Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 253 (2009) 39–54

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Term Rewriting Technique for Decision Graphs

Bahareh Badban[1]

Department of Software Engineering
University of Konstanz
Germany

Abstract

We provide an automatic verification for a fragment of FOL quantifier-free logic with zero, successor and equality. We use BDD representation of such formulas and to verify them, we first introduce a (complete) *term rewrite system* to generate an equivalent *Ordered* $(0, S, =)$-BDD from any given $(0, S, =)$-BDD. Having the ordered representation of the BDDs, one can verify the original formula in constant time. Then, to have this transformation automatically, we provide an algorithm which will do the whole process.

Keywords: Term Rewrite System; First-Order Logic; Decision Procedure; Verification

1 Introduction

In this article we consider the satisfiability and tautology problem for boolean combinations over the equational theory of zero and successor in the natural numbers. The atoms are equations between terms built from variables, zero (0) and successor (S). Formulas are built from atoms by means of negation ($\neg$) and conjunction ($\wedge$). The formulas are quantifier-free, except for the implicit outermost quantifier ($\forall$ when considering tautology checking, and $\exists$ when considering satisfiability).

In general, the decision problem for plain equational theories is unsolvable already, so we must restrict to particular theories. The decision problem for boolean combinations over equational theories can be approached in several ways. *Binary Decision Diagrams* (BDDs) represent boolean functions as directed acyclic graphs [5]. They are of value for validating formulas in propositional logic. In [5] OBDDs (*Ordered* BDDs) are reduced BDDs which accept some ordering on boolean variables. A boolean function is satisfiable if and only if its unique OBDD representation does not correspond to 0. In the BDD-method, a formula is transformed

[1] Email: badban@inf.uni-konstanz.de

doi:10.1016/j.entcs.2009.10.016

to a propositionally equivalent *Ordered Binary Decision Diagram* (OBDD) which can be seen as a large if-then-else (ITE) tree with shared subterms (see Section 2). Although in principle also OBDD representations are exponentially big, it appears that in practice many formulas have a succinct OBDD-representation. Furthermore, boolean operations, such as negation and conjunction, can be computed on OBDDs very cheaply. Together with the fact that (due to sharing) many practical boolean functions have a small OBDD representation, OBDDs are very popular in verification of hardware design, and play a major role in symbolic model checking.

In order to solve the satisfiability or tautology problem, each path in the OBDD has to be checked for consistency, with respect to the underlying equational theory. A path represents a conjunction of (negated) equations, on which the aforementioned decision procedures can be applied. All inconsistent paths can be removed, resulting in an OBDD with only consistent paths. However, due to sharing subterms, an OBDD can have exponentially many paths, so still there is a computational bottleneck. In the *Encoding method* these steps are reversed. First the formula is transformed to a purely propositional formula. In this translation, facts from the equational theory (e.g. congruence of functions, transitivity of equality and orderings) are encoded into the formula. Then a *finite model property* is used to obtain a finite upperbound on the cardinality of the model. Finally, variables that range over a set of size n are encoded by $log(n)$ propositional variables. The resulting formula can be checked for satisfiability with any existing SAT-technique, for instance based on resolution [7] or on BDDs [5]. An early example is Ackermann's reduction [1], by which second order variables can be eliminated. More optimal versions are in [10,17,6].

To date, several methods have been proposed to reduce different logics into propositional logic, which captures boolean functions. Goel et al. [10] and Bryant et al. [6]. present methods to transform the logic of Equality with Uninterpreted Functions (EUF) into propositional logic. In [18] the theory of *separation predicates* is reduced to propositional logic. In [16] the EUF extended with constrained lambda expressions, ordering, and successor and predecessor functions, is translated to propositional logic. The idea of extending the theory of BDDs was recognized earlier by Groote and van de Pol [11], who presented an algorithm to transform EQ-BDDs to EQ-OBDDs, where EQ-BDDs represent the extension of BDDs with equalities. We extend the method for EQ-BDDs from [11] to a fragment of quantifier free logic FOL. We make a terminating set of rewrite rules on $(0, S, =)$-BDDs, resulting in a $(0, S, =)$-R-OBDD, such that all paths in the $(0, S, =)$-R-OBDD are satisfiable. This property enables us to check tautology, contradiction and satisfiability on $(0, S, =)$-R-OBDDs in constant time. At the end we present an algorithm through which any formula of the logic above is translated to an $(0, S, =)$-R-OBDD.

We define the set of terms as the closure of $\bar{V} = V \cup \{0\}$ (union of the sets of variables and zero) under successor. To be able to have an ordering on BDDs, we will need to define an ordering on terms of the logic. What is the appropriate ordering on terms? The answer, unfortunately, is not obvious. In [2] Chapter 3, two orderings which resulted in failed attempts are explained. One of them does

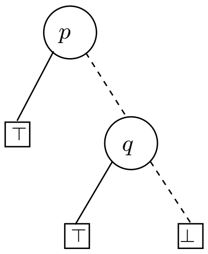

Figure 1. $ITE(p, \top, ITE(q, \top, \bot))$. *Solid* lines denote the left branch of the ITE (when their corresponding guard holds) and *dashed* lines represents their counterpart.

not provide termination, and the other does not omit all unsatisfiable paths.

