that if there is an assignment satisfying one literal in each clause of a CNF ϕ then ϕ is satisfiable.

The set of EUF-CNFs containing only unit clauses is denoted by UCnf.

At first we introduce two binary relations on the set of terms contained in φ.

Definition 5.2 Let $\varphi \in \mathsf{UCnf}$. The binary relation $\sim_\varphi$ is the smallest relation over $\mathsf{Term}(\varphi) \times \mathsf{Term}(\varphi)$ such that:

(i) $s \sim_\varphi t$, if $\{s \approx t\} \in \varphi$.

(ii) $\sim_\varphi$ is reflexive, symmetric, and transitive.

Definition 5.3 The binary relation $\cong_\varphi$ is the smallest relation over $\mathsf{SubTerm}(\varphi) \times \mathsf{SubTerm}(\varphi)$ such that:

(i) $s \cong_\varphi t$, if $\{s \approx t\} \in \varphi$.

(ii) $f(s_1, \ldots, s_n) \cong_\varphi f(t_1, \ldots, t_n)$, if $s_i \cong_\varphi t_i$, $1 \leq i \leq n$, and $f(s_1, \ldots, s_n)$, $f(t_1, \ldots, t_n) \in \mathsf{SubTerm}(\varphi)$.

(iii) $\cong_\varphi$ is reflexive, symmetric, and transitive.

We will prove a lemma stating that for reduced CNFs the introduced binary relations are equivalent.

Lemma 5.4 *Let $\varphi \in \mathsf{UCnf}$ be reduced. Then for each $s, t \in \mathsf{Term}$*

$$s \sim_\varphi t \text{ if and only if } s \cong_\varphi t.$$

Proof.

($\Rightarrow$) Suppose $s \sim_\varphi t$. Then by Definitions 5.2 and 5.3, we obtain $s \cong_\varphi t$.

($\Leftarrow$) Suppose $s \cong_\varphi t$. We give a proof by contradiction.

Assume that

$$s \not\sim_\varphi t.$$

It means that Condition (*ii*) of Definition 5.3 was applied at least one time.

Without loss of generality we can assume that we applied the condition one time. Then there are

$$f(s_1, \ldots, s_n), f(t_1, \ldots, t_n) \in \mathsf{SubTerm}(\varphi)$$

such that

$$s_i \sim_\varphi t_i, 1 \leq i \leq n.$$

In this case there are $u_0, \ldots, u_n \in \mathsf{Term}(\varphi)$ such that

$$\{s_i = (u_0 \approx u_1)\}, \{u_1 \approx u_2\}, \ldots, \{(u_{n-1} \approx u_n) = t_n\} \in \varphi,$$

where $i \in \{1, \ldots, n\}$.

In this case Rule i of the reduction system EUF-REDUCE would be applicable. This contradicts that φ is reduced. We obtain that $s \sim_\varphi t$. □

Lemma 5.5 *Suppose $\varphi \in \mathsf{UCnf}$, φ is reduced and $\{s \not\approx t\} \in \varphi$ for some $s, t \in \mathsf{Term}$. Then $s \not\cong_\varphi t$.*

Proof. At first we prove by contradiction that $s \not\sim_\varphi t$.

Assume that $s \sim_\varphi t$. Then one of the following holds.

- $s = t$. Then $\{s \not\approx s\} \in \varphi$. In this case Rule ii can be applied. This contradicts that φ is reduced.
- There are $u_0, \ldots, u_n \in \mathsf{Term}(\varphi)$ such that

$$\{s = u_0 \approx u_1\}, \{u_1 \approx u_2\}, \ldots, \{u_{n-1} \approx u_n = t\} \in \varphi.$$

Then Rule i can be applied. This contradicts that φ is reduced.

We can conclude that $s \not\sim_\varphi t$.

