Ternary Boolean Diagrams and Their Application to Model Checking

Wenhui Zhang
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing, China

Abstract. Binary decision diagrams (BDDs) are a type of data structure with associated algorithms for manipulation of Boolean functions. It has applications in logic synthesis and formal verification. This work[1] introduces ternary Boolean diagrams (TBDs) for representation and manipulation of Boolean functions. TBDs may be considered as a data structure complementary to BDDs, and may be used in certain cases where BDDs are expensive to use. A connection of TBDs to the problem of model checking concurrent transition systems is established. Experimental evaluation and case studies of the verification approach are discussed with respect to BDD based model checking.

Keywords. Boolean Function Manipulation, Symbolic Model Checking, Ternary Boolean Diagrams

1 Introduction

Binary decision diagrams (BDDs) were introduced for representation of switching circuits in [11]. The full potential for efficient algorithms based on the BDDs was investigated in [4], in which a fixed variable ordering and shared sub-graphs are used for compressed canonical representation. Efficient boolean function manipulation has made BDDs been extensively used in logic synthesis and formal verification, in particular, for combating the state explosion problem of model checking [6, 14]. However, BDD still has problems for succinctly representing many verification problems. It is therefore of great importance to search for alternative or supplementary bases for boolean function manipulation and formal verification. This paper discuses the use of ternary boolean diagrams (TBDs). TBDs are a kind of extension of BDDs such that each node of TBD has 3 out edges instead of 2, and a node may be labeled by a label (representing a variable) or a dual one. The structure is similar to ternary decision diagrams (TDDs) [10], however, the interpretation is different. An essential difference between TBDs and BDDs (or TDDs) is that TBDs are not decision diagrams: given a TBD

[1] This work merges and extends the preliminary works presented in technical reports [19, 20], and it was supported by the National Natural Science Foundation of China under Grant Nos. 60833001 and 61272135, the National Basic Research Program of China under Grant No. 2014CB340701, and the CAS Innovation Program.

representing a set of states, a path of such a TBD does not have sufficient information to decide whether a state constructed from the labels of the path is a state in the set. The advantage of this difference is that it provides possibilities for representing Boolean formulas more succinctly. The contents of this paper are as follows.

- A definition of the structure and semantics of TBD is given in Section 2. The correspondence of TBDs to Boolean formulas is established through the semantics.

  These definitions are meant to be brief, and may not be directly useful for Boolean manipulation, and then a definition of ordered TBDs and related issues are given in Section 2.3.

  Operations including negation and conjunction are defined for TBDs. The correspondence of these operations to Boolean operations is established. Then for efficient TBD manipulation, a set of rewrite rules is presented for rewriting TBDs into a kind of reduced order TBDs.
- The representation succinctness is explained through a comparison of TBDs and BDDs in Section 2.7.

  It is proved that TBDs may be more succinct than BDDs for representation of some formulas. Experimental data are also presented to support that there may be advantages with TBDs in average case, not only in extreme cases.
- The application of TBDs to model checking CTL formulas are presented in Section 3.

  Feasibility of such an application is studied through a comparison of an implementation of this approach with the implementation of BDD based model checking in the well known model checking tool NuSMV [7].

  The comparison contains two parts: one uses random Boolean programs, the other uses well-known protocols as the case studies. The first one is supposed to show some basic properties of the computational aspects of TBDs, while the second one is to show the feasibility of TBD based model checking for practical applications.

  The first one is helpful for reflecting the basic properties of the computational aspects, since random problems do not posses good structures that may be utilized by various strategies in model checking, and such problems may be considered as hard problems to be dealt with.

  The second one is helpful for showing the feasibility to practical applications, since such a problem may involve a relatively large number of variables that are necessary for encoding a realistic problem. For this kind of problems, various strategies have to be used in order to make verification process successful.

  In general, we may conclude that BDD based model checking and TBD based model checking have their own advantages, such that TBDs may be considered as a data structure complementary to BDDs, and they may be used in different situations to achieve the optimal feasibility of model checking approaches.

– Then a section for concluding remarks follows and finally, there are appendices for establishing the correctness of the claims which are not proved in the main sections.

2 Ternary Boolean Diagrams

Let $\mathcal{L}$ be a set of labels. Let $\mathcal{L}^- = \{-x \mid x \in \mathcal{L}\}$. A boolean diagram over $\mathcal{L}$ is a graph with a root node and each node is assigned a label of $\mathcal{L} \cup \mathcal{L}^-$. A ternary boolean diagram is such a graph where the out degree of a node is either 3 or 0, and the out edges of a node is ordered such that the left, the middle and the right out edges of a node can be identified. Formally, ternary Boolean diagrams are defined as follows.

Definition 1 (Ternary Boolean Diagram). *Let $\mathcal{L}$ be a set of labels. A ternary Boolean diagram (TBD) over $\mathcal{L}$ is a quadruple*

$$(N, n_0, E, L)$$

where N is a set of nodes, $n_0 \in N$ is the root node, $E : N \to N^3$ is a partial function that defines the three out edges of a node, $L : N \to \mathcal{L} \cup \mathcal{L}^-$ is a labeling function which assigns each node a label of $\mathcal{L}$ or its negation.

The set of ternary Boolean diagrams over $\mathcal{L}$ is denoted $\mathcal{D}(\mathcal{L})$. For simplicity, we fix an $\mathcal{L}$ and write $\mathcal{D}$ for $\mathcal{D}(\mathcal{L})$.

Notations When $E(n)$ is not defined for a node n, the value of $E(n)$ is denoted by a special triple $(\epsilon, \epsilon, \epsilon)$. A TBD t with n_0 as the root node may be represented by (x, a, b, c) when $L(n_0) = x, E(n_0) = (a, b, c)$. For the clearness of the presentation, additional notations to be used are as follows.

notation	meaning	condition
$x > 0$	$x \in \mathcal{L}$	
$x < 0$	$x \in \mathcal{L}^-$	
$L(t)$	x	where $t = (x, a, b, c)$
$--x$	x	where $x > 0$
$\lvert x \rvert$	x	where $x > 0$
$\lvert x \rvert$	$-x$	where $x < 0$
$-(x, a, b, c)$	$(-x, a, b, c)$	where $x > 0$ or $x < 0$
$x \cdot t$	t	where $x > 0$
$x \cdot t$	$-t$	where $x < 0$
τ_x	$(x, \epsilon, \epsilon, \epsilon)$	where $x > 0$
$\tau_x^{\sim}$	$(x, \epsilon, \epsilon, \epsilon)$	where $x > 0$ or $x < 0$

Example 1. Figure 1 shows graphic representations of (x, a, b, c) and $-(x, a, b, c)$ (or $(-x, a, b, c)$) where a, b, c are TBDs, or ϵ in case the node does not have successors.

Fig. 1.

2.1 Language, Model and Equivalence

Let $\Sigma = (\mathcal{L} \cup \mathcal{L}^-)$.

Definition 2. *Let $r \in \mathcal{D}$ and $\sigma = x_1 \cdots x_k \in \Sigma^*$. σ is accepted by r, denoted $\sigma \models r$, iff there is an $x \in \mathcal{L}$ such that $r\lceil_{x_1} \cdots \lceil_{x_k} = \tau_x$ in which $r\lceil_w$ is defined as follows:*

case	$r\lceil_w$ *when* $w < 0$	$r\lceil_w$ *when* $w > 0$
1	r	r
2	$v \cdot \tau_x^\sim$	$v \cdot \tau_y^\sim$
3	$v \cdot -\tau_z$	$v \cdot -\tau_z$
4	$(v, s\lceil_w, t\lceil_w, u\lceil_w)$	$(v, s\lceil_w, t\lceil_w, u\lceil_w)$

in which the last rule is applied only when the other rules are not applicable, in which the corresponding condition for each of the cases is specified as follows.

case	*condition*
1	$r = \tau_x^\sim$
2	$r = (v, \tau_x^\sim, \tau_y^\sim, \tau_z) \wedge \lvert v\rvert = \lvert w\rvert$
3	$r = (v, \tau_x^\sim, \tau_y^\sim, -\tau_z) \wedge \lvert v\rvert = \lvert w\rvert$
4	$r = (v, s, t, u)$

Cyclicity in a TBD causes non-termination in the computation of $r\lceil_{x_1} \cdots \lceil_{x_k}$ (in which the operator is left associative). In such cases, we have $r\lceil_{x_1} \cdots \lceil_{x_k} \neq \tau_x$ for all x. On the other hand, acyclicity is ensured when using ordered TBDs (cf. Definition 3.1).

Definition 3. *Let $r \in \mathcal{D}$ and $m \subseteq \mathcal{L}$. m is a model of r, denoted $m \models r$, iff there is a sequence $\sigma \in \Sigma^*$ such that $x \in \sigma$ implies $x > 0 \wedge x \in m$ or $x < 0 \wedge -x \notin m$, and $\sigma \models r$.*

Models can be derived from those strings in the language that do not contain conflicting labels, by removing all negative labels for each of such strings.

Example 2. Let $\mathcal{L} = \{p, q, r\}$. Then we have $\Sigma = \{p, q, r, -p, -q, -r\}$. Let τ denote $(r, \epsilon, \epsilon, \epsilon)$. Figure 2 is a graphic representation[2] of $(-p, -\tau, \tau, (q, \tau, -\tau, \tau))$.

[2] Note that the three τ nodes in the figure are the same and should be merged to one, the figure shows three nodes for presentational clarity. Similar for the two $-\tau$ nodes.

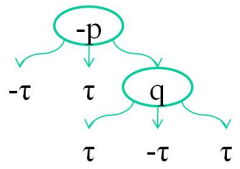

Fig. 2.

– The language defined by this TBD is

$$((\Sigma \setminus \{-q, q\})^* \cdot \{q\} \cdot (\Sigma \setminus \{-p, p\})^* \cdot \{-p, p\} \cdot \Sigma^*) \cup$$
$$((\Sigma \setminus \{-q, q\})^* \cdot \{-q\} \cdot (\Sigma \setminus \{-p, p\})^* \cdot \{-p\} \cdot \Sigma^*).$$

  Taking the first part in the union as an example:
  $(\Sigma \setminus \{-q, q\})^*$ in the expression does not have any effect in the reduction of the TBD, then q reduces the TBD to $(-p, -\tau, \tau, -\tau)$, which remains unchanged after reduction with $(\Sigma \setminus \{-p, p\})^*$, then both $-p$ and p reduces $(-p, -\tau, \tau, -\tau)$ to τ, which remains unchanged after reduction with Σ^*. Therefore the set $(\Sigma \setminus \{-q, q\})^* \cdot \{q\} \cdot (\Sigma \setminus \{-p, p\})^* \cdot \{-p, p\} \cdot \Sigma^*$ is a part of the language of the TBD.
– Then accordingly, the set of models of the TBD is

$$\{\{p, q, r\}, \{q, r\}, \{r\}, \{p, q\}, \{q\}, \{\}\}.$$

Definition 4. *Let $s, t \in \mathcal{D}$ be TBDs. s is equivalent to t, denoted $s \equiv t$, iff for all $m \subseteq \mathcal{L}$, $m \models s$ iff $m \models t$.*

Completeness: We say that r is complete, if for all $m \subseteq \mathcal{L}$, m is a model of r. Then r is complete iff r is equivalent to τ_x for an $x \in \mathcal{L}$.

2.2 Boolean Formulas as TBDs

Let $\Phi(S)$ denote the set of boolean formulas with variables in S. Let $\mathcal{L}_I \subseteq \mathcal{L}$ be a subset of $\mathcal{L}$.

Definition 5. *Let $\varphi \in \Phi(\mathcal{L}_I)$ and $m \subseteq \mathcal{L}$. $m \models \varphi$ iff φ evaluates to true under the assignment such that $x_i = 1$ iff $x_i \in m$.*

Definition 6. *Let $\varphi \in \Phi(\mathcal{L}_I)$ and $r \in \mathcal{D}$. φ is equivalent to r, denoted $\varphi \equiv r$, iff for all $m \subseteq \mathcal{L}$, $m \models \varphi$ iff $m \models r$.*

Then τ_x is equivalent to the logical constant 1, and $-\tau_x$ is equivalent to the logical constant 0.

Example 3. The TBD in Example 2 is equivalent to $(\neg p \vee q)$. It is easily seen that both have the same set of models when the set of Boolean variables are given as $\{p, q, r\}$.

