Journal of Artificial Intelligence Research 58 (2017) 453-521 Submitted 7/16; published 3/17

New Canonical Representations by Augmenting OBDDs with Conjunctive Decomposition

Yong Lai LAIY@JLU.EDU.CN
College of Computer Science and Technology,
Jilin University, Changchun, 130012, P.R. China

Dayou Liu DYLIU@JLU.EDU.CN
Key Laboratory of Symbolic Computation and Knowledge Engineering
of Ministry of Education, Changchun, 130012, P.R. China

Minghao Yin YMH@NENU.EDU.CN
College of Computer Science and Information Technology,
Northeast Normal University, Changchun, 130117, P. R. China

Abstract

We identify two families of canonical knowledge compilation languages. Both families augment ROBDD with conjunctive decomposition bounded by an integer i ranging from 0 to ∞. In the former, the decomposition is finest and the decision respects a chain $\mathcal{C}$ of variables, while both the decomposition and decision of the latter respect a tree $\mathcal{T}$ of variables. In particular, these two families cover three existing languages ROBDD, ROBDD with as many implied literals as possible, and AND/OR BDD. We demonstrate that each language in the first family is complete, while each one in the second family is incomplete with expressivity that does not decrease with incremental i. We also demonstrate that the succinctness does not decrease from the i-th language in the second family to the i-th language in the first family, and then to the $(i+1)$-th language in the first family. For the operating efficiency, on the one hand, we show that the two families of languages support a rich class of tractable logical operations, and particularly the tractability of each language in the second family is not less than that of ROBDD; and on the other hand, we introduce a new time efficiency criterion called rapidity which reflects the idea that exponential operations may be preferable if the language can be exponentially more succinct, and we demonstrate that the rapidity of each operation does not decrease from the i-th language in the second family to the i-th language in the first family, and then to the $(i+1)$-th language in the first family. Furthermore, we develop a compiler for the last language in the first family ($i = \infty$). Empirical results show that the compiler significantly advances the compiling efficiency of canonical representations. In fact, its compiling efficiency is comparable with that of the state-of-the-art compilers of non-canonical representations. We also provide a compiler for the i-th language in the first family by translating the last language in the first family into the i-th language ($i < \infty$). Empirical results show that we can sometimes use the i-th language instead of the last language without any obvious loss of space efficiency.

1. Introduction

Knowledge compilation (KC) is a key approach for dealing with the computational intractability in propositional reasoning (Selman & Kautz, 1996; Darwiche & Marquis, 2002; Cadoli & Donini, 1997). From a theoretical perspective, a core task in the KC field is to identify target languages and evaluate them according to their properties. This paper focuses on four key properties: the expressivity of the language, the canonicity of the results of compiling knowledge bases into the language, the space efficiency of storing the compiled results, and the time efficiency of operating on the 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 a given 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. From a practical perspective, the existence of an efficient compiler is required in order for a language to be widely applied, and developing efficient compilers is therefore another key task of the KC researchers (Darwiche, 2014).

Canonicity, an important property of KC languages, provides equivalence tests with constant time complexity and plays a critical role in the performance of compiling methods (Darwiche, 2011; Van den Broeck & Darwiche, 2015). The reduced ordered binary decision diagram (ROBDD) (Bryant, 1986) over a fixed variable order is one of the most influential canonical languages in the KC literature. The current success of ROBDD owes much to two main advantages. First, ROBDD is one of the most tractable target languages that supports in polytime all query operations and many transformation operations mentioned in the KC map. Second, researchers have developed many efficient ROBDD compilers (see e.g., Somenzi, 2002; Lind-Nielsen, 1996; Huang & Darwiche, 2004; Lv, Su, & Xu, 2013) that facilitate the employment from practical applications.

Despite its current success, ROBDD has a well-known weakness in succinctness, which reflects the explosion in size for many types of knowledge bases in practice. Deterministic, decomposable negation normal form (d-DNNF) (Darwiche, 2001b) is a strict superset of ROBDD that has proven influential in probabilistic reasoning applications (Chavira & Darwiche, 2008). In particular, most efficient compilers (e.g., OBDD compilers) can be seen as special d-DNNF compilers (Huang & Darwiche, 2007). Although d-DNNF is strictly more succinct than ROBDD, it has no canonicity and less tractability. A recent trend in the KC field is to identify new canonical representations in d-DNNF that can mitigate the size explosion problem of ROBDD without sacrificing its main advantages. Mateescu, Dechter, and Marinescu (2008) identified a canonical representation called AND/OR BDD (AOBDD). The sizes of ROBDDs are exponential in pathwidth, while AOBDD has a tighter bound on size in that it is only exponential in treewidth [1]. However, AOBDD is incomplete for a pseudo tree that is not a chain. Darwiche (2011) proposed a generalization of ROBDD called sentential decision diagram (SDD). Its complete subset compressed and trimmed SDD over a fixed vtree $\mathcal{V}$ is canonical (denoted by CSDD$_\mathcal{V}$), and has a tighter size bound that is also exponential in treewidth. Compared with ROBDD, CSDD$_\mathcal{V}$ has less tractability. Lai, Liu, and Wang (2013) proposed a language called OBDD with implied literals (OBDD-L) by associating some implied literals with each non-false vertex in OBDD. OBDD-L has a complete canonical subset called ROBDD with as many implied literals as possible (ROBDD-L_∞), which maintains the two advantages of ROBDD. First, given each operation ROBDD supports in polytime, ROBDD-L_∞ also supports it in polytime in the

