Transforming equality logic to propositional logic

Hans Zantema and Jan Friso Groote

Department of Computer Science, Eindhoven University of Technology

P.O. Box 513, 5600 MB Eindhoven, The Netherlands

h.zantema@tue.nl jfg@win.tue.nl

Abstract

We investigate and compare various ways of transforming equality formulas to propositional formulas, in order to be able to solve satisfiability in equality logic by means of satisfiability in propositional logic. We propose *equality substitution* as a new approach combining desirable properties of earlier methods, we prove its correctness and show its applicability by experiments.

1 Introduction

We consider *equality formulas* being propositional formulas in which the atoms are equalities between variables. Two such formulas are called *equality equivalent*, denoted by $\simeq_E$, if for any interpretation of the variables in any domain they yield the same result. For instance, we have

$$x = y \wedge x = z \quad \simeq_E \quad x = y \wedge y = z$$

since for both formulas the result is true if and only if the variables x, y, z all three have the same interpretation. On the other hand, in propositional logic they are not equivalent: writing p, q, r for $x = y$, $x = z$, $y = z$, respectively, we do not have $p \wedge q \equiv p \wedge r$.

The main question we address is the question of how to check whether two (big) equality formulas are equality equivalent automatically. A direct observation shows that

$$\phi \simeq_E \psi \iff \neg(\phi \leftrightarrow \psi) \simeq_E \mathsf{false},$$

hence checking equivalence of two formulas can be done by checking whether a formula is equivalent to false. The latter is called *satisfiability*, hence we are interested in satisfiability of equality formulas.

This problem plays an important role in hardware verification. In fact there one is interested in a slightly more extensive logic: the logic of equality with uninterpreted functions (UIF, [5]). However, by Ackermann's transformation ([1]) the problem of deciding validity of a formula in UIF is reduced to satisfiability of equality formulas. More recently, an improved transformation serving the same goal was proposed in [3].

One approach was presented in [7], where a variant of BDD-technology (EQ-BDDs) was developed for satisfiability of equality formulas. The given method is complete in the sense that their algorithm always terminates, and decides whether the given formula is satisfiable. Unfortunately, in EQ-BDDs there is no unique representation as is the case in ordinary BDDs for propositional formulas. Another method is proposed in [11]. There a resolution-like method was developed for checking satisfiability formulas in CNF.

A different approach is first transform the equality formula to a propositional formula and then analyze this propositional formula. For propositional formulas a lot of work has been done for efficient satisfiability checking, yielding a variety of efficient and usable implementations. In this paper we concentrate on transformations Ψ from equality formulas to propositional formulas by which satisfiability of equality formulas is transformed to satisfiability of propositional formulas, i.e.,

$$\phi \simeq_E \mathsf{false} \iff \Psi(\phi) \equiv \mathsf{false}.$$

Having such a transformation Ψ then checking satisfiability of an equality formula ϕ proceeds as follows: compute $\Psi(\phi)$ and decide whether $\Psi(\phi) \equiv \mathsf{false}$ by a standard satisfiability checker for propositional formulas. For such a transformation Ψ a number of properties is desirable:

- the size of $\Psi(\phi)$ is not too big;
- the structure of $\Psi(\phi)$ reflects the structure of ϕ;
- the variables of $\Psi(\phi)$ represent equalities in ϕ.

The main goal of these properties is that checking (propositional) satisfiability of $\Psi(\phi)$ by standard techniques is feasible for a reasonable class of formulas ϕ. Roughly speaking two main approaches can be distinguished:

1. Addition of transitivity. In this approach it is analyzed which transitivity properties may be relevant for ϕ, and Ψ is defined by

   $$\Psi(\phi) = \phi \wedge T,$$

   where T is the conjunction of the relevant transitivity properties. This approach is followed in [6, 4, 2].

2. Bit vector encoding. In this approach $\lceil \log(\#A) \rceil$ boolean variables x_i are introduced for every variable x, where $\#A$ is the size of the set A of variables, and $\Psi(\phi)$ is obtained from ϕ by replacing every $x = y$ by

$$\bigwedge_i (x_i \leftrightarrow y_i).$$

In [6] this is already mentioned as a folklore method. Closely related is range allocation [9, 10]. In this approach a formula structure is analyzed to define a small domain for each variable, preferably smaller than $\#A$. Then a standard BDD based tool is used to check satisfiability of the formula under the domain.