2.3 Ordered TBDs

Let $\leq$ be a linear order on $\mathcal{L}$. $x < y$ iff $x \leq y$ and $x \neq y$. Let $\mathcal{L} = \{p_1, ..., p_n, p_{n+1}\}$ such that $p_i < p_{i+1}$ for $i = 1, ..., n$. Let $\tau = \tau_{p_{n+1}} = (p_{n+1}, \epsilon, \epsilon, \epsilon)$. Then τ and $-\tau = (-p_{n+1}, \epsilon, \epsilon, \epsilon)$ are the only terminal nodes of ordered TBDs.

Definition 7. *The set $\mathcal{D}^{\mathcal{O}} \subseteq \mathcal{D}$ of ordered TBDs is defined as follows. $s \in \mathcal{D}^{\mathcal{O}}$ iff $s \in \{\tau, -\tau\}$, or $s = (x, n_0, n_1, n_2) \in \mathcal{D}$ with $|x| < |L(n_i)|$ and n_i ordered for all $i \in \{0, 1, 2\}$.*

Negation: Let $s = (x, a, b, c) \in \mathcal{D}^{\mathcal{O}}$. The negation of s, denoted $\neg s$, is computed by $\neg s = (-x, a, b, c)$.

Conjunction: Let $s = (x, a, b, c)$, $t = (x', a', b', c')$ be ordered TBDs. The conjunction of s and t, denoted $s \wedge t$, is computed as follows.

(1) One of s and t is in $\{\tau, -\tau\}$. Then

$\tau \wedge t \quad = t \wedge \tau \quad = t$
$-\tau \wedge t = t \wedge -\tau = -\tau$

(2) None of s and t are in $\{\tau, -\tau\}$. Let $u \vee v$ denote $\neg(\neg u \wedge \neg v)$. Then

case	$s \wedge t$
1	$(x, a \wedge a', b \wedge b', c \wedge c')$
2	$(x, ((a \wedge c) \vee (a' \wedge c')), ((b \wedge c) \vee (b' \wedge c')), \tau)$
3	$(x, a \wedge \neg(a' \wedge c'), b \wedge \neg(b' \wedge c'), c)$
4	$(x, a, b, c \wedge t)$
5	$(x, \neg(\neg a \wedge t), \neg(\neg b \wedge t), \neg(\neg c \wedge t))$

The corresponding condition for each of the cases is specified as follows.

case	condition
1	$x = x' \wedge x > 0$
2	$x = x' \wedge x < 0$
3	$\|x\| = \|x'\| \wedge x > x'$
4	$\|x\| < \|x'\| \wedge x > 0$
5	$\|x\| < \|x'\| \wedge x < 0$

The cases where $|x| > |x'|$ (with two subcases, one with $x > 0$ and the other with $x < 0$) or $|x| = |x'| \wedge x < x'$ are computed by $s \wedge t = t \wedge s$.

Proposition 1. *Let $s, t \in \mathcal{D}^{\mathcal{O}}$ be TBDs. Then $s \wedge t = t \wedge s$.*

The commutativity of conjunction follows from the definition of the operator.

Abstraction: Let $r \in \{\tau, -\tau, (v, s, t, u)\}$ be an element of $\mathcal{D}^{\mathcal{O}}$. The abstraction of r on x, denoted $abs(x)(r)$, is computed by $abs(x)(r) = r|_{-x} \wedge r|_x$ in which $r|_w$ is computed as follows:

case	$r\|_w$ when $w < 0$	$r\|_w$ when $w > 0$
1	r	r
2	$v \cdot (s \wedge u)$	$v \cdot (t \wedge u)$
3	$(v, s\|_w, t\|_w, u\|_w)$	$(v, s\|_w, t\|_w, u\|_w)$

The corresponding condition for each of the cases is specified as follows.

case	condition
1	$r \in \{\tau, -\tau\}$
2	$r = (v, s, t, u) \wedge \|v\| = \|w\|$
3	$r = (v, s, t, u) \wedge \|v\| \neq \|w\|$

Abstraction corresponds to the application of the universal quantification in a Boolean formula. The existential abstraction of r on x, is defined by $\neg abs(x)(\neg r)$.

Proposition 2. *Let $s_1, s_2, t_1, t_2 \in \mathcal{D}^{\mathcal{O}}$ and $x \in \mathcal{L}$. If $s_1 \equiv s_2$ and $t_1 \equiv t_2$, then $\neg s_1 \equiv \neg s_2$, $s_1 \wedge t_1 \equiv s_2 \wedge t_2$ and $abs(x)(s_1) \equiv abs(x)(s_2)$.*

The proof is presented in Appendix A.1.

On Completeness: The predicate *comp* on $\mathcal{D}^{\mathcal{O}}$ for observation of whether a TBD is complete is defined as follows: $comp(r) = (abs(p_1)abs(p_2)\cdots abs(p_n)(r) = \tau)$.

Proposition 3. *Let $r \in \mathcal{D}^{\mathcal{O}}$. r is complete iff $comp(r)$ holds.*

The proof is presented in Appendix A.2.

2.4 Boolean Formulas as Ordered TBDs

Let $\mathcal{L}_I$ be restricted to a subset of $\mathcal{L} \setminus \{p_{n+1}\}$.

Proposition 4. *Let $x, \varphi_0, \varphi_1 \in \Phi(\mathcal{L}_I)$ and $s, t \in \mathcal{D}^{\mathcal{O}}$. If $x \in \mathcal{L}_I$ is an atomic formula, then $x \equiv (x, -\tau, \tau, \tau)$, and if $\varphi_0 \equiv s$ and $\varphi_1 \equiv t$, then $\neg\varphi_0 \equiv \neg s$ and $\varphi_0 \wedge \varphi_1 \equiv s \wedge t$.*

The proof is presented in Appendix A.3.

Example 4. Let φ be $\neg(p \wedge \neg q) \wedge \neg(\neg p \wedge q)$. Let $\mathcal{L} = \{p, q, z\}$ with $p < q < z$ such that $\mathcal{L}_I = \{p, q\}$. Let $\tau = (z, \epsilon, \epsilon, \epsilon)$ and $-\tau = (-z, \epsilon, \epsilon, \epsilon)$. Following the definitions of negation and conjunction, φ can be transformed to a TBD as follows, with the graphical representation of the final resulting TBD shown in Figure 3.

p	$\equiv (p, -\tau, \tau, \tau)$
$\neg p$	$\equiv (-p, -\tau, \tau, \tau)$
q	$\equiv (q, -\tau, \tau, \tau)$
$\neg q$	$\equiv (-q, -\tau, \tau, \tau)$
$p \wedge \neg q$	$\equiv (p, -\tau, \tau, (q, \tau, -\tau, \tau))$
$\neg p \wedge q$	$\equiv (-p, (q, \tau, -\tau, \tau), \tau, \tau)$
$\neg(p \wedge \neg q)$	$\equiv (-p, -\tau, \tau, (q, \tau, -\tau, \tau))$
$\neg(\neg p \wedge q)$	$\equiv (p, (q, \tau, -\tau, \tau), \tau, \tau)$
φ	$\equiv (p, (q, \tau, -\tau, \tau), (-q, \tau, -\tau, \tau), \tau)$

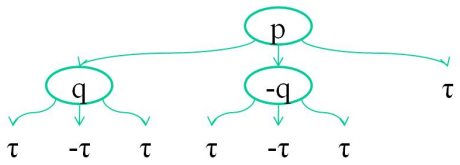

Fig. 3.

Proposition 5. *Let $\varphi \in \Phi(\mathcal{L}_I)$ and $s \in \mathcal{D}^{\mathcal{O}}$. If $\varphi \equiv s$, then $\forall x.\varphi \equiv abs(x)(s)$.*

The proof is presented in Appendix A.4.

Proposition 6. *Let $\varphi \in \Phi(\mathcal{L}_I)$ and $s \in \mathcal{D}^{\mathcal{O}}$. If $\varphi \equiv s$, then φ is valid iff s is complete.*

This follows from the definition of validity (i.e, every set $m \in 2^{\mathcal{L}}$ is a model of φ) and that of completeness (every set $m \in 2^{\mathcal{L}}$ is a model of s).

2.5 Reduced Ordered TBDs

Definition 8. *The set of reduced ordered TBDs, denoted $\mathcal{D}^{\mathcal{R}}$, is a subset of $\mathcal{D}^{\mathcal{O}}$ defined as follows. $\{\tau, -\tau\} \subseteq \mathcal{D}^{\mathcal{R}}$, and $(x, a, b, c) \in \mathcal{D}^{\mathcal{R}}$ iff $a, b, c \in \mathcal{D}^{\mathcal{R}}$ and the following conditions hold:*

$c \neq -\tau$	
$b = -\tau \wedge c = \tau$	$\Rightarrow a = \tau$
$a = -\tau \wedge c = \tau$	$\Rightarrow b = \tau$
$a = b$	$\Rightarrow a, c \notin \{\tau, -\tau\}$
$c \in \{a, b\}$	$\Rightarrow c = \tau \wedge x > 0$
$a, b \in \{\tau, -\tau\}$	$\Rightarrow x > 0$

Proposition 7. *For every $s \in \mathcal{D}^{\mathcal{O}}$, there exists $t \in \mathcal{D}^{\mathcal{R}}$ such that $s \equiv t$.*

This follows from Proposition 9 that states that every TBD not in $\mathcal{D}^{\mathcal{R}}$ can be rewritten into an equivalent one in $\mathcal{D}^{\mathcal{R}}$.

Rewrite Rules Let $x \in \mathcal{L} \cup \mathcal{L}^-$ and $y \in \mathcal{L}$. Let R be a rewrite system with the following set of rules.

$(x, -\tau, -\tau, c)$	$\to (x \cdot -\tau)$
$(x, a, b, -\tau)$	$\to (x \cdot -\tau)$
(x, τ, τ, c)	$\to (x \cdot c)$
(x, a, a, τ)	$\to (x \cdot a)$
$(x, a, -\tau, \tau)$	$\to (x, \tau, -\tau, a)$
$(x, -\tau, b, \tau)$	$\to (x, -\tau, \tau, b)$
(x, a, c, c)	$\to (x, a, \tau, c)$
(x, c, b, c)	$\to (x, \tau, b, c)$
$(-y, \tau, b, \tau)$	$\to (y, -\tau, -b, \tau)$
$(-y, a, \tau, \tau)$	$\to (y, -a, -\tau, \tau)$
$(-y, \tau, -\tau, c)$	$\to (y, -c, \tau, \tau)$
$(-y, -\tau, \tau, c)$	$\to (y, \tau, -c, \tau)$

Regarding the operation $\cdot$, we remind that according to the notation defined at the beginning of this section, $x \cdot \tau$ is $-\tau$ when $x \in \mathcal{L}^-$ and $x \cdot \tau$ is τ when $x \in \mathcal{L}$. It is similar for other TBDs. Let $s \xrightarrow{*} t$ denotes that s rewrites to t in 0 or more steps.

Proposition 8. *For every $s, t \in \mathcal{D}^{\mathcal{O}}$, if $s \xrightarrow{*} t$, then $s \equiv t$.*

The proof is presented in Appendix A.5.

Proposition 9. *For every $s \in \mathcal{D}^{\mathcal{O}}$, there exists one and only one $t \in \mathcal{D}^{\mathcal{R}}$ such that $s \xrightarrow{*} t$.*

This follows from the fact that if an $s \in \mathcal{D}^{\mathcal{O}}$ is not in $\mathcal{D}^{\mathcal{R}}$, then the rewrite rules can be repeatedly applied until a TBD in $\mathcal{D}^{\mathcal{R}}$ is obtained. On the other hand, the rewrite rules does not interfere with each other, such that the order of the applications of the rules does not affect the result of the rewriting.

Example 5. Let $\varphi = \neg(p \wedge \neg q) \wedge \neg(\neg p \wedge q)$ be the same as that of Example 4. Let $\mathcal{L} = \{p, q, z\}$ with $p < q < z$ such that $\mathcal{L}_I = \{p, q\}$. Following the definitions of negation and conjunction, and the application of the rewrite rules, φ can be transformed to a TBD as follows, with the graphical representation of the final resulting TBD shown in Figure 4.

p	$\equiv (p, -\tau, \tau, \tau)$
$\neg p$	$\equiv (p, \tau, -\tau, \tau)$
q	$\equiv (q, -\tau, \tau, \tau)$
$\neg q$	$\equiv (q, \tau, -\tau, \tau)$
$p \wedge \neg q$	$\equiv (p, -\tau, \tau, (q, \tau, -\tau, \tau))$
$\neg p \wedge q$	$\equiv (p, \tau, -\tau, (q, -\tau, \tau, \tau))$
$\neg(p \wedge \neg q)$	$\equiv (p, \tau, (q, -\tau, \tau, \tau), \tau)$
$\neg(\neg p \wedge q)$	$\equiv (p, (q, \tau, -\tau, \tau), \tau, \tau)$
φ	$\equiv (p, (q, \tau, -\tau, \tau), (q, -\tau, \tau, \tau), \tau)$

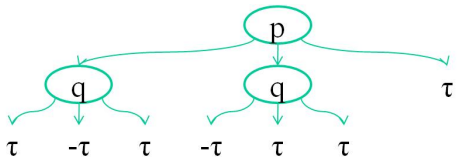

Fig. 4.