1. Given a CNF formula, its pathwidth pw and treewidth w are related by $pw = O(w \log n)$ (see e.g., Bodlaender, 1998).

sizes of the equivalent ROBDDs. Second, ROBDD-L_∞ has a more efficient compiler for ROBDD-L_∞ than those that exist for ROBDD.

Although the current research on canonical representations has done much to mitigate the size explosion problem of ROBDD from both theoretical and practical perspectives, the corresponding compilers cannot yet compile many problems that the state-of-the-art d-DNNF compiler Dsharp can compile (Muise et al., 2012). Furthermore, the relationships between canonical representations, which are indispensable for choosing appropriate representations in practical applications, are not well studied. On the one hand, this paper goes further in mitigating the size explosion problem of ROBDD from both theoretical and practical aspects, and we identify new canonical representations that maintain the main advantages of ROBDD. On the other hand, this paper provides a unified perspective to compare the new representations with some existing canonical languages based on their theoretical properties, especially considering that the tractability of AOBDD is not well studied [2].

Decomposability is an important factor behind the strong succinctness and tractability of d-DNNF. The ideas of ROBDD-L_∞ and AOBDD are to use some special types of conjunctive decomposability to relax the linear variable ordering of ROBDD, where each $\wedge$-decomposition in ROBDD-L_∞ has at most one sub-formula with more than one variable (bounded by one), and each $\wedge$-decomposition in AOBDD respects some tree of variables. We generalize these two types of $\wedge$-decompositions to propose bounded $\wedge$-decomposability parameterized by integer i ($\wedge_i$-decomposition), and $\wedge_i$-decomposition respecting tree $\mathcal{T}$ ($\wedge_{\mathcal{T},i}$-decomposition). We observe that the finest $\wedge_i$-decomposition and $\wedge_{\mathcal{T},i}$-decomposition are unique. Then we generalize OBDD by introducing $\wedge$-decomposition, and call the resulting language OBDD[$\wedge$]. We identify a family of canonical languages in OBDD[$\wedge$] called ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ by imposing three constraints: reducedness, the finest $\wedge_i$-decomposability, and ordered decision respecting a chain $\mathcal{C}$. We identify another family of canonical languages called ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ by imposing reducedness, the finest $\wedge_{\mathcal{T},i}$-decomposability, and ordered decision respecting $\mathcal{T}$. Previous research experience in the KC community has demonstrated that reducedness and orderedness are indispensable for canonicity, while our results show that bounded $\wedge$-decomposability leads to representations with different space-time efficiency. We demonstrate that these two families of languages cover the three previous languages ROBDD, AOBDD and ROBDD-L_∞, as depicted in Figure 1.

We evaluate the theoretical properties of the two families of canonical languages from four aspects, and the obtained results corresponding to the structure in Figure 1 provide an important complement to the current KC map:

(a) We analyze the expressivity and demonstrate that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ is complete while ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ is incomplete. We also demonstrate that if $i \leq j$, ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ is a subset of ROBDD$[\wedge_{\widehat{\mathcal{T},j}}]_{\mathcal{T}}$ (resp. CSDD$_{\mathcal{V}}$); and thus the former is not more expressive than the latter.

(b) We analyze the succinctness and demonstrate that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ (resp. MODS) is strictly less succinct than ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ if $i < j$. We also demonstrate that ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$

2. Mateescu, Dechter, and Marinescu (2008), and Fargier and Marquis (2006) have obtained some tractability results of AOBDD, but the results about **CD**, **FO**, **SFO**, $\wedge$**C**, $\vee$**C**, $\vee$**BC** and $\neg$**C** are not yet studied. Moreover, our results in this paper show that AOBDD has more tractability when imposing some restriction on the tree-structured order of variables.