By Lemma 5.4 we obtain that $s \not\cong_\varphi t$. □

Theorem 5.6 $\varphi \in \mathsf{UCnf}$ *is unsatisfiable if and only if there exist* $s, t \in \mathsf{Term}(\varphi)$ *such that*

$$\{s \not\approx t\} \in \varphi \text{ and } s \cong_\varphi t.$$

Proof. See [16]. □

Theorem 5.7 (**Satisfiability criterion**) *Suppose* φ *is a reduced CNF,* $\bot \notin \varphi$, *and* $\mathsf{Core}(\varphi) = \emptyset$. *Then* φ *is satisfiable.*

Proof. From the assumption of the theorem we conclude that every clause of length more than one contains at least one negative literal. Let $\psi \in \mathsf{Cnf}$ be obtained from φ by removing from all clauses all literals except one negative literal. Hence, ψ is reduced by construction. By Lemma 5.5 and Theorem 5.6 ψ is satisfiable. We can conclude that φ is satisfiable also. □

6 The GDPLL building blocks for the EUF-logic

We now come to the definition of the building blocks for the GDPLL procedure.

The procedure GDPLL for the EUF-logic invokes the following functions.

- We define the function $\mathsf{REDUCE}(\varphi)$ to be any normal form of φ with respect to the reduction system EUF-REDUCE.
- For each $\varphi \in \mathsf{R}$, the function SatCriterion() is defined as follows:

$$\mathsf{SatCriterion}(\varphi) = \begin{cases} \mathsf{true} & \text{if } \mathsf{Core}(\varphi) = \emptyset, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

- For each $\varphi \in \mathsf{R}$, the function Eligible() is defined as below:

$$\mathsf{Eligible}(\varphi) = \mathsf{Lit}(\mathsf{Core}(\varphi)).$$

- For each $\varphi \in \mathsf{R}$ and for each $a \in \mathsf{Eligible}(\varphi)$, the function Filter() is defined as

$$\mathsf{Filter}(\varphi, a) = \{\{a\}\} \cup \varphi|_a \text{ and } \mathsf{Filter}(\varphi, \neg a) = \{\{\neg a\}\} \cup \varphi|_{\neg a}.$$

Example 6.1 As an example we consider the formula originating from the verification of translators (compilers, code generators) [13], where concrete functions have been replaced by uninterpreted function symbols.

$$\varphi_0 = \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\},$$

$$\{z \not\approx g(f(x_1, y_1), f(x_2, y_2))\}\}.$$

After applying Rule i, we obtain

$$\varphi_1 = \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\}, \{z \not\approx g(u_1, f(x_2, y_2))\}\},$$

$$\varphi_2 = \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\}, \{z \not\approx g(u_1, u_2)\}\},$$

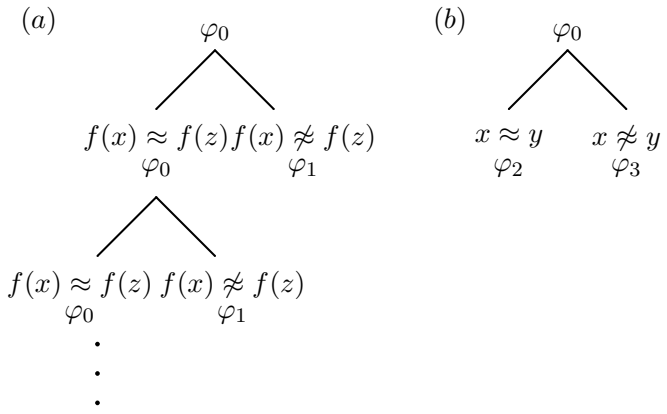

Fig. 2. a) An example of a non-terminating derivation $\varphi_0 = \{\{x \approx y, y \approx z\}, \{f(x) \approx f(z)\}\}$. b) An example of a terminating derivation $\varphi_0 = \{\{x \approx y, y \approx z\}, \{f(x) \approx f(z)\}\}$

$$\varphi_3 = \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\}, \{z \not\approx z\}\}.$$

After applying Rule ii, we obtain

$$\varphi_4 = \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\}, \bot\}.$$

Since $\bot \in \varphi_4$, φ_4 is unsatisfiable and therefore φ_0 is unsatisfiable.