The approach introduced here is in a sence a variant of [3], though our new ordering yields some simpler representation of the terms in the proof settings. Besides, it provides an alternative technique for the OBDD transformation. This, as a result can offer a different method for possible extensions of the background logic (notice that as mentioned above, finding the right order is not easy, and this problem would remain for bigger theories as well). In the current work, substitution rules are certainly different than those of the previous work. In addition, we also introduce an automatic way for transforming any formula (in our FOL fragment) into some Ordered BDD. We do this by means of a so called `sort` algorithm.

Road map. In Section 2, we describe BDDs, and give a formal syntax and semantics of $(0, S, =)$-BDDs. In Section 3 our transformation is presented, leading to the set of $(0, S, =)$-R-OBDDs. First a total and well-founded order on variables is assumed, and extended to a total well-founded order on equalities. Then the rewrite system is presented. Finally, we prove termination and satisfiability over all paths. Section 4 presents an algorithm with the same result as the given term rewrite system. Finally, Section 5 concludes with some remarks on implementation and possible applications.

2 Binary Decision Diagrams

A *binary decision diagram* [5] (BDD) represents a boolean function as a finite, rooted, binary, ordered, directed acyclic graph. The leaves of this graph are labeled $\bot$ and $\top$, and all internal nodes are labeled with boolean variables. A node with label p, left child L and right child R, written $ITE(p, L, R)$, represents the formula *if* p *then* L *else* R.

Given a fixed total order on the propositional variables, a BDD can be transformed to an *Ordered* binary decision diagram (OBDD), in which the propositions along all paths occur in increasing order, redundant tests ($ITE(p, x, x)$) don't occur, and the graph is maximally shared. For a fixed order, each boolean function is represented by a unique reduced OBDD (in the sequel we simply use OBDD to denote a reduced OBDD). For more information on that, one can see [5].

Example 2.1 Figure 1 illustrates a BDD representation of the following formula: $ITE(p, \top, ITE(q, \top, \bot))$ where p and q are propositional variables.

2.1 BDDs with Equality, Zero and Successor

In this section we introduce some basic notations and definitions. We also provide the syntax and semantics of BDDs extended with zero, successor and equality. For our purpose, the sharing information present in the graph is immaterial, so we formalize BDDs by terms (i.e. trees). We show that every formula is representable as a BDD.

We assume V is a set of variables, and define $\bar{V} = V \cup \{0\}$. Sets of terms, formulas, guards and BDDs are defined below:

Definition 2.2 Terms $t \in W$, formulas $\varphi \in \Phi$, guards $g \in G$ and $(0, S, =)$-BDDs $T \in B$ are defined by the following grammar (with $x \in V$):

$$\begin{aligned} t &::= 0 \mid x \mid S(t) \\ \varphi &::= \bot \mid \top \mid t = t \mid \neg\varphi \mid \varphi \wedge \varphi \mid ITE(\varphi, \varphi, \varphi) \\ g &::= \bot \mid \top \mid t = t \\ T &::= \bot \mid \top \mid ITE(g, T, T) \end{aligned}$$

A guard is *trivial* if it is $\bot$ or $\top$, and otherwise it is *non-trivial.* Here are some notations that we will use in this paper: In order to avoid confusion with the =-symbol in guards, we use $\equiv$ to identify syntactic equality between terms or formulas. Symbols $x, y, z, u, \ldots$ denote variables; $r, s, t, \ldots$ range over W; $\varphi, \psi, \ldots$ range over Φ; $f, g, \ldots$ range over guards. $\texttt{Var}(t)$ represents the variable occurring in term t. Furthermore, we will write $x \neq y$ instead of $\neg(x = y)$ and $S^m(t)$ for the m-fold application of S to t, so $S^0(t) \equiv t$ and $S^{m+1}(t) \equiv S(S^m(t))$. Note that each $t \in W$ is of the form $S^m(u)$, for some $m \in \mathbb{N}$ and $u \in \bar{V}$.

We will use some fixed interpretation for the above formulas: Terms are interpreted over the natural numbers ($\mathbb{N}$) and for formulas we use the classical interpretation over $\{0, 1\}$. Given a valuation $v : V \to \mathbb{N}$, we extend v homomorphically to terms and formulas as:

$$\begin{aligned} v(0) &= 0 \\ v(S(t)) &= 1 + v(t) \\ v(\bot) &= 0 \\ v(\top) &= 1 \\ v(s = t) &= 1, \text{ if } v(s) = v(t), \ 0, \text{ otherwise.} \\ v(\neg\varphi) &= 1 - v(\varphi) \\ v(\varphi \wedge \psi) &= \min(v(\varphi), v(\psi)) \\ v(ITE(\varphi, \psi, \chi)) &= v(\psi) \text{ if } v(\varphi) = 1, \ v(\chi) \text{ otherwise.} \end{aligned}$$

It is trivial that the value of a formula under any valuation function is either 0 or

1.

Given a formula φ, we say it is *satisfiable* if there exists a valuation $v : V \to \mathbb{N}$ such that $v(\varphi) = 1$; it is a *contradiction* otherwise. If for all $v : V \to \mathbb{N}$, $v(\varphi) = 1$, then φ is a *tautology*. Finally, if $v(\varphi) = v(\psi)$ for all valuations $v : V \to \mathbb{N}$, then φ and ψ are called *equivalent*. v satisfies φ (or equivalently φ holds under v) is denoted as: $v \models \varphi$.