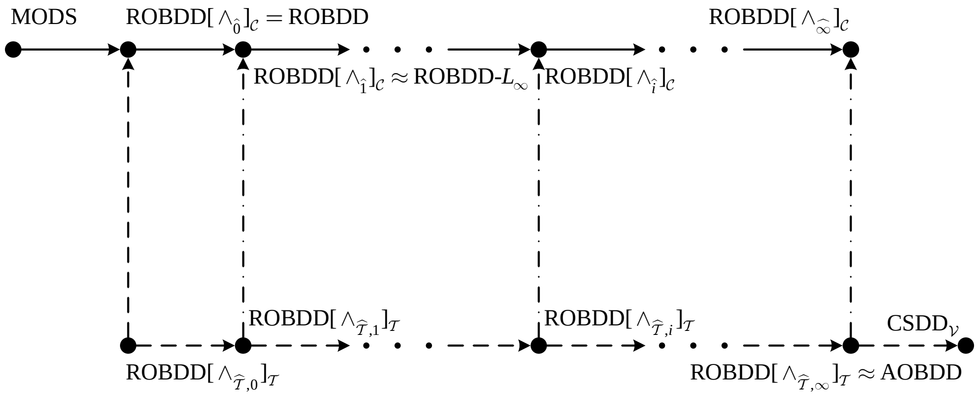

MODS
ROBDD[$\wedge_{\hat{0}}$]$_\mathcal{C}$ = ROBDD
ROBDD[$\wedge_{\hat{\infty}}$]$_\mathcal{C}$
ROBDD[$\wedge_{\hat{1}}$]$_\mathcal{C}$ ≈ ROBDD-L_∞
ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$
ROBDD[$\wedge_{\widehat{\mathcal{T}},1}$]$_\mathcal{T}$
ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_\mathcal{T}$
CSDD$_\mathcal{V}$
ROBDD[$\wedge_{\widehat{\mathcal{T}},0}$]$_\mathcal{T}$
ROBDD[$\wedge_{\widehat{\mathcal{T}},\infty}$]$_\mathcal{T}$ ≈ AOBDD

Figure 1: The relationship between many canonical representations in d-DNNF, where each formula in MODS is a set of models, $\mathcal{T}$ is not a chain, $\mathcal{C}$ is a topological order of $\mathcal{T}$, and $\mathcal{V}$ is the vtree corresponding to $\mathcal{T}$

is strictly less succinct than ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ if $i > 0$ and $\mathcal{T}$ has an infinite depth, and ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_\mathcal{T}$ has the same succinctness as ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ otherwise.

(c) We analyze the time efficiency in terms of tractability. We demonstrate that these two families of languages support a rich class of tractable logical operations. First, these two families of representations qualify as KC languages because they support polytime sentential entailment. Second, ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_\mathcal{T}$ has the same tractability as ROBDD; in particular, ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_\mathcal{T}$ even has more tractability than ROBDD when $\mathcal{T}$ has a finite depth. Third, ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ ($i \geq 2$) maintains the same tractability as ROBDD-L_∞, which is more tractable than d-DNNF.

(d) We also analyze the time efficiency in terms of a new notion called rapidity which reflects an increase of at most polynomial multiples of time cost of an operation. We demonstrate that each ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ (resp. ROBDD[$\wedge_{\widehat{\mathcal{T}},j}$]$_\mathcal{T}$ and MODS) and the equivalent ROBDD[$\wedge_{\hat{j}}$]$_\mathcal{C}$ ($i \leq j$) can be transformed into each other in polytime in the size of ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ (resp. ROBDD[$\wedge_{\widehat{\mathcal{T}},j}$]$_\mathcal{T}$ and MODS). Then we prove that each operation on ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ (resp. ROBDD[$\wedge_{\widehat{\mathcal{T}},j}$]$_\mathcal{T}$ and MODS) is at most as rapid as the same operation on ROBDD[$\wedge_{\hat{j}}$]$_\mathcal{C}$. Moreover, we prove that each operation on ROBDD[$\wedge_{\widehat{\mathcal{T}},j}$]$_\mathcal{T}$ is as rapid as the operation on ROBDD[$\wedge_{\hat{j}}$]$_\mathcal{C}$ if $\mathcal{T}$ has a finite depth.

The results summarized above allow us to make the following statements, which reflect the importance of new canonical representations from a theoretical perspective. First, compared with ROBDD-L_∞, ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ ($i \geq 2$) can further mitigate the size explosion problem without sacrificing the main advantages of ROBDD. Second, given each operation ROBDD (resp. ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_\mathcal{T}$) supports in polytime, ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ also supports it in polytime in the sizes of the equivalent ROBDDs (resp. ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_\mathcal{T}$s); and thus ROBDD (resp. ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_\mathcal{T}$) can serve as an intermediary to perform operations on ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$, since the former is more tractable. Third, ROBDD[$\wedge_{\hat{\infty}}$]$_\mathcal{C}$ is a complete language with the best succinctness and operation rapidity among the two families of languages. Fourth, for a tree with finite depth, ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_\mathcal{T}$ is an incomplete language with the best succinctness, tractability and operation rapidity among the two families of languages.