The TBD for φ has 10 nodes when it is considered as a tree, and it has 5 nodes when it is considered as a graph with shared nodes (in this case, only the 7 terminal nodes are merged into 2 nodes).

2.6 Reordering of Variables

Suppose that we have a TBD t with variable order $x_1, ..., x_n, x_{n+1}$. We may obtain a TBD t' with variable order $x_1, ..., x_{k-1}, x_{k+1}, x_k, x_{k+2}, ..., x_n, x_{n+1}$ (in which $1 \leq k \leq n-1$) by replacing sub-TBDs starting at variable x_k.

Proposition 10. *Let $p > 0$ and $q > 0$. Each of the following pairs represent the same formula with different variable orderings on p and q.*

$(p, (q, a, b, c), c', c'')$ $(q, (p, a, \tau, \tau), (p, b, \tau, \tau), (p, c, c', c''))$
$(p, c, (q, a', b', c'), c'')$ $(q, (p, \tau, a', \tau), (p, \tau, b', \tau), (p, c, c', c''))$
$(p, c, c', (q, a'', b'', c''))$ $(q, (p, \tau, \tau, a''), (p, \tau, \tau, b''), (p, c, c', c''))$
$(p, (q, a, b, c), (q, a', b', c'), c'')$ $(q, (p, a, a', \tau), (p, b, b', \tau), (p, c, c', c''))$
$(p, (q, a, b, c), c', (q, a'', b'', c''))$ $(q, (p, a, \tau, a''), (p, b, \tau, b''), (p, c, c', c''))$
$(p, c, (q, a', b', c'), (q, a'', b'', c''))$ $(q, (p, \tau, a', a''), (p, \tau, b', b''), (p, c, c', c''))$
$(p, (q, a, b, c), (q, a', b', c'), (q, a'', b'', c''))$ $(q, (p, a, a', a''), (p, b, b', b''), (p, c, c', c''))$

The proof is presented in Appendix A.6. These equivalences are useful for reordering of variables.

2.7 Complexity Issues

The representation succinctness is explained through a comparison of TBDs and BDDs. The node size of t, denoted $|t|$, is the number of different nodes of t that may use shared nodes. The tree size of t, denoted $||t||$, is the number of nodes of t where t is expanded as a tree. For TBDs, a TBD tree t has $(2||t|| + 1)/3$ terminal nodes, and therefore $|t| \leq (||t|| - 1)/3 + 2$.

Variable Order Let $v = v_1 \cdots v_m$ denote the partial order $(\{v_1, ..., v_m, z\}, \leq)$ with $v_i < v_j$ iff $i < j$ and $v_i < z$ for all i. An ordering v of variables of ψ is such an ordering where $\{v_1, ..., v_m\}$ is a permutation of variables appearing in ψ (the special variable z is needed for TBDs, it is not necessary for BDDs, however adding this special variable to an ordering of variables would not cause any trouble for BDDs). Let $v[x \to 1]$ denote the ordering v' where the first element of v' is x, and the rest of the elements of v' is the same as v with x removed from v.

Definition 9. *Let $p \leftrightarrow q$ denote $\neg(p \wedge \neg q) \wedge \neg(\neg p \wedge q)$. Let a_i, b_i with $i \in \{1, ..., n\}$ be propositional variables.*

$$\boxed{\begin{array}{l} \varphi_i = a_i \leftrightarrow b_i \\ \varphi = \bigwedge_{i=1}^{n} \varphi_i \end{array}}$$

Proposition 11. *There is some variable ordering, such that the reduced ordered BDD representation of φ has node size $\geq 2^n$, while for all variable orderings, we can construct a TBD representation for φ with tree size and node size linear in n.*

The proof is presented in Appendix B.1.

Definition 10. *Let $p \vee q$ denote $\neg(\neg p \wedge \neg q)$. Let $a_{i,j}$ with $i, j \in \{1, ..., n\}$ and b be propositional variables.*

$$\boxed{\begin{array}{l} \varphi_i = a_{i,1} \leftrightarrow \cdots \leftrightarrow a_{i,n} \\ \varphi'_j = a_{1,j} \leftrightarrow \cdots \leftrightarrow a_{n,j} \\ \hline \varphi = \bigwedge_{i=1}^{n} \varphi_i \\ \varphi' = \bigwedge_{j=1}^{n} \varphi'_j \\ \hline \psi = (b \wedge \varphi) \vee (\neg b \wedge \varphi') \end{array}}$$

Proposition 12. *For all variable orderings, the reduced ordered BDD representation of ψ has node size $\geq 2^n$, while we can construct a TBD representation for ψ with node size polynomial in n.*

This proposition follows from Proposition 13 and Proposition 14, which are to be stated as follows.

Proposition 13. *For any ordering of variables of ψ, if d is a BDD for ψ with such an ordering, then $|d| \geq 3 \cdot 2^{n-1} - 1$.*

The proof is presented in Appendix B.2.

Proposition 14. *For any ordering v of variables of ψ, we can construct a TBD t for ψ with $|t| \leq 256 \cdot (n+1)^{12}$.*

The proof is presented in Appendix B.3.

2.8 Experimental Data

For practical application of TBDs, the average complexity is in many cases more important than best case complexity. This subsection provides experimental data on average complexity for selected types and sizes of random Boolean formulas.

Types of Boolean Formulas Consider Boolean formulas in CNF (conjunctive normal form) and DNF (disjunctive normal form). The number of variables is set to 30. The length of clauses in $\{10, 20, 30\}$ and the number of clauses in $\{100, 200, 300\}$. This makes 9 types of CNF formulas and 9 types of DNF formulas. For each type, we randomly choose 20 formulas, and for each formula, we randomly choose 20 different variable orders.

Experimental Data for CNF Formulas The experimental data for the size of TBDs are presented in Table 1, where *cll* is the clause length, *cln* is the number of clauses in a formula, *min* is the minimum size of the TBD obtained among the 400 TBDs (20 instances with 20 different variable orders), *max* is the maximum size of the TBDs, and *average* is the average size of the TBDs. The TBDs are created according to the computation rules implicitly provided by Proposition 4 and Proposition 9. The experimental data for the size of BDDs are presented in Table 2. In addition, the ratio between the average size of BDDs (in Table 2) and that of TBDs (in Table 1) is calculated for each case. These ratios show that TBDs has significant advantage over BDDs for representing some of the formulas.

cll	cln	min	max	average
10	100	709	756	732.59
	200	1321	1389	1354.85
	300	1893	1977	1934.85
20	100	1472	1560	1516.81
	200	2808	2921	2865.39
	300	4088	4223	4149.23
30	100	1743	1847	1794.53
	200	3104	3252	3187.11
	300	4333	4514	4430.60

Table 1. Size of TBDs for CNF formulas

Remarks The number of variables is 30. When the clause length is 30, each clause represents $2^{30} - 1$ states. Then TBDs and BDDs have the same size for the representation of the whole state space of 2^{30} states except a few hundreds of states. Among the 9 tested types of formulas, TBDs are relatively best when the number of states decreases to roughly 73 percent of a total of 2^{30} states. For such a set of states, there is a huge difference between the sizes of the BDD and TBD representations.

cll	cln	min	max	average	ratio
10	100	82729	215897	136146.64	186.35
	200	577533	1035169	768347.00	567.95
	300	1516800	2512466	1973997.62	1021.29
20	100	2606	3848	3173.77	2.09
	200	6715	9036	7634.59	2.67
	300	11380	15282	12916.41	3.11
30	100	1743	1847	1794.53	1.00
	200	3104	3252	3187.11	1.00
	300	4333	4514	4430.60	1.00

Table 2. Size of BDDs for CNF formulas

Experimental Data for DNF Formulas The experimental data for the size of TBDs and BDDs for the types of DNF formulas are shown in Table 3 and Table 4. These data are similar to those presented in Table 1 and Table 2, and the ratios between size of TBDs and size of BDDs also show that TBDs has significant advantage over BDDs for representing some of the formulas.

cll	cln	min	max	average
10	100	709	758	731.39
	200	1325	1391	1353.56
	300	1893	1964	1933.90
20	100	1478	1551	1518.05
	200	2812	2918	2864.66
	300	4061	4212	4148.40
30	100	1738	1842	1793.01
	200	3107	3267	3186.71
	300	4299	4543	4427.99

Table 3. Size of TBDs for DNF formulas

3 Application to Model Checking

This section concerns the application of TBDs to model checking of finite state concurrent transition system. Given a set of proposition symbols AP. A finite state concurrent transition system may be represented by a Kripke structure $\langle S, R, I, L\rangle$ where S is a set of states, $R \subseteq S \times S$ is a total transition relation, $I \subseteq S$ is the set of initial states, and $L : S \rightarrow 2^{AP}$ is the labeling function which assigns each state a set of propositions that are true on the state. The properties of such a system may be specified by Computation Tree Logic (CTL), a propositional branching-time temporal logic [9] introduced by Emerson and Clarke as a specification language for finite state systems.

cll	cln	min	max	average	ratio
10	100	80947	223120	135831.05	186.22
	200	576519	1050933	765908.19	566.68
	300	1561373	2481051	1966855.62	1018.09
20	100	2743	3728	3171.91	2.09
	200	6386	9028	7596.83	2.65
	300	11067	14691	12866.37	3.10
30	100	1738	1842	1793.01	1.00
	200	3107	3267	3186.71	1.00
	300	4299	4543	4427.99	1.00

Table 4. Size of BDDs for DNF formulas

For symbolic representation of a finite state concurrent transition system, a state can be represented by an assignment of a set of boolean variables. Suppose that $V = \{v_1, ..., v_m\}$ is such a set of variables. Let $\Phi(x_1, ..., x_k)$ denote the set of boolean formulas with variables in $\{x_1, ..., x_k\}$. Then a finite state transition system can be represented by $\langle V, \zeta_I, \zeta_R, L_\zeta \rangle$, where $\zeta_I \in \Phi(v_1, ..., v_m)$ is a formula representing the set of initial states I, $\zeta_R \in \Phi(v_1, ..., v_m, v'_1, ..., v'_m)$ a formula representing the transition relation R, and $L_\zeta : AP \to \Phi(v_1, ..., v_m)$ a function that assigns each proposition a formula representing the set of states in which the proposition is true according to the labeling function L.

Let $\vec{v}$ and $\vec{v}'$ denote respectively $v_1, ..., v_m$ and $v'_1, ..., v'_m$. Let $ex : \Phi(v_1, ..., v_m) \to \Phi(v_1, ..., v_m)$ be a function defined by $ex(A) = \exists \vec{v}'.(\zeta_R \wedge A^{\vec{v}'}_{\vec{v}})$.

Let φ, ψ be CTL formulas. By interpreting a CTL formula φ as a set of states, represented by the Boolean formula $[[\varphi]]$, the set of states of the transition system $\langle S, R, I, L \rangle$ satisfying a CTL formula can be computed[3] as follows [8]:

$[[p]]$	$= L_\zeta(p)\ for\ p \in AP$
$[[\varphi \wedge \psi]]$	$= [[\varphi]] \wedge [[\psi]]$
$[[\neg\varphi]]$	$= \neg[[\varphi]]$
$[[EX\varphi]]$	$= ex([[\varphi]])$
$[[EG\varphi]]$	$= \nu Z([[\varphi]] \wedge ex(Z))$
$[[E(\varphi U \psi)]]$	$= \mu Z([[\psi]] \vee ([[\varphi]] \wedge ex(Z)))$

By transforming CTL formulas into the ones that only use the above logical connectives and temporal operators, every CTL property can be computed by applying the above equations, together with the transformation rules [8]. Then checking whether $\langle S, R, I, L \rangle$ satisfies a CTL formula φ is the same as checking the implication $\zeta_I \to [[\varphi]]$.

