An Algorithm to Verify Formulas by means of (0,S,=)-BDDs

Bahareh Badban *Jaco van de Pol*
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
{Bahareh.Badban,Jaco.van.de.Pol}@cwi.nl

Abstract

In this article we provide an algorithm to verify formulas of the fragment of first order logic, consisting of quantifier free logic with zero, successor and equality. We first develop a rewrite system to extract an equivalent Ordered $(0, S, =)$-BDD from any given $(0, S, =)$-BDD. Then we show completeness of the rewrite system. Finally we make an algorithm with the same result as the rewrite system. Given an Ordered $(0, S, =)$-BDDs we are able to see in constant time whether the formula is a tautology, a contradiction, or only satisfiable.

Keywords: *Algorithm, Decision Diagrams, Equality, Theorem proving, Decision Procedure, Natural numbers.*

1 Introduction

BDDs (binary decision diagrams) represent boolean functions as directed acyclic graphs. BDDs are of value for validating formulas in propositional logic. Although BDDs are a great tool to verify formulas, a representative BDD of a formula can be exponentially large. In [4] OBDDs (*Ordered* BDDs) are reduced BDDs which obey some ordering on boolean variables. A boolean function is satisfiable if and only if its unique OBDD representation does not correspond to 0.

To date many attempts have been made to verify the satisfiability of different logics using the BDD approach. With this intention, several methods have been proposed to reduce these logics into propositional logic, which captures boolean functions. Goel et al. [7] and Bryant et al. [5]. present methods to transform the logic of Equality with Uninterpreted Functions (EUF) into propositional logic. In [10] the theory of *separation predicates* is reduced to propositional logic. In [9] the EUF extended with constrained lambda expressions, ordering, and successor and predecessor functions, is translated to propositional logic.

What we do in this paper is in the opposite direction, which is extending the theory of BDDs, with the intention of enriching it to enable us to verify more expressive logics directly, with no need for any reduction techniques. The idea of extending the theory of BDDs was recognized earlier by Groote and van de Pol [8], 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 [8] to a more expressive hierarchy of first order logic, which consists of quantifier free logic with zero, successor and equality, called $(0, S, =)$-BDDs. We make a terminating set of rewrite rules on $(0, S, =)$-BDDs, resulting in a $(0, S, =)$-OBDD, such that all paths in the $(0, S, =)$-OBDD are satisfiable. This property enables us to check tautology, contradiction and satisfiability on $(0, S, =)$-OBDDs in constant time. At the end we present an algorithm through which any formula of the logic above is translated to an $(0, S, =)$-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. Finding an appropriate ordering on terms is not an easy task. In [2] you will find two orderings resulting in failed attempts. One of those does not provide termination, and the other does not omit unsatisfiable paths necessarily.

In [3] we also describe a (different) rewrite system to transform $(0, S, =)$-BDDs into $(0, S, =)$-OBDDs. There are two main differences between what we do here and the work in [3]. First, here we always place

the smaller term of an equality in an $(0, S, =)$-BDD at the left-hand side. This minor change requires a different ordering on equalities and a different substitution rule in the rewrite system, to provide termination. Second, here we make an algorithm, by which we automatically obtain the equivalent $(0, S, =)$-OBDD of a given formula (this algorithm is not present in [2]). The technical report [2] contains both the work in [3] and the work presented here. This algorithm was adapted and extended from [8].

In Section 2, we first shortly introduce BDDs, and then give a formal syntax and semantics of $(0, S, =)$-BDDs. In Section 3 our transformation is presented, leading to the set of $(0, S, =)$-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 rewrite system. Finally, Section 5 concludes with some remarks on implementation and possible applications.

2 BDDs with Zero, Successor and Equality

A *Binary Decision Diagram* [4] (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 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 OBDD. 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.

As it is described in [8], Ordered EQ-BDDs are not necessarily unique, so we will not make any attempt to obtain a unique representation in ordered form. In this section, we provide the syntax and semantics of BDDs extended with zero, successor and equality. For our purposes, the sharing information present in a graph is not important, so we formalize $(0, S, =)$-BDDs by terms (i.e. trees). We view $(0, S, =)$-BDDs as a restricted subset of formulas, and show that every formula is representable as $(0, S, =)$-BDD. Of course, any implementation should work with the shared representation.