For practical purposes, we developed an ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler with high efficiency. Preliminary experimental results indicate that compared with the state-of-the-art compilers of ROBDD-L_∞ and CSDD$_{\mathcal{V}}$, our compiler has at least an order of magnitude improvement in compiling time for more than 40% of instances. Moreover, ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$, ROBDD-$L_\infty$ and SDD are three canonical subsets of the non-canonical language d-DNNF, and only our compiler is comparable to the state-of-the-art d-DNNF compiler Dsharp (Muise et al., 2012) in both compiling time and resulting sizes (to our knowledge, Dsharp also dominates the compilers of other languages). However, compared with d-DNNF, ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ can support relatively efficient operations; in particular, it has more querying tractability. Moreover, we show that ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ can be efficiently transformed into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. In particular, the methods that first compile knowledge bases into ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ and then convert the results into ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$ provide more efficient compilers for ROBDD-L_∞ and ROBDD, respectively, than the existing ones. We also compare the space efficiency of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ with that of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ from an experimental angle. Our results reveal that we can use ROBDD$[\wedge_{\widehat{2}}]_{\mathcal{C}}$ or ROBDD$[\wedge_{\widehat{3}}]_{\mathcal{C}}$ instead of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$, in some types of applications without obvious loss of space efficiency.

The remainder of this paper is organized as follows. Section 2 provides some technical and notational preliminaries. Section 3 introduces several types of $\wedge$-decompositions. Section 4 defines OBDD$[\wedge]$ and its subsets, and then analyze the relationships between previous languages and subsets of OBDD$[\wedge]$. Sections 5–6 analyze the canonicity, expressivity and succinctness of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$. Section 7 presents a set of tractable operation algorithms, evaluates the tractability of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$, introduces the notion of rapidity, and describes rapidity results for ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$. Section 8 reports the preliminary experimental results. Finally, Section 9 presents related work and Section 10 closes with some concluding remarks. All proofs appear in an appendix.

2. Preliminaries

This paper uses x to denote a propositional or Boolean variable, $PV = \{x_1, \ldots, x_n, \ldots\}$ to denote a countably infinite set of variables, and X to denote a subset of PV. In order to simplify notations, we sometimes assimilate a singleton set with its unique element. A formula is constructed from constants $true$, $false$ and variables in PV using negation operator $\neg$ and conjunction operator $\wedge$. Given a formula φ, we use $Vars(\varphi)$ to denote the set of variables appearing in φ.

An assignment ω over variable set X is a mapping from X to $\{true, false\}$, and the set of all assignments over X is denoted by 2^X. Given any formula φ and assignment ω over a superset of $Vars(\varphi)$, ω satisfies φ (denoted by $\omega \models \varphi$) iff one of the following conditions holds: $\varphi = true$; $\varphi = x$ and $\omega(x) = true$; $\varphi = \neg\psi$ and $\omega \not\models \psi$; or $\varphi = \psi \wedge \psi'$, $\omega \models \psi$ and $\omega \models \psi'$. A model of φ is an assignment over $Vars(\varphi)$ that satisfies φ. The set of models of φ is denoted by $\Omega(\varphi)$. A formula is satisfiable or consistent if it has at least one model, and it is unsatisfiable or inconsistent otherwise. A formula over X is valid if every $\omega \in 2^X$ satisfies it. Given two formulas φ and ψ, φ entails ψ (denoted by $\varphi \models \psi$) iff for each model ω over $Vars(\varphi) \cup Vars(\psi)$, $\omega \models \varphi$ implies $\omega \models \psi$; φ is equivalent to ψ (denoted by $\varphi \equiv \psi$) iff φ and ψ imply each other; and φ and ψ are distinct iff $\varphi \not\equiv \psi$.

Given a formula φ and a variable $x \in Vars(\varphi)$, we say that φ *depends* on x if $\varphi \wedge \neg x \not\equiv \varphi \wedge x$, where x is called an *essential* variable. We say that a formula without any essential variable is *trivial*, and a formula with some inessential variables is *redundant*. Given any formula φ and the set X of essential variables, we can obtain an equivalent formula without any inessential variable by replacing each variable in $Vars(\varphi) \setminus X$ with *true*, and we denote the resulting formula by $\lfloor \varphi \rfloor$.

2.1 Variable Orders

In this paper, we assume that PV is associated with some strict (partial) order $\prec$ which is well-founded and has a dummy least element x_0. That is, x_0 does not appear in any formula and is less than all other variables over $\prec$; and every nonempty variable set X has a greatest lower bound $glb_{\prec}(X)$, where $\prec$ can be omitted in an explicit context. In particular, we assume each variable x satisfies $x \prec glb(\emptyset)$. Hereafter we abbreviate $glb(Vars(\varphi))$ as $glb(\varphi)$, and write $x \prec X$ for $\forall x' \in X. x \prec x'$.