Lemma 2.3 *Every formula in* Φ *is equivalent to at least one* $(0, S, =)$*-BDD.*

3 Representant-Ordered $(0, S, =)$-BDDs

The first step to make a BDD ordered, is to simplify all its guards, in isolation. Here, simplification on guards will be done by Definition 3.2. In Section 3.1 we present a new order on terms. In Definition 3.8 we define an ordering on guards. Notice that these definitions are different from those of [3]. Thereafter, we will introduce a term rewrite system. Using this system we simplify BDDs to their most reduced form, denoted as $(0, S, =)$-R-OBDD.

3.1 Definition of $(0, S, =)$-R-OBDDs

We consider a fixed total and well-founded ordering on V. Below we assume that the variables x, y and z are ordered as $x \prec y \prec z$.

Definition 3.1 [ordering definition] We extend $\prec$ to a total order on W:

- $0 \prec u$ for each element u of V
- $S^m(x) \prec S^n(y)$ if and only if $x \prec y$ or $(x \equiv y$ and $m < n)$ for each two elements $x, y \in \bar{V}$

As of now, we may use the term OBDD instead of $(0, S, =)$-R-OBDD, for simplicity.

Definition 3.2 Suppose g is a guard. By $g\downarrow$ we mean the normal form of g obtained after applying the following rewrite rules on it:

$$\begin{aligned}
x = x &\to \top \\
S(x) = S(y) &\to x = y \\
0 = S(x) &\to \bot \\
x = S^{m+1}(x) &\to \bot \qquad \text{for all } m \in \mathbb{N} \\
t = r &\to r = t \quad \text{for all } r, t \in W \text{ such that } r \prec t.
\end{aligned}$$

We call g simplified if it cannot be further simplified, i.e. $g \equiv g\downarrow$. A $(0, S, =)$-BDD T is called simplified if all guards in it are simplified.

Lemma 3.3 *If* $g \in G$ *is simplified to* g' *using Definition 3.2, then* g *and* g' *are equivalent.*

Next lemma shows possible shapes of a simplified guard. In contrast to [3] the smaller term sits on the left.

Lemma 3.4 *If g is a simplified guard, then it has one of the following shapes:*

- $S^m(0) = x$ *for some* $x \in V$
- $S^m(x) = S^n(y)$ *for some* $x, y \in V,\ x \prec y,\ m = 0$ *or* $n = 0$
- $\top$ *or* $\bot$

It is worth mentioning that as a result each guard has only one simplified form.

In order to be able to substitute one term by another, we may often need to upraise the atom which includes the term by applying some few additional successors, and then to do the replacement. Below, we explain our strategy for doing this:

Definition 3.5 Let $m \in \mathbb{N}$. For terms $r, t \in W$, a variable $y \in V$ and a guard $g \in G$ we define:

$$(r = t)\uparrow^m := S^m(r) = S^m(t) \quad \textit{(lifting)}$$

Definition 3.6 Suppose g is a simplified non-trivial guard, $y \in V$ and $t, r \in W$. We define:

$$g|_{r=S^m(y)} := \begin{cases} (g\uparrow^m [S^m(y) := r])\downarrow & \textit{if } y \textit{ occurs in } g \\ g & \textit{otherwise} \end{cases}$$

$$g|_{t \neq r} := \begin{cases} \bot & \textit{if } g \equiv (t = r)\downarrow \\ g & \textit{otherwise} \end{cases}$$

The following lemma shows the soundness of the operations above:

Lemma 3.7 *For any guard g and a positive natural number m, $g\uparrow^m$ and g are equivalent terms. Moreover, for a guard f, if $v \models f$ for some valuation v then $v(g) = v(g|_f)$.*

To have ordered BDDs, we need to impose some order on simplified guards. Below is what we use as the ultimate order on such guards:

Definition 3.8 [order] We define a total order $\prec$ on simplified guards as:

- $\bot \prec \top \prec g$, for all simplified guards g different from $\top$, $\bot$.
- $(S^p(x) = S^q(y)) \prec (S^m(u) = S^n(v))$ iff:

$$\begin{aligned} &i)\ x \prec u \text{ or} \\ &ii)\ x \equiv u,\ p < m \text{ or} \\ &iii)\ x \equiv u,\ p \equiv m\ , y \prec v \text{ or} \\ &iv)\ x \equiv u,\ p \equiv m,\ y \equiv v,\ q < n \end{aligned}$$

According to this definition $(r_1 = t_1) \prec (r_2 = t_2)$ iff $(r_1, t_1) \prec_{lex} (r_2, t_2)$, in which $\prec_{lex}$ is a lexicographic order on quadruples of the total, well-founded orders $(\bar{V}, \prec) \times (\mathbb{N}, <) \times (\bar{V}, \prec) \times (\mathbb{N}, <)$, and therefore it is well-founded and total. This way without getting into the structures of the involved terms, only by knowing the order between them, one could determine the order of the guards.

Now we can build the term rewrite system, which will be applied on $(0, S, =)$-BDDs and will generate an ordered version of them.