Assume V is a set of variables, and define $\bar{V} = V \cup \{0\}$. We define sets of terms, formulas, guards and BDDs as follows.

Definition 1 *The sets of terms* (W)*, formulas* (Φ)*, guards* (G) *and* $(0, S, =)$*-BDDs* (B) *are defined as below:*

$$\begin{array}{rcl} W & ::= & 0 \mid V \mid S(W) \\ \Phi & ::= & \bot \mid \top \mid W = W \mid \neg\Phi \mid \Phi \wedge \Phi \mid ITE(\Phi, \Phi, \Phi) \\ G & ::= & \bot \mid \top \mid W = W \\ B & ::= & \bot \mid \top \mid ITE(G, B, B) \end{array}$$

We will use the following conventions: letters $x, y, z, u, \ldots$ denote variables; $r, s, t, \ldots$ will range over W; $\varphi, \psi, \ldots$ range over Φ. 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 $S^0(t) = t$ and $S^{m+1}(t) = S(S^m(t))$. Note that each $t \in W$ is of the form $S^m(u)$, for some $m \in \mathbb{N}$ and $u \in \bar{V}$. Finally, throughout this paper we will use $\equiv$ to denote syntactic equality between terms or formulas, in order to avoid confusion with the $=$-symbol in guards.

We will use a fixed interpretation of the above formulas throughout this paper. Terms are interpreted over the natural numbers $(\mathbb{N})$ and for formulas we use classical interpretation over $\{0, 1\}$. In particular, ITE denotes the If-Then-Else function. Given a valuation $v : V \to \mathbb{N}$, we extend v homomorphically to terms and formulas, please refer to [2]. 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*.

Lemma 2 *Every formula defined above is equivalent to at least one* $(0, S, =)$*-BDD.*

Proof. We use induction on the Formula. More details are in [2]. ■

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

We now introduce a total ordering on guards. Sorting the guards along the paths of a BDD according to this order, leads to the set of *Ordered* Binary Decision Diagrams ($(0, S, =)$-OBDDs). Next we prove that all $(0, S, =)$-BDDs (and hence all formulas) can be transformed to $(0, S, =)$-OBDDs by rewriting. Finally, we show that all paths in $(0, S, =)$-OBDDs have a special property, which make them well suited for deciding satisfiability and contradiction of propositional formulas over zero, successor and equality.

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

From now on, we use the term "BDD" as an abbreviation for "$(0, S, =)$-BDD". In this section, we define the set of *Ordered* BDDs. Now, before applying an ordering on BDDs, we need some ordering on guards and to define the latter we use a total ordering on the variables. In the sequel, we consider a fixed total and well-founded order on V (for instance $x \prec y \prec z$).

Definition 3 *(ordering definition) We extend $\prec$ to an order on W:*

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

We use term rewriting systems (TRS), being collections of rewrite rules, in order to specify reductions on guards and BDDs. The reduction relation induced by such a system is the closure of the rules under substitution and context. See [1] for a formal definition. A *normal form* is a term to which no rule applies. A TRS is terminating if all its reduction sequences are finite.

The first step to make a BDD ordered, is to simplify all its guards in isolation. Simplification on guards is defined by the following rules:

Definition 4 *Suppose g is a guard. By $g\downarrow$ we mean the normal form of g obtained after applying the following TRS (Term Rewriting System) simplification rules on it:*

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

note that this is a major distinction with [3].

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. An immediate consequence of the last definition is the following:

Corollary 5 *Each simplified non-trivial guard has one of the following forms:*

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

Definition 6 *Assume m is a natural number, define:*

$$(r = t)^m := S^m(r) = S^m(t) \qquad \text{for each } r, t \in W$$

Using the definition above we define:

Definition 7 *Suppose g is a simplified non-trivial guard, $y \in V$ and $t, r \in W$. Regarding the Definition 6, We define:*

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

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

As it was mentioned before, to impose an ordering on BDDs, first we need a total ordering on guards. Since we are going to deal with simplified guards, we limit our definition to the simplified guards.

