Augmenting Ordered Binary Decision Diagrams with Conjunctive Decomposition

Yong Lai\textsuperscript{1,2,*}, Dayou Liu\textsuperscript{1,2}, Minghao Yin\textsuperscript{2,3}

\textsuperscript{1}College of Computer Science and Technology, Jilin University, Changchun 130012, P.R. China
\textsuperscript{2}Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun 130012, P.R. China
\textsuperscript{3}College of Computer Science and Information Technology, Northeast Normal University, Changchun, P. R. China, 130117

laiy@jlu.edu.cn; liudy@jlu.edu.cn; ymh@nenu.edu.cn

Abstract

This paper augments OBDD with conjunctive decomposition to propose a generalization called OBDD[$\wedge$]. By imposing reducedness and the finest $\wedge$-decomposition bounded by integer $i$ ($\wedge_i$-decomposition) on OBDD[$\wedge$], we identify a family of canonical languages called ROBDD[$\wedge_i$], where ROBDD[$\wedge_0$] is equivalent to ROBDD. We show that the succinctness of ROBDD[$\wedge_i$] is strictly increasing when $i$ increases. We introduce a new time-efficiency criterion called rapidity which reflects that exponential operations may be preferable if the language can be exponentially more succinct, and show that the rapidity of each operation on ROBDD[$\wedge_i$] is increasing when $i$ increases; particularly, the rapidity of some operations (e.g., conjoining) is strictly increasing. Finally, our empirical results show that: a) the size of ROBDD[$\wedge_i$] is normally not larger than that of its equivalent ROBDD[$\wedge_{i+1}$]; b) conjoining two ROBDD[$\wedge_i$]s is more efficient than conjoining two ROBDD[$\wedge_0$]s in most cases, where the former is NP-hard but the latter is in P; and c) the space-efficiency of ROBDD[$\wedge_\infty$] is comparable with that of d-DNNF and that of another canonical generalization of ROBDD called SDD.

Introduction

Knowledge Compilation (KC) is a key approach for dealing with the computational intractability in propositional reasoning [Selman and Kautz 1996; Darwiche and Marquis 2002]. A core issue in the KC community is to identify target languages and then to evaluate them according to their properties. This paper focuses on three key properties: the canonicity of results of compiling knowledge bases into the language, the space-efficiency of storing compiled results, and the time-efficiency of operating compiled results. (Darwiche and Marquis 2002) proposed a KC map to characterize space-time efficiency by succinctness and tractability, where succinctness refers to the polysize transformation between languages, and tractability refers to the set of polytime operations a language supports. For an application, the KC map argues that one should first locate the necessary operations, and then choose the most succinct language that supports these operations in polytime.

Ordered Binary Decision Diagram (OBDD) is one of the most influential KC languages in the literature [Bryant 1986], due to twofold main theoretical advantages. First, its subset Reduced OBDD (ROBDD) is a canonical representation. Second, ROBDD is one of the most tractable target languages which supports all the query operations and many transformation operations (e.g., conjoining) mentioned in the KC map in polytime. Despite its current success, a well-known problem with OBDD is its weak succinctness, which reflects the explosion in size for many types of knowledge bases. Therefore, (Lai et al. 2013) generalized OBDD by associating some implied literals with each non-false vertex to propose a more succinct language called OBDD with implied literals (OBDD-L). They showed that OBDD-L maintains both advantages of OBDD. First, its subset ROBDD with implied literals as many as possible (ROBDD-$L_\infty$) is also canonical. Second, given each operation ROBDD supports in polytime, ROBDD-$L_\infty$ also supports it in polytime in the sizes of the equivalent ROBDDs.

In order to further mitigate the size explosion problem of ROBDD without loss of its theoretical advantages, we generalize OBDD-$L$ by augmenting OBDD with conjunctive decomposition to propose a language called OBDD[$\wedge$]. We then introduce a special type of $\wedge$-decomposition called finest $\wedge_i$-decomposition bounded by integer $i$ ($\wedge_i$-decomposition), and impose reducedness and $\wedge_{i+1}$-decomposition on OBDD[$\wedge$] to identify a family of canonical languages called ROBDD[$\wedge_i$]. In particular, ROBDD[$\wedge_0$] and ROBDD[$\wedge_2$] are respectively equivalent to ROBDD and ROBDD-$L_\infty$. We show that the succinctness of ROBDD[$\wedge_i$] is strictly stronger than ROBDD[$\wedge_j$] if $i > j$. Our empirical results verify this property and also show that the space-efficiency of ROBDD[$\wedge_\infty$] is comparable with that of deterministic Decomposable Negation Normal Form (d-DNNF, a superset of OBDD[$\wedge$]) [Darwiche 2001] and that of another canonical subset called Sentential Decision Diagram (SDD) [Darwiche 2011] in d-DNNF.