Definition 3.9 [$(0, S, =)$-R-OBDD] An $(0, S, =)$-R-OBDD (Representant-Ordered $(0, S, =)$-BDD) is a simplified $(0, S, =)$-BDD (i.e. all its guards are simplified) which is a normal form with respect to the following term rewrite system:

(i) $ITE(\top, T_1, T_2) \to T_1$

(ii) $ITE(\bot, T_1, T_2) \to T_2$

(iii) $ITE(g, T, T) \to T$

(iv) $ITE(g, ITE(g, T_1, T_2), T_3) \to ITE(g, T_1, T_3)$

(v) $ITE(g, T_1, ITE(g, T_2, T_3)) \to ITE(g, T_1, T_3)$

(vi) $ITE(g_1, ITE(g_2, T_1, T_2), T_3) \to ITE(g_2, ITE(g_1, T_1, T_3), ITE(g_1, T_2, T_3))$ *if* $g_1 \succ g_2$

(vii) $ITE(g_1, T_1, ITE(g_2, T_2, T_3)) \to ITE(g_2, ITE(g_1, T_1, T_2), ITE(g_1, T_1, T_3))$ *if* $g_1 \succ g_2$

(viii) *for every simplified* $(0, S, =)$*-BDD* C*, if* y *occurs in* g *and* $S^n(x) = S^m(y) \prec g$ *then*:
$ITE(S^n(x) = S^m(y),\ C[g], T\ ) \to ITE(S^n(x) = S^m(y),\ C[\ g|_{S^n(x)=S^m(y)}], T\ )$

In the viiith rule, one of m or n must be 0, since according to the assumption $S^n(x) = S^m(y)$ is a simplified guard (Lemma 3.4).

Obviously the result of applying any rule (of Definition 3.9) on a simplified BDD is a simplified BDD. The BDD which can no longer be simplified is called of *normal* form. In the sequel we show that each BDD has a normal form. The next lemma in immediate from this definition:

Lemma 3.10 *Suppose* $T \in B$ *is a* $(0, S, =)$*-BDD which becomes* T' *after applying any arbitrary rule of Definition 3.9 on it. Then* T *and* T' *are equivalent. As a result each* $(0, S, =)$*-BDD is equivalent with any of its normal forms.*

Example 3.11 Let $x \prec y \prec z$. Below we present our simplification technique over $ITE(S(y) = z, ITE(x = S^2(y), \top, \bot), \bot)$ (Figure 2).

$$\begin{array}{ll}
 & ITE(S(y) = z, ITE(x = S^2(y), \top, \bot), \bot) \\
\xrightarrow{6} & ITE(x = S^2(y), ITE(S(y) = z, \top, \bot), ITE(S(y) = z, \bot, \bot)) \\
\xrightarrow{3} & ITE(x = S^2(y), ITE(S(y) = z, \top, \bot), \bot) \\
\xrightarrow{8} & ITE(x = S^2(y), ITE(\{S^3(y) = S^2(z)[S^2(y) := x]\} \downarrow, \top, \bot), \bot) \\
\overset{substitution}{\equiv} & ITE(x = S^2(y), ITE(\{S(x) = S^2(z)\} \downarrow, \top, \bot), \bot) \\
\equiv & ITE(x = S^2(y), ITE(x = S(z), \top, \bot), \bot)
\end{array}$$

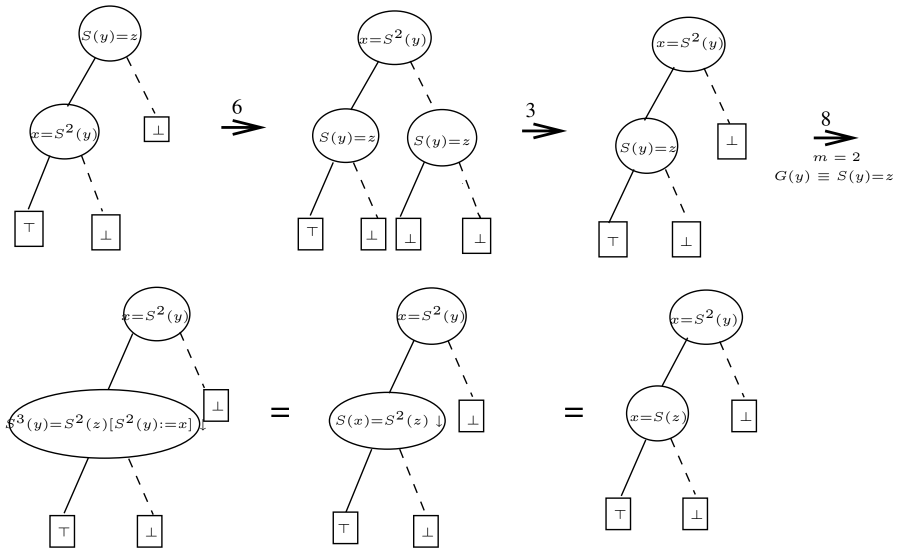

Figure 2. Derivation in Example 3.11

3.2 Termination

To show that our system is terminating we first prove some properties on $\prec$.

Lemma 3.12 *Let $f \equiv S^n(x) = S^m(y)$ and $g \equiv S^k(v) = S^l(w)$. If $f \prec g$ and $f \equiv f\downarrow$ and $g \equiv g\downarrow$ and $y \in \{v, w\}$, then $g|_f \prec g$.*

Definition 3.13 [recursive path order for BDDs] Let S and T be simplified BDDs. Then $S \equiv f(S_1, S_2) \succ_{rpo} g(T_1, T_2) \equiv T$ if and only if

(I) $S_1 \succeq_{rpo} T$ or $S_2 \succeq_{rpo} T$; or

(II) $f \succ g$ and $S \succ_{rpo} T_1, T_2$; or

(III) $f \equiv g$ and $S \succ_{rpo} T_1, T_2$ and either $S_1 \succ_{rpo} T_1$, or ($S_1 \equiv T_1$ and $S_2 \succ_{rpo} T_2$).