Definition 8 *(order on simplified guards) We define total order $\prec$ on simplified guards as below*

- $\bot \prec \top \prec g$, *for all guards* g, *different from* $\top$, $\bot$.
- $(r_1 = t_1) \prec (r_2 = t_2)$, *iff* $(r_1, t_1) \prec_{lex} (r_2, t_2)$.

Definition 9 *An* $(0, S, =)$*-OBDD (Ordered* $(0, S, =)$*-BDD) is a simplified* $(0, S, =)$*-BDD which is a normal form regarding to the following term rewrite system:*

1. $ITE(\top, T_1, T_2) \to T_1$
2. $ITE(\bot, T_1, T_2) \to T_2$
3. $ITE(g, T, T) \to T$
4. $ITE(g, ITE(g, T_1, T_2), T_3) \to ITE(g, T_1, T_3)$
5. $ITE(g, T_1, ITE(g, T_2, T_3)) \to ITE(g, T_1, T_3)$
6. $ITE(g_1, ITE(g_2, T_1, T_2), T_3) \to ITE(g_2, ITE(g_1, T_1, T_3), ITE(g_1, T_2, T_3))$
   provided $g_1 \succ g_2$
7. $ITE(g_1, T_1, ITE(g_2, T_2, T_3)) \to ITE(g_2, ITE(g_1, T_1, T_2), ITE(g_1, T_1, T_3))$
   provided $g_1 \succ g_2$
8. *for every simplified* $(0, S, =)$*-BDD* C, *we have* 8_C :
   $ITE(S^n(x) = S^m(y),\ C[g], T\ ) \to ITE(S^n(x) = S^m(y),\ C[\ g|_{S^n(x)=S^m(y)}], T\ )$
   provided y *occurs in* g *and* $S^n(x) = S^m(y) \prec g$

In case 8 of the definition above always one of m or n is 0 (Corollary 5). Rules 1–7 are the normal rules for simplifying BDDs for plain propositional logic, which remove redundant tests, and ensure that guards along paths occur in increasing order. Rule 8 allows to substitute equals for equals. This is needed to take care of transitivity of equality. Other properties of equality, such as reflexivity, symmetry, and injectivity of successor, are dealt with by the simplification rules.

Example 10 *Let* $x \prec y \prec z$ *(See Figure 1)*

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

3.2 Termination

Now we come to the first main claim that every BDD with zero, successor and equality, has a normal form with respect to the TRS above, which means each given BDD has at least one equivalent OBDD. It suffices to prove termination: We apply TRS rules to a given BDD, until we reach a normal form after a finite number of steps, which is guaranteed by termination. The so derived BDD is the OBDD.

We prove termination by means of a powerful tool, the *recursive path ordering* ($\prec_{rpo}$) [6, 11]. This is a standard way to extend a (total) well-founded order on a set of labels to a (total) well-founded order on trees over these labels. To this end, we view guards as labels, ordered by Definition 8, and BDDs are viewed as *binary* trees, so $ITE(g, T_1, T_2)$ corresponds to the tree $g(T_1, T_2)$.

Definition 11 *(recursive path order for BDDs).* $S \equiv f(S_1, S_2) \succ_{rpo} g(T_1, T_2) \equiv T$ *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:* $x \succ_{rpo} y$ *or* $x \equiv y$, *also* $S \succ_{rpo} T_1, T_2$ *means:* $S \succ_{rpo} T_1$ *and* $S \succ_{rpo} T_2$.

