ELSEVIER

Artificial Intelligence 136 (2002) 189–213

Artificial Intelligence

www.elsevier.com/locate/artint

Ordered binary decision diagrams as knowledge-bases

Takashi Horiyama [a,*], Toshihide Ibaraki [b]

[a] *Graduate School of Information Science, Nara Institute of Science and Technology, Nara 630-0101, Japan*
[b] *Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan*

Received 28 April 2000; received in revised form 22 November 2001

Abstract

We consider the use of ordered binary decision diagrams (OBDDs) as a means of realizing knowledge-bases, and show that, from the view point of space requirement, the OBDD-based representation is more efficient and suitable in some cases, compared with the traditional CNF-based and/or model-based representations. We then present polynomial time algorithms for the two problems of testing whether a given OBDD represents a unate Boolean function, and of testing whether it represents a Horn function. © 2002 Published by Elsevier Science B.V.

Keywords: Knowledge representation; Automated reasoning; Ordered binary decision diagrams (OBDDs); Recognition problems; Unate functions; Horn functions

1. Introduction

Logical formulae are one of the traditional means of representing knowledge in artificial intelligence (AI) [24]. However, it is known that deduction from a knowledge-base that consists of a set of propositional clauses is co-NP-complete and abduction is Σ_2^p-complete [12]. Recently, an alternative way of representing knowledge, i.e., by a subset of its models, which are called characteristic models, has been proposed (see, e.g., [17,18,20, 21]). By restricting a knowledge-base to be Horn, deduction in this model-based approach can be performed in linear time, and abduction is also performed in polynomial time [17].

* Corresponding author.
E-mail addresses: horiyama@is.aist-nara.ac.jp (T. Horiyama), ibaraki@i.kyoto-u.ac.jp (T. Ibaraki).

0004-3702/02/$ – see front matter
PII: S0004-3702(02)00119-4

In addition to these favorable properties on the computational complexity, this approach has good evaluation in the practical sense [18,19].

In this paper, we propose yet another knowledge representation, i.e., the use of ordered binary decision diagrams (OBDDs) [1,4,25]. An OBDD is a directed acyclic graph representing a Boolean function, and can be considered as a variant of a decision tree. By restricting the order of variable appearances and by sharing isomorphic subgraphs, OBDDs have the following useful properties:

(1) When an ordering of variables is specified, an OBDD has the unique reduced canonical form for each Boolean function.
(2) Many Boolean functions appearing in practice can be compactly represented.
(3) When an OBDD is given, satisfiability and tautology of the represented function can be easily checked in constant time.
(4) There are efficient algorithms for many other Boolean operations on OBDDs.

As a result of these properties, OBDDs are widely used for various practical applications, especially in computer-aided design and verification of digital systems (see, e.g., [6,7,27]). One of the notable advantages of OBDDs is that, in the practical sense, minimization of DNFs (and also CNFs) can be done considerably faster than other approaches [8]. These observations encourage the use of OBDDs as knowledge-bases. The manipulation of knowledge-bases by OBDDs (e.g., deduction and abduction) was first discussed by Madre and Coudert [23].

We first compare the above three representations, i.e., formula-based, model-based, and OBDD-based, on the basis of their sizes. This will give a foundation for analyzing and comparing time and space complexities of various operations. Comparisons between these representations have been attempted in different communities. In AI community, it was shown that formula-based and model-based representations are incomparable with respect to space requirement [17]. Namely, each of them sometimes allows exponentially smaller sizes than the other, depending on the functions. In theoretical computer science and VLSI design communities, it was pointed out that formula-based and OBDD-based representations are also incomparable [14]. However, the three representations have never been compared on the same ground. We show in this paper that, in some cases, an OBDD-based representation requires exponentially smaller space than the other two, while there are also cases in which each of the other two requires exponentially smaller space than that of an OBDD. Thus, OBDDs can find their place in knowledge-bases. We also point out an unfortunate result that there exists a Horn function which requires an exponential size for any of the three representations.

OBDDs are known to be efficient for such knowledge-base operations as deduction and abduction [23]. Given two OBDDs as a knowledge-base and a deductive query, it can be decided in polynomial time whether the query is a consequence of the knowledge, where the knowledge can be a general Boolean function [23]. As for abduction, we have a polynomial time algorithm for Horn knowledge-bases by introducing some constraints on its assumption set, while it remains NP-complete for the general case [15]. By restricting a knowledge-base to be Horn, OBDDs can be translated into their CNFs and into their characteristic models, respectively, in polynomial time (more specifically in output

polynomial time), and vice versa [16]. Since a negative function is Horn, the technique can be applied to a unate OBDD after changing the polarities of some variables so that all variables become negative.

We investigate in this paper two fundamental recognition problems of OBDDs, that is, testing whether a given OBDD represents a unate Boolean function, and testing whether it represents a Horn function. We show that these recognition problems can be solved in polynomial time for both the unate and Horn cases. We often encounter these problems, since a knowledge-base representing some real-world phenomenon is sometimes required to be unate or Horn, from the hypothesis posed on the phenomenon and/or from the investigation of the mechanism causing the phenomenon. For example, if the knowledge-base represents a data set of test results with various physical measurements (e.g., body temperature, blood pressure, number of pulses and so on), it is often the case that the diagnosis of a certain disease is monotonically depending on each test result. The dependency may have the unate property (i.e., some of the tests may have negative polarity). Also in AI, it is common to assume Horn knowledge-bases as they can be processed efficiently in many respects (for example, deduction from a set of Horn clauses can be done in linear time [10]). These recognition problems also play a fundamental role in the area of learning and identifying meaningful structures in empirical data [9,13,29]. We emphasize here that OBDD-based approach is suitable for various tasks of structure identification discussed in [9]; e.g., finding effective representations [4,12,29], devising decompositions of database schema [22,30], synthesizing simple Boolean expressions [3, 11], and casting logical theories that render subsequent processing tractable [15,26].

The rest of this paper is organized as follows. The next section gives fundamental definitions and concepts. We compare the three representations in Section 3, and consider the problems of recognizing unate and Horn OBDDs in Sections 4 and 5, respectively.

2. Preliminaries

2.1. Notations and fundamental concepts

We consider a Boolean function $f:\{0,1\}^n \to \{0,1\}$. An *assignment* is a vector $a \in \{0,1\}^n$, whose ith coordinate is denoted by a_i. A *model of* f is a satisfying assignment a of f, i.e., $f(a)=1$, and the *theory* $\Sigma(f)$ *representing* f is the set of all models of f. Given $a, b \in \{0,1\}^n$, we denote by $a \leqslant b$ the usual bitwise (i.e., componentwise) ordering of assignments; $a_i \leqslant b_i$ for all $i=1,2,\ldots,n$, where $0<1$. Given a subset $E \subseteq \{1,2,\ldots,n\}$, χ^E denotes the characteristic vector of E; the ith coordinate ${\chi^E}_i$ equals 1 if $i \in E$ and 0 if $i \notin E$.

Let $x_1, x_2, \ldots, x_n$ be the n variables of f, where each x_i corresponds to the ith coordinate of assignments and evaluates to either 0 or 1. Negation of a variable x_i is denoted by $\bar{x}_i$. Variables and their negations are called literals. A *clause* is a disjunction of some literals, and a conjunction of clauses is called a *conjunctive normal form* (*CNF*). We say that f *is represented by a CNF* φ, if $f(a)=\varphi(a)$ holds for all $a \in \{0,1\}^n$. Any Boolean function can be represented by some CNF, which may not be unique.

We sometimes do not make a distinction among a function f, its theory $\Sigma(f)$, and a CNF φ that represents f, unless confusion arises. We define a *restriction* of f by replacing a variable x_i by a constant $a_i \in \{0,1\}$, and denote it by $f|_{x_i=a_i}$. Namely,

$$f|_{x_i=a_i}(x_1,\ldots,x_n) = f(x_1,\ldots,x_{i-1},a_i,x_{i+1},\ldots,x_n)$$

holds. Restriction may be applied to many variables. We also define $f \leqslant g$ (respectively, $f < g$) by $\Sigma(f) \subseteq \Sigma(g)$ (respectively, $\Sigma(f) \subset \Sigma(g)$).

Lemma 2.1. *Relation* $\leqslant$ *has the following properties*:

(1) $f \leqslant g$ *holds if and only if* $f|_{x_i=a_i} \leqslant g|_{x_i=a_i}$ *holds for both* $a_i = 0$ *and* 1.
(2) $f \vee g \leqslant h$ *holds if and only if* $f \leqslant h$ *and* $g \leqslant h$ *hold.*

For an assignment $p \in \{0,1\}^n$, we define $a \leqslant_p b$ if $(a \oplus_{bit} p) \leqslant (b \oplus_{bit} p)$ holds, where $\oplus_{bit}$ denotes the bitwise (i.e., componentwise) exclusive-or operation. A Boolean function f is *unate* with polarity p if $f(a) \leqslant f(b)$ holds for all assignments a and b such that $a \leqslant_p b$. A theory Σ is *unate* if Σ represents a unate function. A clause is *unate* with polarity p if $p_i = 0$ for all positive literals x_i and $p_i = 1$ for all negative literals $\bar{x}_i$ in the clause. A CNF is *unate* with polarity p if it contains only unate clauses with polarity p. It is known that a theory Σ is unate if and only if Σ can be represented by some unate CNF. A unate function is *positive* (respectively, *negative*) if its polarity is $(00\cdots0)$ (respectively, $(11\cdots1)$).

A theory Σ is *Horn* if Σ is closed under operation $\wedge_{bit}$, where $a \wedge_{bit} b$ is bitwise AND of two models $a, b \in \{0,1\}^n$. For example, if $a = (0011)$ and $b = (0101)$, then $a \wedge_{bit} b = (0001)$. The *closure* of a theory Σ with respect to $\wedge_{bit}$, denoted by $Cl_{\wedge_{bit}}(\Sigma)$, is defined as the smallest set that contains Σ and is closed under $\wedge_{bit}$. We also use the operation $\wedge_{bit}$ as a set operation; $\Sigma(f) \wedge_{bit} \Sigma(g) = \{a \mid a = b \wedge_{bit} c$ holds for some $b \in \Sigma(f)$ and $c \in \Sigma(g)\}$. We often denote $\Sigma(f) \wedge_{bit} \Sigma(g)$ by $f \wedge_{bit} g$, for convenience. Note that the two functions $f \wedge g$ and $f \wedge_{bit} g$ are different.

A Boolean function f is *Horn* if $\Sigma(f)$ is Horn; equivalently if $f \wedge_{bit} f = f$ holds (as sets of models). A clause is *Horn* if the number of positive literals in it is at most one, and a CNF is *Horn* if it contains only Horn clauses. It is known that a theory Σ is Horn if and only if Σ can be represented by some Horn CNF. By definition, a negative function is Horn, but not conversely.