Here $x \succeq_{rpo} y$ means that $x \succ_{rpo} y$ or $x \equiv y$, and $S \succ_{rpo} T_1, T_2$ is shorthand for $S \succ_{rpo} T_1$ and $S \succ_{rpo} T_2$.

This definition forces an order on BDDs, as shown in [4] Chapter 6.

Lemma 3.14 *Let f, g be two simplified guards, such that $f \prec g$, and C is a $(0, S, =)$-BDD. If g occurs at least once in C, then $C[g] \succ_{rpo} C[f]$.*

Proof This holds because of the monotonical behaviour of $\succ_{rpo}$ ([4] Chapter 6).□

Next lemma shows that if a sub-tree of a BDD is replaced by a smaller tree, then the whole tree will become smaller.

Lemma 3.15 *If T is a simplified BDD, and S a sub BDD of it, and S' is another simplified BDD where $S \succ_{rpo} S'$, then if we replace S by S' in T and derive T', we will have $T \succ_{rpo} T'$*

Proof It is easy by induction on the structure of T and Definition 3.13(III). □

Applying any rewrite rule on a BDD results in a smaller BDD with respect to the $\succ_{rpo}$ order:

Lemma 3.16 *Each rewrite rule is contained in $\succ_{rpo}$.*

Proof The only non straightforward case is rule 8, wherefore we use Lemmas 3.12 and 3.14. □

Now, we can prove that our term rewrite system (Definition 3.9) always terminates:

Theorem 3.17 (Termination) *The rewrite system defined in Definition 3.9 is terminating on simplified $(0, S, =)$-BDDs.*

Proof According to the previous lemma all rewrite rules are contained in $\succ_{rpo}$. This implies termination, because $\succ_{rpo}$ is a reduction order, i.e. well-founded, and closed under substitutions and contexts [4] Chapter 6. □

As an immediate result of termination, each BDD has a normal form:

Corollary 3.18 *Every $(0, S, =)$-BDD is equivalent to at least one $(0, S, =)$-R-OBDD.*

3.3 Satisfiability of paths in $(0, S, =)$-R-OBDDs

We consider α, β, γ to represent finite sequences of (possibly negated) guards. Let us denote the empty sequence by ε and the concatenation of sequences α and β by $\alpha.\beta$.

Definition 3.19 We define the set of *Paths* in a $(0, S, =)$-BDD by:

- $Pat(\top) = Pat(\bot) = \varepsilon$
- $Pat(ITE(g, T_1, T_2)) = \{g.\alpha \mid \alpha \in Pat(T_1)\} \cup \{\neg g.\beta \mid \beta \in Pat(T_2)\}$

α is an *ordered* path if it occurs in some $(0, S, =)$-OBDD. We are going to prove that al paths in an OBDD are satisfiable.

The next two lemmas give syntactical properties on OBDDs, which can be used for proving satisfiability of each path in an OBDD.

Lemma 3.20 *Let $T \equiv ITE(S^m(x) = S^n(z), T_1, T_2)$ be a $(0, S, =)$-R-OBDD. Let α be a path in T_2 and $H = \{S^{j_i}(x) = r_i \mid 1 \leq i \leq k\}$ be the set of all positive guards on α which have x as their left-hand side variable. Then for each positive guard on α with a variable which occurs in an atom in H, we can conclude that the guard belongs to H.*

The next lemma says that in an OBDD, the left-most variable of each guard will not occur at the right-hand side of any guard underneath it.

Lemma 3.21 *Let $T \equiv ITE(S^m(x) = r, T_1, T_2)$ be a $(0, S, =)$-R-OBDD. Then for all guards $s = t$ occurring in T_1 or T_2 we have $x \not\equiv \texttt{Var}(t)$ (i.e. $t \not\equiv S^k(x)$ for any k).*

Now we prove the second main theorem, which is satisfiability of each path in an OBDD.

Theorem 3.22 (Paths are satisfiable) *Each path in a $(0, S, =)$-R-OBDD is satisfiable.*

Satisfiability of paths in OBDDs results in:

Corollary 3.23

- $\top$ *is the only tautological* $(0, S, =)$*-R-OBDD.*
- $\perp$ *is the only contradictory* $(0, S, =)$*-R-OBDD.*
- *Every other* $(0, S, =)$*-R-OBDD is satisfiable.*

Proof Each path in a tautological OBDD should end in a $\top$. Because if T is a tautological OBDD, containing a path α which ends in a $\perp$, then according to Theorem 3.22, there is a valuation v which satisfies α. But then $v(T) = 0$, which is impossible since T is a tautology. Therefore, if T has more than one leaf, rule 3 of Definition 3.9 is applicable on a tautological OBDD which is not $\top$, and this contradicts the orderedness. So $T \equiv \top$. Similarly for a contradictory one. □

4 The Transformer Algorithm

In this section we present an algorithm to transform any formula in our logic into an equivalent OBDD. One could consider an algorithm which applies the rules of our term rewrite system one by one, on the given formula, until it reaches an OBDD. Although this is possible, it is not efficient, since in the process a lot of unnecessary cases will be checked on the formula, until it can reach a normal form. We instead extend the algorithm in [12], which is based in Shannon's expansion with the smallest equation $x = y$:

$$\varphi \Longleftrightarrow (x = y \wedge \varphi|_{x=y}) \vee (x \neq y \wedge \varphi|_{x \neq y}).$$

Since the set of BDDs is a subset of the set of formulas,so in this section we may use BDDs wherever we work with set of formulas. In order to simplify formulas, we extend the reducing method in Definition 3.6 over all formulas:

Definition 4.1 For any formula φ and any simplified literal (guard) l, we define:

$$\begin{aligned}(\neg\varphi)|_l &:= \neg(\varphi|_l)\\ (\varphi \wedge \psi)|_l &:= (\varphi|_l) \wedge (\psi|_l)\\ ITE(\varphi_1, \varphi_2, \varphi_3)|_l &:= ITE((\varphi_1)|_l, (\varphi_2)|_l, (\varphi_3)|_l)\end{aligned}$$

As a result the corresponding lemma (i.e. Lemma 3.7) is extendible to all formulas, as well:

Lemma 4.2 *If $v \models l$ for a literal l and a valuation v, then $v(\varphi) \equiv v(\varphi|_l)$.*

Proof By induction on the structure of φ and using Lemma 3.7, it is straightforward. □

Applying an $|_l$ operation on a BDD will not increase the size of the BDD.

Lemma 4.3 *Let T be a simplified BDD. Suppose l is a simplified guard possibly occurring on T. If l is no bigger than the guards occurring on T then $T \succeq_{rpo} T|_l$.*

Proof According to Lemma 3.12 and Definition 3.6, the guards do not get bigger. Now, by using induction over the structure of T and Definitions 4.1, the proof is trivial. □

Intuitively, the operation $|_l$ replaces the rightmost variable occurring in the (positive) literal l with the left most term sitting in l. So, one would expect that the rightmost variable would not appear in the formula after this operation is applied:

Lemma 4.4 *Let l be a simplified guard of the form $r = S^m(y)$ in which $y \in V$. Then $y \notin T|_l$.*

Proof It is trivial by Definitions 4.1 and 3.6. □

In the next definition we generalize the simplification method over guards in Definition 3.2 to all formulas, because in practice we also need to make the guards, occurring inside BDDs and formulas, smaller if possible:

Definition 4.5 We extend the simplification rules of Definition 3.2 to all formulas below:

$$\begin{array}{rlrl}
g \longrightarrow g\downarrow & \textit{(if } g \textit{ is not simplified)} & & \\
\neg g \longrightarrow \neg(g\downarrow) & \textit{(if } g \textit{ is not simplified)} & & \\
(\varphi \wedge \bot) \longrightarrow \bot & & (\bot \wedge \varphi) \longrightarrow \bot & \\
(\varphi \wedge \top) \longrightarrow \varphi & & (\top \wedge \varphi) \longrightarrow \varphi & \\
(\neg\top) \longrightarrow \bot & & (\neg\bot) \longrightarrow \top & \\
ITE(\top,\ \varphi,\ \psi) \longrightarrow \varphi & & ITE(\bot,\ \varphi,\ \psi) \longrightarrow \psi & \\
ITE(g,\ \psi,\ \psi) \longrightarrow \psi & & &
\end{array}$$

$\varphi \Downarrow$ represents a most simplified version of φ.

Similar to the Lemma 3.3, it can be proved that:

Lemma 4.6 *If φ is simplified to φ' using Definition 4.5, then φ and φ' are equivalent.*

Lemmas 4.2 and 4.6 together will lead to:

Lemma 4.7 *If $v \models l$ for a literal l and a valuation v, then $v(\varphi) \equiv v((\varphi|_l)\Downarrow)$.*

Theorem 4.8 *Let T be a simplified BDD. If g is the smallest guard occurring in T, then $T \succ_{rpo} (T|_l)\Downarrow$ for $l \in \{g, \neg g\}$.*

Proof According to the last case of Definition 4.1, in order to calculate $T|_l$ we could apply the operation $|_l$ to its sub-trees. Let us consider a case distinction over l:

- $l \equiv g$. g occurs in T. Hence there is a sub-tree of T of the form $ITE(g, T_1, T_2)$. Let T' be one of these. Therefore:

$$\begin{aligned}
(T'|_g)\Downarrow &\equiv ITE(g|_g, T_1|_g, T_2|_g)\Downarrow && \text{(Definition 4.1)}\\
&\equiv (T_1|_g)\Downarrow && \text{(Definition 4.5)}\\
&\preceq_{rpo} T_1|_g && \text{(using Lemma 3.16 for Definition 4.5)}\\
&\preceq_{rpo} T_1 && \text{(Lemma 4.3)}\\
&\prec_{rpo} T' && \text{(Lemma 3.13(I))}
\end{aligned}$$

  Above, $A \preceq_{rpo} B$ means $B \succeq_{rpo} A$. Now according to Lemma 3.15, our original tree is bigger. Meaning that $T \succ_{rpo} (T|_l)\Downarrow$. In this last conclusion we also used Lemma 3.16 and Lemma 4.3 implicitly.

- $l \equiv \neg g$. Similar. □

The following function, called `sort`, is introduced to take the smallest guard occurring in a formula and bring it to the topmost place, and sort and simplify the formula afterwards.

Definition 4.9 We define a function `sort` on simplified formulas, which sorts and simplifies the formulas with respect to their smallest guard.

- $\texttt{sort}(\bot) \equiv \bot$
- $\texttt{sort}(\top) \equiv \top$
- Let g be the smallest guard occurring (positively or negatively) in φ. Then

$$\texttt{sort}(\varphi) \equiv \begin{cases} \texttt{sort}(\varphi|_g \Downarrow) & \text{if } \texttt{sort}(\varphi|_g \Downarrow) \equiv \texttt{sort}(\varphi|_{\neg g} \Downarrow)\\ ITE(g,\ \texttt{sort}(\varphi|_g \Downarrow),\ \texttt{sort}(\varphi|_{\neg g} \Downarrow)) & \text{otherwise} \end{cases}$$

Since each BDD is a formula, therefore the function `sort` can be recursively used. The following lemmas are immediately derived from this definition.