Therefore, the function REDUCE() is defined by means of reduction rules, that can be applied in any order. The reduction rules allow to replace equals for equals, and simplifying a formula by removing all equalities of the form $t \approx t$. So, all work specific for the EUF-logic is done by the function REDUCE.

The function Eligible() allows us to choose literals from the purely positive clauses of length more than one, i.e. from the core of a formula. Hence, we may terminate with SAT as soon as the core of a reduced formula is empty and the formula does not contain the empty clause.

In Example 6.2 we show that choosing to split on an arbitrary positive literal can lead to a non-terminating derivation.

Example 6.2 The figure 2(a) shows sample derivations from a CNF $\varphi_0 = \{\{x \approx y, y \approx z\}, \{f(x) \approx f(z)\}\}$ in tree notation. Splitting on a literal $f(x) \approx f(z)$ can lead to a non-terminating derivation since for a positive branch φ_0 is derived. For a negative branch the CNF $\varphi_1 \equiv \{\{x \approx y\}, \{y \approx z\}\}$ is derived.

Splitting on a literal contained in Core(φ) leads to a terminating derivation. For the given CNF φ_0, Core(φ_0) $= \{x \approx y, y \approx z\}$. A terminating derivation is depicted in Figure 2(b). After splitting on a literal $x \approx y$, for a positive branch a reduced CNF $\varphi_2 \equiv \{\{x \approx y\}, \{f(x) \approx f(z)\}\}$ is derived and for a negative branch a reduced CNF $\varphi_3 \equiv \{\{y = z\}, \{f(x) \approx f(z)\}\}$ is derived. Both φ_2 and φ_3 are satisfiable according Theorem 5.7.

7 Soundness and completeness

In this section we prove that the GDPLL procedure for the EUF-logic is sound and complete. One can see that the rules of the reduction system EUF-REDUCE preserve (un)satisfiability of a formula.

Lemma 7.1 *Let $\varphi \in \mathsf{Cnf}$. Then φ is satisfiable if and only if $\mathsf{REDUCE}(\varphi)$ is satisfiable.*

Proof. We show that every reduction step preserves (un)satisfiability. So, let $\varphi \to \psi$.

($\Rightarrow$) Suppose φ is satisfiable. Let I be an arbitrary E-interpretation such that $\mathsf{I} \models \varphi$.

(i) Suppose $\varphi \equiv \{\{s \approx t\}\} \uplus \varphi'$, where $s,t \in \mathsf{SubTerm}(\varphi')$ and $s \notin \mathsf{SubTerm}(t)$, and $\psi \equiv \{\{s \approx t\}\} \uplus \varphi'[s := t]$.
Taking into account $\mathsf{I} \models s \approx t$, we obtain $\mathsf{I} \models \varphi'[s := t]$. Hence, $\mathsf{I} \models \psi$.

(ii) Let $\psi \equiv \varphi\downarrow$.
Hence, $\psi = \{C\downarrow \mid (C \in \varphi) \wedge (\forall t \in \mathsf{Term} : (t \approx t) \notin \mathsf{C})\}$. Then for each $C \in \varphi$, $\mathsf{I} \models C$. Since for each $t \in \mathsf{Term}$, $\mathsf{I} \not\models t \not\approx t$, we obtain that $\mathsf{I} \models C\downarrow$. Therefore, $\mathsf{I} \models \psi$.

($\Leftarrow$) Suppose ψ is satisfiable. Let I be an arbitrary E-interpretation such that $\mathsf{I} \models \psi$.