By addition of transitivity the variables of $\Psi(\phi)$ represent equalities in ϕ, but the structure of $\Psi(\phi)$ does not reflect the structure of ϕ. For instance, if ϕ is a formula over n variables then the size of T is $\Theta(n^3)$ which can be much bigger than the size of ϕ itself. On the other hand by bit vector encoding the structure of $\Psi(\phi)$ reflects the structure of ϕ, but the variables of $\Psi(\phi)$ do not represent equalities in ϕ. Moreover, although the size of the transformed formula is small, it often turns out that the efficiency of proving unsatisfiability of this formula by standard approaches is very bad.

In this paper we define *equality substitution* eqs as an alternative transformation that combines both desired properties. The emphasis is on proving correctness: both for the earlier approaches and equality substitution we prove the basic correctness property $\phi \simeq_E \mathsf{false} \iff \Psi(\phi) \equiv \mathsf{false}$. We are not aware of earlier full proofs for the earlier approaches. In the last section we report some experiments showing that equality substitution outperforms the bit vector encoding for a class of formulas similar to the pigeon hole formulas. Comparison of equality substitution to addition of transitivity shows a similar performance, but equality substitution yields much smaller formulas.

2 Basic definitions and properties

Let A be a finite set of variable symbols. We define an *equality formula* by the syntax

$$\begin{array}{lll} V & ::= & x \mid y \mid z \mid \cdots \quad \text{where } A = \{x, y, z, \ldots\} \\ E & ::= & V = V \mid \mathsf{true} \mid \mathsf{false} \mid \neg E \mid (E \vee E) \mid (E \wedge E) \mid (E \rightarrow E) \mid (E \leftrightarrow E) \end{array}$$

Hence an equality formula consists of *equations* $x = y$ for $x, y \in A$ and usual boolean connectives. As usual redundant parentheses will be omitted. For instance, if $x, y, z \in A$ then $(x = y \wedge y = z) \rightarrow x = z$ is an equality formula.

A *domain* D is defined to be a non-empty set. For any domain D we call a function $\epsilon : A \rightarrow D$ an *assignment* to D. For any assignment ϵ we define its interpretation $\overline{\epsilon}$ on equality formulas inductively as follows:

$$\overline{\epsilon}(x = y) = \begin{cases} \mathsf{true} & \text{if } \epsilon(x) = \epsilon(y) \\ \mathsf{false} & \text{if } \epsilon(x) \neq \epsilon(y) \end{cases} \qquad \begin{array}{rcl} \overline{\epsilon}(\neg\phi) & = & \neg\overline{\epsilon}(\phi) \\ \overline{\epsilon}(\phi \vee \psi) & = & \overline{\epsilon}(\phi) \vee \overline{\epsilon}(\psi) \\ \overline{\epsilon}(\phi \wedge \psi) & = & \overline{\epsilon}(\phi) \wedge \overline{\epsilon}(\psi) \end{array}$$

$$\begin{array}{rcl} \overline{\epsilon}(\mathsf{true}) & = & \mathsf{true} \\ \overline{\epsilon}(\mathsf{false}) & = & \mathsf{false} \end{array} \qquad \begin{array}{rcl} \overline{\epsilon}(\phi \rightarrow \psi) & = & \overline{\epsilon}(\phi) \rightarrow \overline{\epsilon}(\psi) \\ \overline{\epsilon}(\phi \leftrightarrow \psi) & = & \overline{\epsilon}(\phi) \leftrightarrow \overline{\epsilon}(\psi) \end{array}$$

Two equality formulas ϕ, ψ are called *equality equivalent*, denoted as $\phi \simeq_E \psi$, if $\overline{\epsilon}(\phi) = \overline{\epsilon}(\psi)$ for every domain D and every assignment ϵ to D. For instance, one can check that

$$(x = y \wedge y = z) \rightarrow x = z \ \simeq_E \ \mathsf{true}.$$

We will concentrate on the question how to decide whether $\phi \simeq_E \psi$ for arbitrary equality formulas ϕ, ψ. It is easily checked that

$$\phi \simeq_E \psi \iff \neg(\phi \leftrightarrow \psi) \simeq_E \mathsf{false}$$

hence we may and shall concentrate on the question whether $\phi \simeq_E \mathsf{false}$ for a given equality formula ϕ.

Fix a total order $<$ on A. For an equality formula ϕ write $R(\phi)$ for the equality formula obtained from ϕ by replacing every $x = x$ by true and replacing $x = y$ by $y = x$ if $y < x$, for all $x, y \in A$. Clearly $R(\phi) \simeq_E \phi$ for every equality formula ϕ. An equality formula ϕ is called *reduced* if $\phi = R(\phi)$, i.e., it only contains equations $x = y$ satisfying $x < y$. By applying this reduction our question of deciding $\phi \simeq_E \mathsf{false}$ for arbitrary equality formulas reduces to the question of deciding $\phi \simeq_E \mathsf{false}$ for a reduced equality formula ϕ.