We evaluate the tractability of ROBDD[$\wedge_i$] and show that ROBDD[$\wedge_i$] ($i > 0$) does not satisfy SE (resp. SFO, $\Lambda$BC and $\nu$BC) unless $P = NP$. According to the viewpoint of KC map, the applications which need the operation $OP$ corresponding to SE (resp. SFO, $\Lambda$BC and $\nu$BC) will prefer to ROBDD[$\wedge_0$] than ROBDD[$\wedge_i$]. In fact, the latter is strictly more succinct than the former, and also supports $OP$ in...
polytime in the sizes of the equivalent formulas in the former \cite{Lai et al. 2013}. In order to fix this “bug”, we propose an additional time-efficiency evaluation criterion called rapidity which reflects an increase of at most polynomial multiples of time cost of an operation. We show that each operation on ROBDD[$\land_i$] is at least as rapid as that on ROBDD[$\land_i^0$] if $i \geq j$. In particular, some operations (e.g., conjoining) on ROBDD[$\land_i^0$] are strictly more rapid than those on ROBDD[$\land_i$] if $i > j$. Our empirical results verify that conjoining two ROBDD[$\land_i^0$]s is more efficient than conjoining two ROBDD[$\land_i$]s in most cases, where the former is NP-hard but the latter is in P.

**Basic Concepts**

We denote a propositional variable by $x$, and a denumerable variable set by $PV$. A formula $\varphi$ is constructed from constants true, false and variables using negation operator $\neg$, conjunction operator $\land$ and disjunction operator $\lor$, and we denote by $Vars(\varphi)$ the set of its variables and by $PI(\varphi)$ the set of its prime implicants. The conditioning of $\varphi$ on assignment $\omega(\varphi)$ is the formula obtained by replacing each appearance of $x$ in $\varphi$ by true (false) if $x = true$ (false) $\in \omega$. $\varphi$ depends on a variable $x$ iff $\varphi|_{x=\text{false}} \neq \varphi|_{x=\text{true}}$. $\varphi$ is redundant iff it does not depend on some $x \in Vars(\varphi)$. $\varphi$ is trivial iff it depends on no variable.

**Definition 1** ($\land$-decomposition). A formula set $\Psi$ is a $\land$-decomposition of $\varphi$, iff $\varphi \equiv \bigwedge_{\psi \in \Psi} \psi$ and $\{Vars(\psi) : \psi \in \Psi\}$ partitions $Vars(\varphi)$. A decomposition $\Psi$ is finer than another $\Psi'$ iff $\{Vars(\psi) : \psi \in \Psi\}$ is a refinement of $\{Vars(\psi) : \psi \in \Psi'\}$ and $\Psi$ is strict iff $|\Psi| > 1$.

Let $\varphi$ be a non-trivial formula. If $\varphi$ is irredundant and $\{\psi_1, \ldots, \psi_m\}$ is its $\land$-decomposition, $PI(\varphi) = PI(\psi_1) \cup \cdots \cup PI(\psi_m)$. If $\varphi$ does not depend on $x \in Vars(\varphi)$ and $\Psi$ is a $\land$-decomposition of $\varphi|_{x=\text{true}}$, we can get a strict $\land$-decomposition of $\varphi$ by adding $\neg x \lor x$ to $\Psi$. Therefore,

**Proposition 1.** From the viewpoint of equivalence, each non-trivial formula $\varphi$ has exactly one finest $\land$-decomposition.

**Definition 2** ($\land_i$-decomposition). A $\land$-decomposition $\Psi$ is bounded by an integer $0 \leq i \leq \infty$ ($\land_i$-decomposition) iff there exists at most one formula $\psi \in \Psi$ satisfying $|Vars(\psi)| > i$.

Given a $\land$-decomposition $\Psi$, we can get an equivalent $\land_i$-decomposition by conjoining the formulas in $\Psi$ which has more than $i$ variables. According to Proposition[1] we have:

**Proposition 2.** For any non-trivial formula $\varphi$ and integer $0 \leq i \leq \infty$, $\varphi$ has exactly one finest $\land_i$-decomposition from the viewpoint of equivalence.

Hereafter the finest $\land_i$-decomposition is denoted by $\land_i^{\ast}$-decomposition.

**BDD[$\land_i$] and Its Subsets**

In this section, we define binary decision diagram with conjunctive decomposition (BDD[$\land_i$]) and some of its subsets.

**Definition 3** (BDD[$\land_i$]). A BDD[$\land_i$] is a rooted directed acyclic graph. Each vertex $v$ is labeled with a symbol $\text{sym}(v)$: if $v$ is a leaf, $\text{sym}(v) = \bot / \top$; otherwise, $\text{sym}(v) = \land$ (decomposition vertex) or $\text{sym}(v) \in PV$ (decision vertex). Each internal vertex $v$ has a set of children $Ch(v)$; for a decision vertex, $Ch(v) = \{lo(v), hi(v)\}$, where $lo(v)$ and $hi(v)$ are called low and high children and connected by dashed and solid edges, respectively. Each vertex represents the following formula:

$$\vartheta(v) = \begin{cases} \text{false/false} & \text{sym}(v) = \bot / \top; \\
\land_{w \in Ch(v)} \vartheta(w) & \text{sym}(v) = \land; \end{cases}$$

where $\{\vartheta(w) : w \in Ch(v)\}$ is a strict $\land$-decomposition of $\vartheta(v)$ if $\text{sym}(v) = \land$, and $\varphi \circ_x \psi = (\neg x \lor \varphi) \lor (x \land \psi)$. The formula represented by the BDD[$\land_i$] is defined as the one represented by its root.