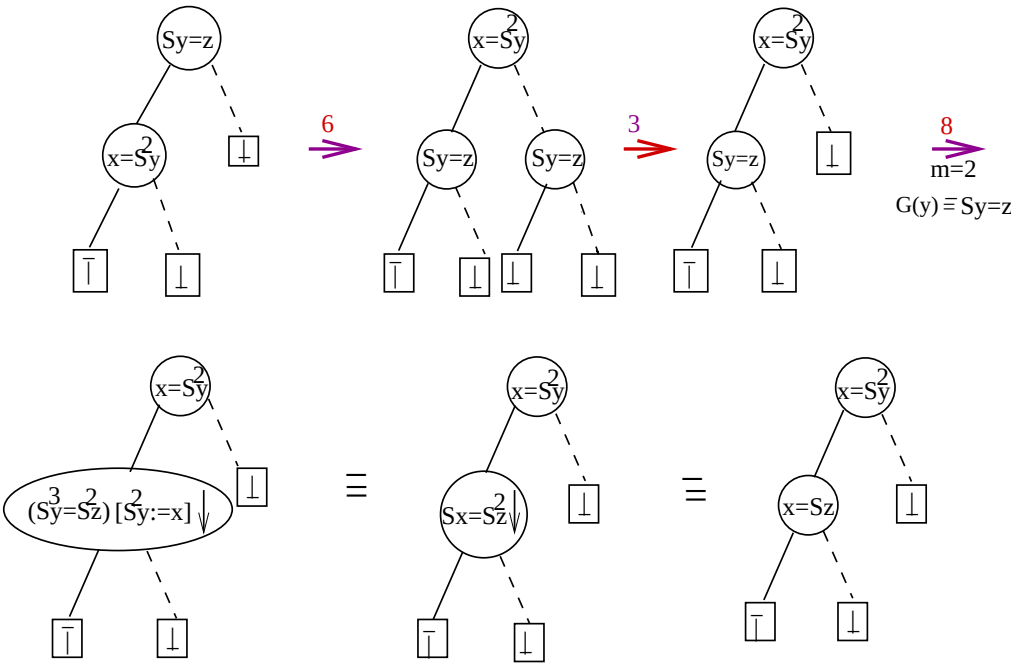

Figure 1: Derivation of Example 10

This definition yields an order, as it is shown in [11]. In order to prove termination, we will show that each rewrite rule (of Definition 9) is indeed a reduction rule regarding $\succ_{rpo}$. The next lemma will be very helpful to show that this reduction property really holds.

Lemma 12 *Each rewrite rule is contained in* $\succ_{rpo}$.

Proof. Please look at [2]. ■

Termination of the rewrite system is an immediate consequence of this Lemma.

Theorem 13 *The rewrite system defined in Definition 9 is terminating.*

Proof. Regarding Lemma 12 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 [11]. ■

This theorem says that by repeated applications of the rewrite rules on an arbitrary simplified BDD, after finitely many iterations we will obtain the normal form of it, which is its equivalent ordered form, so

Corollary 14 *Every* $(0, S, =)$*-BDD is equivalent to at least one OBDD.*

3.3 Satisfiability of paths in OBDDs

For a given formula, by constructing a BDD representation and then making it ordered by TRS rules then just looking at the result, we will be able to figure out whether the formula is a tautology, a contradiction or a consistency (satisfiable). Below we will explain the details:

NOTATION. Let α, β, γ range over finite sequences of guards and negations of guards. We write ε for the empty sequence, and $\alpha.\beta$ for the concatenation of sequences α and β. If the order of a sequence is unimportant, we sometimes view it as a set, and write $g \in \alpha$, or even $\alpha \cup \beta$. The latter denotes the set of all guards or negations of guards that occur somewhere on α or β.

Definition 15 Literals *are guards or negations of guards.* Paths *are sequences of literals. We define the set of paths of a BDD* T*:*

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

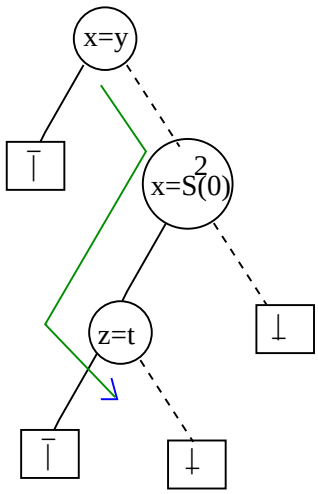

Figure 2: A path from Example 16

Valuation $v : V \to \mathbb{N}$ *satisfies* α if $v(g) = 1$ for all literals $g \in \alpha$. α is *satisfiable* if a valuation v that satisfies it exists.