We write $\equiv$ for logical equivalence in the sense of propositional logic; if applied to equality formulas this means that an equation $x = y$ is considered as a propositional atom.

Write T for the conjunction of all formulas

$$\neg R(x = y) \vee \neg R(y = z) \vee R(x = z)$$

for which $x, y, z \in A$ are all three distinct.

Theorem 1 *Let ϕ be a reduced equality formula. Then $\phi \simeq_E \mathsf{false}$ if and only if $\phi \wedge T \equiv \mathsf{false}$.*

Proof: First assume that $\phi \wedge T \equiv \mathsf{false}$. Let $\epsilon : A \rightarrow D$ be arbitrary; we have to prove that $\overline{\epsilon}(\phi) = \mathsf{false}$. By transitivity of equality in D we obtain that

$$\overline{\epsilon}(\neg R(x = y) \vee \neg R(y = z) \vee R(x = z)) = \mathsf{true}.$$

As a consequence we obtain $\overline{\epsilon}(T) = \mathsf{true}$. Hence

$$\overline{\epsilon}(\phi) = \overline{\epsilon}(\phi) \wedge \overline{\epsilon}(T) = \overline{\epsilon}(\phi \wedge T) = \mathsf{false};$$

the last step follows from $\phi \wedge T \equiv \mathsf{false}$ and the definition of $\overline{\epsilon}$.

Conversely assume that $\phi \simeq_E \mathsf{false}$ holds and $\phi \wedge T \not\equiv \mathsf{false}$; we have to derive a contradiction. Since $\phi \wedge T$ is satisfiable there is an assignment δ on the atoms of the shape $x = y$ to the booleans such that $\overline{\delta}(\phi \wedge T) = \mathsf{true}$, where $\overline{\delta}$ is the interpretation corresponding to δ. Hence $\overline{\delta}(\phi) = \overline{\delta}(T) = \mathsf{true}$. Define the relation $\simeq$ in A as follows:

$$x \simeq y \iff \overline{\delta}(R(x = y)).$$

From the definition of R it follows that $\simeq$ is reflexive and symmetric; since $\overline{\delta}(T) = \mathsf{true}$ we conclude that $\simeq$ is transitive. Hence $\simeq$ is an equivalence relation. By injectively mapping the equivalence classes of $\simeq$ to some domain D we obtain an assignment $\epsilon : A \to D$ satisfying

$$x \simeq y \iff \epsilon(x) = \epsilon(y).$$

By construction we now have $\overline{\epsilon}(\phi) = \overline{\delta}(\phi) = \mathsf{true}$, contradicting the assumption $\phi \simeq_E \mathsf{false}$. $\Box$

Theorem 1 shows that addition of transitivity is a valid approach for transforming equality formulas to propositional formulas by which satisfiability of equality formulas is transformed to satisfiability of propositional formulas. The next theorem states validity of the bit vector encoding approach.

Fix N to be the smallest number satisfying $2^N \geq \#A$. For every $x \in A$ introduce N boolean variables $x_1, \ldots, x_N$. Write A_N for the set of all of these $N * \#A$ boolean variables. The *bit vector encoding* bve transforming equality formulas over A to propositional formulas over A_N is defined as follows:

$$\mathsf{bve}(x = y) = \bigwedge_{i=1}^{N} (x_i \leftrightarrow y_i),$$

$$\mathsf{bve}(\mathsf{true}) = \mathsf{true}, \quad \mathsf{bve}(\mathsf{false}) = \mathsf{false}, \quad \mathsf{bve}(\neg\phi) = \neg\mathsf{bve}(\phi),$$

$$\mathsf{bve}(\phi \diamond \psi) = \mathsf{bve}(\phi) \diamond \mathsf{bve}(\psi)$$

for $x, y \in A$, $\diamond \in \{\vee, \wedge, \to, \leftrightarrow\}$.

Theorem 2 *Let ϕ be an equality formula over A. Then $\phi \simeq_E$* false *if and only if* $\mathsf{bve}(\phi) \equiv \mathsf{false}$.