Hereafter we denote a leaf vertex by $(\bot / \top)$, a decomposition vertex $(\land$-vertex for short) by $(\land \varphi \land v)$, and a decision vertex ($\lor$-vertex for short) by $\langle \land \varphi \lor v \rangle$. We abuse $\langle \land \{w\} \rangle$ to denote $\langle \land \rangle$, $\langle \land \{\top\} \rangle$ to denote $\langle \top \rangle$, and $\langle \land \{\{\top\}\} \cup \{v\} \rangle$ to denote $\langle \bot \rangle$. Given a BDD[$\land_i$] $\Gamma$, $|\Gamma|$ denotes the size of $\Gamma$ defined as the number of its edges. In addition, we use $\Gamma_v$ to denote the BDD[$\land_i$] rooted at $v$, and occasionally abuse $v$ to denote $\vartheta(v)$. Now we define the subsets of BDD[$\land_i$]:

**Definition 4** (subsets of BDD[$\land_i$]). A BDD[$\land_i$] is ordered over a linear order of variables $\prec$ over $PV$ (OBDD[$\land_i$]) iff each $\lor$-vertex $u$ and its $\lor$-descendant $v$ satisfy $\text{sym}(u) \prec \text{sym}(v)$. An OBDD[$\land_i$] is reduced (ROBDD[$\land_i$]), iff no two vertes are identical (having the same symbol and children) and no $\lor$-vertex has two identical children. An OBDD[$\land_i$] is $\land_{i}$-decomposable (OBDD[$\land_{i}^0$]) iff each $\land$-vertex is a $\land_{i}$-decomposition. An ROBDD[$\land_i$] is $\land_{i}^{\ast}$-decomposable (ROBDD[$\land_{i}^{\ast}$]), if each $\land$-vertex is a $\land_{i}^{\ast}$-decomposition and the $\land_{i}^{\ast}$-decomposition of each $\lor$-vertex is $\{v\}$.

For any two OBDD[$\land_i$]s, unless otherwise stated, hereafter assume that they are over the same variable order and $x_k$ is the $k$th variable. In the following, we mainly focus on ROBDD[$\land_i$], and analyze its canonicity and space-time efficiency. Obviously, ROBDD is equivalent to ROBDD[$\land_i^0$]. In addition, since a BDD vertex labelled with a set of implied literals in \cite{Lai et al. 2013} can be seen as a $\land_1$-vertex, it is easy to show ROBDD[$\land_\infty$] is equivalent to ROBDD[$\land_1$]. Figures[1][3][6] respectively depict an ROBDD[$\land_1$] and an ROBDD[$\land_\infty$] representing $\varphi = (x_1 \leftrightarrow x_3 \leftrightarrow x_5) \land (x_2 \leftrightarrow x_4 \leftrightarrow x_6)$. Note that for simplicity, we draw multiple copies of vertices, denoted by dashed boxes, but they represent the same vertex. Figure[3] is not an OBDD[$\land_1$] since vertex $v$ is not bounded by one. If we extend $\varphi$ to the following formula, the number of vertices labelled with $x_{1+n}$ in ROBDD[$\land_i^0$] will equal $2^n$, while the number of vertices in ROBDD[$\land_{i+1}$] ($i > j$) will be $(2j + 5)n$. That is, the size of ROBDD[$\land_{i+1}$] representing Eq. \ref{ex1} is exponential in $n$, while the size of the

\footnote{Each internal vertex in ROBDD[$\land_i$] is non-trivial.}
Given a formula, there is exactly one ROBDD also easily understood, since we can transform ROBDD into ROBDD.

We close this section by pointing out that ROBDD is canonical and complete. The canonicity is immediately from the uniqueness of ROBDD-decomposition. The completeness is also easily understood, since we can transform ROBDD into ROBDD (see the next section).

**Proposition 3.** Given a formula, there is exactly one ROBDD to represent it.

**Space-Efficiency Analysis**

We analyze the space-efficiency in terms of succinctness ([Gogic et al. 1995], [Darwiche and Marquis 2002]). The succinctness results is given as follows, where \( L \leq_s L' \) denotes that \( L \) is not more succinct than \( L' \). Due to space limit we just briefly explain them, from two aspects.

**Proposition 4.** \( \text{ROBDD}[\land_i] \leq_s \text{ROBDD}[\land_j] \) iff \( i \leq j \).

First, we show the direction from right to left by proposing an algorithm called \textsc{Decompose} (in Algorithm [1]) which can transform an OBDD into the equivalent ROBDD in polynome. Note that each ROBDD is an OBDD. \textsc{Decompose} immediately provides a compiling method of ROBDD, that is, transforming ROBDD into ROBDD. Some functions used in \textsc{Decompose} are explained as follows:

- **\textsc{Finest}(u)**: While there exists some \( v \in Ch(u) \) with \( sym(u') = sym(v) = \land \), we repeat removing \( v \) from \( Ch(u') \) and then adding all children of \( v \) to \( Ch(u') \).