Example 16 *Let* $T \equiv ITE(x = y, \top, ITE(x = S^2(0), ITE(z = t, \top, \bot), \bot)$ *then the following is a path(Figure 2):*

$$x \neq y.\ x = S^2(0).\ z \neq t$$

Theorem 17 *Each path in an OBDD is satisfiable.*

Proof. We use induction over OBDDs. Suppose $T \equiv ITE(S^m(x) = S^n(z), T_1, T_2)$ is an OBDD, and each path which belongs to T_1 or T_2 is satisfiable, we will show that each path in T is satisfiable as well.

Suppose α is a satisfiable path in T_1 or T_2, so there is a valuation v which satisfies α. Let D be the set of those elements of $\bar{V}$ which occur in α. We modify this valuation, in a way that it satisfies $S^m(x) = S^n(z)$ — or its negation, depending on whether α is in T_1 or T_2 — and also still satisfies α. For more details please refer to [2]. ■

Corollary 18 *An immediate consequence of Theorem 17 is*

- $\top$ *is the only tautological OBDD.*
- $\bot$ *is the only contradictory OBDD.*
- *Every other OBDD is satisfiable (only).*

Proof. Please look at [2]. ■

In the next section an algorithm will be presented which will produce an OBDD out of any formula(the algorithm is not present in[2]). This algorithm is an extension of the one in [8].

4 An Algorithm

Starting from an arbitrary formula, not just BDD, we define a system to simplify the formulas, then we will make an algorithm, through which a given formula will be transformed to an equivalent OBDD.

Definition 19 *We define a set of rewrite rules to be applied on formulas, called simplification:*

$$(u = v) \to (u = v) \downarrow$$

$$(\varphi \wedge \bot) \to \bot \qquad (\bot \wedge \varphi) \to \bot$$

$$(\varphi \wedge \top) \to \varphi \qquad (\top \wedge \varphi) \to \varphi$$

$$(\neg\top) \to \bot \qquad (\neg\bot) \to \top$$

$$ITE(\top,\ \varphi,\ \psi) \to \varphi \qquad ITE(\bot,\ \varphi,\ \psi) \to \psi$$

$$ITE(g,\ \psi,\ \psi) \to \psi$$

We identify the most simplified version of φ *with* $\varphi \Downarrow$.

Definition 20 *We extend the function of Definition 7, on formulas, for any literal l:*

$$(\neg\varphi)|_l := \neg(\varphi|_l)$$
$$(\varphi \wedge \psi)|_l := (\varphi|_l) \wedge (\psi|_l)$$
$$ITE(g, \varphi_1, \varphi_2)|_l := ITE(g|_l, \varphi_1|_l, \varphi_2|_l)$$

Remark 21 *An immediate consequence of the definition above, is that:*
considering $l \equiv S^m(y) = r$ as a guard in which $y \in V$, it is the case that $y \notin T|_l$ moreover $l, \neg l \notin T|_{\neg l}$.

Lemma 22 *Suppose T is a simplified BDD. let g be the smallest guard occuring in T. Then $T \succ_{rpo} (T|_l) \Downarrow$, where $l \in \{g, \neg g\}$.*

Proof. Regarding the assumption, T has a sub-BDD T' of the shape $ITE(g, T_1, T_2)$. Therefore T' is replaced with T_1 in $(T|_g) \Downarrow$ and with T_2 in $(T|_{\neg g}) \Downarrow$. $T' \succ_{rpo} T_1, T_2$ regarding the Definition 11(I), all other rewrite steps will make T smaller in $\succ_{rpo}$(See also lemma 12). Thus it can be easily concluded that $T \succ_{rpo} (T|_l) \Downarrow$. ■

Definition 23 *We define a function* `sort` *on simplified formulas, which sorts out and simplifies the given formula regarding the smallest contained guard.*

- $\texttt{sort}(\bot) \equiv \bot$
- $\texttt{sort}(\top) \equiv \top$
- *Consider g as the smallest guard occuring positively or negatively in φ. Then*

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

Remark 24 *According to the definition above, it is obvious that* $\texttt{sort}(\varphi)$ *is a BDD.*

Lemma 25 *Let T be any simplified BDD. Then:*

1. $\texttt{sort}(T) \Downarrow \equiv \texttt{sort}(T)$.
2. $T \succeq_{rpo} \texttt{sort}(T)$.

Proof.