Proof: For the 'if' part we take an arbitrary assignment $\epsilon : A \to D$ satisfying $\overline{\epsilon}(\phi) = \mathsf{true}$ and we prove that this gives rise to a satisfying assignment for $\mathsf{bve}(\phi)$. Since $\#\epsilon(A) \leq \#A \leq 2^N$ there exists an injective map $\alpha : \epsilon(A) \to \{\mathsf{false}, \mathsf{true}\}^N$. Define $\overline{\alpha} : A_N \to \{\mathsf{false}, \mathsf{true}\}$ by

$$\alpha(\epsilon(x)) = (\overline{\alpha}(x_1), \ldots, \overline{\alpha}(x_N))$$

for all $x \in A$. Extend $\overline{\alpha}$ to propositional formulas over A_N by defining

$$\overline{\alpha}(\mathsf{true}) = \mathsf{true}, \quad \overline{\alpha}(\mathsf{false}) = \mathsf{false}, \quad \overline{\alpha}(\neg\phi) = \neg\overline{\alpha}(\phi),$$

$$\overline{\alpha}(\phi \diamond \psi) = \overline{\alpha}(\phi) \diamond \overline{\alpha}(\psi)$$

for $x, y \in A$, $\diamond \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$. For $x, y \in A$ we obtain

$$\begin{array}{lcl}
\overline{\epsilon}(x = y) = \mathsf{true} & \iff & \epsilon(x) = \epsilon(y) \\
& \iff & \alpha(\epsilon(x)) = \alpha(\epsilon(y)) \quad \text{(since } \alpha \text{ is injective)} \\
& \iff & (\overline{\alpha}(x_1), \ldots, \overline{\alpha}(x_N)) = (\overline{\alpha}(y_1), \ldots, \overline{\alpha}(y_N)) \\
& \iff & \overline{\alpha}(x_1) = \overline{\alpha}(y_1) \wedge \cdots \wedge \overline{\alpha}(x_N)) = \overline{\alpha}(y_N) \\
& \iff & \overline{\alpha}(\bigwedge_{i=1}^{N}(x_i \leftrightarrow y_i)) = \mathsf{true} \\
& \iff & \overline{\alpha}(\mathsf{bve}(x = y)) = \mathsf{true}.
\end{array}$$

This holds for every equality $x = y$. Hence,

$$\overline{\alpha}(\mathsf{bve}(\phi)) = \overline{\epsilon}(\phi) = \mathsf{true}.$$

So, we have a satisfying assignment $\overline{\alpha}$ for $\mathsf{bve}(\phi)$, which we had to prove.

For the converse assume $\overline{\alpha} : A_N \rightarrow \{\mathsf{false}, \mathsf{true}\}$ is a satisfying assignment for $\mathsf{bve}(\phi)$. Let $D = \{\mathsf{false}, \mathsf{true}\}^N$. Define $\epsilon : A \rightarrow D$ by $\epsilon(x) = (\overline{\alpha}(x_1), \ldots, \overline{\alpha}(x_N))$. Similarly as above we obtain $\overline{\alpha}(\mathsf{bve}(x = y)) = \mathsf{true} \iff \overline{\epsilon}(x = y) = \mathsf{true}$, hence from $\overline{\alpha}(\mathsf{bve}(\phi)) = \mathsf{true}$ we may conclude $\overline{\epsilon}(\phi) = \mathsf{true}$, contradicting the assumption $\phi \simeq_E \mathsf{false}$. □

The requirement $2^N \geq \#A$ is essential for the validity of Theorem 2 as is shown by the following example. Let $A = \{x_1, \ldots, x_n\}$ and $n > 2^N$. Then

$$\bigwedge_{1 \leq i < j \leq n} \neg(x_i = x_j) \quad \not\simeq_E \quad \mathsf{false},$$

while

$$\mathsf{bve}(\bigwedge_{1 \leq i < j \leq n} \neg(x_i = x_j)) \quad = \quad \bigwedge_{1 \leq i < j \leq n} \neg(\bigwedge_{k=1}^{N}(x_{ik} \leftrightarrow x_{jk})) \quad \equiv \quad \mathsf{false}.$$

3 Equality substitution

In this section equality substitution eqs is introduced for transforming equality formulas to propositional formulas, combining desired properties of the two transformations considered until now. Just like in bit vector encoding a substitution is applied on the equalities in the formula, and the rest of the formula remains unchanged. The main point is to define $\mathsf{eqs}(x = y)$ for variables x, y such that $\phi \simeq_E \psi \iff \mathsf{eqs}(\phi) \equiv \mathsf{eqs}(\psi)$.

Let $<$ on A be the order that we already fixed for defining R. It is convenient to number the elements of A with respect to this order, i.e., we assume $A = \{x_1, x_2, \ldots, x_n\}$ for $n = \#A$, satisfying

$$x_i < x_j \iff i < j.$$

For every i, j satisfying $1 \leq i < j \leq n$ we introduce a fresh propositional variable p_{ij}; the set of all these $\frac{n(n-1)}{2}$ variables is denoted by P_A.