We mainly focus on the tree-structured orders which are defined as the ancestor-descendant relationships on trees of variables. Hereafter we use $V(\mathcal{T})$ and $V(\mathcal{G})$ to denote the respective sets of vertices in a tree $\mathcal{T}$ and a graph $\mathcal{G}$. Given a tree $\mathcal{T}$ over variables, we denote its depth by $dep(\mathcal{T})$, and the corresponding tree-structured order by $\prec_{\mathcal{T}}$. For a variable $x \in V(\mathcal{T})$, we denote its children in $\mathcal{T}$ by $Ch_{\mathcal{T}}(x)$, and the subtree rooted at x by $\mathcal{T}_x$. Hereafter we assume that for each set X of two variables, $glb(X)$ is already known. Therefore, for each set X of variables, $glb(X)$ can be computed in $O(|X|)$. Given two variables $x \prec_{\mathcal{T}} x'$, there exists exactly one path from x to x' (denoted by $x \rightsquigarrow x'$), and we denote the child variable of x on this path by $ch_{\mathcal{T}}(x \rightsquigarrow x')$. We also assume that for each pair x and x', $ch_{\mathcal{T}}(x \rightsquigarrow x')$ is already known. Figure 2a–2c depicts three trees $\mathcal{C}$, $\mathcal{T}$ and $\mathcal{T}'$ over PV, respectively. Clearly, $\mathcal{C}$ is a chain with $\prec_{\mathcal{T}} \subset \prec_{\mathcal{C}}$ and $\prec_{\mathcal{T}'} \subset \prec_{\mathcal{C}}$, $glb_{\prec_{\mathcal{T}}}(\{x_2, x_3\}) = x_1$, and $ch_{\mathcal{T}'}(x_1 \rightsquigarrow x_4) = x_2$.

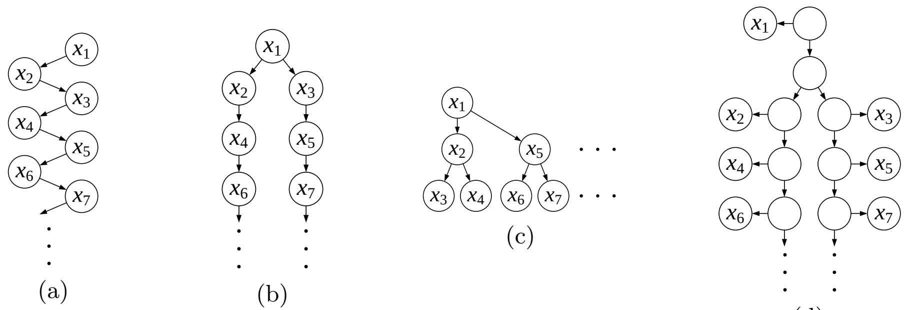


Figure 2: Three trees and a vtree over PV, where (a) is a chain and (d) is the vtree

Vtree is another way of organizing variables, which is adopted in some related work (see e.g., Darwiche, 2011). Each vtree is a full binary tree whose leaves are in one-to-one correspondence with the variables. It is obvious that each tree over variable set can be transformed into a vtree. For example, the tree in Figure 2b can be transformed into the vtree in Figure 2d.

2.2 Other Logical Operations

It is well known that $\neg$ and $\wedge$ are complete for any propositional theory. Here we introduce some other logical operations that are defined using $\neg$ and $\wedge$. Note that the first two operators, as well as $\wedge$, can easily extend to multi-parameter cases.

- Disjunction operator: $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$.
- Equality operator: $\varphi \leftrightarrow \psi = (\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi)$.
- Decision operator: $\varphi \diamond_x \psi = (\neg x \wedge \varphi) \vee (x \wedge \psi)$.
- Conditioning operator: The conditioning of formula φ on assignment ω, denoted by $\varphi|_\omega$, is a formula obtained by replacing each x in φ with $true$ (resp. $false$) if $x = true \in \omega$ (resp. $x = false \in \omega$).
- Forgetting operator: Let φ be a formula and X be a variable set. The forgetting of X from φ, denoted by $\exists X.\varphi$, is a formula that does not mention any variable in X and for every formula ψ that does not mention any variable in X, we have $\varphi \models \psi$ precisely when $\exists X.\varphi \models \psi$.

2.3 Specific Types of Formulas

We next present some specific types of formulas used as background knowledge or related work in the rest of the paper:

- A *literal* is either a variable x or its negation $\neg x$. Given a literal l, its negation $\neg l$ is $\neg x$ if l is x, and $\neg l$ is x otherwise. A *clause* δ (resp. *term* γ) is a set of literals representing their disjunction (resp. conjunction).
- A formula in *conjunctive normal form* (CNF) is a set of clauses representing their conjunction, and a formula in *disjunctive normal form* (DNF) is a set of terms representing their disjunction. MODS is a subset of DNF where each disjunct of every formula φ is a model of φ. Hereafter, we assume that each variable in MODS is essential, and MODS is therefore canonical.
- An *implicate* (resp. *implicant*) of a formula φ is an invalid clause δ (resp. a consistent term γ) satisfying $\varphi \models \delta$ (resp. $\gamma \models \varphi$). Among all the implicates (resp. implicants) of φ, the *prime* implicates (resp. implicants) are the minimal sets, and we use $PI(\varphi)$ (resp. $IP(\varphi)$) to denote the set of prime implicates (resp. implicants). We say that a partition $\{\Psi_1, \cdots, \Psi_m\}$ of $PI(\varphi)$ (resp. $IP(\varphi)$) is disjoint if $Vars(\Psi_i) \cap Vars(\Psi_j) = \emptyset$ for $1 \leq i \neq j \leq m$.
- A *binary decision diagram* (BDD) (Bryant, 1986) is a rooted DAG. Each internal vertex v has two children that are called *low* child $lo(v)$ and *high* child $hi(v)$, and connected by dashed and solid arcs, respectively. Each vertex v is labeled with a symbol $sym(v)$. If v is a leaf, $sym(v) = \bot$ or $\top$, and v represents $false$ or $true$. Otherwise, $sym(v)$ is a variable, and v represents a formula $\vartheta(v) = \vartheta(lo(v)) \diamond_{sym(v)} \vartheta(hi(v))$. A BDD is *ordered* (OBDD) over a chain $\mathcal{C}$ if each internal vertex v and its parent u have $sym(u) \prec_{\mathcal{C}} sym(v)$. An OBDD is *reduced* (ROBDD) if no two vertices are identical (i.e., having the same symbol and children) and no $\diamond$-vertex has two identical children.
- *OBDD with implied literals* (OBDD-L) (Lai, Liu, & Wang, 2013) is a generalization of OBDD in which implied literals are associated with each non-false vertex. Each leaf vertex v in OBDD-L is $\langle\bot\rangle$ or $\langle L(v)\rangle$, and each internal vertex v is denoted by $\langle sym(v), lo(v), hi(v), L(v)\rangle$, where $L(v)$ represents a consistent term. Each internal ver-

tex v represents a formula $\vartheta(v) = [\vartheta(lo(v)) \diamond_{sym(v)} \vartheta(hi(v))] \wedge \bigwedge l \in L(v)$, and each implied literal is a unit implicant of $\vartheta(v)$ whose variable appears in neither $\vartheta(lo(v))$ nor $\vartheta(hi(v))$, and is less than the variables of other unit implicants. An OBDD-L has as many as possible implied literals (OBDD-L_∞) if for any internal vertex v, $L(v)$ includes all unit implicants. An OBDD-L_∞ is reduced (ROBDD-L_∞) if no two distinct vertices have identical variables, children and implied literals, and no vertex has two identical children.

- *AND/OR Multi-Valued Decision Diagram* (AOMDD) (Mateescu, Dechter, & Marinescu, 2008) is a generalization of ROBDD representing graphical models. Here we focus on the binary version of unweighted AOMDD called AOBDD. Let $\mathcal{T}$ be a tree over some variable set (called a pseudo tree by Mateescu, Dechter, & Marinescu, 2008). A meta-node u can be either: (1) a leaf node $\langle \bot \rangle$ or $\langle \top \rangle$; or (2) an internal node which is a combination of a $\diamond$-vertex and its two $\wedge$-children. In an internal meta-node with a $\diamond$-vertex labeled by x, each $\wedge$-vertex v is $\wedge_{\mathcal{T}}$-decomposable and satisfies $x \prec_{\mathcal{T}} glb(v)$. Two meta-nodes are independent if they are from two disjoint subtrees of $\mathcal{T}$. Each AOBDD is a set of DAGs consisting of meta-nodes satisfying that any two roots are independent, and each DAG has neither a meta-node with two identical $\wedge$-vertices nor a meta-node identical with another meta-node.
- A formula in *negation normal form* (NNF) is constructed from $false$, $true$ and literals using only conjoining and disjoining operators. An NNF formula can be represented as a rooted, directed acyclic graph (DAG) where each leaf vertex is labeled with $false$, $true$ or literal; and each internal vertex is labeled with $\wedge$ or $\vee$ and can have arbitrarily many children.
- An NNF formula φ is *decomposable* (DNNF) if for each conjunction in φ, the conjuncts do not share variables. A DNNF formula φ is *deterministic* (d-DNNF) if any two children u and v of a $\vee$-vertex satisfy $\vartheta(u) \wedge \vartheta(v) \models false$. A d-DNNF formula is called a *Decision-DNNF* (Oztok & Darwiche, 2014) if each $\vee$-vertex has exactly two $\wedge$-children representing formulas $\neg x \wedge \varphi$ and $x \wedge \varphi'$, respectively; that is, each $\vee$-vertex is equivalent to a $\diamond$-vertex labeled by x.
- A *sentential decision diagram* over some vtree $\mathcal{V}$ (SDD$_\mathcal{V}$) (Darwiche, 2011) is a special d-DNNF formula. Each $\vee$-vertex in SDD$_\mathcal{V}$ takes the form $(\varphi_1 \wedge \psi_1) \vee \cdots \vee (\varphi_n \wedge \psi_n)$, and satisfies the following conditions: $\varphi_i \not\equiv false$, $\varphi_i \wedge \varphi_j \equiv false$ $(i \neq j)$, and $\bigvee_{1 \leq i \leq n} \varphi_i \equiv true$; and there exists some vertex in $\mathcal{T}$ with two children v and w such that each variable of φ_i is in $\mathcal{T}_v$ and each variable of ψ_i is in $\mathcal{T}_w$. An SDD$_\mathcal{V}$ is *compressed* iff $\psi_i \not\equiv \psi_j$ when $i \neq j$; and it is *trimmed* iff it does not take the form $(\varphi \wedge true) \vee (\neg\varphi \wedge false)$, and each $\varphi_i \neq true$. Compressed and trimmed SDD is canonical (denoted by CSDD$_\mathcal{V}$). A *Decision-CSDD$_\mathcal{V}$* (Oztok & Darwiche, 2015) is an CSDD$_\mathcal{V}$ where each $\vee$-vertex takes the form $(\varphi_1 \wedge \psi_1) \vee (\varphi_2 \wedge \psi_2)$ and satisfies one of the following conditions: φ_1 and φ_2 are literals; ψ_1 is a constant; or $\psi_1 \equiv \neg\psi_2$.

