Augmenting Ordered Binary Decision Diagrams with Conjunctive Decomposition

Yong Lai$^{1,2,*}$, Dayou Liu$^{1,2}$, Minghao Yin$^{2,3}$

$^1$College of Computer Science and Technology, Jilin University, Changchun 130012, P.R. China
$^2$Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun 130012, P.R. China
$^3$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$\land_i$. By imposing reducedness and the finest $\land$-decomposition bounded by integer $i$ ($\land_i$-decomposition) on OBDD$\land_i$, we identify a family of canonical languages called ROBDD$\land_i$, where ROBDD$\land_0$ is equivalent to ROBDD. We show that the succinctness of ROBDD$\land_i$ 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$\land_i$ 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$\land_i$ is normally not larger than that of its equivalent ROBDD$\land_{i+1}$; b) conjoining two ROBDD$\land_i$s is more efficient than conjoining two ROBDD$\land_0$s in most cases, where the former is NP-hard but the latter is in P; and c) the space-efficiency of ROBDD$\land_\infty$ 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 poly-size 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 nonfalse 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$\land_i$. We then introduce a special type of $\land$-decomposition called finest $\land_i$-decomposition bounded by integer $i$ ($\land_i$-decomposition), and impose reducedness and $\land_i$-decomposition on OBDD$\land_i$ to identify a family of canonical languages called ROBDD$\land_i$. In particular, ROBDD$\land_0$ and ROBDD$\land_\infty$ are respectively equivalent to ROBDD and ROBDD-$L_\infty$. We show that the succinctness of ROBDD$\land_i$ is strictly stronger than ROBDD$\land_j$ if $i > j$. Our empirical results verify this property and also show that the space-efficiency of ROBDD$\land_\infty$ is comparable with that of deterministic Decomposable Negation Normal Form (d-DNNF, a superset of OBDD$\land_\infty$) (Darwiche 2001) and that of another canonical subset called Sentential Decision Diagram (SDD) (Darwiche 2011) in d-DNNF.

We evaluate the tractability of ROBDD$\land_i$ and show that ROBDD$\land_i$ ($i > 0$) does not satisfy SE (resp. SFO, $\land BC$ and $\lor BC$) unless $P = NP$. According to the viewpoint of KC map, the applications which need the operation $OP$ corresponding to SE (resp. SFO, $\land BC$ and $\lor BC$) will prefer to ROBDD$\land_6$ than ROBDD$\land_1$. 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{Lai2013}. 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_j] if \( i \geq j \). In particular, some operations (e.g., conjoining) on ROBDD[\land_i] are strictly more rapid than those on ROBDD[\land_j] if \( i > j \). Our empirical results verify that conjoining two ROBDD[\land_i]'s is more efficient than conjoining two ROBDD[\land_0]'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 \( \text{true}, \text{false} \) and variables using negation operator \( \neg \), conjunction operator \( \land \) and disjunction operator \( \lor \), and we denote by \( \text{Vars}(\varphi) \) the set of its variables and by \( PI(\varphi) \) the set of its prime implicates. The conditioning of \( \varphi \) on assignment \( \omega(\varphi) \) is the formula obtained by replacing each appearance of \( x \) in \( \varphi \) by \( \text{true} \) if \( x = \text{true} \) \( \in \omega \). \( \varphi \) 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 \( x \in \text{Vars}(\varphi) \). \( \varphi \) is trivial iff it depends on no variable.

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

Let \( \psi \) 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 \( x \in \text{Vars}(\varphi) \) and \( \Psi \) is a \( \land \)-decomposition of \( \varphi|_{x=\text{true}} \), we can get a strict \( \land \)-decomposition of \( \varphi \) by adding \( \neg x \times x \) 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 \( |\text{Vars}(\psi)| > i \).

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 \( \land_i \)-decomposition.