For $1 \leq k \leq i < j \leq n$ we define $P(k, i, j)$ inductively by

$$P(i, i, j) = p_{ij}$$

for all i, j satisfying $1 \leq i < j \leq n$, and

$$P(k, i, j) = (p_{ki} \wedge p_{kj}) \vee (\neg p_{ki} \wedge \neg p_{kj} \wedge P(k + 1, i, j))$$

for all k, i, j satisfying $1 \leq k < i < j \leq n$. We will use these formulas only for $k = 1$; the formula $P(1, i, j)$ is a propositional formula over P_A of size $O(i)$. For instance, $P(1, 3, 5)$ is equal to

$$(p_{13} \wedge p_{15}) \vee (\neg p_{13} \wedge \neg p_{15} \wedge ((p_{23} \wedge p_{25}) \vee (\neg p_{23} \wedge \neg p_{25} \wedge p_{35}))).$$

We define the transformation eqs from equality formulas over A to propositional formulas over P_A as follows:

$$\mathsf{eqs}(x_i = x_j) = \begin{cases} \mathsf{true} & \text{if } i = j, \\ P(1, i, j) & \text{if } i < j, \\ P(1, j, i) & \text{if } j < i, \end{cases}$$

$$\mathsf{eqs}(\mathsf{true}) = \mathsf{true}, \quad \mathsf{eqs}(\mathsf{false}) = \mathsf{false}, \quad \mathsf{eqs}(\neg\phi) = \neg\mathsf{eqs}(\phi),$$

and

$$\mathsf{eqs}(\phi \diamond \psi) = \mathsf{eqs}(\phi) \diamond \mathsf{eqs}(\psi)$$

for $\diamond \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$.

It is hard to give an intuition for eqs other than what follows directly from its definition; surprisingly the original intuition we had for eqs turned out to be wrong. Many modifications of eqs turned out to violate the essential property below.

Theorem 3 *Let ϕ, ψ be arbitrary equality formulas over A. Then*

$$\phi \simeq_E \psi \iff \mathsf{eqs}(\phi) \equiv \mathsf{eqs}(\psi).$$

Indeed, $\mathsf{eqs}((x_1 = x_2 \wedge x_2 = x_3) \rightarrow x_1 = x_3)$ is equal to

$$(p_{12} \wedge ((p_{12} \wedge p_{13}) \vee (\neg p_{12} \wedge \neg p_{13} \wedge p_{23}))) \rightarrow p_{13}$$

which is logically equivalent to $\mathsf{eqs}(\mathsf{true}) = \mathsf{true}$.

In the remainder of this section we prove Theorem 3. We start by proving

$$\phi \simeq_E \psi \iff \mathsf{eqs}(\phi) \equiv \mathsf{eqs}(\psi).$$

We assume that

$$\overline{\delta}(\mathsf{eqs}(\phi)) = \overline{\delta}(\mathsf{eqs}(\psi))$$

for all $\delta : P_A \rightarrow \mathsf{Bool}$, and we have to prove that $\overline{\epsilon}(\phi) = \overline{\epsilon}(\psi)$ for every domain D and every assignment $\epsilon : A \rightarrow D$. This follows from the following lemma, proving the $\Leftarrow$-part of Theorem 3.

For an assignment $\epsilon : A \rightarrow D$ we define $\delta_\epsilon : P_A \rightarrow \mathsf{Bool}$ by

$$\delta_\epsilon(p_{ij}) \iff \epsilon(x_i) = \epsilon(x_j).$$

Lemma 4 *Let ϕ be an equality formula and let $\epsilon : A \rightarrow D$ be any assignment. Then*

$$\overline{\epsilon}(\phi) = \overline{\delta}_\epsilon(\mathsf{eqs}(\phi)).$$

Proof: Due to the compositional definition of eqs it suffices to prove this for ϕ being of the shape $x_i = x_j$. In case of $i = j$ this holds since $\overline{\epsilon}(x_i = x_i) = \mathsf{true} = \overline{\delta}_\epsilon(\mathsf{true}) = \overline{\delta}_\epsilon(\mathsf{eqs}(x_i = x_i))$. In the remaining case $i \neq j$ we may assume $i < j$ by

$$\overline{\epsilon}(x_i = x_j) = \overline{\epsilon}(x_j = x_i)$$

and symmetry in the definition of eqs. Since $\mathsf{eqs}(x_i = x_j)$ is equal to $P(1, i, j)$ it remains to prove

$$\overline{\epsilon}(x_i = x_j) \iff \overline{\delta}_\epsilon(P(1, i, j)).$$

We prove this by proving the stronger claim

$$\overline{\epsilon}(x_i = x_j) \iff \overline{\delta}_\epsilon(P(k, i, j)).$$

for all $k = 1, 2, \ldots, i$ by reverse induction on k. For $k = i$ this holds by definition. As the induction hypothesis we now assume

$$\epsilon(x_i) = \epsilon(x_j) \iff \overline{\epsilon}(x_i = x_j) \iff \overline{\delta}_\epsilon(P(k+1, i, j)).$$