3. Definitions and Properties of $\wedge$-Decompositions

In this section, we define several types of $\wedge$-decompositions and point out some properties that will be used in subsequent sections. Note that some types of $\wedge$-decompositions mentioned in this paper have been presented in previous work in explicit or implicit forms.

3.1 Definition and Properties of $\wedge$-Decomposition

The notion of $\wedge$-decomposition is well known among KC researchers; in fact, it was used to define decomposable NNF (Darwiche, 2001a). We present its definition as follows:

Definition 1 ($\wedge$-decomposition)**.** Given a formula φ, a formula set Ψ is its $\wedge$-*decomposition*, iff $\varphi \equiv \bigwedge_{\psi \in \Psi} \psi$ and $\{Vars(\psi) : \psi \in \Psi\}$ partitions $Vars(\varphi)$.

Given a $\wedge$-decomposition Ψ of φ, each $\psi \in \Psi$ is a *factor*; Ψ is *strict* iff $|\Psi| > 1$; Ψ is *finer* than another $\wedge$-decomposition Ψ' of φ iff $\{Vars(\psi) : \psi \in \Psi\}$ is a finer partition of $Vars(\varphi)$ than $\{Vars(\psi) : \psi \in \Psi'\}$; Ψ is *equivalent* to Ψ', iff for each $\psi \in \Psi$, there exists some $\psi' \in \Psi'$ such that $\psi \equiv \psi'$ and $Vars(\psi) = Vars(\psi')$; and one *sub-decomposition* of Ψ is a subset of Ψ.

Example 1. $\Psi_1 = \{x_2 \vee \neg x_2, x_3 \leftrightarrow x_5, x_4 \leftrightarrow x_6\}$, $\Psi_2 = \{x_2 \vee \neg x_2, (x_3 \leftrightarrow x_5) \wedge (x_4 \leftrightarrow x_6)\}$ and $\Psi_3 = \{(x_2 \vee \neg x_2) \wedge (x_4 \leftrightarrow x_6), x_3 \leftrightarrow x_5\}$ are three $\wedge$-decompositions of $\varphi = (x_2 \vee \neg x_2) \wedge [(x_3 \wedge x_5) \vee (\neg x_3 \wedge \neg x_5)] \wedge (x_4 \leftrightarrow x_6)$, where Ψ_1 is strictly finer than both Ψ_2 and Ψ_3. $\Psi_4 = \{x_{k+0\cdot n} \leftrightarrow \cdots \leftrightarrow x_{k+i\cdot n} : 1 \leq k \leq n\}$ is a $\wedge$-decomposition of the following formula:

$$\bigwedge_{1 \leq k \leq n} x_{k+0\cdot n} \leftrightarrow \cdots \leftrightarrow x_{k+i\cdot n} \tag{1}$$

We will use Equation (1) several more times throughout this paper. For notational convenience, $\emptyset$ is defined as the unique $\wedge$-decomposition of *true*. For a non-constant formula φ, $\{\varphi\}$ is obviously a $\wedge$-decomposition of φ, and we call it a *unit* $\wedge$-decomposition. Given a $\wedge$-decomposition Ψ of a consistent formula, we define the function: $\lfloor \Psi \rfloor = \{\lfloor \psi \rfloor : \psi \text{ is a non-trivial factor in } \Psi\}$. In Example 1, $\lfloor \Psi_1 \rfloor = \{x_3 \leftrightarrow x_5, x_4 \leftrightarrow x_6\}$. Then we can make the following observations:

Observation 1. Given a consistent formula φ and its $\wedge$-decomposition $\Psi = \{\psi_1, \ldots, \psi_m\}$,

(a) $\lfloor \Psi \rfloor$ is a $\wedge$-decomposition of $\lfloor \varphi \rfloor$;

(b) If each factor in Ψ is invalid, $PI(\varphi) = PI(\psi_1) \cup \cdots \cup PI(\psi_m)$ and $IP(\varphi) = \{\gamma_1 \wedge \cdots \wedge \gamma_m : \gamma_i \in IP(\psi_i)\}$; and

(c) Ψ is equivalent to another $\wedge$-decomposition Ψ' of φ, iff both Ψ is finer than Ψ' and Ψ' is finer than Ψ.