**BDD[\land] and Its Subsets**

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

**Definition 3** (BDD[\land]). A BDD[\land] 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:

\[
\varphi(v) = \begin{cases} 
\text{false}/\text{true} & \text{sym}(v) = \bot/\top; \\
\bigwedge_{w \in Ch(v)} \varphi(w) & \text{sym}(v) = \land; \\
\varphi(lo(v)) \land \varphi(hi(v)) & \text{otherwise}.
\end{cases}
\]

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

Hereafter we denote a leaf vertex by \( (\bot/\top) \), a decomposition vertex \( (\land \text{-vertex for short}) \) by \( (\land, Ch(v)) \), and a decision vertex \( (\lor \text{-vertex for short}) \) by \( (\text{sym}(v), lo(v), hi(v)) \). We abuse notation \( (\land, \{w\}) \) to denote \( w \), \( (\land, \emptyset) \) to denote \( \top \), \( \{\land, \emptyset\} \cup V \) to denote \( \land, V \), and \( \{\land, \{\bot\}\} \cup V \) to denote \( \land \). Given a BDD[\land] \( G \), \( \mathcal{G} \) denotes the size of \( G \) defined as the number of its edges. In addition, we use \( G_i \) to denote the BDD[\land] rooted at \( v \), and occasionally abuse \( v \) to denote \( \varphi(v) \). Now we define the subsets of BDD[\land]:

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

For any two OBDD[\land]'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], and analyze its canonicity and space-time efficiency. Obviously, ROBDD is equivalent to ROBDD[\land]. In addition, since a BDD vertex labelled with a set of implied literals in \( Lai et al. 2013 \) can be seen as a \( \land_i \)-vertex, it is easy to show ROBDD[\land] is equivalent to ROBDD[\land_i]. Figures \ref{fig:1a} and \ref{fig:1b} respectively depict an ROBDD[\land] and an ROBDD[\land_i] 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 \ref{fig:1c} is not an OBDD[\land] since vertex \( v \) is not bounded by one. If we extend \( \varphi \) to the following formula, the number of vertices labelled with \( x_{1+i} \) in ROBDD[\land] \((i > j) \) will equal \( 2^n \), while the number of vertices in ROBDD[\land] \((i > j) \) will be \((2j + 5)n \). That is, the size of ROBDD[\land] \((i > j) \) representing Eq. \ref{eq:1} is exponential in \( n \), while the size of the

---

\footnote{Each internal vertex in ROBDD[\land] is non-trivial.}
corresponding ROBDD[∧_2] is only linear in n.
\[ x_{k+0\cdot n} \leftrightarrow \cdots \leftrightarrow x_{k+(j+1)\cdot n} \]  

(1)

Figure 1: An ROBDD[∧_1] (a) and an ROBDD[∧_2] (b)

We close this section by pointing out that ROBDD[∧_2] is canonical and complete. The canonicity is immediately from the uniqueness of ∧_2-decomposition. The completeness is also easily understood, since we can transform ROBDD into ROBDD[∧_2] (see the next section).

**Proposition 3.** Given a formula, there is exactly one ROBDD[∧_2] to represent it.

**Space-Efficiency Analysis**

We analyze the space-efficiency in terms of succinctness. The succinctness results is given as follows, where L = \{∧, \{sym(v), (⊥), \{⊥\}, hi(u)\} \equiv u'. Then we call FINEST(v) to get an equivalent ROBDD[∧_2] vertex.

- **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 sym(v_2) = \∧. Without loss of generality, assume v_1 = lo(u'). Let v = \{sym(u'), (⊥), \{⊥\}, hi(u')\} \equiv u'. If v < |Vars(v_1)| and i < |Vars(v)|, then u' is already an ROBDD[∧_1] vertex. Otherwise, u' is an ROBDD[∧_2] vertex.

- **EXTRACTSHARE(u')**: This function handles the case where lo(u') and hi(u') share some children. That is, sym(lo(u')) = sym(hi(u')) = \∧ 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') \setminus Vars(V)| > i, we remove v from V. If V = \emptyset, u' is already an ROBDD[∧_2] vertex. Otherwise, let v = \{sym(u'), lo(u') \setminus V, hi(u') \setminus V\}, and then \{\∧, V \cup \{v\}\} is an ROBDD[∧_2] vertex equivalent to u'.