Now we have

$\overline{\delta}_\epsilon(P(k, i, j)) \quad \iff \quad$ (by definition)

$\overline{\delta}_\epsilon((p_{ki} \wedge p_{kj}) \vee (\neg p_{ki} \wedge \neg p_{kj} \wedge P(k+1, i, j))) \quad \iff \quad$ (by definition)

$(\epsilon(x_k) = \epsilon(x_i) \wedge \epsilon(x_k) = \epsilon(x_j)) \vee (\epsilon(x_k) \neq \epsilon(x_i) \wedge \epsilon(x_k) \neq \epsilon(x_j) \wedge \overline{\delta}_\epsilon(P(k+1, i, j)))$

$\iff$ (by the induction hypothesis)

$(\epsilon(x_k) = \epsilon(x_i) \wedge \epsilon(x_k) = \epsilon(x_j)) \vee (\epsilon(x_k) \neq \epsilon(x_i) \wedge \epsilon(x_k) \neq \epsilon(x_j) \wedge \epsilon(x_i) = \epsilon(x_j))$

$\iff$ (by transitivity of $=$)

$(\epsilon(x_k) = \epsilon(x_i) \wedge \epsilon(x_i) = \epsilon(x_j)) \vee (\epsilon(x_k) \neq \epsilon(x_i) \wedge \epsilon(x_k) \neq \epsilon(x_j) \wedge \epsilon(x_i) = \epsilon(x_j))$

$\iff$ (proposition logic)

$(\epsilon(x_k) = \epsilon(x_i) \vee \epsilon(x_k) \neq \epsilon(x_j) \vee \epsilon(x_i) \neq \epsilon(x_j)) \wedge \epsilon(x_i) = \epsilon(x_j)$

$\iff$ (by transitivity of $=$)

$\epsilon(x_i) = \epsilon(x_j) \quad \iff \quad \bar{\epsilon}(x_i = x_j)$

which we had to prove. □

The hard part of Theorem 3 is the $\Rightarrow$-part. For that we need a lemma.

Lemma 5 *Let T be the conjunction of all formulas*

$$\neg R(x = y) \vee \neg R(y = z) \vee R(x = z)$$

for which $x, y, z \in A$ are all three distinct. Then $\mathsf{eqs}(T) \equiv \mathsf{true}$.

Proof: We have to prove that $\mathsf{eqs}(\neg R(x = y) \vee \neg R(y = z) \vee R(x = z)) \equiv \mathsf{true}$. Let j, k, m satisfying $1 \leq j < k < m \leq n$ be the numbers of the variables x, y, z in some order. Then the required property is one of the following three propositional equivalences:

$$\neg P(1, j, k) \vee \neg P(1, j, m) \vee P(1, k, m) \equiv \mathsf{true},$$
$$\neg P(1, j, k) \vee P(1, j, m) \vee \neg P(1, k, m) \equiv \mathsf{true},$$
$$P(1, j, k) \vee \neg P(1, j, m) \vee \neg P(1, k, m) \equiv \mathsf{true}.$$

We will prove the more general property that for every i satisfying $1 \leq i \leq j$ the following three propositional equivalences hold:

$$\neg P(i, j, k) \vee \neg P(i, j, m) \vee P(i, k, m) \equiv \mathsf{true},$$
$$\neg P(i, j, k) \vee P(i, j, m) \vee \neg P(i, k, m) \equiv \mathsf{true},$$
$$P(i, j, k) \vee \neg P(i, j, m) \vee \neg P(i, k, m) \equiv \mathsf{true}.$$