1. We prove it with induction on $\succ_{rpo}$. If $T \in \{\top, \bot\}$ then that will hold. Suppose the lemma holds for any T', such that $T \succ_{rpo} T'$.
   - If $\texttt{sort}(T) \equiv \texttt{sort}(T|_g \Downarrow)$ then regarding the Lemma 22 and the hypothesis, $\texttt{sort}(T) \Downarrow \equiv \texttt{sort}(T)$.
   - If $\texttt{sort}(T) \equiv ITE(g,\ \texttt{sort}(T|_g \Downarrow),\ \texttt{sort}(T|_{\neg g} \Downarrow))$ then it is obvious that none of the rules of the Definition 19 can be applied, since T is simplified, and both $\texttt{sort}(T|_g \Downarrow)$ and $\texttt{sort}(T|_{\neg g} \Downarrow)$ already establish the lemma. Thus $\texttt{sort}(T) \Downarrow \equiv \texttt{sort}(T)$.
2. For the two trivial cases it obviously holds. Now using induction, we suppose that it holds for any $T \succ_{rpo} T'$. We show that it also holds for T:
   If $\texttt{sort}(T) \equiv \texttt{sort}(T|_g \Downarrow)$ then regarding the Lemma 22 and the hypothesis, $T \succeq_{rpo} \texttt{sort}(T)$. Otherwise $\texttt{sort}(T) \equiv ITE(g,\ \texttt{sort}(T|_g \Downarrow),\ \texttt{sort}(T|_{\neg g} \Downarrow))$, moreover $T \equiv ITE(f, T_1, T_2)$ in which $f \succeq g$, using the Definition 11(II,III) it is obvious that either one of those two holds, and hence $T \succ_{rpo} \texttt{sort}(T)$ or $T \equiv \texttt{sort}(T)$. In any case $T \succeq_{rpo} \texttt{sort}(T)$.

■

Theorem 26 *Suppose T is a simplified BDD, if $T \equiv \texttt{sort}(T)$ then T is ordered.*

Proof. We prove it inductively

- $T \in \{\top, \bot\}$, then it is ordered.

- $T \equiv ITE(f, T_1, T_2)$, we assume that the theorem holds for any $T \succ_{rpo} T'$. Now we distinguish two cases:

  - $\mathtt{sort}(T) \equiv \mathtt{sort}(T|_g \Downarrow)$. We show that this case will never happen.
    Regarding Remark 21 $g \notin T|_g$, hence $g \notin \mathtt{sort}(T|_g \Downarrow)$. Therefore $T \not\equiv \mathtt{sort}(T|_g \Downarrow)$, which is a contradiction because $g \in T$ and $T \equiv \mathtt{sort}(T)$ according to the hypothesis.
  - $\mathtt{sort}(T) \equiv ITE(g,\ \mathtt{sort}(T|_g \Downarrow),\ \mathtt{sort}(T|_{\neg g} \Downarrow))$.
    Since $T \equiv \mathtt{sort}(T)$, we have got the two followings:
    1. $f \equiv g,\ T_1 \equiv \mathtt{sort}(T|_g \Downarrow),\ T_2 \equiv \mathtt{sort}(T|_{\neg g} \Downarrow))$ and
    2. $T_i|_{g_i} \equiv T_i$, using Remark 21, where $g_1 \equiv g$ and $g_2 \equiv \neg g$. Now:

$$\begin{array}{ll} T_i \equiv \mathtt{sort}(T|_{g_i} \Downarrow) & \text{regarding 1} \\ \equiv \mathtt{sort}(T_i|_{g_i} \Downarrow) & \text{definition of } T \\ \equiv \mathtt{sort}(T_i \Downarrow) & \text{above} \\ \equiv \mathtt{sort}(T_i) & \text{regarding Lemma 25(1) and(1)} \end{array}$$

  Using the induction hypothesis, T_1 and T_2 are ordered. Hence rules 1-7 of the Definition 9 are not applicable on T, which means T will not be ordered only if the rule $8'$ is applicable . This rule can not be applied either, since $T_1|_g \equiv T_1$. Therefore T is ordered.