Theorem 4.10 `sort`(φ) *is terminating over any formula* φ.

Proof Using induction over the structure of φ. □

Lemma 4.11 *The set of all variables in* `sort`(φ) *is a subset of the set of all variables in* φ.

Lemma 4.12 `sort`(φ) *is a BDD for any formula* φ. *Moreover* φ *is equivalent to* `sort`(φ), *i.e.* $v(\varphi) \equiv v(\texttt{sort}(\varphi))$ *for any valuation* v.

One application of `sort` does not always yield an OBDD:

Example 4.13 Let $\varphi \equiv ITE(x = S(z), ITE(y = z, \top, \bot), \bot)$; we show how the `OBDD` algorithm finds an equivalent OBDD for this φ.

φ is simplified already, so that $\psi = \varphi \Downarrow = \varphi$. Now $\psi \neq \bot$, hence we must enter the **while**-*loop: we first need to calculate* $\texttt{sort}(\varphi)$. *$x = S(z)$ is the smallest guard.* $\texttt{sort}(\psi|_{x=S(z)}) \equiv ITE(x = S(y), \top, \bot)$ *and* $\texttt{sort}(\psi|_{x\neq S(z)}) \equiv \bot$. *Hence*

$$\begin{aligned}\texttt{sort}(\psi) &= ITE(x = S(z),\ \texttt{sort}(\psi|_{x=S(z)}),\ \texttt{sort}(\psi|_{x\neq S(z)}))\\ &= ITE(x = S(z),\ ITE(x = S(y), \top, \bot), \bot) \qquad \textit{(above)}\end{aligned}$$

Now $\psi \neq$ $\texttt{sort}(\psi)$, hence we must repeat the **while**-*loop: $x = S(y)$ is the smallest guard.* $\texttt{sort}(\psi|_{x=S(y)}) \equiv ITE(x = S(z), \top, \bot)$ *and* $\texttt{sort}(\psi|_{x\neq S(y)}) \equiv \bot$. *Hence*

$$\begin{aligned}\texttt{sort}(\psi) &= ITE(x = S(y),\ \texttt{sort}(\psi|_{x=S(y)}),\ \texttt{sort}(\psi|_{x\neq S(y)}))\\ &= ITE(x = S(y),\ ITE(x = S(z), \top, \bot), \bot) \qquad \textit{(above)}\end{aligned}$$

Again $\psi \neq$ $\texttt{sort}(\psi)$, hence we must repeat the **while**-*loop: $x = S(y)$ is the smallest guard.* $\texttt{sort}(\psi|_{x=S(y)}) \equiv ITE(x = S(z), \top, \bot)$ *and* $\texttt{sort}(\psi|_{x\neq S(y)}) \equiv \bot$. *Hence*

$$\begin{aligned}\texttt{sort}(\psi) &= ITE(x = S(y),\ \texttt{sort}(\psi|_{x=S(y)}),\ \texttt{sort}(\psi|_{x\neq S(y)}))\\ &= ITE(x = S(y),\ ITE(x = S(z), \top, \bot), \bot) \qquad \textit{(above)}\end{aligned}$$

This time $\psi =$ $\texttt{sort}(\psi)$, hence we must leave the **while**-*loop, and stop with $\psi = ITE(x = S(y),\ ITE(x = S(z), \top, \bot), \bot)$ as the outcome.*

As a result, we need to have some algorithm that can recursively apply `sort` until a fixed point is reached. This is what we look for with the next algorithm:

Definition 4.14 The following algorithm, generates a $(0, S, =)$-R-OBDD for any formula:

```
OBDD($\varphi$)
    $\psi$:= $\varphi\Downarrow$ ;
    $\varphi$:= $\bot$ ;
    while $\varphi \neq \psi$ do
        $\varphi$:= $\psi$ ;
        $\psi$:= sort($\psi$) ;
    od
    return $\psi$
```

Later, in Theorem 4.17, we will prove that this algorithm always returns an OBDD. The following two lemmas describe some properties of the `sort` function which will be used to prove termination of the `OBDD` algorithm.

Lemma 4.15 *Let T be any simplified BDD. Then:*

(i) $\texttt{sort}(T)\Downarrow \equiv \texttt{sort}(T)$.

(ii) $T \succeq_{rpo} \texttt{sort}(T)$.

One can easily check that the `OBDD` algorithm (i.e. Definition 4.14) terminates as soon as $T \equiv \texttt{sort}(T)$. Below, we claim that such a T is ordered.

Theorem 4.16 *If T is a simplified BDD for which $T \equiv \texttt{sort}(T)$, then T is an ordered BDD.*

Now we can prove the main theorem which is termination of the algorithm:

Theorem 4.17 (Termination of the algorithm) *The algorithm given in Definition 4.14 is terminating, and $\texttt{OBDD}(\varphi)$ is a $(0, S, =)$-R-OBDD equivalent to φ, for any given formula φ.*

Proof According to the Lemma 4.12 $\texttt{sort}(\varphi)$ will be a BDD equivalent to φ. The $\succeq_{rpo}$ ordering is well-founded, therefore using Lemma 4.15(ii) we know that after finitely many steps we will reach a fixed point of $\texttt{sort}(\psi) \equiv \psi$, for some ψ. Now, using Theorem 4.16, this ψ is an ordered BDD, which on the other-hand is the outcome of the $\texttt{OBDD}(\varphi)$ (by definition), and is equivalent to φ (by Lemma 4.12). □

Below we show, on a simple example, how the `OBDD` algorithm operates of formulas.