For any Horn theory Σ, a model $a \in \Sigma$ is called *characteristic* if it cannot be produced by bitwise AND of other models in Σ; $a \notin Cl_{\wedge_{bit}}(\Sigma - \{a\})$. The set of all characteristic models of a Horn theory Σ, which we call the *characteristic set of* Σ, is denoted by $Char(\Sigma)$. Note that every Horn theory Σ has a unique characteristic set $Char(\Sigma)$, which satisfies $Cl_{\wedge_{bit}}(Char(\Sigma)) = \Sigma$. The set of *minimal models* of f with respect to $p \in \{0,1\}^n$ is defined as

$$min_p(f) = \{a \in \Sigma(f) \mid \text{ there exists no } b \in \Sigma(f) \text{ satisfying } b <_p a\},$$

where $b <_p a$ denotes that $b \leqslant_p a$ and $b \neq a$ hold. The following lemma gives an upper bound on the size (i.e., cardinality) of the characteristic set.

Lemma 2.2 [21]. *Let f be a Horn function on n variables. Then, the characteristic set of f has size at most $\sum_{p\in B_n} |min_p(f)|$, where $B_n = \{\chi^{E_{n,i}} \mid i = 0, 1, \ldots, n\}$ and $\chi^{E_{n,i}}$ is the characteristic vector of the set $E_{n,i} \subseteq \{0, 1, \ldots, n\}$ given by*

$$\begin{cases} E_{n,0} = \{1, 2, \ldots, n\} & \text{for } i = 0 \text{ (i.e., } \chi^{E_{n,0}} = (11\cdots1)), \\ E_{n,i} = E_{n,0} - \{i\} & \text{for } i = 1, 2, \ldots, n. \end{cases}$$

2.2. Ordered binary decision diagrams

An *ordered binary decision diagram* (OBDD) is a directed acyclic graph that represents a Boolean function. It has two sink nodes 0 and 1, called the 0-*node* and the 1-*node*, respectively (which are together called the *constant nodes*). Other nodes are called *variable nodes*, and each variable node v is labeled by one of the variables $x_1, x_2, \ldots, x_n$. Let $var(v)$ denote the label of node v. Each variable node has exactly two outgoing edges, called a 0-*edge* and a 1-*edge*, respectively. One of the variable nodes becomes the unique source node, which is called the *root node*. Let $X = \{x_1, x_2, \ldots, x_n\}$ denote the set of n variables. A *variable ordering* is a total ordering $(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$, associated with each OBDD, where π is a permutation $\{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$. The *level*[1] of a node v, denoted by $level(v)$, is defined by its label; if node v has label $x_{\pi(i)}$, $level(v)$ is defined to be $n - i + 1$. That is, the root node is in level n and has label $x_{\pi(1)}$, the nodes in level $n - 1$ have label $x_{\pi(2)}$ and so on. The level of the constant nodes is defined to be 0. On every path from the root node to a constant node in an OBDD, each variable appears at most once in the decreasing order of their levels.

Every node v of an OBDD also represents a Boolean function f_v, defined by the subgraph consisting of those edges and nodes reachable from v. If node v is a constant node, f_v equals to its label. If node v is a variable node, f_v is defined as $\overline{var(v)}\, f_{0\text{-}succ(v)} \vee var(v)\, f_{1\text{-}succ(v)}$ by Shannon's expansion, where 0-$succ(v)$ and 1-$succ(v)$, respectively, denote the nodes pointed by the 0-edge and the 1-edge of node v. The function f represented by an OBDD is the one represented by the root node. Fig. 1 illustrates three OBDDs representing $x_3x_2 \vee x_1$ with a variable ordering (x_3, x_2, x_1). Given an assignment a, the value of $f(a)$ is determined by following the corresponding path from the root node to a constant node in the following manner: at a variable node v, one of the outgoing edges is selected according to the assignment $a_{var}(v)$ to the variable $var(v)$. The value of the function is the label of the final constant node.

When two nodes u and v in an OBDD represent the same function, and their levels are the same, they are called *equivalent*. A node whose 0-edge and 1-edge both point to the same node is called *redundant*. An OBDD is called *dense* if every variable node v satisfy $level(0\text{-}succ(v)) = level(1\text{-}succ(v)) = level(v) - 1$ (i.e., all paths from the root node to constant nodes visit $n + 1$ nodes). A dense OBDD which has no equivalent nodes is *quasi-reduced*. An OBDD which has no mutually equivalent nodes and no redundant nodes is *reduced*. The OBDDs (a), (b) and (c) in Fig. 1 are dense, quasi-reduced and reduced, respectively. A reduced OBDD is obtained from a quasi-reduced OBDD by deleting redundant nodes v and changing their incoming edges $e = (u, v)$ to $(u, 0\text{-}succ(v))$.

[1] This definition of level may be different from its common use.

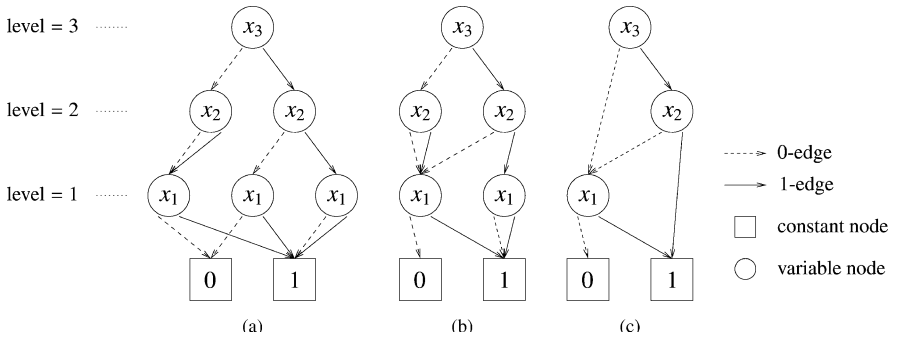

Fig. 1. OBDDs representing $x_3x_2 \vee x_1$.

In the following, we assume that all OBDDs are reduced, unless otherwise stated. The *size* of an OBDD is the number of nodes in the OBDD. Given a function f and a variable ordering, its reduced OBDD is unique and has the minimum size among all OBDDs with the same variable ordering. The minimum sizes of OBDDs representing a given Boolean function depends on the variable orderings [4].

Given an OBDD that represents f, the OBDDs of $f|_{x_i=0}$ and $f|_{x_i=1}$ can be obtained in $O(|f|)$ time, where $|f|$ denotes the size of the OBDD of f [2]. The size does not increase by a restriction. Given two OBDDs representing f and g, applying fundamental logic operators, e.g., $f \wedge g$, $f \vee g$, $f \oplus g$ and $f \to g$, can be performed in $O(|f| \cdot |g|)$ time, and property $f \leqslant g$ can be also checked in $O(|f| \cdot |g|)$ time [4].

A *partition* for f is a pair of sets (L, R) satisfying $L, R \subseteq X = \{x_1, x_2, \ldots, x_n\}$, $L \cup R = X$ and $L \cap R = \emptyset$. L is called a *left partition* and R is called a *right partition*. Let l denote an assignment to the variables in L, and r denote an assignment to the variables in R. Then, $l \cdot r$ denotes the complete assignment obtained by combining l and r. Let X' be a subset of X, and ω be a positive number satisfying $0 < \omega < 1$. Then, a partition (L, R) is called *ω-balanced for X'*, if it satisfies $\lfloor \omega|X'| \rfloor \leqslant |X' \cap L| \leqslant \lceil \omega|X'| \rceil$. Given a partition (L, R), a set A of assignments l^i for L and r^i for R, $i = 1, 2, \ldots, h$, is called a *fooling set* if it satisfies

(1) $f(l^i \cdot r^i) = a$ for all i,
(2) $f(l^i \cdot r^j) \neq a$ or $f(l^j \cdot r^i) \neq a$ for all $i \neq j$,

for some $a \in \{0, 1\}$. The next lemma tells that the size h of a fooling set gives a lower bound on the size of an OBDD that represents f.

Lemma 2.3 [5][2]. *Let f be a Boolean function on n variables, X' be some subset of the variables and ω be some positive number satisfying $0 < \omega < 1$. If f has a fooling set of size at least h for every ω-balanced partition (L, R) for X', then the size of OBDD representing f is at least h for any variable ordering.*

[2] Although the original lemma (Lemma 2 in [5]) states the case when h is at least c^n for some constant $c > 1$, its proof can be applied to any h in a straightforward manner.

3. Three approaches for knowledge-base representation

In this section, we compare three knowledge-base representations: CNF-based, model-based, and OBDD-based. It is known that CNF-based and model-based representations play orthogonal roles with respect to space requirement. Namely, each of them sometimes allows exponentially smaller sizes than the other, depending on the functions. We show that OBDD-based representation is incomparable to the other two in the same sense.

We start with relations between OBDD and CNF representations.

Lemma 3.1. *There exists a negative theory on n variables, for which OBDD and CNF both require size* $O(n)$*, while its characteristic set requires size* $\Omega(2^{n/2})$.

Proof. Consider a function

$$f_A = \bigwedge_{i=1}^{m} (\bar{x}_{2i-1} \vee \bar{x}_{2i}),$$

where $n = 2m$. The size of this CNF is obviously $O(n)$. The characteristic set is given by $\{a \in \{0,1\}^{2m} |$ exactly one of a_{2i-1} or a_{2i} is 0 for all $i = 1, 2, \ldots, m\}$, whose size is $\Omega(2^{n/2})$ [17]. The OBDD representing f_A is illustrated in Fig. 2, with a variable ordering $(x_n, x_{n-1}, \ldots, x_1)$. The size of this OBDD is $O(n)$. □

Lemma 3.2. *There exists a negative theory on n variables, for which OBDD requires size* $O(n)$ *and the characteristic set requires size* $O(n^2)$*, while CNF requires size* $\Omega(2^{n/2})$.

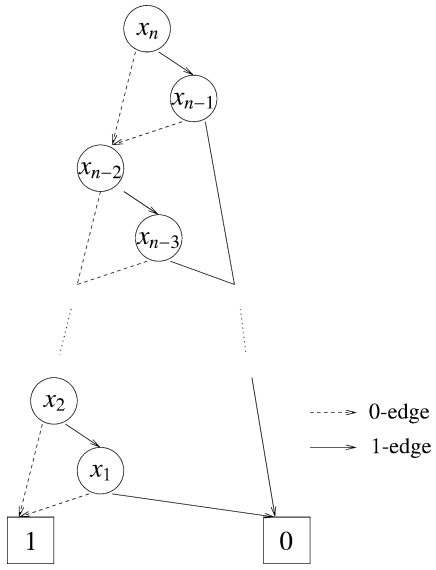

Fig. 2. OBDD representing $f_A = \bigwedge_{i=1}^{m} (\bar{x}_{2i-1} \vee \bar{x}_{2i})$.

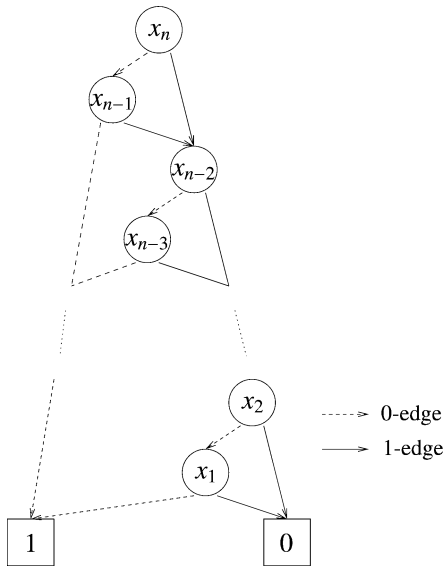

Fig. 3. OBDD representing $f_B = \bigvee_{i=1}^{m}(\bar{x}_{2i-1} \wedge \bar{x}_{2i})$.