- **\textsc{ExtractLeaf}(u)**: The input \( u' \) of this function, as well as the next two functions, is a \( \lor \)-vertex whose children are ROBDD vertices; we employ these functions to get an ROBDD vertex equivalent to \( u' \). \textsc{ExtractLeaf} handles the case where \( \lor \in Ch(u') \) and \( |Ch(u')| > 1 \). Without loss of generality, assume \( lo(u') = \lor \). If \( i = 0 \), \( u' \) is already an ROBDD vertex. Otherwise, let \( v = \langle \land, \{\{\text{sym}(u'), \land, \lor\}, hi(u')\} \rangle = u' \). Then we call \textsc{Finest}(v) to get an equivalent ROBDD vertex.

- **\textsc{ExtractPart}(u)**: This function handles the case where one child of \( u' \) is a part of the other. That is, \( Ch(u') = \{v_1, v_2\} \) satisfies \( v_1 \in Ch(v_2) \) and \( \text{sym}(v_2) = \land \). Without loss of generality, assume \( v_1 = lo(u') \). Let \( v = \langle \text{sym}(u'), \lor, \land \rangle \). Then \( u'' = \langle \land, \{v_1, v\} \rangle = u' \). If \( i < |vars(v_1)| \) and \( i < |vars(v)| \), then \( u'' \) is already an ROBDD vertex. Otherwise, \( u'' \) is an ROBDD vertex.

- **\textsc{ExtractShare}(u)**: This function handles the case where \( lo(u') \) and \( hi(u') \) share some children. That is, \( \text{sym}(lo(u')) = \text{sym}(hi(u')) = \land \) and \( V = Ch(lo(u')) \cap Ch(hi(u')) \neq \emptyset \). If \( V = Ch(lo(u')) \), we return \( lo(u') \). Otherwise, if there exists some \( v \in V \) with \( |vars(v)| > i \) and \( |vars(u') \cap Vars(V)| > i \), we remove \( v \) from \( V \). If \( V = \emptyset \), \( u' \) is already an ROBDD vertex. Otherwise, let \( v = \langle \text{sym}(u'), lo(u') \rangle \land V, hi(u') \rangle \). Then \( \langle \land, V \cup \{v\} \rangle \) is an ROBDD vertex equivalent to \( u' \).

**Algorithm 1: \textsc{Decompose}(u)**

Input: an OBDD vertex \( u \)

Output: the ROBDD vertex which is equivalent to \( u \)

1. if \( H(u) \neq \text{nil} \) then return \( H(u) \)
2. if \( u \) is a leaf vertex then \( u' \leftarrow u \)
3. else
4. \( u' \leftarrow \langle \text{sym}(u), \{\text{Decompose}(v) \in Ch(u)\} \rangle \)
5. if \( u' \) is a \( \land \)-vertex then
6. if \( (\land, v) \in Ch(u') \) and \( |Vars(v)| > 1 \) then
7. \( u' \leftarrow \text{ExtractLeaf}(u') \)
8. else if one child of \( u \) is a part of the other then
9. \( u' \leftarrow \text{ExtractPart}(u) \)
10. else if the children of \( u \) share some children then
11. \( u' \leftarrow \text{ExtractShare}(u') \)
12. else if \( lo(u') = hi(u') \) then \( u' \leftarrow lo(u') \)
13. else \( u' \leftarrow \text{Finest}(u) \)
end
15. if a previous vertex \( u'' \) identical with \( u' \) appears then \( H(u) \leftarrow u'' \)
16. else \( H(u) \leftarrow u' \)
17. return \( H(u) \)

Second, we show the direction from left to right by counterexample: If \( i > j \), Eq. (1) can be represented by an ROBDD in linear size, but the size of the equivalent ROBDD is exponential in \( n \).

**Time-Efficiency Analysis**

We analyze the time-efficiency of operating ROBDD in terms of tractability ([Darwiche and Marquis 2002]) and a new perspective. First we present the operations mentioned in this paper:

**Definition 5** (operation). An operation \( OP \) is a relation between \( \Delta_\rho \times \Delta_\gamma \) and \( \Gamma \), where \( \Delta_\rho \) denotes the primary information of \( OP \) which is a set of sequences of formulas, \( \Delta_\rho \) denotes the supplementary information customized for \( OP \), and \( \Gamma \) is the set of outputs of \( OP \). \( OP \) on language \( L \), denoted by \( OP(L) \), is the subset \( \{ ((\varphi_1, \ldots, \varphi_n), \alpha, \beta) \in OP : \varphi_i \in L \text{ for } 1 \leq i \leq n \text{ and } \beta \in L \text{ if it is a formula} \} \).
Hereafter, we abbreviate \(((\varphi_1, \ldots, \varphi_n), \alpha, \beta) \in OP\) as \((\varphi_1, \ldots, \varphi_n, \alpha, \beta) \in OP\). According to the above definition, we can easily formalize the query operations (CO, VA, CE, IM, EQ, SE, CT and ME) and transformation operations (CD, SFO, FO, \& BC, \& C, \lor BC, \lor C and \neg C) mentioned in the KC map. We say an algorithm \(ALG\) performs operation \(OP(L)\), iff for each \((\varphi_1, \ldots, \varphi_n, \alpha, \beta) \in OP(L)\), \((\varphi_1, \ldots, \varphi_n, \alpha, ALG(\varphi_1, \ldots, \varphi_n, \alpha)) \in OP(L)\).