First assume that the first equivalence does not hold. Then there is an assignment such that the propositions

$$P(i, j, k) = (p_{ij} \wedge p_{ik}) \vee (\neg p_{ij} \wedge \neg p_{ik} \wedge P(i + 1, j, k)),$$
$$P(i, j, m) = (p_{ij} \wedge p_{im}) \vee (\neg p_{ij} \wedge \neg p_{im} \wedge P(i + 1, j, m)),$$
$$\neg P(i, k, m) = (\neg p_{ik} \vee \neg p_{im}) \wedge (p_{ik} \vee p_{im} \vee \neg P(i + 1, k, m))$$

all three hold. If p_{ij} holds then we conclude from the validity of the first two propositions that p_{ik} and p_{im} both hold too, contradicting the validity of the third proposition. Hence $\neg p_{ij}$ holds. Then by validity of all three propositions we conclude that $\neg p_{ik}$, $\neg p_{im}$, $P(i+1, j, k)$, $P(i+1, j, m)$ and $\neg P(i+1, k, m)$ all hold. Repeating the same argument $j - i$ times yields that

$$P(j, j, k) = p_{jk}, \quad P(j, j, m) = p_{jm},$$

$$\neg P(j, k, m) = (\neg p_{jk} \vee \neg p_{jm}) \wedge (p_{jk} \vee p_{jm} \vee \neg P(j+1, k, m))$$

all three hold, contradiction. Hence the first equivalence to be proved holds.

Next assume that the second equivalence does not hold. Then in a similar way after $j - i$ steps we obtain that

$$P(j, j, k) = p_{jk}, \quad \neg P(j, j, m) = \neg p_{jm},$$

$$P(j, k, m) = (p_{jk} \wedge p_{jm}) \vee (\neg p_{jk} \wedge \neg p_{jm} \wedge P(j+1, k, m))$$

all three hold, contradiction.

Finally assuming that the third equivalence does not hold yields in a similar way that

$$\neg P(j, j, k) = \neg p_{jk}, \quad P(j, j, m) = p_{jm},$$

$$P(j, k, m) = (p_{jk} \wedge p_{jm}) \vee (\neg p_{jk} \wedge \neg p_{jm} \wedge P(j+1, k, m))$$

all three hold, contradiction. $\Box$

Now we prove the $\Rightarrow$-part of Theorem 3.

Assume $\phi \simeq_E \psi$. Then $\neg(\phi \leftrightarrow \psi) \simeq_E \mathsf{false}$. From Theorem 1 we conclude that $\neg(\phi \leftrightarrow \psi) \wedge T \equiv \mathsf{false}$. In this equivalence the equalities are considered as propositional variables. Since eqs has been defined as a substitution on these variables we conclude $\mathsf{eqs}(\neg(\phi \leftrightarrow \psi) \wedge T) \equiv \mathsf{false}$. We obtain

$$\begin{array}{rcll}
\neg(\mathsf{eqs}(\phi) \leftrightarrow \mathsf{eqs}(\psi)) & \equiv & \mathsf{eqs}(\neg(\phi \leftrightarrow \psi)) & \\
& \equiv & \mathsf{eqs}(\neg(\phi \leftrightarrow \psi)) \wedge \mathsf{eqs}(T) & \text{(by Lemma 5)} \\
& = & \mathsf{eqs}(\neg(\phi \leftrightarrow \psi) \wedge T) & \\
& \equiv & \mathsf{false} &
\end{array}$$

hence $\mathsf{eqs}(\phi) \equiv \mathsf{eqs}(\psi)$, which concludes the proof of Theorem 3.

4 Experimental results

In this section we report some experimental results comparing addition of transitivity, bit vector encoding and equality substitution, all three in combination with various propositional satisfiability provers.

We consider the formulas form_n from [11] that are related to the pigeon hole formulas in proposition calculus. Just like pigeon hole formulas these are parameterized by a number n, they are easily seen to be contradictory by a meta argument, and each of the formulas is the conjunction of two subformulas. The formulas are defined as follows.

$$\mathsf{form}_n \;\equiv\; (\bigwedge_{1\leq i<j\leq n} x_i \neq x_j) \;\wedge\; \bigwedge_{j=1}^{n}(\bigvee_{i\in\{1,\ldots,n\},i\neq j} x_i = y)$$

There are $n+1$ variables $x_1,\ldots,x_n,y$. The first subformula states that all values of $x_1,\ldots,x_n$ are different. The second subformula states that the value of y occurs in every subset of size $n-1$ of $\{x_1,\ldots,x_n\}$, hence it will occur at least twice in $\{x_1,\ldots,x_n\}$, contradicting the property of the first subformula. Hence the total formula is unsatisfiable. This is a non-trivial kind of unsatisfiability in the following sense: the whole formula is a conjunction of a great number of formulas, and for every of these conjuncts it holds that the formula is satisfiable after removing the conjunct. Moreover, for every pair of variables the equality between these variables occurs in the formula, either positively or negatively. Since pigeon hole like formulas are well-known to be notoriously hard in propositional logic, we consider this formula to be an interesting candidate for experiments for techniques for checking satisfiability of equality formulas. We did our experiments on the formula form_n for n having the values 10, 15, 20, 30, 40, 50, 60.