**Algorithm 1: DECOMPOSE(u)**

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if H(u) \neq \emptyset then return H(u)</td>
</tr>
<tr>
<td>2</td>
<td>if u is a leaf vertex then u' \leftarrow u</td>
</tr>
<tr>
<td>3</td>
<td>else</td>
</tr>
<tr>
<td>4</td>
<td>u' \leftarrow {DECOMPOSE(v) : v \in Ch(u')}</td>
</tr>
<tr>
<td>5</td>
<td>if u' is a o-vertex then</td>
</tr>
<tr>
<td>6</td>
<td>if (\⊥) \in Ch(u') and</td>
</tr>
<tr>
<td>7</td>
<td>u' \leftarrow EXTRACTLEAF(u')</td>
</tr>
<tr>
<td>8</td>
<td>else if one child of u is a part of the other then</td>
</tr>
<tr>
<td>9</td>
<td>u' \leftarrow EXTRACTPART(u')</td>
</tr>
<tr>
<td>10</td>
<td>else if the children of u share some children then</td>
</tr>
<tr>
<td>11</td>
<td>u' \leftarrow EXTRACTSHARE(u')</td>
</tr>
<tr>
<td>12</td>
<td>else if lo(u') = hi(u') then u' \leftarrow lo(u')</td>
</tr>
<tr>
<td>13</td>
<td>else u' \leftarrow FINEST(u)</td>
</tr>
<tr>
<td>14</td>
<td>end</td>
</tr>
<tr>
<td>15</td>
<td>if a previous vertex u'' identical with u' appears then H(u) \leftarrow u''</td>
</tr>
<tr>
<td>16</td>
<td>else H(u) \leftarrow u'</td>
</tr>
<tr>
<td>17</td>
<td>return H(u)</td>
</tr>
</tbody>
</table>

Second, we show the direction from left to right by counterexample: If i > j, Eq. (1) can be represented by an ROBDD[∧_1] in linear size, but the size of the equivalent ROBDD[∧_2] is exponential in n.

**Time-Efficiency Analysis**

We analyze the time-efficiency of operating ROBDD[∧_2] 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 \( \Delta_p \times \Delta_s \) and \( \Gamma \), where \( \Delta_p \) denotes the primary information of \( OP \) which is a set of sequences of formulas, \( \Delta_s \) denotes the supplementary information customized for \( OP \), and \( \Gamma \) is the set of outputs of \( OP \). \( OP \) on language \( \Lambda \), denoted by \( OP(\Lambda) \), is the subset \( \{(\varphi_1, \ldots, \varphi_n, \alpha, \beta) \in \Lambda : \varphi_i \in \Lambda \text{ for } 1 \leq i \leq n \text{ and } \beta \in \Lambda \text{ if it is a formula}\} \).
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, ∨BC, ∨C and ¬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 polynomial 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.

<table>
<thead>
<tr>
<th>(L)</th>
<th>CO</th>
<th>VA</th>
<th>CE</th>
<th>IM</th>
<th>EQ</th>
<th>SE</th>
<th>CT</th>
<th>ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROBDD[(\land_0)]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>ROBDD[(\land_i)]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>ROBDD[(\land_0)]</td>
<td>CD</td>
<td>FO</td>
<td>SFO</td>
<td>∧C</td>
<td>∧BC</td>
<td>∨C</td>
<td>∨BC</td>
<td>¬C</td>
</tr>
<tr>
<td>ROBDD[(\land_i)]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

Since ROBDD[\(\land_i\)] 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[\(\land_i\)] \((i > 0)\) does not satisfy SE unless \(P = NP\), which implies that ROBDD[\(\land_i\)] does not satisfy ∧BC (resp. ∧C, ∨BC, ∨C, SFO and FO) unless \(P = NP\). Observation 1 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[\(\land_i\)].