**Tractability Evaluation**

As the KC map, we say language \(L\) satisfies \(OP\) iff there exists some polytime algorithm performing \(OP(L)\). The tractability results are shown in Table 1 and due to space limit we will only discuss proofs of the less obvious ones.

| \(L\) | CO | VA | CE | IM | EQ | SE | CT | ME |
|------|----|----|----|----|----|----|----|----|
| ROBDD[\(\lambda_0\)] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| ROBDD[\(\lambda_i\)] (\(i > 0\)) | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| ROBDD[\(\lambda_0\)] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| ROBDD[\(\lambda_i\)] (\(i > 0\)) | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |

Since ROBDD[\(\lambda_i\)] is a subset of d-DNNF, it supports each query operation which is tractable for d-DNNF, in polytime. According to the following observation, ROBDD[\(\lambda_i\)] (\(i > 0\)) does not satisfy SE unless \(P = NP\), which implies that ROBDD[\(\lambda_i\)] does not satisfy \& BC (resp. \& C, \lor BC, \lor C, SFO and FO) unless \(P = NP\). Observation 2 can be proved by modifying the proof of Theorem 3.1 in [Fortune et al. 1978], since both free BDDs in the proof can be replaced by ROBDD[\(\lambda_i\)].

**Observation 1.** Given any two ROBDD[\(\lambda_i\)] (\(i > 0\)) vertices \(u\) and \(v\), the problem of deciding whether \(u \equiv v\) holds is co-NP-complete.

Given a BDD \([\lambda]\) vertex \(u\) and an assignment \(\omega\), we can get a vertex \(v'\) equivalent to \(u_\omega\) by replacing each \((x, v, v')\) appearance in \(G_u\) with \((x, v, v')\) for each \(x = false\) (true) \(\in \omega\). We can call DECOMPOSE to transform \(v'\) into ROBDD[\(\lambda_i\)] in polytime if \(u\) is in ROBDD[\(\lambda_i\)]. That is, ROBDD[\(\lambda_i\)] satisfies CD. If \(u\) is in ROBDD[\(\lambda_i\)], we use \(u_i\) to denote the ROBDD[\(\lambda_i\)] vertex which is equivalent to \(u_\omega\). Finally, the ROBDD[\(\lambda_i\)] (\(i > 0\)) \(G\) representing Eq. 1 has a linear size, but the negation of \(G\) has an exponential size. That is, ROBDD[\(\lambda_i\)] does not satisfy \neg C.

**A New Perspective About Time-Efficiency**

Due to distinct succinctness, it is sometimes insufficient to compare the time-efficiency of two canonical languages by comparing their tractability. For example, according to the tractability results mentioned previously, ROBDD[\(\lambda_i\)] does not satisfy SE (resp. SFO, \& BC and \lor BC) unless \(P = NP\). From the perspective of KC map, the applications which need the operation \(OP \in \{SE, SFO, \& BC, \lor BC\}\) will prefer to ROBDD[\(\lambda_0\)] than ROBDD[\(\lambda_i\)]. In fact, the latter is strictly more succint than the former, and also supports \(OP\) in polytime in the sizes of the equivalent formulas in the former [Lai et al. 2013]. In order to fix this “bug”, we propose an additional time-efficiency evaluation criterion tailored for canonical languages.

**Definition 6 (rapidity).** Given an operation \(OP\) and two canonical languages \(L\) and \(L'\), \(OP(L)\) is at least as rapid as \(OP(L')\) (\(L' \leq_r OP\ L\)), iff for each algorithm \(ALG'\) performing \(OP(L')\), there exists some polynomial \(p\) and some algorithm \(ALG\) performing \(OP(L)\) such that for every input \((\varphi_1, \ldots, \varphi_n, \alpha) \in OP(L)\) and its equivalent input \((\varphi'_1, \ldots, \varphi'_n, \alpha) \in OP(L')\), \(ALG(\varphi_1, \ldots, \varphi_n, \alpha)\) can be done in time \(p(t + |\varphi'_1| + \cdots + |\varphi'_n| + |\alpha|)\), where \(t\) is the running time of \(ALG'(\varphi_1, \ldots, \varphi_n, \alpha)\).

Note that the rapidity relation is reflexive and transitive. Let \(OP\) be an operation, and \(L\) and \(L'\) be two canonical languages, where \(L \leq_r OP\ L'\). Given each input \((\varphi_1, \ldots, \varphi_n, \alpha) \in OP(L)\) and its equivalent input \((\varphi'_1, \ldots, \varphi'_n, \alpha) \in OP(L')\), time cost of performing \(OP\) on \((\varphi_1, \ldots, \varphi_n, \alpha)\) increases at most polynomial times than that of performing \(OP\) on \((\varphi'_1, \ldots, \varphi'_n, \alpha)\). In particular, if \(L'\) supports \(OP\) in polytime, then \(L\) also supports \(OP\) in polytime in sizes of the equivalent formulas in \(L'\). Thus for applications needing canonical languages, we suggest that one first identify the set \(OP\) of necessary operations, second identify the set \(L\) of canonical languages meeting the tractability requirements, third add any canonical language \(L\) satisfying \(\exists L' \subseteq_l OP \in OP^L \leq_r OP\ L'\) to \(L\), and finally choose the most succinct language in \(L\). Now we present the rapidity results:

**Proposition 5.** ROBDD[\(\lambda_i\)] \(\leq_r OP\ ROBDD[\lambda_j]\) if \(i \leq j\). In particular, for \(OP \in \{CD, FO, SFO, \& C, \& BC, \lor BC, \lor C\}\), ROBDD[\(\lambda_j\)] \(\not<_{r} OP\ ROBDD[\lambda_j]\) if \(i > j\).