Example 4.18 We consider the formula of Example 3.11:

$$\varphi \equiv ITE(S(y) = z, ITE(x = S^2(y), \top, \bot), \bot).$$

φ is simplified already, therefore $\varphi \Downarrow \equiv \varphi$ and $\psi := \varphi$. We enter the loop since $\bot \neq \varphi$. Here, $\texttt{sort}(\psi)$ is:

$$ITE(x = S^2(y),\ \texttt{sort}(\varphi|_{x=S^2(y)}),\ \texttt{sort}(\varphi|_{x\neq S^2(y)})).$$

The innermost formulas are to be computed here. We will have:

$$\texttt{sort}(\varphi|_{x=S^2(y)}) \equiv ITE(x = S(z), \top, \bot) \text{ and } \texttt{sort}(\varphi|_{x\neq S^2(y)}) \equiv \bot.$$

Substituting these two in the formula, we obtain

$$\texttt{sort}(\psi) \equiv ITE(x = S^2(y), ITE(x = S(z), \top, \bot), \bot).$$

Once more applying the `sort` function over this formula will result in $\texttt{sort}(\texttt{sort}(\psi)) \equiv \texttt{sort}(\psi)$. This is a fixed point, therefore $\texttt{OBDD}(\varphi)$ is $ITE(x = S^2(y), ITE(x = S(z), \top, \bot), \bot)$. This is an OBDD for the original formula $ITE(S(y) = z, ITE(x = S^2(y), \top, \bot), \bot)$.

5 Conclusion

In this paper we provided another sound and complete method for verification of a fragment of quantifier-free FOL. This fragment contains equality with zero and successors. We introduced an algorithm which transforms each formula into (one of) its equivalent ordered BDD (s). In an Ordered BDD all paths are satisfiable so a formula is a tautology (contradictory) if and only if the derived OBDD is q $\top$ ($\perp$), and is satisfiable otherwise.

Although the logic that we tackled is rather small, many verification problems can be expressed in this logic. A lot of research has directed towards model checking techniques for verification of large systems, with huge state spaces. In this plot, BDDs have also been of use [8,9]. Among these, symbolic model checking [15,14] have been of much interest. The idea is to use OBDDs for reducing the size of the representive state space. Although propositional logic have been much into considerations, our technique has the ability to provide a more expressive logic. This would prevent necessary obligations for finding proper transformation functions from bigger languages into propositional logic [13]. SMT solvers or also UPPAAL which is a tool used for verification of real time systems, seem closer to our purpose. UPPAAL uses separation logic for modeling real time systems. Formulas are sets of constraints over expressions like $x > y$, $x \leq 6 + z$ or $2 < x \leq 6$, etc.

Another line of research could be extension of BDD-method (current results) to other algebras. Some interesting extensions are incorporation of addition (+), or an investigation of other free algebras (such as LISP-list structures based on null and cons).

6 Acknowledgement

I would like to thank Jaco van de Pol for his support and good comments on the paper.

References

[1] W. Ackermann. *Solvable Cases of the Decision Problem.* Studies in Logic and the Foundations of Mathematics. North-Holland, 1954.

[2] B. Badban. *Verification Techniques for Extensions of Equality Logic.* PhD thesis, Amsterdam Vrij University, 2006.

[3] B. Badban and J.C. van de Pol. Zero, successor and equality in binary decision diagrams. *Annals of Pure and Applied Logic*, 133(1-3):101–123, 2005.

[4] M.A. Bezem, J.W. Klop, and R.C. de Vrijer, editors. *Term Rewriting Systems.* Cambridge University Press, 2003.

[5] R.E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[6] 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*, 2001.

[7] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[8] Stefan Edelkamp. Heuristic search planning with bdds. In *PuK*, 2000.

[9] Stefan Edelkamp and Peter Kissmann. Limits and Possibilities of BDDs in State Space Search. In *AAAI*, pages 1452–1453, 2008.

[10] A. Goel, K. Sajid, H. Zhou, and A. Aziz. BDD based procedures for a theory of equality with uninterpreted functions. In *Proc. CAV*, 1998.

[11] J.F. Groote and J.C. van de Pol. Equational binary decision diagrams. In *Proc. of LPAR 2000*, pages 161–178, 2000.

[12] J.F. Groote and J.C. van de Pol. Equational binary decision diagrams. In *Proc. Conference on Logic for Programming and Automated Reasoning*, 2000.

[13] Shuvendu K. Lahiri, Randal E. Bryant, and Byron Cook. A symbolic approach to predicate abstraction. In *CAV*, pages 141–153, 2003.

[14] S.K. Lahiri, T. Ball, and B. Cook. Predicate abstraction via symbolic decision procedures. *CoRR'06*, abs/cs/0612003.

[15] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, 1992.

[16] G. Nelson and D.C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.

[17] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small domains instantiations. In *Proc. CAV*, pages 455–469, 1999.

[18] O. Strichman, S.A. Seshia, and R.E. Bryant. Deciding separation formulas with SAT. In *Proc. CAV*, pages 209–222, 2002.