(i) Suppose $\varphi \equiv \{\{s \approx t\}\} \uplus \varphi'$, where $s,t \in \mathsf{SubTerm}(\varphi')$ and $s \notin \mathsf{SubTerm}(t)$, and $\psi \equiv \{\{s \approx t\}\} \uplus \varphi'[s := t]$.
Taking into account $\mathsf{I} \models s \approx t$, we obtain $\mathsf{I} \models \varphi'$. Hence, $\mathsf{I} \models \varphi$.

(ii) Assume $\psi \equiv \varphi\downarrow$. Then for each $C \in \varphi$ one of the following holds.
- $C \subseteq D$, where $D \in \psi$. Since $\mathsf{I} \models \psi$ then $\mathsf{I} \models D$. Hence, $\mathsf{I} \models C$.
- $(t \approx t) \in C$, where $t \in \mathsf{Term}$. Since for each $t \in \mathsf{Term}$, $\mathsf{I} \models t \approx t$ then $\mathsf{I} \models C$.

Since for each $C \in \varphi$, $\mathsf{I} \models C$, we conclude that $\mathsf{I} \models \varphi$. □

Lemma 7.2 *Let $\varphi \in \mathsf{Cnf}$, and $s,t \in \mathsf{Term}$. Then φ is unsatisfiable iff both $\{\{s \approx t\}\} \cup \varphi|_{s\approx t}$ and $\{\{s \not\approx t\}\} \cup \varphi|_{s\not\approx t}$ are unsatisfiable.*

Proof. ($\Rightarrow$) Let I be an arbitrary E-interpretation such that $\mathsf{I} \not\models \varphi$. Hence, $\mathsf{I} \not\models \{\{s \approx t\}\} \cup \varphi$ and $\mathsf{I} \not\models \{\{s \not\approx t\}\} \cup \varphi$.

Without loss of generality we can consider the case when $\mathsf{I} \models s \approx t$. In this case $\mathsf{I} \not\models \{\{s \not\approx t\}\} \cup \varphi|_{s\not\approx t}$.

Consider $\{\{s \approx t\}\} \cup \varphi|_{s\approx t}$. Since $\mathsf{I} \models s \approx t$, there is some $C \in \varphi$ such that $\mathsf{I} \not\models C\backslash\{s \not\approx t\}$. Hence, $\mathsf{I} \not\models \{\{s \approx t\}\} \cup \varphi|_{s\approx t}$.

($\Leftarrow$) Let I be an arbitrary E-interpretation. Since both $\{\{s \approx t\}\} \cup \varphi|_{s\approx t}$ and $\{\{s \not\approx t\}\} \cup \varphi|_{s\not\approx t}$ are unsatisfiable, $\mathsf{I} \not\models \{\{s \approx t\}\} \cup \varphi|_{s\approx t}$ and $\mathsf{I} \not\models \{\{s \approx t\}\} \cup \varphi|_{s\approx t}$.

By definition, either $\mathsf{I} \models s \approx t$ or $\mathsf{I} \models s \not\approx t$. Without loss of generality we can assume that $\mathsf{I} \models s \approx t$. Therefore, $\mathsf{I} \not\models \varphi$, and φ is unsatisfiable. □

Theorem 7.3 (**Soundness and Completeness**) *A CNF φ is unsatisfiable if and only if the* GDPLL(φ) *returns "unsatisfiable".*

Proof. In order to apply Theorem 5, we have to check the following properties.

Properties i and ii have been proved in Lemma 7.1 and in Lemma 7.2. Properties iv and Property v have been proved in Theorem 5.7.

To prove Property iii we define a well-founded order $\prec$ as follows. For all $\varphi, \psi \in \mathsf{REDUCE(Cnf)}$ $\psi_1 \prec \psi_2$ if $\sum_{C\in\mathsf{Core}(\psi_1)} |C| < \sum_{C\in\mathsf{Core}(\psi_2)} |C|$.