We emphasize an interesting observation here. It was mentioned that for \(OP \in \{SE, SFO, \& BC, \lor C\}\), \(OP(ROBDD[\lambda_0])\) can be performed in polytime but \(OP(ROBDD[\lambda_j])\) (\(i > 0\)) cannot be performed in polytime unless \(P = NP\). Therefore, if we only consider the tractability of \(OP\), it may lead to the illusion that the time efficiency of performing \(OP(ROBDD[\lambda_i])\) is pessimistically lower than that of performing \(OP(ROBDD[\lambda_0])\). In fact, Proposition 5 shows that \(OP(ROBDD[\lambda_j])\) can also be performed in polytime in the sizes of equivalent ROBDD[\(\lambda_i\)]. That is, according to our new perspective, the applications which need \(OP\) will prefer to ROBDD[\(\lambda_j\)] than ROBDD[\(\lambda_0\)].

To explain the first conclusion in Proposition 5 we first propose an algorithm called CONVERTDOWN (in Algorithms 2) to transform ROBDD[\(\lambda_i\)] into ROBDD[\(\lambda_j\)]. CONVERTDOWN terminates in polytime in the size of output, due to the facts that ROBDD[\(\lambda_i\)] \(\leq_{r} ROBDD[\lambda_j]\) and that ROBDD[\(\lambda_i\)] is canonical and satisfies CD. CONVERTDOWN, together with DECOMPOSE, provide new methods to answer query and to perform transformation on ROBDD[\(\lambda_j\)]s. First, we call CONVERTDOWN to transform ROBDD[\(\lambda_j\)]s into ROBDD[\(\lambda_i\)]s. Next, we answer query using the outputs of the first step, or perform transformation on the outputs and then transform the result into ROBDD[\(\lambda_j\)].
by DECOMPOSE. Since the time complexities of DECOMPOSE and CONVERTDOWN are polynomial in the sizes of ROBDD[∧]_{ij} s, we know ROBDD[∧]_{ij} \leq_{OP} ROBDD[∧]_{ij}.

Algorithm 2: CONVERTDOWN(u)

Input: an ROBDD[∧]_{ij} rooted at u
Output: the ROBDD[∧]_{ij} representing \( \theta(u) \), where \( i \leq j \)

1. If \( H(u) \neq nil \) then return \( H(u) \)
2. If \( u \) is a leaf then \( H(u) \leftarrow u \)
3. Else if \( u \) is a &-vertex then
   4. CONVERTDOWN(lo(v)); CONVERTDOWN(hi(v))
   5. \( H(u) \leftarrow (\text{sym}(u), H(lo(u)), H(hi(u))) \)
4. Else
   7. \( V \leftarrow \{ v \in Ch(u) : |Var(s(v))| > i \} \)
   8. If \( |V| \leq 1 \) then \( H(u) \leftarrow u \)
   9. Else
      10. Let \( v \) be \( (\text{sym}(u), V) \) and \( x \) be the least variable in \( Var(s(v)) \)
      11. \( v' \leftarrow \text{CONVERTDOWN}(x, v \mid x = false, v \mid x = true) \)
      12. \( H(u) \leftarrow (\text{sym}(u), (Ch(u) \setminus V) \cup \{v'\}) \)
13. end
14. end
15. Return \( H(u) \)

Now we turn to explain the second conclusion in Proposition 5. Due to the facts that \( L \leq_{FO} L' \) iff \( L \leq_{BC} L' \), \( L \leq_{FO} L' \) implies \( L \leq_{FO} L' \), \( L \leq_{BC} L' \) implies \( L \leq_{BC} L' \), and \( L \leq_{BC} L' \) implies \( L \leq_{BC} L' \), we just need to show the cases when \( OP \in \{CD, \land BC, \lor BC \} \).

ROBDD[∧]_{ij} \leq_{CD} ROBDD[∧]_{ij}:

Conditioning the ROBDD[∧]_{ij} representing Eq. (1) on \( \{x_{n+0-n} = true, \ldots, x_{n+k+(j+1)-n} = true\} \) will introduce an exponential (in the size of the equivalent ROBDD[∧]_{ij}) number of new vertices.

ROBDD[∧]_{ij} \leq_{BC} ROBDD[∧]_{ij}:

Consider the formulas

\[ \bigwedge_{1 \leq k \leq n} x_{k+0-n} \land (x_{k+1-n} \land \cdots \land x_{k+(j+1)-n}) \land \bigwedge_{1 \leq k \leq n} \neg x_{k+0-n} \lor (x_{k+1-n} \land \cdots \land x_{k+(j+1)-n}) \]

It is easy to design an algorithm to conjoin the ROBDD[∧]_{ij} s representing the above two formulas in linear time. However, the conjunction of two ROBDD[∧]_{ij} s representing the two formulas will generate the ROBDD[∧]_{ij} representing Eq. (1) with an exponential number of new vertices.