Proof. Consider a function

$$f_B = \bigvee_{i=1}^{m}(\bar{x}_{2i-1} \wedge \bar{x}_{2i}) = \bigwedge_{(r_1,r_2,\ldots,r_m)\in S_B}(\bar{r}_1 \vee \bar{r}_2 \vee \cdots \vee \bar{r}_m),$$

where $n = 2m$ and $S_B = \{(r_1, r_2, \ldots, r_m) \mid r_i \in \{x_{2i-1}, x_{2i}\} \text{ for all } i = 1, 2, \ldots, m\}$. f_B is dual to f_A. The smallest CNF representation of f_B, which is given above, has $\Omega(2^{n/2})$ clauses. The characteristic set is $\{\chi^{\{1,2,\ldots,2m\}-S} \in \{0,1\}^{2m} \mid S = \{2i-1, 2i\} \text{ or } S = \{2i-1, 2i, j\} \text{ for } i \in \{1, 2, \ldots, m\} \text{ and } j(\neq 2i-1, 2i) \in \{1, 2, \ldots, 2m\}\}$, whose size is $O(n^2)$ [17]. The OBDD representing f_B is illustrated in Fig. 3, with a variable ordering $(x_n, x_{n-1}, \ldots, x_1)$. Note that, as f_B is dual to f_A, this OBDD is obtained by negating input variables (i.e., exchanging the roles of 0-edges and 1-edges) and negating output (i.e., exchanging the roles of the 0-node and the 1-node) of the OBDD in Fig. 2. The size of this OBDD is $O(n)$. □

By combining Lemmas 3.1 and 3.2, we show that, for some theory, OBDD can be exponentially smaller than its characteristic set and CNF representations.

Theorem 3.1. *There exists a negative theory on n variables, for which OBDD requires size $O(n)$, while both of the characteristic set and CNF require sizes $\Omega(2^{n/4})$.*

Proof. Consider a function

$$f_C = \left(\bigwedge_{i=1}^{m}(\bar{x}_{2i-1} \vee \bar{x}_{2i})\right) \wedge \left(\bigvee_{i=m+1}^{2m}(\bar{x}_{2i-1} \wedge \bar{x}_{2i})\right),$$

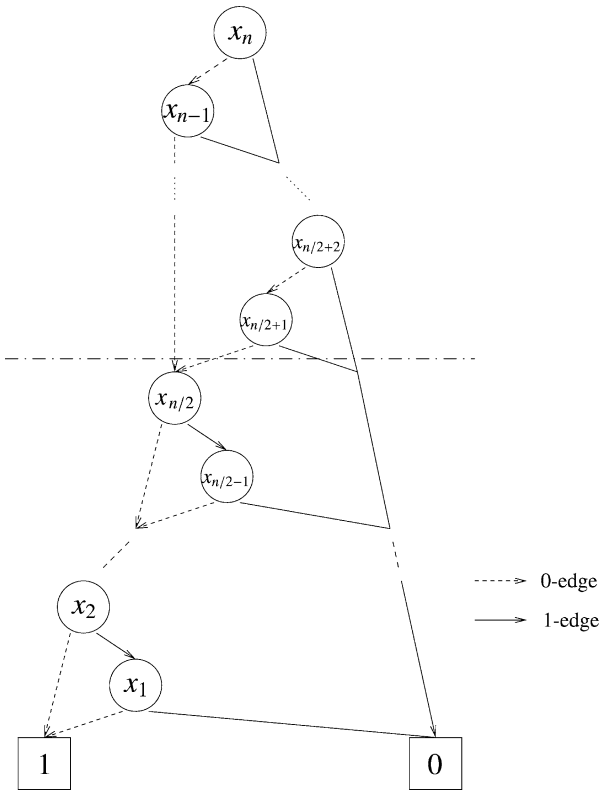

Fig. 4. OBDD representing $f_C = (\bigwedge_{i=1}^{m}(\bar{x}_{2i-1} \vee \bar{x}_{2i})) \wedge (\bigvee_{i=m+1}^{2m}(\bar{x}_{2i-1} \wedge \bar{x}_{2i}))$.

where $n = 4m$. As shown in Lemma 3.1, the characteristic set requires size $\Omega(2^{n/4})$ to represent the first half. Also by Lemma 3.2, CNF representation always requires size $\Omega(2^{n/4})$ to represent the second half. Note the first and second halves are independent since the variables in the first half do not appear in the second half and vice versa. Therefore, the above lower bounds of the characteristic set and CNF are valid also for f_C. An OBDD that represents f_C is illustrated in Fig. 4, with a variable ordering $(x_n, x_{n-1}, \ldots, x_1)$. The size of this OBDD is $O(n)$. □

We now turn to the opposite direction, i.e., CNF and the characteristic set can be exponentially smaller than the size of OBDD.

Lemma 3.3. *The size of the characteristic set is* $O(n)$ *for the following Horn function on* n *variables* $x_{i,j}$, $1 \leqslant i, j \leqslant m+1$, *where* $n = (m+1)^2$:

$$f_D = \left(\bigwedge_{i=1}^{m+1}\left(x_{i,m+1} \vee \bigvee_{j=1}^{m} \bar{x}_{i,j}\right)\right) \wedge \left(\bigwedge_{j=1}^{m+1}\left(x_{m+1,j} \vee \bigvee_{i=1}^{m} \bar{x}_{i,j}\right)\right).$$

Proof. Consider $\{\chi^{E_{n,0}}\} \cup \{\chi^{E_{n,i,j}} \mid 1 \leqslant i, j \leqslant m+1\}$ as the set B_n defined in Lemma 2.2, for convenience, where $E_{n,0} = \{(i,j) \mid 1 \leqslant i, j \leqslant m+1\}$ and $E_{n,i,j}$ is the set $E_{n,0} - \{(i,j)\}$ corresponding to variable $x_{i,j}$. $f_D(\chi^{E_{n,0}}) = 1$ holds for the characteristic vector $\chi^{E_{n,0}}$. Thus, $|min_{\chi^{E_{n,0}}}(f_D)| = 1$. Similarly, $|min_{\chi^{E_{n,i,j}}}(f_D)| = 1$ holds for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, m$, since $f_D(\chi^{E_{n,i,j}}) = 1$.

Next, since $f_D(\chi^{E_{n,i,m+1}}) = 0$ is implied by

$$\left(x_{i,m+1} \vee \bigvee_{j=1}^{m} \bar{x}_{i,j}\right)\left(\chi^{E_{n,i,m+1}}\right) = 0$$

for $i = 1, 2, \ldots, m$, we enumerate all minimal models for each $\chi^{E_{n,i,m+1}}$. By definition, we obtain $\chi^{E_{n,0}}$ by flipping the $(i, m+1)$th coordinate of $\chi^{E_{n,i,m+1}}$. This $\chi^{E_{n,0}}$ is a minimal model for $\chi^{E_{n,i,m+1}}$ since $f_D(\chi^{E_{n,0}}) = 1$. When the $(i, m+1)$th coordinate is fixed to 0, the clause $(x_{i,m+1} \vee \bigvee_{j=1}^{m} \bar{x}_{i,j})$ is satisfied by flipping at least one of the (i,j)th coordinates among $j = 1, 2, \ldots, m$. However, if two or more (i,j)th coordinates are flipped, the corresponding vector is not minimal. Thus, we have $|min_{\chi^{E_{n,i,m+1}}}(f_D)| = m+1$ for $i = 1, 2, \ldots, m$. Similarly, we have $|min_{\chi^{E_{n,m+1,j}}}(f_D)| = m+1$ for $j = 1, 2, \ldots, m$.

We also enumerate all minimal models for $\chi^{E_{n,m+1,m+1}}$ since $f_D(\chi^{E_{n,m+1,m+1}}) = 0$. We obtain $\chi^{E_{n,0}}$ by flipping the $(m+1, m+1)$th coordinate. When the $(m+1, m+1)$th coordinate is fixed to 0, minimal models are obtained by flipping exactly one of the $(i, m+1)$th coordinates among $i = 1, 2, \ldots, m$ and exactly one of the $(m+1, j)$th coordinates among $j = 1, 2, \ldots, m$. Thus, we have $|min_{\chi^{E_{n,m+1,m+1}}}(f_D)| = m^2 + 1$. In total, we have $\sum_{a \in B_n} |min_a(f_D)| = \mathrm{O}(m^2)$, i.e., $\mathrm{O}(n)$. By Lemma 2.2, this means that the size of the characteristic set of f_D is $\mathrm{O}(n)$. □

Lemma 3.4 [28]. *Let f be a Boolean function on n variables $x_{i,j}$, $1 \leqslant i, j \leqslant m$, where $n = m^2$. Then, for any partition (L, R) satisfying $|L| = |R| = n/2$, either of the following properties holds*:

(1) *There are at least $m/\sqrt{2}$ different i's satisfying $\{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\} \cap L \neq \emptyset$ and $\{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\} \cap R \neq \emptyset$.*
(2) *There are at least $m/\sqrt{2}$ different j's satisfying $\{x_{1,j}, x_{2,j}, \ldots, x_{m,j}\} \cap L \neq \emptyset$ and $\{x_{1,j}, x_{2,j}, \ldots, x_{m,j}\} \cap R \neq \emptyset$.*

Lemma 3.5. *The size of OBDD representing the following negative function f_E on n variables $x_{i,j}$, $1 \leqslant i, j \leqslant m$, where $n = m^2$, is $\Omega(2^{m/\sqrt{2}})$ for any variable ordering*:

$$f_E = \left(\bigwedge_{i=1}^{m}\left(\bigvee_{j=1}^{m} \bar{x}_{i,j}\right)\right) \wedge \left(\bigwedge_{j=1}^{m}\left(\bigvee_{i=1}^{m} \bar{x}_{i,j}\right)\right).$$

Proof. We prove this by Lemma 2.3 in Section 2.2. Let us consider that the set X' in Lemma 2.3 is given by the set of all variables, and $\omega = 1/2$. Then, for every balanced partition (L, R), assuming case (1) of Lemma 3.4 without loss of generality, we have at least $m/\sqrt{2}$ different i's satisfying $\{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\} \cap L \neq \emptyset$ and $\{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\} \cap$

$R \neq \emptyset$. We select $m/\sqrt{2}$ of these i's, $I = \{i_1, i_2, \ldots, i_{m/\sqrt{2}}\}$. For every $i_k \in I$, we can select two variables $x_{i_k,l_k} \in L$ and $x_{i_k,r_k} \in R$. We construct a set A of assignments such that each assignment satisfies the following restrictions:

(1) For every $i_k \in I$, $(x_{i_k,l_k}, x_{i_k,r_k})$ is assigned either $(0, 1)$ or $(1, 0)$.
(2) For every $i_k \in I$, all variables in $\{x_{i_k,1}, x_{i_k,2}, \ldots, x_{i_k,m}\} - \{x_{i_k,l_k}, x_{i_k,r_k}\}$ are assigned 1.
(3) Other variables are assigned 0.