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

Given a BDD \(\land\) vertex \(u\) and an assignment \(\alpha\), we can get a vertex \(v'\) equivalent to \(u|_\alpha\) by replacing each \((x, v, v')\) appearance in \(\land\) with \((x, v, v')\) \(\land (x, v, v')\) for each \(x \equiv false (true) \in \alpha\). We can call DECOMPOSE to transform \(v'\) into ROBDD[\(\land\)] in polytime if \(v\) is in ROBDD[\(\land\)]. That is, ROBDD[\(\land\)] satisfies CD. If \(u\) is in ROBDD[\(\land\)], we use \(u|_\alpha\) to denote the ROBDD[\(\land\)] vertex which is equivalent to \(u|_\alpha\). Finally, the ROBDD[\(\land_i\)] \((i > 0)\) \(\land\) representing Eq. 1 has a linear size, but the negation of \(\land\) has an exponential size. That is, ROBDD[\(\land_i\)] does not satisfy ¬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[\(\land_0\)] does not satisfy SE (resp. SFO, ∧BC and ∨BC) unless \(P = NP\). From the perspective of KC map, the applications which need the operation \(OP \in \{SE, SFO, ∧BC, ∨BC\}\) will prefer to ROBDD[\(\land_0\)] than ROBDD[\(\land_i\)]. 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 [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 \(OP(L')\) if \(L \leq_{OP} L'\), 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 \(L \leq_{OP} L'\). 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 \(\exists L' \in L\forall OP \in OP.L \leq_{OP} L'\) to \(L\), and finally choose the most succinct language in \(L\). Now we present the rapidity results:

**Proposition 5.** ROBDD[\(\land_i\)] \(\leq_{OP} ROBDD[\land_j]\) if \(i \leq j\). In particular, for \(OP \in \{CD, FO, SFO, ∧C, ∧BC, ∨BC, ∨C\}\), ROBDD[\(\land_j\)] \(\leq_{OP} ROBDD[\land_j]\) if \(i > j\).

We emphasize an interesting observation here. It was mentioned that for \(OP \in \{SE, SFO, ∧BC, ∨BC\}\), \(OP(ROBDD[\land_0])\) can be performed in polytime but \(OP(ROBDD[\land_0])\) \((i > 0)\) 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 \(OP(ROBDD[\land_0])\) is pessimistically lower than that of performing \(OP(ROBDD[\land_0])\). In fact, Proposition 5 shows that \(OP(ROBDD[\land_0])\) can also be performed in polytime in the sizes of equivalent ROBDD[\(\land_0\)]. That is, according to our new perspective, the applications which need \(OP\) will prefer to ROBDD[\(\land_j\)] than ROBDD[\(\land_0\)]

To explain the first conclusion in Proposition 5 we first propose an algorithm called CONVERTDOWN (in Algorithms 2) to transform ROBDD[\(\land_j\)] into ROBDD[\(\land_j\)]. CONVERTDOWN terminates in polytime in the size of output, due to the facts that ROBDD[\(\land_i\)] \(\leq_{OP} ROBDD[\land_j]\) and that ROBDD[\(\land_j\)] is canonical and satisfies CD. CONVERTDOWN, together with DECOMPOSE, provide new methods to answer query and to perform transformation on ROBDD[\(\land_j\)]. First, we call CONVERTDOWN to transform ROBDD[\(\land_j\)] into ROBDD[\(\land_j\)]. Next, we answer query using the outputs of the first step, or perform transformation on the outputs and then transform the result into ROBDD[\(\land_j\)].


by DECOMPOSE. Since the time complexities of DECOMPOSE AND CONVERTDOWN are polynomial in the sizes of ROBDD[\Lambda_i]S, we know ROBDD[\Lambda_i] \leq_{OP} ROBDD[\Lambda_j].