[3] The computation of fixpoints requires a partial order on $\Phi(v_1, ..., v_m)$ which is defined by $\varphi \leq \psi$ iff $\varphi \to \psi$.

3.1 TBD Manipulations

Let $\mathcal{L} = \{v_1, ..., v_m, v'_1, ..., v'_m, z\}$. Let $\leq$ be a linear order on $\mathcal{L}$ such that z is the largest element of L. Let $\mathcal{L}_I = \{v_1, ..., v_m, v'_1, ..., v'_m\}$. Then we can transform the problem of checking whether the finite state transition system (S, R, I, L) satisfies a CTL formula into the problem of manipulation of ordered TBDs as follows.

Proposition 4 provides a way to represent a formula of $\Phi(v_1, ..., v_m, v'_1, ..., v'_m)$ by an equivalent ordered TBD of $\mathcal{D}(\mathcal{L})$; Proposition 5 establishes the relation between quantified boolean formulas and TBD abstraction; Proposition 6 together with Proposition 3 provides a way for checking whether the set of states represented by a formula is empty. This is sufficient for the transformation of the problem.

For efficient manipulation of TBDs, Proposition 7, Proposition 8, and Proposition 9 established the relation between ordered TBDs and reduced ordered TBDs, and together with Proposition 2, they provide the possibility for the use of reduced ordered TBDs instead of ordered TBDs.

3.2 Experimental Evaluation

This subsection contains selected data and a summary of an experimental evaluation of the TBD based model checking implemented in VERDS version 1.30[4]. The experimental evaluation compares this implementation with NuSMV version 2.5.0[5], and is based on two types of random boolean programs[6].

Types of Random Programs Two sets of random programs and 24 CTL properties are formulated for the experimental evaluation.

Programs with Concurrent Processes The parameters of the first set of random boolean programs are as follows:

a: number of processes
b: number of all variables
c: number of share variables
d: number of local variables in a process

The shared variables are initially set to a random value in $\{0, 1\}$, and the local variables are initially set to 0. For each process, the shared variables and the local variables are assigned the negation of a variable randomly chosen from these variables.

[4] http://lcs.ios.ac.cn/∼zwh/verds/
[5] http://nusmv.irst.itc.it/
[6] Details at http://lcs.ios.ac.cn/∼zwh/verds/files/randombp12.rar

Programs with Concurrent Sequential Processes The parameters of the second set of random boolean programs are as follows, in addition to a, b, c, d specified above.

t: number of transitions in a process
p: number of parallel assignment in each transition

Besides the b boolean variables, for each process, there is a local variable representing program locations, with e possible values. The shared variables are initially set to a random value in $\{0, 1\}$, and the local variables are initially set to 0. For each transition of a process, p pairs of shared variables and local variables are randomly chosen among the shared variables and the local variables, such that the first element of such a pair is assigned the negation of the second element of the pair. Transitions are numbered from 0 to $t-1$, and are executed consecutively, and when the end of the sequence of the transitions is reached, it loops back to the execution of the transition numbered 0.

Types of Properties The properties are specified by CTL formulas. 24 properties are formulated for testing. These properties involve AG, AF properties and properties specified with different combinations of the 10 CTL operators with one or two levels of nesting (with two levels of nesting when AX or EX is involved). Properties p_{01} to p_{12} are as follows, in which v_i are global variables.

p_{01} :	$AG(\bigvee_{i=1}^{c} v_i)$
p_{02} :	$AF(\bigvee_{i=1}^{c} v_i)$
p_{03} :	$AG(v_1 \rightarrow AF(v_2 \wedge \bigvee_{i=3}^{c} v_i))$
p_{04} :	$AG(v_1 \rightarrow EF(v_2 \wedge \bigvee_{i=3}^{c} v_i))$
p_{05} :	$EG(v_1 \rightarrow AF(v_2 \wedge \bigvee_{i=3}^{c} v_i))$
p_{06} :	$EG(v_1 \rightarrow EF(v_2 \wedge \bigvee_{i=3}^{c} v_i))$
p_{07} :	$A(v_1\, U\, A(v_2\, U\, \bigvee_{i=3}^{c} v_i)$
p_{08} :	$A(v_1\, U\, E(v_2\, U\, \bigvee_{i=3}^{c} v_i)$
p_{09} :	$A(v_1\, U\, A(v_2\, R\, \bigvee_{i=3}^{c} v_i)$
p_{10} :	$A(v_1\, U\, E(v_2\, R\, \bigvee_{i=3}^{c} v_i)$
p_{11} :	$A(AXv_1\, R\, AX\, A(v_2\, U\, \bigvee_{i=3}^{c} v_i)$
p_{12} :	$A(EXv_1\, R\, EX\, E(v_2\, U\, \bigvee_{i=3}^{c} v_i)$

Properties p_{13} to p_{24} are similar to p_{01} to p_{12} with the difference of $\wedge$ and $\bigvee$ replaced by respectively $\vee$ and $\bigwedge$.

Experimental Data For the programs with concurrent processes, we let b vary over the set of values $\{12, 24, 36\}$, $a = 3$, $c = b/2$, and $d = c/a$. We use $cp12, cp24, cp36$ to denote respectively these subtypes of models for later references.

For the programs with concurrent sequential processes,we let b vary over the set of values $\{12, 16, 20\}$, $a = 2$, $c = b/2$, $t = c$, $p = 4$, and $d = c/a$. We use $sp12, sp16, sp20$ to denote respectively these subtypes.

Then each of the 20 properties is tested on 20 test cases for each of the subtypes. For brevity, we chose to present the experimental data for two properties for each type of the random programs. The two properties are selected by looking at the one NuSMV performs relatively best, and at the one VERDS performs relatively best, with respect to the ratios of the average times of their performance on the 20 test cases.

The data are shown in Table 3, in which the numbers represent times in seconds. The table is a summary of the data obtained for testing of 240 cases. The leftmost 2 columns of the table indicates the subtype of models, and the selected property. Then the left part of the table is the data obtained for model checking with NuSMV running on a Linux platform, using the command "NuSMV -dcx filename" in which the option means that counter example generation is not to be performed. The right part of the table is the data obtained for model checking with VERDS, using the command "verds filename".

m b	p	max	aver	max	aver	ratio
cp 12	p_{11}	0.0	0.0	0.1	0.0	0.83
cp 24	p_{11}	20.4	4.6	33.6	5.0	0.92
cp 36	p_{11}	1038.1	326.0	1028.7	175.7	1.86
cp 12	p_{02}	0.0	0.0	0.0	0.0	4.21
cp 24	p_{02}	15.7	3.8	0.1	0.1	61.43
cp 36	p_{02}	945.7	282.3	2.6	0.8	356.56
sp 12	p_{11}	0.6	0.2	5.8	2.0	0.11
sp 16	p_{11}	7.4	2.7	107.9	16.3	0.16
sp 20	p_{11}	192.0	62.8	644.7	176.4	0.36
sp 12	p_{02}	0.2	0.1	0.0	0.0	4.09
sp 16	p_{02}	3.7	1.8	0.1	0.1	15.78
sp 20	p_{02}	82.1	37.8	0.8	0.6	65.20

Table 5. Selected Experimental Data

Let $T(b, i)$ be the time for verification of the i-th ($i \in \{1, ..., 20\}$) test case with b variables for each of the *cp* or *sp* models, and let $lt.aver$ and $rt.aver$ denote the *aver* value in respectively the left part and the right part of the table. The explanation of $max, aver, ratio$ is as follows.

$$\begin{array}{|l} max(b) = \max_{1 \le i \le 20} T(b, i) \\ aver(b) = \sum_{1 \le i \le 20} T(b, i)/20 \\ ratio(b) = lt.aver(b)/rt.aver(b) \end{array}$$

Summary Comparing with the BDD based model checking algorithm implemented in NuSMV, VERDS has advantages in the evaluation with both types of random programs combined with a set of 24 CTL properties: with respect

to the first type of programs, VERDS has advantages in all 24 cases; with respect to the second type of programs, VERDS has advantages in 20 of the 24 cases. In total, VERDS has advantages in 44 of the 48 cases. Although NuSMV still has advantages in 4 of 24 cases with the second type of random programs, the advantages decrease when the size of the problem increases[7]. On the other hand, in most cases where VERDS has advantages, the differences seem to yield a coefficient of an exponential factor in the number of boolean variables (when testing a property with a set of models of different sizes).

3.3 Experimental Case Studies

Experimental case studies and comparisons of VERDS with NuSMV have already been reported in [13]. It was reported that one has advantages in some of the cases, while the other one has advantage in the other cases. This is a reasonable conclusion, since it is expected that BDD is very efficient, if a problem can be represented compactly (such problems certainly exist), while TBD may be more efficient when BDD is not able to represent a problem compactly. In this subsection, we provided in addition, two case studies, one on a mutual exclusion algorithm, and another on a cache coherent protocol referred to as GERMAN 2004 that has been used for case studies in various situations [16, 1, 12, 3].

Case Study with a Mutual Exclusion Algorithm

This subsection provides experimental data on model checking a mutual exclusion algorithm for $N+1$ pairs of processes with $N \in \{6, 7, 8, 9, 10\}$.

Description of the Model There are three global variables $s0[0..N]$, $s1[0..N]$ and $turn[0..N]$. These are arrays of length $N+1$, the first two are of enumeration type $\{noncr, trying, cr\}$, and the last is of type Boolean. The algorithm in the input format of NuSMV is as follows, in which N is to be replaced by a number in $\{6, 7, 8, 9, 10\}$ and "..." is to be replaced by the corresponding statements.

```
MODULE main
VAR
s0:    array 0..N of {noncr,trying,cr};
s1:    array 0..N of {noncr,trying,cr};
turn:  array 0..N of boolean;
p0_0:  process prc(0,s0,s1,turn,0);
p1_0:  process prc(0,s1,s0,turn,1);
.
.
.
p0_N:  process prc(N,s0,s1,turn,0);
p1_N:  process prc(N,s1,s0,turn,1);
```

[7] Details at http://lcs.ios.ac.cn/∼zwh/verds/files/ex130.pdf

```
ASSIGN
init(turn[0]):=0; ..... init(turn[N]):=0;

MODULE prc(i,s0,s1,turn,t0)
ASSIGN
init(s0[i]):=noncr;
next(s0[i]):=
case
  (s0[i]=noncr): {trying,noncr};
  (s0[i]=trying)&(s1[i]=noncr): cr;
  (s0[i]=trying)&(s1[i]=trying)&(turn[i]=t0): cr;
  (s0[i]=cr): {cr,noncr};
  1: s0[i];
esac;
next(turn[i]):=
case turn[i]=t0&s0[i]=cr:!turn[i]; 1:turn[i]; esac;
FAIRNESS running
FAIRNESS !(s0[i]=cr)
```

Property Specification The CTL properties to be verified are shown as follows, of which the first one is not valid (indicating that the mutual exclusion property is valid), the second and the third are valid, while the last two are not valid.

p_1 $EF(\bigvee_{i=0}^{N}((s0[i]=cr)\&(s1[i]=cr)))$
p_2 $AG(\bigwedge_{i=0}^{N}((s0[i]=trying)\rightarrow AF(s0[i]=cr)))$
p_3 $AG(\bigwedge_{i=0}^{N}((s1[i]=trying)\rightarrow AF(s1[i]=cr)))$
p_4 $AG(\bigwedge_{i=0}^{N}((s0[i]=cr)\rightarrow A[(s0[i]=cr)U$ $(!(s0[i]=cr)\&A[!(s0[i]=cr)U(s1[i]=cr)])]))$
p_5 $AG(\bigwedge_{i=0}^{N}((s1[i]=cr)\rightarrow A[(s1[i]=cr)U$ $(!(s1[i]=cr)\&A[!(s1[i]=cr)U(s0[i]=cr)])]))$

Experimental Data The experimental data for the time of model checking the mutual exclusion algorithm by NuSMV, VERDS and the ratio between the times are presented in Table 13 for $N = 8$ (with 18 processes). In order to have an idea on asymptotic behavior, we choose two properties: the one NuSMV performs relatively best (i.e., property p_4), and the one VERDS performs relatively best (i.e., property p_1), and present experimental data for $N \in \{6, 7, 8, 9, 10\}$ with time-out (denoted *t.o.*) set to 1 day (86400 seconds) in Table 14. The left part of the table is data for p_1 and the right part of the table is data for p_4.

Summary The ratios show that in both cases, the advantage of VERDS increases (or the disadvantage decreases) as the size of the problem instances increases, and VERDS has significant advantage over NuSMV when N is large for this kind of model checking problem instances.