The size of the set A is $2^{m/\sqrt{2}}$ since there are choices in restriction (1). Let $l^h \cdot r^h$ denote the assignment satisfying $h = \sum_{k=1}^{m/\sqrt{2}} a_k \cdot 2^{k-1}$, where $x_{i_k,l_k} = a_k \in \{0, 1\}$ (and $x_{i_k,r_k} = \bar{a}_k$) for each $i_k \in I$. h satisfies $0 \leqslant h < 2^{m/\sqrt{2}}$.

Now, we prove that set A is a fooling set, defined just before Lemma 2.3. First, we show $f_E(l^h \cdot r^h) = 1$ for all h. Since one of x_{i_k,l_k} and x_{i_k,r_k} is assigned 0, $\bigvee_{j=1}^{m} \bar{x}_{i_k,j} = 1$ holds for all $i_k \in I$. For $i_k \notin I$, since $x_{i,1}, x_{i,2}, \ldots, x_{i,m}$ are assigned 0, we have $\bigvee_{j=1}^{m} \bar{x}_{i,j} = 1$. Also $\bigvee_{i=1}^{m} \bar{x}_{i,j} = 1$ holds for all $j \in \{1, 2, \ldots, m\}$. Thus, we have $f_E(l^h \cdot r^h) = 1$ for all h.

Next, we show that $f_E(l^h \cdot r^{h'}) = 0$ holds for $h > h'$. Since $h > h'$, there exists at least one variable x_{i_k,l_k} which is assigned 1 by $l^h \cdot r^h$ and 0 by $l^{h'} \cdot r^{h'}$. By restriction (1), x_{i_k,r_k} is then assigned 0 by $l^h \cdot r^h$ and 1 by $l^{h'} \cdot r^{h'}$. Therefore, x_{i_k,l_k} and x_{i_k,r_k} are assigned 1 by assignment $l^h \cdot r^{h'}$, implying that $\bigvee_{j=1}^{m} \bar{x}_{i_k,j} = 0$ holds. This proves that A is a fooling set. Since the size of this fooling set is at least $2^{m/\sqrt{2}}$ for any balanced partition, this lemma follows from Lemma 2.3. □

Theorem 3.2. *There exists a Horn theory on n variables, for which both of the CNF and the characteristic set require sizes $O(n)$, while the size of the smallest OBDD representation is $\Omega(2^{\sqrt{n}/\sqrt{2}})$.*

Proof. Consider the function f_D in Lemma 3.3. As stated in Lemma 3.3, the size of its characteristic set is $O(n)$. Also the size of the CNF is obviously $O(n)$. The function f_E in Lemma 3.5 is obtained by restricting $x_{1,m+1}, \ldots, x_{m,m+1}$, $x_{m+1,1}, \ldots, x_{m+1,m}$ and $x_{m+1,m+1}$ of f_D to 0. Since the size of OBDD does not increase by a restriction, the size of the smallest OBDD of f_D is $\Omega(2^{\sqrt{n}/\sqrt{2}})$. □

The above results show that none of the three representations always dominate the other two. Therefore, OBDDs can find a place in knowledge-bases as they can represent some theories more efficiently than others.

Unfortunately, by combining Theorems 3.1 and 3.2, we can construct the following function, which is exponential for all representations.

Corollary 3.1. *There exists a Horn function on n variables, for which both of the characteristic set and CNF require sizes $\Omega(2^{n/8})$ and the size of the smallest OBDD representation is $\Omega(2^{\sqrt{n}/2})$.*

Proof. Consider the conjunction $f_C \wedge f_D$, where f_C (respectively, f_D) was defined in the proof of Theorem 3.1 (respectively, Theorem 3.2). Note that f_C and f_D both have $n/2$

variables, but share none of the variables. Then, similarly to the case of Theorem 3.1, the stated lower bounds for the three representations are easily obtained. □

4. Checking unateness of OBDDs

In this section, we discuss the problem of checking whether a given OBDD represents a unate function. We assume, without loss of generality, that the variable ordering is always $(x_n, x_{n-1}, \ldots, x_1)$. The following well-known property indicates that this problem can be solved in polynomial time.

Property 4.1. *Let f be a Boolean function on n variables $x_1, x_2, \ldots, x_n$. Then, f is unate with polarity $p = (p_1, p_2, \ldots, p_n)$ if and only if $f|_{x_i=0} \leqslant f|_{x_i=1}$ for $p_i = 0$ (respectively, $f|_{x_i=0} \geqslant f|_{x_i=1}$ for $p_i = 1$) holds for every $i = 1, 2, \ldots, n$.*

As noted in Section 2.2, an OBDD representing $f|_{x_i=0}$ (respectively, $f|_{x_i=1}$) can be obtained in $O(|f|)$ time from the OBDD representing f, where $|f|$ denotes its size. The size does not increase by a restriction $f|_{x_i=0}$ or $f|_{x_i=1}$. Since the property $g \leqslant h$ can be checked in $O(|g| \cdot |h|)$ time, the unateness of f can be checked in $O(n|f|^2)$ time by checking the conditions $f|_{x_i=0} \leqslant f|_{x_i=1}$ and $f|_{x_i=0} \geqslant f|_{x_i=1}$ for all $i = 1, 2, \ldots, n$.

The following well-known property is useful to reduce the computation time.

Property 4.2. *Let f be a Boolean function on n variables $x_1, x_2, \ldots, x_n$. Then, f is unate with polarity $p = (p_1, p_2, \ldots, p_n)$ if and only if* (i) *both $f|_{x_n=0}$ and $f|_{x_n=1}$ are unate with same polarity $(p_1, p_2, \ldots, p_{n-1})$, and* (ii) *$p_n = 0$ implies $f|_{x_n=0} \leqslant f|_{x_n=1}$ and $p_n = 1$ implies $f|_{x_n=0} \geqslant f|_{x_n=1}$.*

The unateness of functions $f|_{x_n=0}$ and $f|_{x_n=1}$ can be checked by applying Property 4.2 recursively, but we also have to check an additional condition that $f|_{x_n=0}$ and $f|_{x_n=1}$ have the same polarity. Our algorithm is similar to the implementation of OBDD-manipulation-systems by Bryant [4], in the sense that we cache all intermediate computational results to avoid duplicate computation. In Bryant's idea, different computational results may be stored to the same memory in order to handle different operations, and hence the same computation may be repeated more than once. However, as our algorithm only aims at checking the unateness, it can avoid such cache conflict by explicitly preparing memory for each result. This is a key to reduce the computation time.

We check the unateness of f in the bottom-up manner by checking unateness of all nodes corresponding to intermediate results. Note that the property $f|_{x_n=0} \leqslant f|_{x_n=1}$ (respectively, $f|_{x_n=0} \geqslant f|_{x_n=1}$) can be also checked in the bottom-up manner, since $g \leqslant h$ holds if and only if $g|_{x_i=0} \leqslant h|_{x_i=0}$ and $g|_{x_i=1} \leqslant h|_{x_i=1}$ hold for some i.

Algorithm CHECK-UNATE in Fig. 5 checks the unateness and the polarity of a given OBDD in the manner as described above. We use an array $p[\ell]$ to denote the polarity of f with respect to x_ℓ in level ℓ; each element stores 0, 1 or $*$ (not checked yet). We also use a two-dimensional array $imp[u, v]$ to denote whether $f_u \leqslant f_v$ holds or not; each element stores YES, NO or $*$ (not checked yet). In Step 2, the unateness with the polarity

Algorithm CHECK-UNATE

Input: An OBDD representing f with a variable ordering $(x_n, x_{n-1} \ldots, x_1)$.
Output: "yes" and its polarity if f is unate; otherwise, "no".
Step 1 (initialize). Set $p[i] := *$ for all $i = 1, 2, \ldots, n$;

$$imp[u, v] := \begin{cases} \text{NO} & \text{if } (f_u, f_v) = (1, 0); \\ \text{YES} & \text{if } (f_u, f_v) = (0, 0), (0, 1), (1, 1); \\ * & \text{otherwise;} \end{cases}$$

$\ell := 1$.

Step 2 (check unateness in level ℓ and compute $p[\ell]$). For each node v in level ℓ (i.e., labeled with x_ℓ), apply Steps 2-1 and 2-2.

Step 2-1. Set $pol := 0$ if $imp[0\text{-}succ(v), 1\text{-}succ(v)] = \text{YES}$ holds; set $pol := 1$ if $imp[1\text{-}succ(v), 0\text{-}succ(v)] = \text{YES}$ holds; otherwise, output "no" and halt.

Step 2-2. If $p[\ell] = *$, then set $p[\ell] := pol$. If $p[\ell] \neq *$ and $p[\ell] \neq pol$ hold, then output "no" and halt.

Step 3 (compute *imp* in level ℓ). For each pair of nodes (u, v) (where (u, v) is an ordered pair) such that $level(u) \leqslant \ell$ and $level(v) \leqslant \ell$, and at least one of $level(u)$ and $level(v)$ is equal to ℓ, set $imp[u, v] := \text{YES}$ if both $imp[0\text{-}suss'(u), 0\text{-}succ'(v)]$ and $imp[1\text{-}suss'(u), 1\text{-}succ'(v)]$ are YES; otherwise, set $imp[u, v] := \text{NO}$.

Step 4 (iterate). If $\ell = n$, where n is the level of the root node, then output "yes" and polarity $p = (p[1], p[2], \ldots, p[n])$, and halt. Otherwise set $\ell := \ell + 1$ and return to Step 2.

Fig. 5. Algorithm CHECK-UNATE to check the unateness of an OBDD.

specified by $p[\ell]$ is checked for the nodes in level ℓ. More precisely, the unateness for them is checked in Step 2-1, and the consistency of their polarities is checked in Step 2-2. In Step 3, $imp[u, v]$ is computed for the functions f_u and f_v in levels up to ℓ.

The unateness check of f_v in Step 2-1 can be easily done, since both $f_{0\text{-}succ(v)}$ (i.e., $f_v|_{x_\ell=0}$) and $f_{1\text{-}succ(v)}$ (i.e., $f_v|_{x_\ell=1}$) have already been checked to be unate with the same polarity, and both $imp[0\text{-}succ(v), 1\text{-}succ(v)]$ and $imp[1\text{-}succ(v), 0\text{-}succ(v)]$ have been computed in the previous iteration. Note that constant functions 0 and 1 are considered to be unate. The polarity of f_v with respect to x_ℓ in level ℓ is temporarily stored in *pol* in this step.

In Step 2-2, the polarity consistency of x_ℓ is checked by comparing the polarity of node v (which is *pol*) and $p[\ell]$. If $p[\ell] = *$ (i.e., v is the first node checked in level ℓ), we store *pol* in $p[\ell]$. Otherwise, *pol* is checked against $p[\ell]$ and "no" is output if they are not consistent. Note that CHECK-UNATE outputs $p[\ell] = *$ if there are no nodes in level ℓ (i.e., f does not depend on x_ℓ).