■

Now using the following algorithm, we can make an Ordered BDD, for any given formula:

```
OBDD(Φ)
 Ψ:= Φ⇓ ;
 Φ:= ⊥ ;
 while Φ ≠ Ψ do
     Φ:= Ψ ;
     Ψ:=sort(Ψ) ;
 od
 return Ψ;
```

Theorem 27 *The algorithm given above is terminating, and* $\mathtt{OBDD}(\Phi)$ *is an OBDD equivalent to* Φ*, for any given formula* Φ*.*

Proof. Regarding Remark 24 $\mathtt{sort}(\Phi)$ will be a BDD, for any given formula Φ. The $\succeq_{rpo}$ ordering is well-founded, therefore using Lemma 25(2) we know that after finitely many steps we will get $\mathtt{sort}(\Psi) \equiv \Psi$, for some BDD Ψ. Now using Theorem 26, Ψ, is ordered, and it is $\mathtt{OBDD}(\Phi)$, regarding the algorithm.
■

Example 28 *We give an example to see how this algorithm works, also to show that* $\mathtt{OBDD}(\Phi)$ *gives the same result as the rewrite system. We work with the formula of Example 10. Suppose*

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

then: $\Phi \Downarrow \equiv \Phi$*. Moreover* $\mathtt{sort}(\Phi)$ *will be:*

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

Reader will be able to derive that:
$\mathtt{sort}(\Phi|_{x=S^2(y)}) \equiv ITE(x = S(z), \top, \bot)$ *and* $\mathtt{sort}(\Phi|_{x\neq S^2(y)}) \equiv \bot$
Replacing these two in the algorithm, we will obtain

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

It can also be checked that $\mathtt{sort}(\mathtt{sort}(\Phi)) \equiv \mathtt{sort}(\Phi)$*, so this, is also the result of* $\mathtt{OBDD}(\Phi)$*.*

5 Conclusion

We developed the basics for a decision procedure for boolean combinations of equations with zero and successor. First, a formula is transformed into an $(0, S, =)$-BDD. A rewrite systems on $(0, S, =)$-BDDs is presented, which yields a normal form. It is terminating, and the normal form has the desirable property that all paths are satisfiable. As a consequence, if a formula ϕ is a contradiction (i.e. equivalent to $\bot$), then it reduces to $\bot$ in both systems. Similarly for tautologies. Therefore, our method can be used to decide tautology and satisfiability. Then we provided an algorithm, through which any formula is transformed to an equivalent OBDD. This algorithm gives the same OBDD for a given BDD, as the rewrite system, but it is more efficient than the rewrite system. We proved that the algorithm is sound and complete.

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[2] B. Badban and J.C. van de Pol. Two solutions to incorporate zero, successor and equality in binary desicion diagrams. Technical Report SEN-R0231, CWI, Amsterdam, The Netherlands, 2002. Available at http://citeseer.nj.nec.com/badban02two.html.

[3] B. Badban and J.C. van de Pol. Zero, successor and equality in binary desicion diagrams. *Accepted for Annals of Pure and Applied Logic*, 2004.

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

[5] R.E. Bryant, S. German, and M.N. Velev. Processor verification using efficient reductions of the logic of uninterpreted functions to propositional logic. *ACMTCL: ACM Transactions on Computational Logic*, 2, 2001.

[6] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1–2):69–115, 1987.

[7] A. Goel, K. Sajid, H. Zhou, and A. Aziz. BDD based procedures for a theory of equality with uninterpreted functions. In *Proc. of Computer Aided Verification, CAV'98*, LNCS 1427, pages 244–255. Springer, 1998.

[8] J.F. Groote and J.C. van de Pol. Equational binary decision diagrams. In M. Parigot and A. Voronkov, editors, *Proc. of LPAR 2000*, LNAI 1955, pages 161–178. Springer, 2000.

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

[10] O. Strichman, S.A. Seshia, and R.E. Bryant. Deciding separation formulas with SAT. In E. Brinksma and K.G. Larsen, editors, *Proc. of Computer-Aided Verification (CAV)*, LNCS 2404, pages 209–222. Springer, 2002.

[11] H. Zantema. Termination of term rewriting. In M.A. Bezem, J.W. Klop, and R.C. de Vrijer, editors, *Term Rewriting Systems*, chapter 6. Cambridge University Press, 2003 (to appear).