	NuMSV	VERDS	ratio
p_1	26399.4	130.4	202.43
p_2	15557.2	4372.3	3.56
p_3	14261.5	3090.8	4.61
p_4	14205.5	9868.8	1.44
p_5	20075.9	9193.1	2.18

Table 6. Mutual Exclusion, $N = 8$

N	NuSMV	VERDS	ratio	NuSMV	VERDS	ratio
6	85.9	25.9	3.31	90.8	1672.6	0.05
7	973.1	55.9	17.42	979.5	4173.1	0.23
8	26399.4	130.4	202.43	14205.5	9868.8	1.44
9	t.o.	279.8	-	t.o.	56880.3	-
10	t.o.	600.5	-	t.o.	t.o.	-

Table 7. Mutual Exclusion, p_1 and p_4

Case Study with a Cache Coherent Protocol

This subsection provides experimental data on model checking a cache coherent protocol referred to as GERMAN 2004 for N processes with $N \in \{2, 3, 4\}$. For verification of such a complicated protocol, plain implementations of BDD based techniques or TBD based techniques are not sufficiently efficient[8]. Therefore, in the case of NuSMV, we use the option -coi for removing the variables not relevant to the given properties, and -AG for the use of the specially designed algorithm for checking AG properties; in the case of VERDS, corresponding techniques are also applied.

Property Specification Let $N[i].cs$ denote the the cache state of node i, cex denote the constant cache_exclusive, csh denote cache_shared, cin denote cache_invalid. The properties considered are safety ones as follows.

1	$AG!(N[0].cs = cex \,\&\, N[1].cs = cex)$
2	$AG!(N[0].cs = cex \,\&\, N[1].cs = csh)$
3	$AG!(N[0].cs = csh \,\&\, N[1].cs = cex)$
4	$AG(N[0].cs = cex \rightarrow N[1].cs = cin)$
5	$AG(N[1].cs = cex \rightarrow N[0].cs = cin)$

These properties represent relations between cache state of one node and that of another node. The average time (in seconds) for checking each property separately for models with different number of nodes by NuSMV and VERDS are as follows.

[8] For the case with 4 nodes, both programs run a week without reaching a conclusion.

nodes	NuSMV	VERDS	variables
2	0.67	16.01	94
3	33.78	154.72	134
4	8376.43	2288.47	195

The number of variables in the table represents that of Boolean variables involved in VERDS after removing the variables not relevant to the given properties. The increase of Boolean variables for each node is not a constant, since adding a node both adds a set of enumerative variables and increases the range of some enumeration types.

Summary In this case study, the verification times for both NuSMV and VERDS increase roughly exponentially in the number of Boolean variables (not in the number of nodes). However, the increase rate of NuSMV is higher (with the base of the exponential being 1.10 versus 1.05), and VERDS has advantage when the number of nodes reaches 4.

4 Concluding Remarks

Ternary Boolean diagrams are introduced and their connection to boolean formulas has been established. Complexity analysis has shown that there are cases TBDs can be more compact than BDDs for representation of Boolean functions. Experimental data are also reported, and support that there may be advantages with TBDs.

A connection of ternary Boolean diagrams to the problem of verification of finite state concurrent transition systems is established. The feasibility of such a verification approach has been demonstrated on two types of random boolean programs and experimental case studies. While in general, BDD based model checking and TBD based model checking have their own advantages. Experimental data in Section 3 have provided evidence that TBD based model checking may be more efficient in the given types of model checking problems. Apparently, such advantage is due to the compact representation of formulas and verification problems.

Regarding the efficiency of model checking, in the recent years, there is a lot of work on SAT based bounded model checking [2, 15, 17, 18] as an alternative to BDD based symbolic model checking. SAT based bounded model checking and BDD based model checking have their own advantages, i.e., SAT based one may be quick when the validity or falsity of a property can be determined with a relatively small bound, otherwise, BDD based one is still better when a large bound is necessary to reach a conclusion. Therefore all these approaches, including SAT-based model checking, BDD-based model checking and the proposed one, may be considered complementary, for enhancing the applicability of model checking to formal verification purposes.

Acknowledgement The author would like to thank Yi Lv and Ming Ma for providing sources of the case-study protocol, and Huimin Lin, Guangyuan Li and Zhilin Wu for helpful discussions and comments on this work.

A Proofs of Propositions of Section 2.3-2.6.

Before presenting the proofs, we define some notations as follows. We use $2^{\mathcal{L}}$ to denote the power set of $\mathcal{L}$; use $\mathcal{Y}$ to denote the set of elements such that $a \in \mathcal{Y}$ iff the size of a is the same as that of $\mathcal{L}$ and if $p_i \in \mathcal{L}$ then either p_i or $-p_i$ is in a, i.e., an elements of $\mathcal{Y}$ is of the form $\{y_1, ..., y_n, y_{n+1}\}$ such that $\{y_1, ..., y_n, y_{n+1}\} \subseteq \Sigma$ and $y_i \in \{p_i, -p_i\}$; use $ps(Y)$ to denote the set of positive elements of Y for $Y \in \mathcal{Y}$, such that $ps(Y) \in 2^{\mathcal{L}}$ for $Y \in \mathcal{Y}$. Additional notations to be used are as follows.

notation	meaning
$[[s]]$	the set of models of s
$[[s]] \oplus x$	$\{a \cup \{x\} \mid a \in [[s]]\}$
$[[s]] \ominus x$	$\{a \setminus \{x\} \mid a \in [[s]]\}$
$[[s]] \Cup W$	$\{a \cup w' \mid a \in [[s]], w \subseteq W\}$
$2^{\mathcal{L}}_{\oplus x}$	$(2^{\mathcal{L}} \oplus x)$
$2^{\mathcal{L}}_{\ominus x}$	$(2^{\mathcal{L}} \ominus x)$

Then $[[s]] \oplus x$ denotes the set of models obtained by adding x to each model of the set of models $[[s]]$, and $[[s]] \ominus x$ denotes the set of models obtained by removing x from each model of $[[s]]$. The precedence of the above operators is defined to be the lowest in an expression (to avoid excessive use of parenthesis).

Properties of $\ominus$ and $\oplus$ Let $C \subseteq 2^{\mathcal{L}}$. Let $x \in \mathcal{L}$. Let $A, B \subseteq 2^{\mathcal{L}}$ such that $Y \ominus x \in A$ iff $Y \oplus x \in A$ and $Y \ominus x \in B$ iff $Y \oplus x \in B$ for every $Y \in \mathcal{Y}$. The following are properties of $\ominus$ and $\oplus$.

$(A \cap B) \ominus x = (A \ominus x) \cap (B \ominus x)$ $(A \cap B) \oplus x = (A \oplus x) \cap (B \oplus x)$
$(C \oplus x) \cap (C \ominus x) = \emptyset$ $(C \oplus x) \cup (C) \quad = C \Cup \{x\}$
$((C \cap 2^{\mathcal{L}}_{\oplus x} \ominus x) \cap C) \Cup \{x\} = 2^{\mathcal{L}} \Leftrightarrow C = 2^{\mathcal{L}}$

A.1 Proof of Proposition 2

Proposition 2 Let $s_1, s_2, t_1, t_2 \in \mathcal{D}^{\mathcal{O}}$ and $x \in \mathcal{L}$. If $s_1 \equiv s_2$ and $t_1 \equiv t_2$, then $\neg s_1 \equiv \neg s_2$, $s_1 \wedge t_1 \equiv s_2 \wedge t_2$ and $abs(x)(s_1) \equiv abs(x)(s_2)$.

Proof: This proposition follows from the Lemma 4, Lemma 8 and Lemma 11.

Lemma 1. *Let $s \in \mathcal{D}^{\mathcal{O}}$. Let $\{y_1, ..., y_n, y_{n+1}\} \in \mathcal{Y}$. Then $s\lceil_{y_{n+1}} \cdots \lceil_{y_1} \in \{\tau, -\tau\}$*

Proof: Since $s \in \mathcal{D}^{\mathcal{O}}$, in general, we obtain that the labels remain in the $s\lceil_{y_{n+1}} \cdots \lceil_{y_k}$ is a subset of
$\{p_1, ..., p_{k-1}, p_{n+1}, -p_1, ..., -p_{k-1}, -p_{n+1}\}$. When k is 1, the labels remain in $s\lceil_{y_{n+1}} \cdots \lceil_{y_1}$ is a subset of $\{p_{n+1}, -p_{n+1}\}$, which is the same as $\{\tau, -\tau\}$. The lemma follows from this fact.

Lemma 2. *Let $s \in \mathcal{D}^{\mathcal{O}}$. Let $Y = \{y_1, ..., y_n, y_{n+1}\} \in \mathcal{Y}$. Then $ps(Y) \in [[s]]$ iff $s\lceil_{y_{n+1}} \cdots \lceil_{y_1} = \tau$.*

Proof: By definition, we have $s\lceil_{y_{n+1}} \cdots \lceil_{y_1} = \tau$ implies $ps(Y) \in [[s]]$. We only need to show that $ps(Y) \in [[s]]$ implies $s\lceil_{y_{n+1}} \cdots \lceil_{y_1} = \tau$. Suppose that this does not hold. Then we have $ps(Y) \in [[s]]$ and $s\lceil_{y_{n+1}} \cdots \lceil_{y_1} \neq \tau$. According to Lemma 1, we have $ps(Y) \in [[s]]$ and $s\lceil_{y_{n+1}} \cdots \lceil_{y_1} = -\tau$. This means that there is a sequence of elements of Y (allowing an element of Y appear 0 or more times in different places of the sequence) reducing s to τ and another sequence of elements of Y reducing s to $-\tau$. This is not possible, since in the reduction process, reduction on one element does not interfere with the reduction on another element (although if the sequence is not selected properly, we may not be able to reduce s completely to τ or $-\tau$).

Lemma 3. *Let $s \in \mathcal{D}^{\mathcal{O}}$. Then we have $s\lceil_{z_1} \cdots \lceil_{z_k} = \tau$ iff $\neg s\lceil_{z_1} \cdots \lceil_{z_k} = -\tau$.*

Proof: The reduction process of $s\lceil_{z_1} \cdots \lceil_{z_k}$ and that of $\neg s\lceil_{z_1} \cdots \lceil_{z_k}$ are the same except the last step in which the sign of the result is calculated according to the sign of the topmost label of s and that of $\neg s$.

Lemma 4. *Let $s \in \mathcal{D}^{\mathcal{O}}$. Then $[[\neg s]] = 2^{\mathcal{L}} \setminus [[s]]$.*

Proof: Let $Y = \{y_1, ..., y_n, y_{n+1}\} \in \mathcal{Y}$. Then the following holds.

$$\begin{aligned} [[s]] &= \{ps(Y) \mid s\lceil_{y_{n+1}} \cdots \lceil_{y_1} = \tau\} \\ [[\neg s]] &= \{ps(Y) \mid \neg s\lceil_{y_{n+1}} \cdots \lceil_{y_1} = \tau\} \\ &= \{ps(Y) \mid s\lceil_{y_{n+1}} \cdots \lceil_{y_1} = -\tau\} \\ &= 2^{\mathcal{L}} \setminus [[s]] \end{aligned}$$

The first and the second equalities hold, according to Lemma 2. The third holds, according to Lemma 3. The fourth holds, according to Lemma 1.

Lemma 5. *Let $s = (x, a, b, c) \in \mathcal{D}^{\mathcal{O}}$. Let $y > |x|$. Let $m \in 2^{\mathcal{L}}$. $m \cup \{y\} \in [[s]]$ iff $m \setminus \{y\} \in [[s]]$.*

Proof: This follows from the definition of models, since y does affect the evaluation of s in the process of deciding whether m is a model of s.

Lemma 6. *$[[\tau]] = 2^{\mathcal{L}}$ and $[[-\tau]] = \emptyset$.*

Proof: This follows from the definition of models.

Lemma 7. *Let $s = (x, a, b, c) \in \mathcal{D}^{\mathcal{O}} \setminus \{\tau, -\tau\}$. Then*

$$[[s]] = \begin{cases} ([[a]] \cap [[c]]) \ominus x) \cup ([[b]] \cap [[c]] \oplus x) \text{ for } x > 0 \\ 2^{\mathcal{L}} \setminus ([[a]] \cap [[c]]) \ominus -x) \cup ([[b]] \cap [[c]] \oplus -x) \text{ for } x < 0 \end{cases}$$

Proof: We prove the case with $x > 0$. The other case follows from this case and Lemma 4, since $(x, a, b, c) = \neg(-x, a, b, c)$.