In Step 3, comparison between f_u and f_v is also performed easily, since the comparisons between $f_u|_{x_\ell=a_\ell}$ and $f_v|_{x_\ell=a_\ell}$ for both $a_\ell = 0$ and 1 have already been completed. Here we use the convention that $0\text{-}succ'(v)$ (respectively, $1\text{-}succ'(v)$) denotes $0\text{-}succ(v)$ (respectively, $1\text{-}succ(v)$) if $level(v) = \ell$, but denotes v itself if $level(v) < \ell$. This is because $f_v|_{x_\ell=0} = f_{0\text{-}succ(v)}$ and $f_v|_{x_\ell=1} = f_{1\text{-}succ(v)}$ hold if $level(v) = \ell$, and $f_v|_{x_\ell=0} = f_v|_{x_\ell=1} = f_v$ holds if $level(v) < \ell$. Note that $f_u = f_v$ holds if and only if u and v are the same node. After Step 3 is done for some ℓ, we know $imp[u, v]$ for all pairs of nodes u and v such that

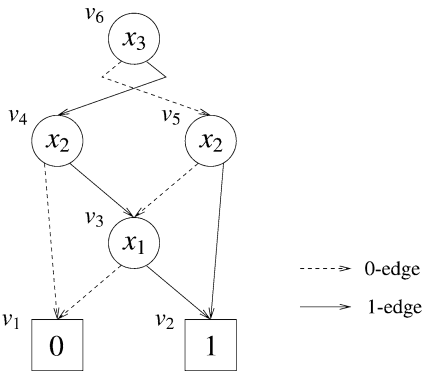

Fig. 6. OBDD representing $f_F = x_1x_2 \vee x_2\bar{x}_3 \vee \bar{x}_3x_1$.

level$(u) \leqslant \ell$ and *level*$(v) \leqslant \ell$. We store all the results, although some of them may not be needed.

Example 4.1. Consider the OBDD of $f_F = x_1x_2 \vee x_2\bar{x}_3 \vee \bar{x}_3x_1$ with variable ordering (x_3, x_2, x_1). As shown in Fig. 6, the OBDD has 6 nodes $v_1, v_2, \ldots, v_6$, which respectively represent the following functions:

$$
\begin{aligned}
f_{v_1} &= 0,\\
f_{v_2} &= 1,\\
f_{v_3} &= x_1,\\
f_{v_4} &= x_1x_2,\\
f_{v_5} &= x_1 \vee x_2,\\
f_{v_6} &= x_1x_2 \vee x_2\bar{x}_3 \vee \bar{x}_3x_1.
\end{aligned}
$$

Algorithm CHECK-UNATE starts from the initializing step. Here we have

$$
\begin{aligned}
&p[i] := * \quad \text{for all } i = 1, 2, \ldots, n\\
&\mathit{imp}[v_1, v_1] = \mathit{imp}[v_1, v_2] = \mathit{imp}[v_2, v_2] = \text{YES},\\
&\mathit{imp}[v_2, v_1] = \text{NO} \quad \text{and} \quad \ell = 1.
\end{aligned}
$$

In the rest of this example, we pay attention to *imp*[ ]'s which are evaluated to be YES. In Step 2 of the first iteration, we have $p[1] = 0$ because v_3 is the unique node in level $\ell = 1$ and *imp*[0-*succ*(v_3), 1-*succ*(v_3)] = *imp*$[v_1, v_2] =$ YES holds. In Step 3 of the first iteration, we have

$$\mathit{imp}[v_1, v_3] = \mathit{imp}[v_3, v_3] = \mathit{imp}[v_3, v_2] = \text{YES},$$

which can be confirmed by the implication from $0 \leqslant x_1 \leqslant x_1 \leqslant 1$. In the computation of *imp*$[v_1, v_3]$, 0-*succ*$'(v_1)$ and 1-*succ*$'(v_1)$ are interpreted as node v_1 itself, while 0-*succ*$'(v_3)$ and 1-*succ*$'(v_3)$ are interpreted as 0-*succ*(v_3) (i.e., v_1) and 1-*succ*(v_3) (i.e., v_2), respectively. In the second iteration (level $\ell = 2$), we have $p[2] = 0$ because

$imp[0\text{-}succ(v_4), 1\text{-}succ(v_4)] = imp[v_1, v_3] = \text{YES}$ and $imp[0\text{-}succ(v_5), 1\text{-}succ(v_5)] = imp[v_3, v_2] = \text{YES}$. We also have

$$\begin{aligned} imp[v_1, v_4] &= imp[v_1, v_5] = imp[v_3, v_5] = imp[v_4, v_2] = imp[v_4, v_3] \\ &= imp[v_4, v_4] = imp[v_4, v_5] = imp[v_5, v_2] = imp[v_5, v_5] = \text{YES}. \end{aligned}$$

In the third iteration (level $\ell = 3$), we have $p[3] = 1$ because $imp[1\text{-}succ(v_6), 0\text{-}succ(v_6)] = imp[v_4, v_5] = \text{YES}$. Then, Algorithm CHECK-UNATE outputs the answer "yes" and the polarity $p = (0, 0, 1)$ of f_F.

Now, we consider the computation time of this algorithm. In Step 2, the computation for each node v is performed in constant time from the data already computed in the previous Step 3. Thus, the total time of Step 2 is $O(|f|)$. In Step 3, the comparison between f_u and f_v for each pair (u, v) is also performed in constant time. The number of pairs compared in Step 3 during the entire computation is $O(\binom{|f|}{2}) = O(|f|^2)$, which requires $O(|f|^2)$ time. The time for the rest of the computation is minor.

Theorem 4.1. *Given an OBDD representing a Boolean function f, checking whether f is unate can be done in $O(|f|^2)$ time, where $|f|$ is the size of the given OBDD.*

If we start Algorithm CHECK-UNATE with initial condition $p[i] := 0$ (respectively, $p[i] := 1$) for all $i = 1, 2, \ldots, n$, we can check the positivity (respectively negativity) of f. This is because f is positive (respectively, negative) if and only if the polarities of all nodes are 0 (respectively, 1).

Corollary 4.1. *Given an OBDD representing a Boolean function f, checking whether f is positive (respectively, negative) can be done in $O(|f|^2)$ time, where $|f|$ is the size of the given OBDD.*

As stated above, it may not be necessary to compute $imp[u, v]$'s for all pairs (u, v) of nodes. The following theorem however gives an unfortunate instance which requires the computation of $imp[u, v]$'s for $\Theta(|f|^2)$ pairs.

Theorem 4.2. *There exists an OBDD of a positive function f which requires to check $\Theta(|f|^2)$ $imp[u, v]$'s, where $|f|$ is the size of the given OBDD.*

Proof. Consider the function f_G with $n = 3m + 1$ variables as defined below:

$$\begin{aligned} f_G &= \bar{x}_{3m+1} f_m \vee x_{3m+1} g_m, \\ f_i &= \begin{cases} (x_{2m+i} \vee f_{i-1}) x_{m-i+1} & (i = 1, 2, \ldots, m) \\ 0 & (i = 0), \end{cases} \\ g_i &= \begin{cases} x_{m+i} g_{i-1} \vee x_{m-i+1} & (i = 1, 2, \ldots, m) \\ 0 & (i = 0). \end{cases} \end{aligned}$$

The OBDDs of f_i and g_i are illustrated in Figs. 7(a) and (b), respectively. As shown in Fig. 8, the OBDD of f_G has $4m + 2$ nodes.

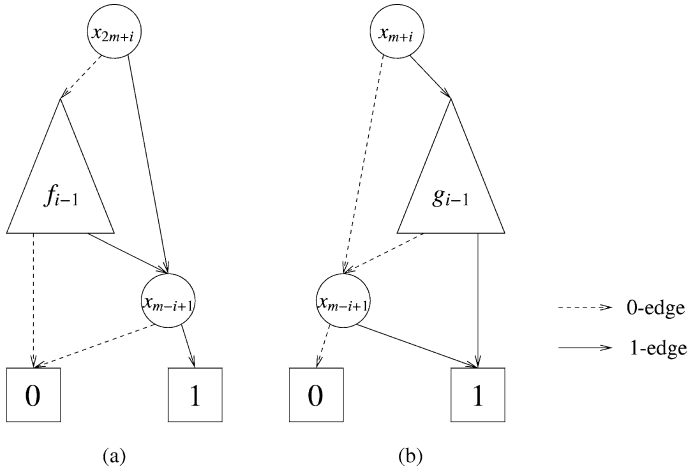

Fig. 7. OBDDs representing (a) $f_i = (x_{2m+i} \vee f_{i-1})x_{m-i+1}$ and (b) $g_i = x_{m+i}\, g_{i-1} \vee x_{m-i+1}$.

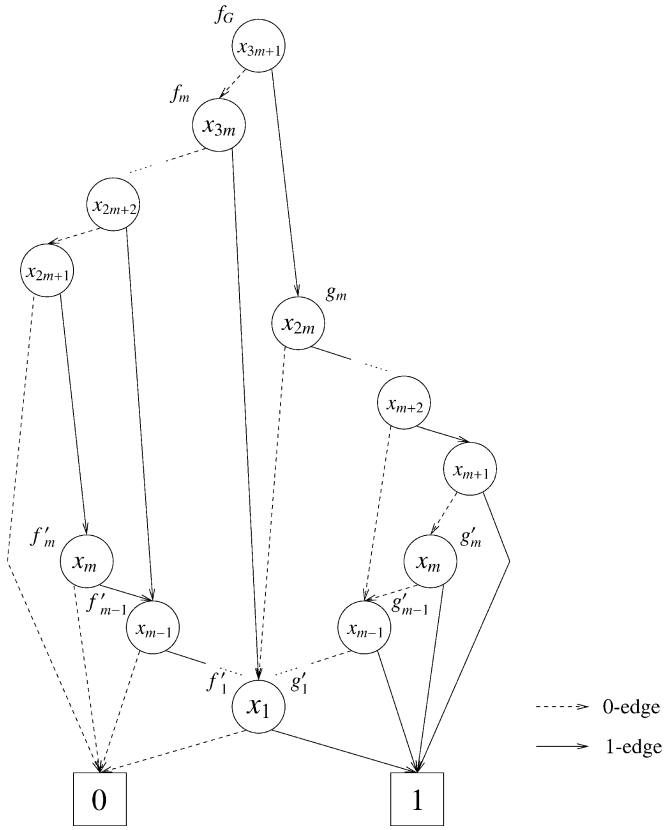

Fig. 8. OBDD representing $f_G = \bar{x}_{3m+1} f_m \vee x_{3m+1} g_m$.

The upper bound $O(|f|^2)$ is obvious from Theorem 4.1. Now, in order to prove the lower bound, we show that f_G requires to check at least m^2 *imp*[ ]'s. Since f_m and g_m are positive functions, Algorithm CHECK-UNATE executes iterations of Steps 2 to 4 from the initial level $\ell = 1$ to level $\ell = 3m$. In level $\ell = 3m+1$, in order to obtain the result $p[3m+1] =$ YES in Step 2, it is necessary to have the property $f_m \leqslant g_m$. By applying Lemma 2.1 recursively, property $f_m \leqslant g_m$ is confirmed by checking the properties $f'_i \leqslant g_m$ for all $i = 1, 2, \ldots, m$, where $f'_i = \bigwedge_{k=1}^{i} x_k$. Similarly, property $f'_i \leqslant g_m$ is confirmed by checking the properties $f'_i \leqslant g'_j$ for all $j = 1, 2, \ldots, m$, where $g'_j = \bigvee_{k=1}^{j} x_k$. Therefore, we need to check $f'_i \leqslant g'_j$ for all $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, m$; i.e., m^2 *imp*[ ]'s. □

5. Checking the Horn property of OBDDs

In this section, we discuss the problem of checking whether a given OBDD represents a Horn function. After examining the condition for the Horn property in the next subsection, an algorithm will be given in Section 5.2.