By definition of the function Filter(), for all $l \in \mathsf{Eligible}(\varphi)$

$$\mathsf{Core}(\mathsf{Filter}(\varphi, l)) \equiv \{C \in \mathsf{Core}(\varphi) |\ l \notin C\}.$$

By definition of the function Eligible(), there is some $C \in \mathsf{Core}(\varphi)$ such that $l \in C$. Hence, for every $l \in \mathsf{Eligible}(\varphi)$, $\mathsf{Filter}(\varphi, l) \prec \varphi$.

Consider $\mathsf{Filter}(\varphi, \neg l)$. By definition of the function Filter(), for all $l \in \mathsf{Eligible}(\varphi)$

$$\mathsf{Core}(\mathsf{Filter}(\varphi, \neg l)) \equiv \{C\backslash\{l\} |\ C \in \mathsf{Core}(\varphi)\}$$

Since there is some $C \in \mathsf{Core}(\varphi)$ such that $l \in C$, we conclude that for every $l \in \mathsf{Eligible}(\varphi)$, $\mathsf{Filter}(\varphi, \neg l) \prec \varphi$. □

8 Implementation

The algorithm was implemented in C++. The ATerm Library [17] was used to represent the literals. Operations that occur frequently on the literals in the algorithm are equivalence checking of terms, and substitution of subterms. These operations can be implemented efficiently using ATerms. The ATerm implementation uses maximal subterm sharing, which makes checking the syntactic equivalence of terms a constant time operation (pointer equality). For efficiency reasons, the subsumption step during simplification was made optional.

We have experimented with a formula related to the pigeon hole formula in proposition calculus

$$\Phi_n = \bigwedge_{1\le i<j\le n} x_i \neq x_j \wedge \bigwedge_{j=1}^{n} \left(\bigvee_{i\in\{x_1,\cdots,x_n\}, i\neq j} x_i = y \right)$$

and a formula, which is a generation of a formula in [13],

$$\Psi_{m,n} = \bigwedge_{1\le i<j\le m} \left(\left(\bigvee_{k=1}^{n} x_{ik} \neq x_{jk} \right) \vee f_i = f_j \right) \wedge \left(\left(\bigvee_{i=1}^{n} u_i \neq f_i \right) \vee g_1 = g_2 \right) \wedge \left(\bigwedge_{i=1}^{n} u_i = f_i \right) \wedge z = g_1 \wedge z \neq g_2$$

In both cases simple meta arguments can be used to show that these formulas are unsatisfiable. Our algorithm was able to solve Φ_n for values up to $n = 100$ and $\Psi_{m,n}$ for values $m = 50, n = 100$ within a couple of seconds.

Furthermore we have experimented with random formulas. Starting with a set random clauses, a number of random substitutions were applied (like $x := f(x)$), to ensure that reduction could be applied. The problem size was about 1000 clauses and 10 different symbols. The EufDpll algorithm was able to solve all these problems. However, the BarcelogicTools program (winner of the SMT-COMP 2005 competition) [11] could do the same and was significantly faster in determining the solution.

9 Conclusions and future work

In the paper we presented the algorithm to solve satisfiability problem for EUF logic which is based on the generalized DPLL procedure. Our approach is implemented in the EufDpll tool.

DPLL-based systems are really efficient only in combination with good optimization strategies. The procedure can incorporate some optimization techniques developed by the SAT community for the DPLL method. Not all approaches suitable for propositional logic work automatically for the EUF logic. Some techniques might become incorrect, and others being still correct, may lose their efficiency. A 'full-strength' version of EufDpll would include heuristics for good branching strategies. EufDpll is a prototype implementation. We have tested it on a diverse set of benchmarks, including the benchmarks from SMT-COMP'05. It is too early for serious comparison to other provers since some trimming to better performance should be done. Nevertheless, our approach looks very promising. It can also be easily extended to compute an interpretation as a model of a set of clauses. Another direction for future research can be extending to a non-clausal procedure.