Suppose that $s = (p_i, a, b, c)$ with $1 \leq i \leq n$.

Considered any given $Y = \{y_1, ..., y_n, y_{n+1}\} \in \mathcal{Y}$.

Then $ps(Y) \in [[s]]$ iff $ps(Y) \in ([[b]] \wedge [[c]]) \oplus p_i$ or $ps(Y) \in ([[a]] \wedge [[c]]) \ominus p_i$ is proved as follows.

Since (p_i, a, b, c) is an ordered TBD,
$a\lceil_{y_{n+1}} \cdots \lceil_{y_{i+1}} = a\lceil_{y_{n+1}} \cdots \lceil_{y_1}$,
$b\lceil_{y_{n+1}} \cdots \lceil_{y_{i+1}} = b\lceil_{y_{n+1}} \cdots \lceil_{y_1}$,
$c\lceil_{y_{n+1}} \cdots \lceil_{y_{i+1}} = c\lceil_{y_{n+1}} \cdots \lceil_{y_1}$.
Let u' denote $u\lceil_{y_{n+1}} \cdots \lceil_{y_{i+1}}$ where u denotes one of a, b, c.
Then $a', b', c' \in \{\tau, -\tau\}$ and $s\lceil_{y_{n+1}} \cdots \lceil_{y_{i+1}} = (p_i, a', b', c')$.
Therefore
$ps(Y) \in [[s]]$
iff $p_i \in Y$ and $b' = c' = \tau$ or $-p_i \in Y$ and $a' = c' = \tau$
iff $ps(Y) \in ([[b]] \wedge [[c]]) \oplus p_i$ or $ps(Y) \in ([[a]] \wedge [[c]]) \ominus p_i$.

Lemma 8. *Let* $s, t \in \mathcal{D}^{\mathcal{O}}$. *Then* $[[s \wedge t]] = [[s]] \cap [[t]]$.

Proof: Proof by induction on the structure of t.

- Base case:
  There are 4 cases with $t \in \{\tau, -\tau\}$ or $s \in \{\tau, -\tau\}$. We only consider the first two cases. The other cases are similar.
  Subcase 1: $t = \tau$.
  $[[s \wedge t]] = [[s \wedge \tau]] = [[s]] = [[s]] \cap [[\tau]] = [[s]] \cap [[t]]$.
  Subcase 2: $t = -\tau$.
  $[[s \wedge t]] = [[s \wedge -\tau]] = [[-\tau]] = [[s]] \cap [[-\tau]] = [[s]] \cap [[t]]$.
- Inductive step: $s = (x, a, b, c)$, $t = (x', a', b', c')$, and $s, t \notin \{\tau, -\tau\}$.
  Conjunction is defined with 8 cases (5 explicitly defined and 3 implicitly). We only consider the first 2 cases. The other cases are similar.
  Subcase 1: $x = x' \wedge x > 0$.
  $[[s \wedge t]]$
  $= [[(x, a \wedge a', b \wedge b', c \wedge c')]]$
  $= ([[a \wedge a']] \cap [[c \wedge c']] \ominus x) \cup ([[b \wedge b']] \cap [[c \wedge c']] \oplus x)$
  $= (([[a]] \cap [[a']]) \cap ([[c]] \cap [[c']]) \ominus x) \cup (([[b]] \cap [[b']]) \cap ([[c]] \cap [[c']]) \oplus x)$
  $= (([[a]] \cap [[c]]) \cap ([[a']] \cap [[c']]) \ominus x) \cup (([[b]] \cap [[c]]) \cap ([[b']] \cap [[c']])) \oplus x)$
  $= (([[a]] \cap [[c]] \ominus x) \cap ([[a']] \cap [[c']] \ominus x)) \cup (([[b]] \cap [[c]]) \oplus x) \cap ([[b']] \cap [[c']] \oplus x)$
  $= (([[a]] \cap [[c]] \ominus x) \cup ([[b]] \cap [[c]] \oplus x)) \cap (([[a']] \cap [[c']] \ominus x) \cup ([[b']] \cap [[c']] \oplus x))$
  $= [[s]] \cap [[t]]$
  The second equality holds, according to Lemma 7. The fourth holds, according to the induction hypothesis. The fifth holds, according to Lemma 5 and the properties of $\ominus$ and $\oplus$. The sixth holds, according to the properties of $\ominus$ and $\oplus$.
  Subcase 2: $x = x' \wedge x < 0$.
  Let $y = -x$.
  $[[s \wedge t]]$

$= [[(x, ((a \wedge c) \vee (a' \wedge c')), ((b \wedge c) \vee (b' \wedge c')), \tau)]]$
$= 2^{\mathcal{L}} \setminus$
$((([[((a \wedge c) \vee (a' \wedge c'))]] \cap [[\tau]]) \ominus y) \cup$
$(([[((b \wedge c) \vee (b' \wedge c'))]] \cap [[\tau]]) \oplus y)$
$= 2^{\mathcal{L}} \setminus$
$((([[a]] \cap [[c]]) \cup ([[a']] \cap [[c']]) \ominus y) \cup$
$(([[b]] \cap [[c]]) \cup ([[b']] \cap [[c']]) \oplus y)$
$= 2^{\mathcal{L}} \setminus$
$((([[a]] \cap [[c]] \ominus y) \cup ([[a']] \cap [[c']] \ominus y)) \cup$
$(([[b]] \cap [[c]] \oplus y) \cup ([[b']] \cap [[c']] \oplus y))$
$= 2^{\mathcal{L}} \setminus$
$((([[a]] \cap [[c]] \ominus y) \cup ([[b]] \cap [[c]] \ominus y)) \cup$
$(([[a']] \cap [[c']] \ominus y) \cup ([[b']] \cap [[c']] \oplus y))$
$= (((2^{\mathcal{L}} \setminus [[a]] \cap [[c]]) \ominus y) \cup ([[b]] \cap [[c]] \oplus y)) \cap$
$(((2^{\mathcal{L}} \setminus [[a']] \cap [[c']] \ominus y) \cup ([[b']] \cap [[c']]) \oplus y))$
$= [[s]] \cap [[t]]$

Lemma 9. *Let* $s = (x, a, b, c) \in \mathcal{D}^{\mathcal{O}} \setminus \{\tau, -\tau\}$. *Let* $x > 0$. *Then* $[[s]] \cap 2^{\mathcal{L}}_{\oplus x} = [[b \wedge c]] \oplus x$ *and* $[[s]] \cap 2^{\mathcal{L}}_{\ominus x} = [[a \wedge c]] \ominus x$.

Proof: The proof is as follows.

- Case 1:
  $[[s]] \cap 2^{\mathcal{L}}_{\oplus x}$
  $= (([[a]] \cap [[c]] \ominus x) \cup ([[b]] \cap [[c]] \oplus x)) \cap 2^{\mathcal{L}}_{\oplus x}$
  $= ([[b]] \cap [[c]] \oplus x) \cap 2^{\mathcal{L}}_{\oplus x}$
  $= ([[b]] \cap [[c]] \oplus x)$
  $= ([[b \wedge c]] \oplus x)$
  The second equality holds, according to Lemma 7.
- Case 2:
  $[[s]] \cap 2^{\mathcal{L}}_{\ominus x}$
  $= (([[a]] \cap [[c]] \ominus x) \cup ([[b]] \cap [[c]] \oplus x)) \cap 2^{\mathcal{L}}_{\ominus x}$
  $= ([[a]] \cap [[c]]) \ominus x) \cap 2^{\mathcal{L}}_{\ominus x}$
  $= ([[a]] \cap [[c]] \ominus x)$
  $= ([[a \wedge c]] \ominus x)$

Lemma 10. *Let* $s \in \mathcal{D}^{\mathcal{O}}$. *Let* $w \in \Sigma$. *Then*
$[[s|_w]] = \begin{cases} ([[s]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\} \text{ for } w > 0 \\ ([[s]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\} \text{ for } w < 0 \end{cases}$

Proof: Proof by induction on the structure of s.

- Base case: $s \in \{\tau, -\tau\}$.
  Subcase 1: $w > 0$.
  $[[\tau|_w]] = [[\tau]] = 2^{\mathcal{L}} = ([[\tau]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\}$
  $[[(-\tau)|_w]] = [[-\tau]] = \emptyset = ([[-\tau]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\}$
  Subcase 2: $w < 0$.
  $[[\tau|_w]] = [[\tau]] = 2^{\mathcal{L}} = ([[\tau]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\}$
  $[[(-\tau)|_w]] = [[-\tau]] = \emptyset = ([[-\tau]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\}$

– Inductive step: $s = (x, a, b, c)$ and $s \notin \{\tau, -\tau\}$.
Subcase 1: $|x| = |w|$.
Subcase 1a: $w > 0$ and $x > 0$. Note that $x = w$.
$$\begin{aligned}[[s|_w]] &= [[b \wedge c]] \\ &= ([[b \wedge c]] \oplus w) \cup ([[b \wedge c]] \ominus w) \\ &= ([[b \wedge c]] \oplus w) \cup (([[b \wedge c]] \oplus w) \ominus w) \\ &= ([[s]] \cap 2^{\mathcal{L}}_{\oplus w}) \cup (([[s]] \cap 2^{\mathcal{L}}_{\oplus w}) \ominus w) \\ &= (([[s]] \cap 2^{\mathcal{L}}_{\oplus w}) \ominus w) \Cup \{w\}\end{aligned}$$
The second equality holds, according to Lemma 5 and the properties of $\ominus$ and $\oplus$. The fourth holds, according to Lemma 9.
Subcase 1b: $w < 0$ and $x > 0$. Note that $x = -w$.
$$\begin{aligned}[[s|_w]] &= [[a \wedge c]] \\ &= ([[a \wedge c]] \ominus -w) \cup ([[a \wedge c]] \oplus -w) \\ &= ([[a \wedge c]] \ominus -w) \cup (([[a \wedge c]] \ominus -w) \oplus -w) \\ &= ([[s]] \cap 2^{\mathcal{L}}_{\ominus -w}) \cup (([[s]] \cap 2^{\mathcal{L}}_{\ominus -w}) \oplus -w) \\ &= ([[s]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\}\end{aligned}$$
Subcase 1c: $w > 0$ and $x < 0$. Let $t = -s = (w, a, b, c)$.
$$\begin{aligned}[[s|_w]] &= [[x \cdot (b \wedge c)]] \\ &= 2^{\mathcal{L}} \setminus [[b \wedge c]] \\ &= 2^{\mathcal{L}} \setminus (([[t]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\}) \\ &= ((2^{\mathcal{L}} \setminus [[t]]) \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\} \\ &= ([[s]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\}\end{aligned}$$
Subcase 1d: $w < 0$ and $x < 0$. Let $t = -s = (-w, a, b, c)$.
$$\begin{aligned}[[s|_w]] &= [[x \cdot (a \wedge c)]] \\ &= 2^{\mathcal{L}} \setminus [[a \wedge c]] \\ &= 2^{\mathcal{L}} \setminus (([[t]] \cap 2^{\mathcal{L}}_{\ominus -w} \Cup \{-w\}) \\ &= (((2^{\mathcal{L}} \setminus [[t]]) \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\} \\ &= ([[s]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\}\end{aligned}$$
Subcase 2: $|x| \neq |w|$. Then $s|_w = (x, a|_w, b|_w, c|_w)$.
According to the induction hypothesis, we have the following equality for $u \in \{a, b, c\}$.
$$[[u|_w]] = \begin{cases} ([[u]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\} \text{ for } w > 0 \\ ([[u]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\} \text{ for } w < 0 \end{cases}$$
Subcase 2a: $w > 0$ and $x > 0$.
$$\begin{aligned}&[[s|_w]] \\ &= ([[a|_w]] \cap [[c|_w]] \ominus x) \cup ([[b|_w]] \cap [[c|_w]] \oplus x) \\ &= ((([[a \wedge c]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\}) \ominus x) \cup ((([[b \wedge c]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\}) \oplus x) \\ &= (((([[a \wedge c]] \ominus x) \cup ([[b \wedge c]] \oplus x)) \cap 2^{\mathcal{L}}_{\oplus w}) \ominus w) \Cup \{w\} \\ &= ([[s]] \cap 2^{\mathcal{L}}_{\oplus w}) \ominus w) \Cup \{w\}\end{aligned}$$
Subcase 2b: $w < 0$ and $x > 0$.
$$\begin{aligned}&[[s|_w]] \\ &= ([[a|_w]] \cap [[c|_w]] \ominus x) \cup ([[b|_w]] \cap [[c|_w]] \oplus x)) \\ &= ((([[a \wedge c]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{\emptyset, -w\}) \ominus x) \cup ((([[b \wedge c]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\}) \oplus x) \\ &= (((([[a \wedge c]] \ominus x) \cup ([[b \wedge c]] \oplus x)) \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\} \\ &= ([[s]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\}\end{aligned}$$
Subcase 2c: $w > 0$ and $x < 0$.