ROBDD[∧]_{ij} \leq_{BC} ROBDD[∧]_{ij}:

By replacing the two formulas mentioned above with the following ones: \( x_{1+0-n} \land (x_{1+1-n} \land \cdots \land x_{1+(j+1)-n}) \land \bigwedge_{2 \leq k \leq n} x_{k+0-n} \land \cdots \land x_{k+(j+1)-n} \land (x_{k+0-n} \land \cdots \land x_{k+(j+1)-n}) \), we can prove this conclusion in a similar way.

**Preliminary Experimental Results**

In this section, we report some preliminary experimental results of ROBDD[∧]_{ij} (0 \leq i \leq \infty), to verify several previous theoretical properties. In our experiments about space-efficiency, each CNF formula was first compiled into ROBDD[∧]_{ij} by the ROBDD-L_{\infty} compiler in [Lai et al. 2013] under the min-fill heuristic, then the resulting ROBDD[∧]_{ij} was transformed into ROBDD[∧]_{ij} by DECOMPOSE, and finally the resulting ROBDD[∧]_{ij} was transformed into ROBDD[∧]_{ij} by CONVERTDOWN. We also compared the conjunctive efficiency of ROBDD[∧]_{ij} with that of ROBDD[∧]_{ij}. All experiments were conducted on a computer with a two-core 2.99GHz CPU and 3.4GB RAM.

First, we compared the sizes of ROBDD[∧]_{ij} s (0 \leq i \leq 20) which represent random 3-CNF formulas over 50 variables, where each instance has 50, 100, 150 or 200 clauses. Figure 2a depicts the experimental results. Each point is the mean value obtained over 100 instances with the same parameters. The experimental results show that the size of ROBDD[∧]_{ij} is normally not smaller than that of its equivalent ROBDD[∧]_{ij+1}, which is in accordance with the previous succinctness results.

Second, we implemented two conjunctive algorithms of ROBDD and ROBDD-L_{\infty} [Lai 2013] to conjoin two ROBDD[∧]_{ij} s and ROBDD[∧]_{ij} s, respectively. Note that a single conjunction is normally performed very fast. Taking into account the fact that bottom-up compilation of a CNF formula can be viewed as just performing conjunctions, we compare the bottom-up compiling time instead. Figure 2b depicts the experimental results. Here we also used random instances with the same parameters as Figure 2a, except that the numbers of clauses vary from 10 to 250. The experimental results show that for most instances, the time efficiency of conjoining ROBDD[∧]_{ij} is outperformed that of conjoining ROBDD[∧]_{ij} s, which accords with the rapidity results.

Last, we compared the sizes of ROBDD[∧]_{ij} with those of SDD (d-DNNF) on five groups of benchmarks: emptyroom, flat100 (flat200), grid, iscas89, and sortnet. We used a state-of-the-art compiler Dsharp [Muise et al. 2012] to generate d-DNNF, and the SDD compiler in [Choi and Darwiche 2013] to generate SDD. Note that since the SDD compiler had some difficulty in compiling flat200, we used flat100 instead. Figure 2c (2d) depicts the results both the ROBDD[∧]_{ij} and SDD (d-DNNF) compilers compiled in one hour. Due to space limit and readability consideration, we remove point (9, 4) from Figure 2c, and then all other sizes are in the range of \([10^2, 10^5]\). Also, we remove points (1192941, 82944), (9, 3) and (20104922, 6422) from Figure 2d. Note that Figure 2d has more points than Figure 2c, since the compiling efficiency of SDD is normally lower than that of ROBDD[∧]_{ij} and d-DNNF. The experimental results show that the space-efficiency of ROBDD[∧]_{ij} is comparable with that of d-DNNF and SDD. Note that from the theoretical aspect, we can show that the succinctness relation between ROBDD[∧]_{ij} (i > 0) and SDD is incomparable, which accords with the experimental results.

**Related Work**

This study is closely related to three previous KC languages which also augment BDD with decomposition.

First, [Mateescu et al. 2008] proposed a relaxation of ROBDD called AND/OR Multi-value Decision Diagram by adding tree-structured \land decomposition and ranking \lor vertices on the same tree-structured order. It is easy to see that for an AND/OR BDD (i.e., AOBDD), if we remove all \land vertices with only one child, the result is an OBDD[∧]. And it is easy to show that AOBDD is strictly less succinct than ROBDD[∧]_{ij}. In addition, AOBDD is incomplete for
non-chain trees.

Second, [Lai et al. 2013] proposed a language called \textit{OBDD with implied literals} (OBDD-L) by associating each non-false vertex in OBDD with a set of implied literals, and then obtained a canonical subset called ROBDD-$L_{\infty}$ by imposing reducedness and requiring that every internal vertex has as many as possible implied literals. They designed an algorithm called L2Inf which can transform OBDD-L into ROBDD-$L_{\infty}$ in polytime in the size of input, and another algorithm called Inf2ROBDD which can transform ROBDD-$L_{\infty}$ into ROBDD in polytime in the size of output. Obviously, each non-false vertex in OBDD-L can be seen as a $\land$-vertex. Therefore, OBDD-L (ROBDD-$L_{\infty}$) is equivalent to ROBDD[$\land_i$] (ROBDD[$\land_0$]), and L2Inf and Inf2ROBDD are two special cases of \textsc{Decompose} and \textsc{ConvertDown}, respectively.