References

[1] Ackermann, W. *Solvable cases of the decision problem.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1954.

[2] Badban, R., de Pol, J., Tveretina O., and Zantema, H. *Generalizing DPLL and Satisfiability for Equalities. Information and Computation*, vol. 205, issue 8, (August 2007), pp. 1188-1211.

[3] Barrett, C. W., Dill, D., and Levitt, J. Validity checking for combinations of theories with equality. In *Formal Methods in Computer-Aided Design (FMCAD'96)* (November 1996), M. Srivas and A. Camilleri, Eds., vol. 1166 of *LNCS*, Springer-Verlag, pp. 187–201.

[4] Bryant, R., and Velev, M. Boolean satisfiability with transitivity constraints. *ACM Transactions on Computational Logic 3*, 4 (October 2002), pp. 604–627.

[5] Burch, J.R. and Dill, D. L. Automatic verification of pipelined microprocessor control. in *Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 18, Springer-Verlag, Berlin, 1994.

[6] Davis, M., Logemann, G., and Loveland, D. A machine program for theorem proving. *Communications of the Association for Computing Machinery 7* (July 1962), pp. 394–397.

[7] Goel, A., Sajid, K., Zhou, H., Aziz, A., and Singhal, V. BDD based procedures for a theory of equality with uninterpreted functions. In *Computer-Aided Verification (CAV'98)* (1998), A. J. Hu and M. Y. Vardi, Eds., vol. 1427 of *LNCS*, Springer-Verlag, pp. 244–255.

[8] Groote, J., and van de Pol, J. Equational binary decision diagrams. In *Logic for Programming and Reasoning (LPAR'2000)* (2000), M. Parigot and A. Voronkov, Eds., vol. 1955 of *LNAI*, pp. 161–178.

[9] Meir, O., and Strichman, O. Yet another decision procedure for equality logic proceedings. In *17th International Conference on Computer Aided Verification (CAV 2005)* (2005), vol. 3576 of *LNCS*, pp. 307–320.

[10] Nieuwenhuis, R., and Oliveras, A. Congruence closure with integer offsets. In *10h Int. Conf. Logic for Programming, Artif. Intell. and Reasoning (LPAR)* (2003), M. Vardi and A. Voronkov, Eds., LNAI 2850, pp. 78–90.

[11] Nieuwenhuis, R. and Oliveras, A. Decision Procedures for SAT, SAT Modulo Theories and Beyond. The BarcelogicTools. In *Proc. of 12th Int. Conf. Logic for Programming, Artificial Intelligecne and Reasoning (LPAR)*, LNAI, pp. 23-46, 2005.

[12] Pnueli, A., Rodeh, Y., and Shtrichman, O. Range allocation for equivalence logic. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01)* (December 2001), R. Hariharan, M. Mukund, and V. Vinay, Eds., vol. 2245 of *LNCS*, Springer-Verlag, pp. 317–333.

[13] Pnueli, A., Rodeh, Y., Shtrichman, O., and Siegel, M. Deciding equality formulas by small domains instantiations. In *Computer-Aided Verification (CAV'99)* (1999), LNCS, Springer-Verlag.

[14] Pnueli, A., Rodeh, Y., Shtrichman, O., and Siegel, M. The small model property: how small can it be? *Information and Computation 178*, 1 (October 2002), 279 – 293.

[15] Rodeh, Y., and Shtrichman, O. Finite instantiations in equivalence logic with uninterpreted functions. In *Computer Aided Verification (CAV'01)* (July 2001), vol. 2102 of *LNCS*, Springer-Verlag, pp. 144–154.

[16] Shostak, R. An algorithm for reasoning about equality. *Communications of the ACM 21* (July 1978), 583–585.

[17] M.G.J. van den Brand and H.A. de Jong and P. Klint and P.A. Olivier Efficient Annotated Terms. in *Software, Practise and Experience*, vol. 30(3), pp. 259-291, 2000.