5.1. Conditions for the Horn property

We assume, without loss of generality, that the variable ordering is always $(x_n, x_{n-1}, \ldots, x_1)$. Denoting $f|_{x_n=0}$ and $f|_{x_n=1}$ by f_0 and f_1 for simplicity, f is given by

$$f = \bar{x}_n f_0 \vee x_n f_1,$$

where f_0 and f_1 are Boolean functions on $n-1$ variables $x_1, x_2, \ldots, x_{n-1}$. By definition, we can determine whether f is Horn by checking the condition $f \wedge_{bit} f = f$. For this, we may first construct an OBDD of $f \wedge_{bit} f$, and then check the equivalence between $f \wedge_{bit} f$ and f. However, the following theorem says that this check may require exponential time and hence is intractable in general.

Theorem 5.1. *There exists a Boolean function f on n variables, for which OBDD requires size $O(n^2)$, while the OBDD representing $f \wedge_{bit} f$ requires $\Omega(2^{n/4})$ for the same variable ordering.*

Proof. Consider a function

$$f_H = \bigvee_{i=1}^{2m} f_i,$$

$$f_i = g_i \wedge x_{i+2m} \wedge \left(\bigwedge_{j \in \{1,2,\ldots,2m\}-\{i\}} \bar{x}_{j+2m} \right) \quad (i = 1, 2, \ldots, 2m),$$

$$g_i = \begin{cases} \bar{x}_{i-m} \wedge \bar{x}_i & (i = m+1, m+2, \ldots, 2m) \\ \bar{x}_i \wedge \bar{x}_{i+m} & (i = 1, 2, \ldots, m), \end{cases}$$

where $n = 4m$. We first prove the upper bound on the size of the OBDD of f_H. We assume that the variable ordering is $(x_{4m}, x_{4m-1}, \ldots, x_1)$. f_H can be rewritten as

$$f_H = h_{2m},$$
$$h_i = \begin{cases} x_{i+2m}\left(g_i \wedge \left(\bigwedge_{j\in\{1,2,\ldots,i-1\}} \bar{x}_{j+2m}\right)\right) \vee \bar{x}_{i+2m} h_{i-1} & (i = 1, 2, \ldots, 2m) \\ 0 & (i = 0) \end{cases}$$

by Shannon's expansion. The OBDD of h_i $(i = 1, 2, \ldots, 2m)$ has the root node labeled by x_{i+2m}, and the 1-edge and the 0-edge pointing to the OBDD of $h_i|_{x_{i+2m}=1}$ and that of $h_i|_{x_{i+2m}=0}$, respectively. Thus, the size of the OBDD of h_i has the following upper bound:

$$\begin{aligned} |h_i| &\leqslant |h_i|_{x_{i+2m}=1}| + |h_i|_{x_{i+2m}=0}| + 1 \\ &= \left| g_i \wedge \left(\bigwedge_{j\in\{1,2,\ldots,i-1\}} \bar{x}_{j+2m} \right) \right| + |h_{i-1}| + 1, \end{aligned}$$

where $|h_i|$ denotes the size of the OBDD of h_i. Since $g_i \wedge (\bigwedge_{j\in\{1,2,\ldots,i-1\}} \bar{x}_{j+2m})$ is an AND of less than n negative literals, the size of its OBDD is $O(n)$. By definition, $h_0 = 0$ means that the size of the OBDD of h_0 is 1. By induction, we have

$$|f_H| = |h_{2m}| \leqslant O(n) + |h_{2m-1}| + 1 \leqslant O(2mn) + |h_0| + 2m.$$

Namely, the upper bound is $O(n^2)$.

Now, we consider the second part of the theorem, i.e. the lower bound on $f_H \wedge_{bit} f_H$. First, we show the identity $f_H \wedge_{bit} f_H = f_H \vee g$ by considering their models, where $g = (\bigvee_{i=1}^{m} g_i) \wedge (\bigwedge_{j=1}^{2m} \bar{x}_{j+2m})$. Let b and c be models of f_H. By definition, b (respectively, c) is a model of f_k (respectively, f_ℓ) for some k (respectively, ℓ) $(k, \ell \in \{1, 2, \ldots, 2m\})$. Then, we have two cases: (1) $k = \ell$, and (2) $k \neq \ell$. In case (1), since f_k is Horn, $a = b \wedge_{bit} c$ is also a model of f_k. In case (2), model b satisfies

$$b_k = b_{k'} = 0, \quad b_{k+2m} = 1, \quad \text{and} \quad b_{j+2m} = 0 \quad \text{for all } j \in \{1, 2, \ldots, 2m\} - \{k\},$$

where k' denotes $k + m$ if $k \in \{1, 2, \ldots, m\}$, but denotes $k - m$ if $k \in \{m + 1, m + 2, \ldots, 2m\}$. Similarly, model c satisfies

$$c_\ell = c_{\ell'} = 0, \quad c_{\ell+2m} = 1, \quad \text{and} \quad c_{j+2m} = 0 \quad \text{for all } j \in \{1, 2, \ldots, 2m\} - \{\ell\}.$$

Thus, $k \neq \ell$ implies that $a = b \wedge_{bit} c$ satisfies the following restrictions:

$$a_k = a_{k'} = a_\ell = a_{\ell'} = 0, \quad \text{and} \quad a_{j+2m} = 0 \quad \text{for all } j \in \{1, 2, \ldots, 2m\}.$$

This means that a is a model of g. By considering both cases, we have

$$\Sigma(f_H \wedge_{bit} f_H) \subseteq \Sigma(f_H) \vee \Sigma(g) = \Sigma(f_H \vee g). \tag{1}$$

On the other hand, let a be a model of g. Then, a is of form $a = b \wedge_{bit} c$ where b and c satisfy the following restrictions:

$$b_{k+2m} = 1, \quad b_j = a_j \quad \text{for all } j \in \{1, 2, \ldots, 4m\} - \{k + 2m\},$$
$$c_{k+3m} = 1, \quad c_j = a_j \quad \text{for all } j \in \{1, 2, \ldots, 4m\} - \{k + 3m\}.$$

By definition, a satisfies $a_k = a_{k+m} = 0$ for some $k \in \{1, 2, \ldots, m\}$ and $a_{j+2m} = 0$ for all $j \in \{1, 2, \ldots, 2m\}$. Since both b and c are models of f_H, we have $\Sigma(g) \subseteq \Sigma(f_H \wedge_{bit} f_H)$. Also, the definition of bitwise AND operation implies $\Sigma(f_H) \subseteq \Sigma(f_H \wedge_{bit} f_H)$. Thus, we have

$$\Sigma(f_H \vee g) \subseteq \Sigma(f_H \wedge_{bit} f_H). \tag{2}$$

By combining (1) and (2), we have the identity $f_H \wedge_{bit} f_H = f_H \vee g$.

The size of the OBDD of $\bigvee_{i=1}^{m} g_i = \bigvee_{i=1}^{m} (\bar{x}_i \wedge \bar{x}_{i+m})$ is known to be $\Omega(2^m)$ for the variable ordering $(x_{2m}, x_{2m-1}, \ldots, x_1)$ [4], where this function is obtained from $f_E \wedge_{bit} f_E$ by restricting $x_{4m}, x_{4m-1}, \ldots, x_{2m+1}$ to 0. Since the size does not increase by a restriction, the size of the OBDD of $f_E \wedge_{bit} f_E$ is $\Omega(2^{n/4})$. □

Our main result however shows that the condition $f \wedge_{bit} f = f$ can be checked in polynomial time without explicitly constructing the OBDD of $f \wedge_{bit} f$. For this goal, the following lemmas tell a key property that the problem can be divided into two subproblems, and hence can be solved by a divide-and-conquer approach.

Lemma 5.1. *Let f be a Boolean function on n variables $x_1, x_2, \ldots, x_n$, which is expanded as $f = \bar{x}_n f_0 \vee x_n f_1$. Then, f is Horn if and only if both f_0 and f_1 are Horn and $f_0 \wedge_{bit} f_1 \leqslant f_0$ holds.*

Proof. We first prove the only-if-part. Let b and c be models of f, where b and c may be identical. Then, by definition of a Horn function, $a = b \wedge_{bit} c$ is also a model of f. By considering the nth bits b_n and c_n of models b and c, without loss of the generality, we have the following three cases:

(1) Both are 0's, i.e., $b = (b', 0)$ and $c = (c', 0)$, where (b', b_n) is a concatenation of b' $(\in \{0,1\}^{n-1})$ and b_n.
(2) Both are 1's, i.e., $b = (b', 1)$ and $c = (c', 1)$.
(3) One is 0 and the other is 1, i.e., $b = (b', 0)$ and $c = (c', 1)$.

Case (1) implies $(b', 0) \wedge_{bit} (c', 0) = (a', 0)$. Namely, for any two models b' and c' of f_0, $b' \wedge_{bit} c'$ is also a model of f_0. By definition, this says that f_0 is Horn. Similarly, case (2) implies that f_1 is Horn. Finally, case (3) implies $(b', 0) \wedge_{bit} (c', 1) = (a', 0)$. Namely, for any models b' of f_0 and c' of f_1, $b' \wedge_{bit} c'$ is always a models of f_0. Thus, we have the property $\Sigma(f_0) \wedge_{bit} \Sigma(f_1) \subseteq \Sigma(f_0)$, i.e., $f_0 \wedge_{bit} f_1 \leqslant f_0$.

Now, the proof of the if-part is trivial. For any two models b and c of f, one of the above three cases (1), (2) and (3) holds. In case (1) (respectively, case (2)), since f_0 (respectively, f_1) is Horn, $(a', 0) = (b', 0) \wedge_{bit} (c', 0)$ (respectively, $(a', 1) = (b', 1) \wedge_{bit} (c', 1)$) is also a model of f. In case (3), since $f_0 \wedge_{bit} f_1 \leqslant f_0$ holds, $(a', 0) = (b', 0) \wedge_{bit} (c', 1)$ is also a model of f. Since $a = b \wedge_{bit} c$ is always a model of f, by definition, f is Horn. □

The Horn property of f_0 and f_1 can be checked by applying Lemma 5.1 recursively. The following lemma says that the condition $f_0 \wedge_{bit} f_1 \leqslant f_0$ in Lemma 5.1 can be also checked recursively.