Algorithm 2: CONVERTDOWN(u)

Input: an ROBDD[\Lambda_i] rooted at u
Output: the ROBDD[\Lambda_j] 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 \lor-vertex then
   4. CONVERTDOWN(lo(v)); CONVERTDOWN(hi(v))
5. H(u) \leftarrow (\sum(u), H(lo(u)), H(hi(u)))
6. end

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_{BC} L', L \leq_{IFCD} L' implies L \leq_{FO} L', and L \leq_{BC} L' implies L \leq_{IFCD} L', just need to show the cases when \Omega \in \{CD, BC, \lor\}\}. The experimental results show that the size of ROBDD[\Lambda_i] is normally not smaller than that of its equivalent ROBDD[\Lambda_j], which is in accordance with the previous succinctness results.

Second, we implemented two conjunction algorithms of ROBDD and ROBDD-L_\infty (Lai 2013) to conjure two ROBDD[\Lambda_i]S and ROBDD[\Lambda_j]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 2 depicts the experimental results. Here we also used random instances with the same parameters as Figure 2, except that the numbers of clauses vary from 10 to 250. The experimental results show that for most instances, the time efficiency of conjuring ROBDD[\Lambda_i]S is outperformed that of conjuring ROBDD[\Lambda_j]S, which accords with the rapidity results.

Last, we compared the sizes of ROBDD[\Lambda_i] with those of SDD (d-DNNF) on five groups of benchmarks: empty-room, flat100 (flat200), grid, iscas89, and sortnet. We used a state-of-the-art compiler Dsharp (Muide 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 has some difficulty in compiling flat200, we used flat100 instead. Figure 2(2h) depicts the results both the ROBDD[\Lambda_i] and SDD (d-DNNF) compilers compiled in one hour. Due to space limit and readability consideration, we remove point (9.4) from Figure 2, and then all other sizes are in the range of [10^5, 10^6]. Also, we remove points (1192941, 82944), (9.3) and (2010492, 6422) from Figure 2. Note that Figure 2(4) has more points than Figure 2(b), since the compiling efficiency of SDD is normally lower than that of ROBDD[\Lambda_i] and d-DNNF. The experimental results show that the space-efficiency of ROBDD[\Lambda_i] is comparable with that of d-DNNF and SDD. Note that from the theoretical aspect, we can show that the succinctness relation between ROBDD[\Lambda_i] (i > 0) and SDD is incomparable, which accords with the experimental results.

Preliminary Experimental Results

In this section, we report some preliminary experimental results of ROBDD[\Lambda_i] (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[\Lambda_i] by the ROBDD-L_\infty compiler in Lai et al. 2013 under the min-fill heuristic, then the resulting ROBDD[\Lambda_i] was transformed into ROBDD[\Lambda_j] by DECOMPOSE, and finally the resulting ROBDD[\Lambda_j] was transformed into ROBDD[\Lambda_j] by CONVERTDOWN. We also compared the conjunctive efficiency of ROBDD[\Lambda_j] with that of ROBDD[\Lambda_j]. 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[\Lambda_i]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 2 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[\Lambda_i] is normally not smaller than that of its equivalent ROBDD[\Lambda_{i+1}], which is in accordance with the previous succinctness 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 AND-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[\Lambda_i]. And it is easy to show that AOBDD is strictly less succinct than ROBDD[\Lambda_j]. In addition, AOBDD is incomplete for
non-chain trees.

Second, (Lai et al. 2013) proposed a language called 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 $\land_1$-vertex. Therefore, OBDD-L (ROBDD-$L_{\infty}$) is equivalent to ROBDD[$\land_1$] (ROBDD[$\land_1$]), and L2Inf and Inf2ROBDD are two special cases of DECOMPOSE and CONVERTDOWN, respectively.

Last, (Bertacco and Damiani 1996) added the finest negatively-disjunctive-decomposition ($\downarrow$-decomposition) into ROBDD to propose a representation called 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.

**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 \cite{Gogate and Dechter 2011, Gogate and Dechter 2012}), one may need other languages whose decompositions are relatively easy to be captured, for example, ROBDD[∧] whose decompositions can be computed using SAT solver as an oracle \cite{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[∧]. Intrinsic, ROBDD[∧] 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 ∧-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[∧] 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 Science Foundation under grant 2014M561292.

References