We used three different propositional satisfiability checkers. The first one consists of computing the BDD using the package CUDD, see `http://supportweb.cs.bham.ac.uk/documentation/cudd/`. In the table this checker is denoted by 'bdd'. The second one first transforms the formula to CNF using Tseitin's transformation and then applies zChaff, see `http://ee.princeton.edu/~chaff/zchaff.php`. In the table this checker is denoted by 'ch'. The last one is the checker HeerHugo ([8]), denoted by 'hh'. All experiments are carried out under Linux on a 1Ghz. pentium 4.

The following table reports the results. Times are in seconds; '-' means that more than 600 seconds were required. Size indicates the number of binary symbols in the propositional formula.

	add transitivity				bit vector encoding				equality substitution			
n	size	bdd	ch	hh	size	bdd	ch	hh	size	bdd	ch	hh
10	1619	1	0	0	1079	56	1	113	794	0	0	0
15	5354	-	0	0	2519	-	7	-	2554	1	0	1
20	12539	-	0	0	5699	-	91	-	5889	20	0	1
30	41759	-	0	1	13049	-	-	-	19284	-	0	4
40	98279	-	1	3	28079	-	-	-	44979	-	1	16
50	191099	-	2	6	44099	-	-	-	86974	-	2	49
60	329219	-	4	11	63719	-	-	-	149269	-	5	123

About the bdd experiments with addition of transitivity we note that the order in which the big conjunction is computed is of great influence on the result. In the table we first computed the bdds of form_n and T separately and then computed the conjunction, as is suggested by the the shape of the formula. Only computing the bdd of T is already very expensive: for 12 variables the resulting bdd has over one million nodes. However, by computing the bdd of form_n and then consecutively taking conjunction with each of the transitivity properties gives a much better result: then unsatisfiability of form_{60} is proved in 62 seconds.

As a conclusion from the table we may state that the best results are obtained by the two transformations addition of transitivity and equality substitution, both in combination with zChaff: then unsatisfiability of form_{60} is proved in only a few seconds. Among these two transformations equality substitution gives rise to the smallest formulas. Although bit vector encoding gives rise to much smaller formulas, it gives a very bad performance on proving unsatisfiability.

5 Concluding Remarks

We proposed equality substitution as a new transformation by which the satisfiability problem for equality logic is transformed to the satisfiability problem for propositional logic. Both for earlier approaches and for this new approach we gave proofs for correctness. We did some experiments on pigeon hole like formulas showing that equality substitution serves well for proving unsatisfiability of equality formulas in combination with the propositional prover zChaff. Although this involves only one particular class of formulas, it is an indication for practical applicability.

References

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

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

[3] R.E. Bryant, S. German, , and M.N. Velev. Processor verification using efficient reductions of the logic of uninterpreted functions to propositional logic. *ACM Transactions on Computational Logic*, 2(1):93–134, January 2001.

[4] R.E. Bryant and M.N. Velev. Boolean satisfiability with transitivity constraints. In E.A. Emerson and A.P. Sistla, editors, *Computer-Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 85–98. Springer-Verlag, July 2000.

[5] J.R. Burch and D.L. Dill. Automated verification of pipelined microprocesoor control. In D.L. Dill, editor, *Computer-Aided Verification (CAV'94)*, volume 818 of *LNCS*, pages 68–80. Springer-Verlag, June 1994.

[6] A. Goel, K. Sajid, H. Zhou, A. Aziz, and V. Singhal. BDD based procedures for a theory of equality with uninterpreted functions. In *Proceedings of Conference on Computer-Aided Verification (CAV)*, volume 1427 of *Lecture Notes in Computer Science*, pages 244–255. Springer, 1998.

[7] J. F. Groote and J. C. van de Pol. Equational binary decision diagrams. In M. Parigot and A. Voronkov, editors, *Logic for Programming and Reasoning (LPAR)*, volume 1955 of *Lecture Notes in Artificial Intelligence*, pages 161–178. Springer Verlag, 2000.

[8] J. F. Groote and J. P. Warners. The propositional formula checker HeerHugo. *Journal of Automated Reasoning*, 24:101–125, 2000.

[9] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small domains instantiations. In *Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 455–469. Springer-Verlag, 1999.

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

[11] O. Tveretina and H. Zantema. A proof system and a decision procedure for equality logic. Technical Report CS-report 03-02, Eindhoven University of Technology, 2003. Available via `http://www.win.tue.nl/~hzantema/TZ.pdf`.