Lemma 5.2. *Let f, g and h be Boolean functions on n variables, which are expanded as $f = \bar{x}_n f_0 \vee x_n f_1$, $g = \bar{x}_n g_0 \vee x_n g_1$ and $h = \bar{x}_n h_0 \vee x_n h_1$, respectively. Then, property $f \wedge_{bit} g \leqslant h$ holds if and only if $f_0 \wedge_{bit} g_0 \leqslant h_0$, $f_0 \wedge_{bit} g_1 \leqslant h_0$, $f_1 \wedge_{bit} g_0 \leqslant h_0$ and $f_1 \wedge_{bit} g_1 \leqslant h_1$ hold.*

Proof. The identity

$$f \wedge_{bit} g = \bar{x}_n\big((f_0 \wedge_{bit} g_0) \vee (f_0 \wedge_{bit} g_1) \vee (f_1 \wedge_{bit} g_0)\big) \vee x_n(f_1 \wedge_{bit} g_1) \tag{3}$$

can be proved in a manner similar to Lemma 5.1 by considering all models. Then, since $f \wedge_{bit} g \leqslant h$ holds if and only if $(f \wedge_{bit} g)|_{x_n=0} = (f_0 \wedge_{bit} g_0) \vee (f_0 \wedge_{bit} g_1) \vee (f_1 \wedge_{bit} g_0) \leqslant h|_{x_n=0} = h_0$ and $(f \wedge_{bit} g)|_{x_n=1} = f_1 \wedge_{bit} g_1 \leqslant h|_{x_n=1} = h_1$ hold, this lemma follows from Lemma 2.1(2). □

Note that the condition of type $f \wedge_{bit} g \leqslant f$ in Lemma 5.1 requires to check the condition of type $f_1 \wedge_{bit} g_0 \leqslant f_0$ (i.e., checking of type $f \wedge_{bit} g \leqslant h$ for three functions f, g and h). The last condition can be checked recursively by Lemma 5.2.

5.2. Algorithm to check the Horn property

Algorithm CHECK-HORN in Fig. 9 checks the Horn property of a given OBDD in the bottom-up manner by applying Lemmas 5.1 and 5.2 recursively. The bottom-up and caching techniques used there are similar to those of CHECK-UNATE. However, we emphasize here that, in the case of unateness, the naive algorithm to check the condition of Property 4.1 was already polynomial time, while a naive algorithm to check the condition of Lemma 5.1 would require exponential time by Theorem 5.1. This CHECK-HORN is the first polynomial time algorithm, which is made possible by using both Lemmas 5.1 and 5.2.

In Algorithm CHECK-HORN, we use an array *horn*[v] to denote whether each node v represents a Horn function or not, and a three-dimensional array *bit-imp*[u, v, w] to denote whether $f_u \wedge_{bit} f_v \leqslant f_w$ holds or not. Each element of the arrays stores YES, NO or $*$ (not

Algorithm CHECK-HORN

Input: An OBDD representing f with a variable ordering $(x_n, x_{n-1}, \ldots, x_1)$.
Output: "yes" if f is Horn; otherwise, "no".

Step 1 (initialize). Set

$$\textit{horn}[v] := \begin{cases} \text{YES} & \text{if } v \text{ is a constant node 0 or 1;} \\ * & \text{otherwise;} \end{cases}$$

$$\textit{bit-imp}[u, v, w] := \begin{cases} \text{NO} & \text{if } (f_u, f_v, f_w) = (1, 1, 0); \\ \text{YES} & \text{if } f_u, f_v, f_w \in \{0, 1\} \text{ and } (f_u, f_v, f_w) \neq (1, 1, 0); \\ * & \text{otherwise;} \end{cases}$$

$\ell := 1$.

Step 2 (check the Horn property in level ℓ). For each node v in level ℓ (i.e., labeled with x_ℓ), set *horn*[v] := YES if all of *horn*[0-*succ*(v)], *horn*[1-*succ*(v)] and *bit-imp*[0-*succ*(v), 1-*succ*(v), 0-*succ*(v)] are YES; otherwise, output "no" and halt.

Step 3 (compute *bit-imp*[$*$] in level ℓ). For each triple (u, v, w) of nodes such that *level*(u) $\leqslant \ell$, *level*(v) $\leqslant \ell$ and *level*(w) $\leqslant \ell$, and at least one of *level*(u), *level*(v) and *level*(w) is equal to ℓ, check whether $f_u \wedge_{bit} f_v \leqslant f_w$ holds according to Fig. 10. Set its result YES or NO to *bit-imp*[u, v, w].

Step 4 (iterate). If $\ell = n$ then output "yes" and halt. Otherwise set $\ell := \ell + 1$ and return to Step 2.

Fig. 9. Algorithm CHECK-HORN to check the Horn property of an OBDD.

YES	if all of *bit-imp*[1-*succ*′(*u*), 1-*succ*′(*v*), 1-*succ*′(*w*)], *bit-imp*[0-*succ*′(*u*), 0-*succ*′(*v*), 0-*succ*′(*w*)], *bit-imp*[0-*succ*′(*u*), 1-*succ*′(*v*), 0-*succ*′(*w*)] and *bit-imp*[1-*succ*′(*u*), 0-*succ*′(*v*), 0-*succ*′(*w*)] are YES.
NO	otherwise.

Fig. 10. Checking *bit-imp*$[u, v, w]$ (i.e., $f_u \wedge_{bit} f_v \leqslant f_w$) for a triple of nodes (u, v, w) in Step 3 of Algorithm CHECK-HORN.

checked yet); *horn*$[v]$ = YES says that f_v is Horn and *bit-imp*$[u, v, w]$ = YES says that $f_u \wedge_{bit} f_v \leqslant f_w$ holds. We note here that, since the given OBDD is reduced, the condition $f_u \wedge_{bit} f_v \leqslant f_w$ may be checked for functions in different levels; in such case, all functions are considered to have l_{max} variables $x_1, x_2, \ldots, x_{l_{max}}$, where l_{max} denotes the maximum level of the nodes u, v and w.

In Step 2 of Algorithm CHECK-HORN, *horn*$[v]$ for each v can be easily computed according to Lemma 5.1. Note that every node v satisfies $f_v|_{x_{level(v)}=0} = f_{0\text{-}succ(v)}$ and $f_v|_{x_{level(v)}=1} = f_{1\text{-}succ(v)}$. Also note that *horn*[0-*succ*(v)], *horn*[1-*succ*(v)] and *bit-imp*[0-*succ*(v), 1-*succ*(v), 0-*succ*(v)] have already been computed in the previous iterations.

Similarly, *bit-imp*$[u, v, w]$ in Step 3 for each triple (u, v, w) can be also computed easily by Fig. 10, which corresponds to Lemma 5.2. Similar to the case of checking unateness, 0-*succ*′(v) (respectively, 1-*succ*′(v)) denotes 0-*succ*(v) (respectively, 1-*succ*(v)) if *level*$(v) = \ell$, but denotes v itself if *level*$(v) < \ell$. Upon completing Step 3 for ℓ, we have the results for all triples (u, v, w) of nodes such that *level*$(u) \leqslant \ell$, *level*$(v) \leqslant \ell$ and *level*$(w) \leqslant \ell$. These contain all the information required in the next iteration, although some of them may not be needed.

Example 5.1. Consider the OBDD of $f_I = (x_3 \vee \bar{x}_2 \vee \bar{x}_1)(\bar{x}_3 \vee x_2)(\bar{x}_3 \vee x_1)$ with variable ordering (x_3, x_2, x_1). As shown in Fig. 11, the OBDD has 7 nodes $v_1, v_2, \ldots, v_7$, which respectively represent the following functions:

$$
\begin{aligned}
f_{v_1} &= 1,\\
f_{v_2} &= 0,\\
f_{v_3} &= \bar{x}_1,\\
f_{v_4} &= x_1,\\
f_{v_5} &= \bar{x}_2 \vee \bar{x}_1,\\
f_{v_6} &= x_2 x_1,\\
f_{v_7} &= (x_3 \vee \bar{x}_2 \vee \bar{x}_1)(\bar{x}_3 \vee x_2)(\bar{x}_3 \vee x_1).
\end{aligned}
$$

Algorithm CHECK-HORN starts from the initializing step:

horn$[v_1]$ = *horn*$[v_2]$ = YES, *horn*$[v_i] = *$ for all $i = 3, 4, \ldots, 7$,

bit-imp$[v_1, v_1, v_2]$ = NO,

bit-imp$[v_i, v_j, v_k]$ = YES for all triples $(v_i, v_j, v_k) \in \{v_1, v_2\}^3$ but (v_1, v_1, v_2),

and $\ell = 1$.

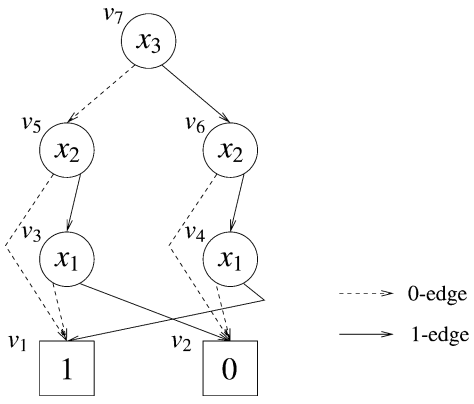

Fig. 11. OBDD representing $f_I = (x_3 \vee \bar{x}_2 \vee \bar{x}_1)(\bar{x}_3 \vee x_2)(\bar{x}_3 \vee x_1)$.

In the rest of this example, we pay attention to *bit-imp*[ ]'s which are needed in the following computation. In Step 2 of the first iteration, we have $\mathit{horn}[v_3] = \mathit{horn}[v_4] = \text{YES}$ because $\mathit{horn}[v_1] = \mathit{horn}[v_2] = \text{YES}$ and $\mathit{bit\text{-}imp}[v_1, v_2, v_1] = \mathit{bit\text{-}imp}[v_2, v_1, v_2] = \text{YES}$ hold. In Step 3 of the first iteration, we have

$$\mathit{bit\text{-}imp}[v_1, v_3, v_1] = \text{YES},$$

which can be confirmed by the property

$$1 \wedge_{\mathit{bit}} x_1 = \{0, 1\} \wedge_{\mathit{bit}} \{1\} = \{0, 1\} = 1.$$

We also have the property

$$\begin{aligned} \mathit{bit\text{-}imp}[v_2, v_4, v_2] &= \mathit{bit\text{-}imp}[v_3, v_2, v_1] = \mathit{bit\text{-}imp}[v_3, v_4, v_3] \\ &= \mathit{bit\text{-}imp}[v_1, v_4, v_1] = \text{YES}. \end{aligned}$$

In the second iteration (level $\ell = 2$), we have $\mathit{horn}[v_5] = \mathit{horn}[v_6] = \text{YES}$ because the properties $\mathit{horn}[v_1] = \mathit{horn}[v_2] = \mathit{horn}[v_3] = \mathit{horn}[v_4] = \text{YES}$ and $\mathit{bit\text{-}imp}[v_1, v_3, v_1] = \mathit{bit\text{-}imp}[v_2, v_4, v_2] = \text{YES}$ hold. We also have the property

$$\mathit{bit\text{-}imp}[v_5, v_6, v_5] = \text{YES}$$

because

$$\begin{aligned} \mathit{bit\text{-}imp}[v_3, v_4, v_3] &= \mathit{bit\text{-}imp}[v_1, v_2, v_1] = \mathit{bit\text{-}imp}[v_1, v_4, v_1] \\ &= \mathit{bit\text{-}imp}[v_3, v_2, v_1] = \text{YES}. \end{aligned}$$

Note that $\mathit{bit\text{-}imp}[v_3, v_2, v_1]$ is an example of the general property that checking the condition of type $f \wedge_{\mathit{bit}} g \leqslant f$ in Lemma 5.1 requires checking the condition of type $f \wedge_{\mathit{bit}} g \leqslant h$. In the third iteration (level $\ell = 3$), we have $\mathit{horn}[v_7] = \text{YES}$ because the properties $\mathit{horn}[v_5] = \mathit{horn}[v_6] = \text{YES}$ and $\mathit{bit\text{-}imp}[v_5, v_6, v_5] = \text{YES}$ hold. Then, Algorithm CHECK-HORN outputs the answer "yes" and halts. □

Now, we consider the computation time of Algorithm CHECK-HORN. In Step 2, *horn*$[v]$ for each node v is computed in constant time. Thus, the total time of Step 2 is $O(|f|)$. In Step 3, *bit-imp*$[u, v, w]$ for each triple (u, v, w) is also computed in constant time. The number of triples to be checked in Step 3 during the entire computation is $O(|f|^3)$. The time for the rest of the computation is minor.