Last, [Bertacco and Damiani 1996] added the finest negatively-disjunctive-decomposition ($\downarrow$-decomposition) into ROBDD to propose a representation called \textit{Multi-Level Decomposition Diagram} (MLDD). For completeness, $\neg$-vertices are sometimes admitted. If we introduce both conjunctive and disjunctive decompositions into ROBDD, then the resulting language will be equivalent to MLDD. However, [Bertacco and Damiani 1996] paid little theoretical attention on the space-time efficiency of MLDD. On the other hand, our empirical results show that there are little disjunctive decomposition in practical benchmarks.

\textbf{Conclusions}

The main contribution of this paper is a family of canonical representations, the theoretical evaluation of their properties based some previous criteria and a new criterion, and the experimental verification of some theoretical properties. Among all languages, ROBDD[$\land_{\infty}$] has the best succinct-
ness and rapidity. It seems to be the optimal option in the application where full compilation is adopted. However, it seems very time-consuming to directly compute the finest decomposition of a knowledge base since there normally exist too many possibilities of decomposition. Therefore, in the application where partial compilation is adopted (e.g., importance sampling for model counting [Gogate and Dechter 2011] [Gogate and Dechter 2012]), one may need other languages whose decompositions are relatively easy to be captured, for example, ROBDD [\(\wedge\)] whose decompositions can be computed using SAT solver as an oracle ([Lai et al. 2013]). The second main contribution of the paper is the algorithms which perform logical operations or transform one language into another. These algorithms provide considerable potential to develop practical compilers for ROBDD [\(\wedge\)]. Intrinsically, ROBDD [\(\wedge\)] can be seen as a data structure which relax the linear orderedness of ROBDD to some extent, and thus a future direction of generalizing this work is to exploit [\(\wedge\)]-decomposition to relax the v-tree-structured order of SDD, which has the potential to identify new canonical languages with more succinctness than both ROBDD [\(\wedge\)] and SDD.

Acknowledgments
We would thank Arthur Choi for providing their 32-bit SDD package. This research is supported by the National Natural Science Foundation of China under grants 61402195, 61133011 and 61202308, and by China Postdoctoral Scientific Foundation of China under grants 61402195, 61133011 and 61202308, and by China Postdoctoral Science Foundation under grant 2014M561292.

References
[Bertacco and Damiani 1996] Valeria Bertacco and Maurizio Damiani. Boolean function representation based on disjoint-support decompositions. In Proceedings of the 14th International Conference on Computer Design (ICCD-96), VLSI in Computers and Processors, pages 27–32, 1996.
[Bryant 1986] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, 35(8):677–691, 1986.
[Cadoli and Donini 1997] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. AI Communications, 10:137–150, 1997.
[Cho and Darwiche 2013] Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI-13), pages 187–194, 2013.
[Darwiche and Marquis 2002] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. Journal of Artificial Intelligence Research, 17:229–264, 2002.
[Darwiche 2001] Adnan Darwiche. On the tractability of counting theory models and its application to truth maintenance and belief revision. Journal of Applied Non-Classical Logics, 11(1–2):11–34, 2001.
[Darwiche 2011] Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence, pages 819–826, 2011.
[Fortune et al. 1978] Steven Fortune, John E. Hopcroft, and Erik Meineche Schmidt. The complexity of equivalence and containment for free single variable program schemes. In Proceedings of the 5th Colloquium on Automata, Languages and Programming, pages 227–240, 1978.
[Gogate and Dechter 2011] Vibhav Gogate and Rina Dechter. SampleSearch: Importance sampling in presence of determinism. Artificial Intelligence, 175:694–729, 2011.
[Gogate and Dechter 2012] Vibhav Gogate and Rina Dechter. Importance sampling-based estimation over and/or search spaces for graphical models. Artificial Intelligence, 184:185:38–77, 2012.
[Gogic et al. 1995] Goran Gogic, Henry Kautz, Christos Papadimitriou, and Bart Selman. The comparative linguistics of knowledge representation. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pages 862–869, 1995.
[Lai et al. 2013] Yong Lai, Dayou Liu, and Shengsheng Wang. Reduced ordered binary decision diagram with implied literals: A new knowledge compilation approach. Knowledge and Information Systems, 35:665–712, 2013.
[Lai 2013] Yong Lai. Ordered Binary Decision Diagram with Implied Literals. PhD thesis, Jilin University, 2013.
[Mateescu et al. 2008] Robert Mateescu, Rina Dechter, and Radu Marinescu. AND/OR Multi-Valued Decision Diagrams (AOMDDs) for Graphical Models. Journal of Artificial Intelligence Research, 33:465–519, 2008.
[Muise et al. 2012] Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-DNNF compilation with sharpSAT. In Proceedings of the 25th Canadian Conference on Artificial Intelligence, pages 356–361, 2012.
[Selman and Kautz 1996] Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. Journal of the ACM, 43:193–224, 1996.