Let $y = -x$. Let $t = -s = (y, a, b, c)$.
$[[s|_w]]$
$= 2^{\mathcal{L}} \setminus ([[a|_w]] \cap [[c|_w]] \ominus y) \cup ([[b|_w]] \cap [[c|_w]] \oplus y)$
$= 2^{\mathcal{L}} \setminus (([[t]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\})$
$= (((2^{\mathcal{L}} \setminus [[t]]) \cap 2^{\mathcal{L}}_{\oplus w}) \ominus w) \Cup \{w\}$
$= ([[s]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\}$
Subcase 2d: $w < 0$ and $x < 0$.
Let $y = -x$. Let $t = -s = (y, a, b, c)$.
$[[s|_w]]$
$= 2^{\mathcal{L}} \setminus ([[a|_w]] \cap [[c|_w]] \ominus y) \cup ([[b|_w]] \cap [[c|_w]] \oplus y)$
$= 2^{\mathcal{L}} \setminus (([[t]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\})$
$= (((2^{\mathcal{L}} \setminus [[t]]) \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\}$
$= ([[s]] \cap 2^{\mathcal{L}}_{\ominus -w}) \Cup \{-w\}$

Lemma 11. *Let $s \in \mathcal{D}^{\mathcal{O}}$. Let $w > 0$. Then* $[[abs(w)(s)]] = (([[s]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \cap [[s]]) \Cup \{w\}$.

Proof: The proof is as follows.
$[[abs(w)(s)]]$
$= [[s|_w]] \cap [[s|_{-w}]]$
$= (([[s]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \Cup \{w\}) \cap (([[s]] \cap 2^{\mathcal{L}}_{\ominus w}) \Cup \{w\})$
$= (([[s]] \cap 2^{\mathcal{L}}_{\oplus w} \ominus w) \cap [[s]]) \Cup \{w\}$
The second equality holds, according to Lemma 10.

A.2 Proof of Proposition 3

Proposition 3 Let $r \in \mathcal{D}^{\mathcal{O}}$. r is complete iff $comp(r)$ holds.

Proof: This follows from Lemma 12.

Lemma 12. *Let $r \in \mathcal{D}^{\mathcal{O}}$. $[[r]] = 2^{\mathcal{L}}$ iff $comp(r)$ holds.*

Proof: The proof is as follows.
$comp(r)$
$\Leftrightarrow abs(p_1) \cdots abs(p_n)(r) = \tau$
$\Leftrightarrow [[abs(p_1) \cdots abs(p_n)(r)]] = 2^{\mathcal{L}}$
$\Leftrightarrow [[r]] = 2^{\mathcal{L}}$
The last equivalence holds, according to the properties of $\ominus$ and $\oplus$, and the repeated use of Lemma 11.

A.3 Proof of Proposition 4

Proposition 4 Let $x, \varphi_0, \varphi_1 \in \Phi(\mathcal{L}_I)$ and $s, t \in \mathcal{D}^{\mathcal{O}}$. If $x \in \mathcal{L}_I$ is an atomic formula, then $x \equiv (x, -\tau, \tau, \tau)$, and if $\varphi_0 \equiv s$ and $\varphi_1 \equiv t$, then $\neg\varphi_0 \equiv \neg s$ and $\varphi_0 \wedge \varphi_1 \equiv s \wedge t$.

Proof: Let $[[\varphi]]$ denote the set of models of φ.

- Let $p_i \in \mathcal{L}_I$ be an atomic formula, then $[[p_i]] = 2^{\mathcal{L}} \oplus p_i$.
  On the other hand, according to Lemma 7,
  $[[(p_i, -\tau, \tau, \tau)]]$
  $= (([[-\tau]] \wedge [[\tau]]) \ominus p_i) \cup (([[\tau]] \wedge [[\tau]]) \oplus p_i)$
  $= 2^{\mathcal{L}} \oplus p_i$.
- $\neg\varphi_0 \equiv \neg s$ follows from that $[[\neg\varphi_0]] = 2^{\mathcal{L}} \setminus [[\varphi_0]]$ and $[[\neg s]] = 2^{\mathcal{L}} \setminus [[s]]$ by inductive arguments.
- $\varphi_0 \wedge \varphi_1 \equiv s \wedge t$ follows from that $[[\varphi_0 \wedge \varphi_1]] = [[\varphi_0]] \wedge [[\varphi_1]]$ and $[[s \wedge t]] = [[s]] \wedge [[t]]$ by inductive arguments.

A.4 Proof of Proposition 5

Proposition 5 Let $\varphi \in \Phi(\mathcal{L}_I)$ and $s \in \mathcal{D}^{\mathcal{O}}$. If $\varphi \equiv s$, then $\forall x.\varphi \equiv abs(x)(s)$.

Proof: The proof is as follows.
$[[\forall x.\varphi]]$
$= ([[\varphi]] \cap ([[\varphi]] \cap (2^{\mathcal{L}}_{\oplus x}) \ominus x)) \Cup \{x\}$
$= ([[s]] \cap ([[s]] \cap (2^{\mathcal{L}}_{\oplus x}) \ominus x)) \Cup \{x\}$
$= [[abs(x)(s)]]$
The first equality holds, according to the semantics of $\forall x.\varphi$ and the last equivalence holds, according to Lemma 11.

A.5 Proof of Proposition 8

Proposition 8 For every $s, t \in \mathcal{D}^{\mathcal{O}}$, if $s \xrightarrow{*} t$, then $s \equiv t$.

Proof: For each rewrite rule, the equivalence follows from Lemma 6 and Lemma 7. As an example, we prove $(x, -\tau, -\tau, c) \equiv (x \cdot -\tau)$.

- Case 1: $x > 0$.
  $[[(x, -\tau, -\tau, c)]]$
  $= ([[-\tau]] \cap [[c]] \ominus x) \cup ([[-\tau]] \cap [[c]] \oplus x) = \emptyset = [[-\tau]] = [[x \cdot -\tau]]$
- Case 2: $x < 0$.
  $[[(x, -\tau, -\tau, c)]]$
  $= 2^{\mathcal{L}} \setminus ([[-\tau]] \cap [[c]] \ominus x) \cup ([[-\tau]] \cap [[c]] \oplus x) = 2^{\mathcal{L}} = [[\tau]] = [[-x \cdot -\tau]]$

A.6 Proof of Proposition 10

Proposition 10 Let $p > 0$ and $q > 0$. Each of the following pairs represent the same formula with different variable orderings on p and q.

$(p,(q,a,b,c),c',c'')$ $(q,(p,a,\tau,\tau),(p,b,\tau,\tau),(p,c,c',c''))$
$(p,c,(q,a',b',c'),c'')$ $(q,(p,\tau,a',\tau),(p,\tau,b',\tau),(p,c,c',c''))$
$(p,c,c',(q,a'',b'',c''))$ $(q,(p,\tau,\tau,a''),(p,\tau,\tau,b''),(p,c,c',c''))$
$(p,(q,a,b,c),(q,a',b',c'),c'')$ $(q,(p,a,a',\tau),(p,b,b',\tau),(p,c,c',c''))$
$(p,(q,a,b,c),c',(q,a'',b'',c''))$ $(q,(p,a,\tau,a''),(p,b,\tau,b''),(p,c,c',c''))$
$(p,c,(q,a',b',c'),(q,a'',b'',c''))$ $(q,(p,\tau,a',a''),(p,\tau,b',b''),(p,c,c',c''))$
$(p,(q,a,b,c),(q,a',b',c'),(q,a'',b'',c''))$ $(q,(p,a,a',a''),(p,b,b',b''),(p,c,c',c''))$

Proof: This follows from Lemma 6 and Lemma 7. As an example, we prove the first equivalence as follows.

$$
\begin{aligned}
&[[p,(q,a,b,c),c',c'')]]\\
&= (([[(q,a,b,c)]] \cap [[c'']] \ominus p) \cup ([[c']] \cap [[c'']] \oplus p)\\
&= ((((([[a]] \cap [[c]] \ominus q) \cup ([[b]] \cap [[c]] \oplus q)) \cap [[c'']]) \ominus p) \cup ([[c']] \cap [[c'']] \oplus p)\\
&= ((([[a]] \cap [[c]] \ominus q) \cap [[c'']] \ominus p) \cup ((([[b]] \cap [[c]] \oplus q) \cap [[c'']] \ominus p) \cup\\
&\quad ([[c']] \cap [[c'']] \oplus p)\\
&= ((([[a]] \cap [[c]] \cap [[c'']] \ominus p) \ominus q) \cup ((([[b]] \cap [[c]] \cap [[c'']] \ominus p) \oplus q) \cup\\
&\quad ([[c']] \cap [[c'']] \oplus p)\\
&= ((([[a]] \cap [[c]] \cap [[c'']] \ominus p) \cup ([[c']] \cap [[c'']] \oplus p)) \ominus q) \cup\\
&\quad ((([[b]] \cap [[c]] \cap [[c'']] \ominus p) \cup ([[c']] \cap [[c'']] \oplus p)) \oplus q)\\
&= ((([[a]] \ominus p) \cup (2^{\mathcal{L}} \oplus p)) \cap (([[c]] \cap [[c'']] \ominus p) \cup ([[c']] \cap [[c'']] \oplus p)) \ominus q) \cup\\
&\quad ((([[b]] \ominus p) \cup (2^{\mathcal{L}} \oplus p)) \cap (([[c]] \cap [[c'']] \ominus p) \cup ([[c']] \cap [[c'']] \oplus p)) \oplus q)\\
&= ([[(p,a,\tau,\tau)]] \cap [[(p,c,c',c'')]] \ominus q) \cup ([[(p,b,\tau,\tau)]] \cap [[(p,c,c',c'')]] \oplus q)\\
&= [[(q,(p,a,\tau,\tau),(p,b,\tau,\tau),(p,c,c',c''))]]
\end{aligned}
$$

B Proofs of Propositions of Section 2.7

Before presenting the proofs, we define some notations as follows. $bdd_v(\varphi)$ denotes the reduced ordered BDD of φ with the variable order specified in v. $positive(t)$ denotes that the TBD t is an ordered TBD where all non-terminal nodes are marked by positive labels. A TBD t is said to be positive, if $positive(t)$ holds.

B.1 Proof of Proposition 11

Proposition 11 There is some variable ordering, such that the reduced ordered BDD representation of φ has node size $\geq 2^n$, while for all variable orderings, we can construct a TBD representation for φ with tree size and node size linear in n.

Proof: It is known that representations of φ by reduced ordered BDDs varies from $3 \cdot n + 2$ to $3 \cdot 2^n - 1$ depending on the ordering of the variables [8]. The second part of this proposition follows from Lemma 15, which is to be established as follows.

Lemma 13. *Let s, t be positive TBDs. Then we can construct a positive TBD for $s \wedge t$ such that $||s \wedge t|| \leq ||s|| + ||t|| - 1$.*

Proof: Since $\tau \wedge u = u \wedge \tau = u$ and $-\tau \wedge u = u \wedge -\tau = -\tau$ for any TBD u, this lemma holds when one of s and t is τ or $-\tau$. Let $s = (x, a, b, c)$ and $t = (x', a', b', c')$. We have three cases $x < x'$, $x = x'$ or $x > x'$.

- If $x < x'$, we have $s \wedge t = (x, a, b, c \wedge t)$.
  Then $||s \wedge t|| \leq ((||s|| - ||a|| - ||b|| - 1) + ||t|| - 1) + ||a|| + ||b|| + 1 = ||s|| + ||t|| - 1$.
- If $x = x'$, we have $s \wedge t = (x, a \wedge a', b \wedge b', c \wedge c')$.
  Then $||s \wedge t|| \leq (||a|| + ||a'|| - 1) + (||b|| + ||b'|| - 1) + (||c|| + ||c'|| - 1) + 1 < ||s|| + ||t|| - 1$.
- If $x > x'$, we have $s \wedge t = (x', a', b', c' \wedge s)$.
  Similar to the first case, $||s \wedge t|| \leq ||s|| + ||t|| - 1$.