Theorem 5.2. *Given an OBDD representing a Boolean function f, checking whether f is Horn can be done in $O(|f|^3)$ time, where $|f|$ is the size of the given OBDD.*

As stated above, it may not be necessary to compute *bit-imp*$[u, v, w]$'s for all triples (u, v, w) of nodes. The following theorem however gives an unfortunate instance which requires the computation of *bit-imp*$[u, v, w]$'s for $\Theta(|f|^3)$ triples.

Theorem 5.3. *There exists an OBDD of a Horn function f which requires to check $\Theta(|f|^3)$ bit-imp$[u, v, w]$'s, where $|f|$ is the size of the given OBDD.*

Proof. By an argument similar to Theorem 4.2, we can prove this theorem. Consider the function f_J with $n = 6m + 2$ variables defined below:

$$
\begin{aligned}
f_J &= \bar{x}_{6m+2} g_0 \vee x_{6m+2} g_1,\\
g_0 &= \bar{x}_{6m+1} h_{A,m} \vee x_{6m+1} h_{B,m},\\
g_1 &= \bar{x}_{6m+1} h_{C,m} \vee x_{6m+1} 0,\\
h_{A,i} &= \begin{cases} (\bar{x}_{3m+i} \vee h_{A,i-1})\bar{x}_{3m-i+1} \vee (\bar{x}_{m+1} \vee \bar{x}_1) & (i = 1, 2, \ldots, m)\\ 0 & (i = 0), \end{cases}\\
h_{B,i} &= \begin{cases} (\bar{x}_{4m+i} \vee h_{B,i-1})\bar{x}_{2m-i+1} & (i = 1, 2, \ldots, m)\\ 0 & (i = 0), \end{cases}\\
h_{C,i} &= \begin{cases} (\bar{x}_{5m+i} \vee h_{C,i-1})\bar{x}_{m-i+1} & (i = 1, 2, \ldots, m)\\ 0 & (i = 0). \end{cases}
\end{aligned}
$$

The OBDD of f_J has $6m + 6$ nodes.

The upper bound $O(|f|^3)$ is obvious from Theorem 5.2. We show that f_J requires to check at least m^3 *bit-imp*[ ]'s, which proves the lower bound. Since $h_{A,m}$, $h_{B,m}$ and $h_{C,m}$ are negative functions (i.e., Horn functions), Algorithm CHECK-HORN executes iterations of Steps 2 to 4 from the initial level $\ell = 1$ to level $\ell = 6m$. In level $\ell = 6m + 1$, in order to obtain the result $g_0 \wedge_{bit} g_1 \leqslant g_0$ in Step 3, it is necessary to have the property $h_{B,m} \wedge_{bit} h_{C,m} \leqslant h_{A,m}$. (Note that the property $g_0 \wedge_{bit} g_1 \leqslant g_0$ is used for checking the Horn property of f_J.) By Lemma 5.2, $g_0 \wedge_{bit} g_1 \leqslant g_0$ holds if and only if

$$
h_{A,m} \wedge_{bit} h_{C,m} \leqslant h_{A,m}, \quad h_{A,m} \wedge_{bit} 0 \leqslant h_{A,m},
$$

$$
h_{B,m} \wedge_{bit} h_{C,m} \leqslant h_{A,m} \quad \text{and} \quad h_{B,m} \wedge_{bit} 0 \leqslant h_{B,m}
$$

hold. In order to confirm $h_{B,m} \wedge_{bit} h_{C,m} \leqslant h_{A,m}$, we need to check $h'_{B,i} \wedge_{bit} h'_{C,j} \leqslant h'_{A,k}$ for all triples $(i, j, k) \in \{1, 2, \ldots, m\}^3$, where

$$h'_{A,i} = \left(\bigwedge_{k=1}^{i} \bar{x}_{2m+k} \right) \vee (\bar{x}_{m+1} \vee \bar{x}_1), \quad h'_{B,i} = \bigwedge_{k=1}^{i} \bar{x}_{m+k} \quad \text{and} \quad h'_{C,i} = \bigwedge_{k=1}^{i} \bar{x}_k.$$

Thus, it is necessary to check m^3 *bit-imp*[ ]'s. □

6. Conclusion

In this paper, we considered to use OBDDs to represent knowledge-bases. We have shown that the conventional CNF-based and model-based representations, and the new OBDD representation are mutually incomparable with respect to space requirement. Thus, OBDDs can find their place in knowledge-bases, as they can represent some theories more efficiently than others.

We then considered the problem of recognizing whether a given OBDD represents a unate Boolean function, and whether it represents a Horn function. It turned out that checking unateness can be done in quadratic time with respect to the size of OBDD, while checking the Horn property can be done in cubic time.

OBDDs are dominatingly used in the field of computer-aided design and verification of digital systems. The reason for this is that many Boolean functions which we encounter in practice can be compactly represented, and that many operations on OBDDs can be efficiently performed. We believe that OBDDs are also useful for manipulating knowledge-bases. Developing efficient algorithms for knowledge-base operations such as deduction and abduction should be addressed in the further work.

Acknowledgement

The authors would like to thank Professor Endre Boros of Rutgers University for his valuable comments. This research was partially supported by the Scientific Grant-in-Aid from Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] S.B. Akers, Binary decision diagrams, IEEE Trans. Comput. C-27 (1978) 509–516.
[2] K.S. Brace, R.L. Rundell, R.E. Bryant, Efficient implementation of a BDD package, in: Proc. ACM/IEEE Design Automation Conference, Orlando, FL, 1990, pp. 40–45.
[3] R.K. Brayton, G.D. Hachtel, A.L. Sangiovanni-Vincentelli, Multilevel logic synthesis, Proc. IEEE 78 (2) (1990) 264–300.
[4] R.E. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Trans. Comput. C-35 (1986) 677–691.
[5] R.E. Bryant, On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication, IEEE Trans. Comput. 40 (1991) 205–213.

[6] P. Buch, A. Narayan, A.R. Newton, A. Sangiovanni-Vincentelli, Logic synthesis for large pass transistor circuits, in: Proc. IEEE/ACM International Conference on Computer Aided Design, San Jose, CA, 1997, pp. 663–670.
[7] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, Sequential circuit verification using symbolic model checking, in: Proc. ACM/IEEE Design Automation Conference, Orlando, FL, 1990, pp. 46–51.
[8] O. Coudert, Doing two-level logic minimization 100 times faster, in: Proc. ACM/SIAM Symposium on Discrete Algorithms, San Francisco, CA, 1995, pp. 112–118.
[9] R. Dechter, J. Pearl, Structure identification in relational data, Artificial Intelligence 58 (1992) 237–270.
[10] W.F. Dowling, J.H. Gallier, Linear time algorithms for testing the satisfiability of Horn formula, J. Logic Programming 3 (1984) 267–284.
[11] R. Drechsler, N. Drechsler, W Günther, Fast exact minimization of BDDs, in: Proc. ACM/IEEE Design Automation Conference, San Francisco, CA, 1998, pp. 200–205.
[12] T. Eiter, G. Gottlob, The complexity of logic-based abduction, J. ACM 42 (1995) 3–42.
[13] C. Glymour, R. Scheines, P. Spirtes, K. Kelly, Discovering Causal Structure, Academic Press, Orlando, FL, 1987.
[14] K. Hayase, H. Imai, OBDDs of a monotone function and of its implicants, in: Proc. International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, Vol. 1178, Springer, Berlin, 1996, pp. 136–145.
[15] T. Horiyama, T. Ibaraki, Reasoning with ordered binary decision diagrams, in: Proc. International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, Vol. 1969, Springer, Berlin, 2000, pp. 120–131.
[16] T. Horiyama, T. Ibaraki, Translation among CNFs, characteristic models and ordered binary decision diagrams, in: Proc. International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, Vol. 2223, Springer, Berlin, 2001, pp. 231–243.
[17] H.A. Kautz, M.J. Kearns, B. Selman, Reasoning with characteristic models, in: Proc. AAAI-93, Washington, DC, 1993, pp. 34–39.
[18] H.A. Kautz, M.J. Kearns, B. Selman, Horn approximations of empirical data, Artificial Intelligence 74 (1995) 129–245.
[19] H.A. Kautz, B. Selman, An empirical evaluation of knowledge compilation by theory approximation, in: Proc. AAAI-94, Seattle, WA, 1994, pp. 155–161.
[20] D. Kavvadias, C. Papadimitriou, M. Sideri, On Horn envelopes and hypergraph transversals, in: Proc. International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, Vol. 762, Springer, Berlin, 1993, pp. 399–405.
[21] R. Khardon, D. Roth, Reasoning with models, Artificial Intelligence 87 (1996) 187–213.
[22] Y.T. Lai, K.R. Pan, M. Pedram, OBDD-based functional decomposition: Algorithms and implementation, IEEE Trans. CAD 15 (8) (1996) 977–990.
[23] J.C. Madre, O. Coudert, A logically complete reasoning maintenance system based on a logical constraint solver, in: Proc. IJCAI-91, Sydney, Australia, 1991, pp. 294–299.
[24] J. McCarthy, P.J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: D. Michie (Ed.), Machine Intelligence, Vol. 4, Edinburgh University Press, Edinburgh, 1969.
[25] S. Minato, N. Ishiura, S. Yajima, Shared binary decision diagram with attributed edges for efficient Boolean function manipulation, in: Proc. ACM/IEEE Design Automation Conference, Orlando, FL, 1990, pp. 52–57.
[26] K. Ravi, K.L. McMillan, T.R. Shiple, F. Somenzi, Approximation and decomposition of binary decision diagrams, in: Proc. ACM/IEEE Design Automation Conference, San Francisco, CA, 1998, pp. 445–450.
[27] N. Takahashi, N. Ishiura, S. Yajima, Fault simulation for multiple faults using BDD representation of fault sets, in: Proc. IEEE/ACM International Conference on Computer Aided Design, Santa Clara, CA, 1991, pp. 550–553.
[28] Y. Takenaga, Theoretical studies on memory-based parallel computation and ordered binary decision diagrams, Ph.D. Thesis, Graduate School of Engineering, Kyoto University, Kyoto, Japan, 1994.
[29] L.G. Valiant, A theory of the learnable, Comm. ACM 27 (11) (1984) 1134–1142.
[30] C. Yang, M. Ciesielski, V. Singhal, BDS: A BDD-based logic optimization system, in: Proc. ACM/IEEE Design Automation Conference, Los Angeles, CA, 2000, pp. 92–97.