Lemma 14. *For any ordering v of variables of φ, we can construct a positive TBD for φ_i with tree size* 10.

Proof: This follows from the construction demonstrated in Example 5 in Section 2.5.

Lemma 15. *For any ordering v of variables of φ, we can construct a positive TBD t for φ with $||t|| \leq 9n + 1$ and $|t| \leq 3n + 2$.*

Proof: According to Lemma 14, for each φ_i, we construct a positive TBD with tree size 10. Then according to Lemma 13, for φ, we construct a positive TBD t with $||t|| \leq n \cdot 10 - (n - 1) = 9n + 1$. This implies $|t| \leq 3n + 2$.

B.2 Proof of Proposition 13

Before presenting the proof of Proposition 13, we introduce two lemmas.

Lemma 16. *Let ϕ be a formula.*
Then $|bdd_v(\phi)| \geq max(|bdd_v(\phi|_{x=0})|, |bdd_v(\phi|_{x=1})|)$ for any x and any ordering v of variables.

Proof: Starting with $bdd_v(\phi)$, in order to create $bdd_v(\phi|_{x=0})$, all pointers to the node labeled with x in $bdd_v(\phi)$ are redirected to the left branch of the node (with possibly further reductions). This modification will not increase the number of nodes.

Lemma 17. *Let ϕ be $(a_1 \leftrightarrow b_1) \wedge \cdots \wedge (a_n \leftrightarrow b_n)$.*
Let $(x_1, ..., x_n)$ and $(y_1, ..., y_n)$ be permutations of respectively $\{a_1, ..., a_n\}$ and $\{b_1, ..., b_n\}$. Let $v = x_1 ... x_n y_1 ... y_n$. Then $|bdd_v(\phi)| = 3 \cdot 2^n - 1$.

Proof: A similar result is known when $v = a_1...a_n b_1...b_n$ [8]. For $v = x_1...x_n y_1...y_n$, the reasoning is as follows.

- Expanding $x_1, ..., x_n$ we have $2^n - 1$ nodes and 2^n different cases (of type 0). 2^{n-1} of the cases (where the variable corresponding with y_1 is assigned false) of type 0 point to false when y_1 assigns false, creating 2^{n-1} new y_1 nodes. 2^{n-1} of the cases (where the variable corresponding with y_1 is assigned true) of type 0 point to false when y_1 assigns true, creating another 2^{n-1} new y_1 nodes. There remains 2^n undecided cases. Half of the cases are the same as (or symmetric with respect to the truth assignments of y_1 and the corresponding x-variable) to the other half of the cases. Therefore there remain 2^{n-1} different cases (of type 1).
- 2^{n-2} of the cases (where the variable corresponding with y_2 is assigned false) of type 1 point to false when y_2 assigns false, creating 2^{n-2} new y_2 nodes. 2^{n-2} of the cases (where the variable corresponding with y_2 is assigned true) of type 1 point to false when y_2 assigns true, creating another 2^{n-2} new y_2 nodes. Similarly, there remain 2^{n-2} different cases (of type 2).
- Generally, at the i-th round, there remains 2^{n-i} different cases of types i. Half of the cases point to false when y_{i+1} assigns false, creating 2^{n-i} new y_{i+1} nodes, and half of the cases point to false when y_{i+1} assigns true, also creating 2^{n-i} new y_{i+1} nodes. This process continues until there remains 2 different cases, requiring two new y_n nodes.
- Therefore there are 2^{n-i+1} y_i nodes, by looking backward, we know that they must be different, and in total, there are $2^{n+1} - 2$ different nodes for $y_1, ..., y_n$. Summing up the number of nodes for $x_1, ..., x_n, y_1, ..., y_n$ and the two terminal nodes, we have $|bdd_v(\phi)| = 3 \cdot 2^n - 1$.

Proposition 13 For any ordering of variables of ψ, if d is a BDD for ψ with such an ordering, then $|d| \geq 3 \cdot 2^{n-1} - 1$.

Proof: Let $o = c_1 \cdots c_m$ be an ordering of variables of ψ. We prove that $|bdd_o(\psi)| \geq 3 \cdot 2^{n-1} - 1$. Let $o = c_1 \cdots c_m$ such that $(c_1, ..., c_m)$ is a permutation of $\{a_{i,j} | 1 \leq i, j \leq n\} \cup \{b\}$. Let $v(\varphi)$ denote the set of variables appearing in φ. Let $o_i = \{c_1, ..., c_i\}$. Let k be the least number such that $|v(\varphi_i) \cap o_m| = n - 1$ or $|v(\varphi'_i) \cap o_m| = n - 1$ for some $0 \leq 1 \leq n$. The proof is as follows.

- Either $|v(\varphi_l) \cap o_k| = n - 1$ or $|v(\varphi'_l) \cap o_k| = n - 1$ for some l.
- Suppose that the former is the case (the latter is symmetric). Let $S = v(\varphi_l) \cap o_k = \{a_{1,l}, ..., a_{x-1,l}, a_{x+1,l}, ..., a_{i,l}\}$.
- For each $a_{i,j} \in S$, we select an $a_{i,y_i} \notin o_k$. Let S' denote the set of these variables, and φ'' denote φ' with every variable not in $S \cup S'$ replaced by 1.
- Then φ'' is a formula corresponding to the one in Lemma 17 with $2(n-1)$ variables.
- According to Lemma 16 and Lemma 17, $|bdd_o(\psi)| \geq |bdd_o(\psi|_{b=0})| = |bdd_o(\varphi')| \geq |bdd_o(\varphi'')| = 3 \cdot 2^{n-1} - 1$.

B.3 Proof of Proposition 14

Before presenting the proof of Proposition 14, we introduce some lemmas.

Lemma 18. *Let s, t be positive TBDs. Let $[t]$ denote the height of the TBD t with $[\tau] = [-\tau] = 0$. If s and t have no variables in common, then we can construct a positive TBD for $s \wedge t$ with at most $[s] + [t]$ new nodes.*

Proof: By induction. This lemma holds when one of s and t is τ or $-\tau$. Let $s = (x, a, b, c)$ and $t = (x', a', b', c')$. We have two cases $x < x'$ or $x > x'$.

- If $x < x'$, we have $s \wedge t = (x, a, b, c \wedge t)$.
  According to the induction hypothesis, $c \wedge t$ requires at most $[c] + [t]$ new nodes, therefore $s \wedge t$ requires at most $[c] + [t] + 1 \leq [s] + [t]$ new nodes.
- If $x > x'$, we have $s \wedge t = (x', a', b', c' \wedge s)$.
  Similarly, $s \wedge t$ requires at most $[c'] + [s] + 1 \leq [s] + [t]$ new nodes.

Lemma 19. *Let ϕ_i be $(a_1 \leftrightarrow \cdots \leftrightarrow a_i)$. We can construct a positive TBD t for ϕ_n with $|t| \leq 2(n+1)$ for any ordering of variables.*

Proof: In fact, we can create a positive TBD for ϕ_n and a positive TBD for $\neg\phi_n$ such that the total number of different nodes in the two TBDs is $2(n+1)$. The reasoning by induction is as follows. The statement is true when $n = 1$. Then we can use the TBD t_0 for ϕ_{n-1} and the TBD t_1 for $\neg\phi_{n-1}$ to build a positive TBD $t'_0 = (a_n, t_1, t_0, \tau)$ for ϕ_n and a positive TBD $t'_1 = (a_n, t_0, t_1, \tau)$ for $\neg\phi_n$ with only two additional new nodes. The construction assumes variable order $v = a_n \cdots a_1$. Since the variables in ϕ_i are all symmetric, this construction can be done for any variable order by rearranging the position of the variables in the formula.

Lemma 20. *For any ordering v of variables of ψ, we can construct a positive TBD t for ψ with $|t| \leq 4n^2(n+1)+1$ for the modified variable ordering $v[b \to 1]$.*

Proof: The proof is as follows.

- According to Lemma 19, for each φ_i, we can construct a corresponding TBD with $2(n+1)$ nodes. According to Lemma 18, for φ, we can construct a corresponding TBD t with $|t| \leq 2(n+1) \cdot n + (n-1) \cdot 2 \cdot n^2 = 2n^2(n+1)$.
- Similarly, for φ', we can construct a corresponding TBD t' with $|t'| \leq 2n^2(n+1)$.
- Then $t'' = (b, t, t', \tau)$ is a TBD for ψ with $|t''| \leq 4n^2(n+1)+1$ different nodes (where τ is not counted, since it must have appeared in t or t') with b as the label of the top node of the TBD compatible with the order $v[b \to 1]$.

Lemma 21. *Let v be a given ordering of variables, and t be a positive TBD with variable ordering $v[b \to 1]$. Then we can construct a TBD t' with variable ordering v such that t and t' represent the same formula and $|t'| \leq |t|^4$.*

Proof: By moving b down 1 level in the reordering of the variables (applying the equivalences in Proposition 10 for reordering), the nodes of the form (b, x, y, z) at the current level is replaced, and the number of such nodes at each level is at most $(|t|-1)^3$ in the replacement process, since x, y, z are chosen among the $|t|-1$ nodes (all of the nodes of t except the top one). Therefore it may create at most $(|t|-1)^3 \cdot t$ new nodes for all of the levels of t. Then $|t'| \leq t + (|t|-1)^3 \cdot t \leq |t|^4$.

Proposition 14 For any ordering v of variables of ψ, we can construct a TBD t for ψ with $|t| \leq 256 \cdot (n+1)^{12}$.

Proof: This follows from Lemma 20 and Lemma 21.

References

1. K. Baukus, Y. Lakhnech, K. Stahl. Parameterized Verification of a Cache Coherence Protocol: Safety and Liveness. VMCAI 2002: 317-330. 2002.
2. A. Biere, A. Cimmatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. LNCS 1579:193-207. TACAS 99.
3. B. Bingham, J. Bingham, M. Greenstreet. Parameterized Verification of Deadlock Freedom in Symmetric Cache Coherence Protocols. FMCAD 2011.
4. R. E. Bryant. Graph based algorithms for boolean function manipulation. IEEE Transaction on Computers 35(8):677-691. 1986.
5. R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. ACM Comput. Surv. 24(3): 293-318. 1992.
6. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. LICS 1990: 428-439.
7. A. Cimatti, E. M. Clarke, F. Giunchiglia, M. Roveri. NuSMV: A New Symbolic Model Verifier. CAV 1999: 495-499.
8. E. M. Clarke, O. Grumberg and D. Peled. Model Checking. The MIT Press. 1999.
9. E. Allen Emerson and E. M. Clarke. Using Branching-time Temporal Logics to Synthesize Synchronization Skeletons. Science of Computer Programming 2(3):241-266. 1982.
10. G. Jennings, J. Isaksson, P. Lindgren. Ordered ternary decision diagrams and the multivalued compiled simulation of unmapped logic. 27th Annual Simulation Symposium:99 - 105. 1994.
11. C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. Bell Systems Technical Journal 38: 985-999. 1959.
12. Yi Lv, Huimin Lin, Hong Pan: Computing Invariants for Parameter Abstraction. MEMOCODE 2007: 29-38.
13. M. Ma: Model Checking for Protocols Using VERDS. TASE 2011:231-234.
14. K. L. McMillan. Symbolic Model Checking. Kluwer Academic Publisher,1993.
15. W. Penczek, B. Wozna, and A. Zbrzezny. Bounded Model Checking for the Universal Fragment of CTL. Fundamenta Informaticae 51:135-156. 2002.
16. A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. TACAS 2001:82-97.
17. W. Zhang. Verification of ACTL Properties by Bounded Model Checking. LNCS 4739 (EUROCAST 2007): 556-563. Springer-Verlag. 2007.
18. W. Zhang. Bounded Semantics of CTL and SAT-based Verification. LNCS 5885 (ICFEM 2009): 286-305. Springer-Verlag. 2009.

19. W. Zhang. Ternary Boolean Diagrams. Technical Report ISCAS-LCS-10-24, Institute of Software, Chinese Academy of Sciences. 2010. Available at http://lcs.ios.ac.cn/∼zwh/tr/2010tr24.pdf.
20. W. Zhang. Complexity Issues of Ternary Boolean Diagrams. Technical Report ISCAS-SKLCS-11-17, Institute of Software, Chinese Academy of Sciences. 2011. Available at http://lcs.ios.ac.cn/∼zwh/tr/2011tr17.pdf.