According to the above observations, we know that partitioning $PI(\varphi)$ into disjoint subsets yields a $\wedge$-decomposition, and the finest disjoint partition corresponds to the finest $\wedge$-decomposition.

Proposition 1. *Let φ be an irredundant consistent formula, and let $\{\Psi_1, \ldots, \Psi_m\}$ be the minimum disjoint partition of $PI(\varphi)$. $\{\bigwedge_{\delta \in \Psi_i} \delta : 1 \leq i \leq m\}$ is the unique finest $\wedge$-decomposition of φ from the viewpoint of equivalence.*

It is a well known fact in the field of logic synthesis that each irredundant consistent formula has a unique finest $\wedge$-decomposition (Bertacco, 2003). For a consistent formula φ, it is obvious that $\lfloor \varphi \rfloor$ is irredundant, and next we will show that the finest $\wedge$-decomposition of φ can be constructed from the finest $\wedge$-decomposition of $\lfloor \varphi \rfloor$. First, we define an auxiliary function as follows:

$$\lceil \Psi \rceil_X = \Psi \cup \bigcup_{x \in X} \{x \vee \neg x\},$$

where Ψ is a formula set and X is a set of variables not appearing in Ψ. Then we have the following property:

Proposition 2. *Let φ be a consistent formula with the set X of essential variables, and let Ψ be the finest $\wedge$-decomposition of $\lfloor\varphi\rfloor$. $\lceil\Psi\rceil_X$ is the unique finest $\wedge$-decomposition of φ from the viewpoint of equivalence, where $X = Vars(\varphi) \setminus Vars(\lfloor\varphi\rfloor)$. Each $\wedge$-decomposition Ψ' of φ is equivalent to $\{\bigwedge_{\psi\in\lceil\Psi\rceil_X \text{ and } Vars(\psi)\subseteq Vars(\psi')} \psi : \psi' \in \Psi'\}$.*

The reader can verify that in Example 1, Ψ_1 is the finest $\wedge$-decomposition of φ. By the above proposition we know that for a consistent formula, we can use its finest $\wedge$-decomposition to construct any other $\wedge$-decomposition of this formula.

3.2 Definition and Property of Bounded $\wedge$-Decomposition

We now introduce a special type of $\wedge$-decomposition called *bounded* $\wedge$-decomposition:

Definition 2 ($\wedge_i$-decomposition)**.** A $\wedge$-decomposition Ψ is bounded by an integer $0 \le i \le \infty$ ($\wedge_i$-decomposition) iff there exists at most one $\psi \in \Psi$ with $|Vars(\psi)| > i$.

Lai, Liu, and Wang (2013) implicitly discussed $\wedge_1$-decomposition, because each vertex in BDD with implied literals can be seen as a $\wedge_1$-decomposition. Ψ_1, Ψ_2, Ψ_3 and Ψ_4 in Example 1 are $\wedge_2$-decomposition, $\wedge_1$-decomposition, $\wedge_1$-decomposition and $\wedge_{i+1}$-decomposition, respectively. Obviously, each $\wedge$-decomposition is a $\wedge_\infty$-decomposition; each nonempty $\wedge_0$-decomposition is a unit decomposition; and each $\wedge_i$-decomposition is a $\wedge_j$-decomposition if $i \le j$. According to Proposition 2, we can obtain the finest $\wedge_i$-decomposition by conjoining the factors with more than i variables in the finest $\wedge$-decomposition:

Proposition 3. *Let φ be a consistent formula, let Ψ be the finest $\wedge$-decomposition of φ, and let i be an integer. From the viewpoint of equivalence, $\{\bigwedge_{\psi\in\Psi \text{ and } |Vars(\psi)|>i} \psi\} \cup \{\psi \in \Psi : |Vars(\psi)| \le i\}$ is the unique finest $\wedge_i$-decomposition of φ.*

Given a consistent formula, the finest $\wedge_i$-decomposition is hereafter denoted by $\wedge_{\widehat{i}}$-decomposition. It is obvious that each finest $\wedge$-decomposition is a $\wedge_{\widehat{\infty}}$-decomposition. Returning to Example 1, the reader can verify that Ψ_2 is the $\wedge_{\widehat{1}}$-decomposition of φ, and it can be obtained from Ψ_1 through the method described in Proposition 3. From the above proposition, we can draw the following two conclusions, which will be used in the rest of the paper:

Corollary 1. *Given two integers $i \le j$, a consistent formula φ, and a $\wedge$-decomposition Ψ of φ, we have the following facts:*

(a) If Ψ is the $\wedge_{\widehat{j}}$-decomposition of φ, then $\{\bigwedge_{\psi\in\Psi \text{ and } |Vars(\psi)|>i} \psi\} \cup \{\psi \in \Psi : |Vars(\psi)| \le i\}$ is the $\wedge_{\widehat{i}}$-decomposition of φ; and

(b) Ψ is finer than the $\wedge_{\widehat{i}}$-decomposition iff each factor in Ψ is not strictly $\wedge_i$-decomposable, and Ψ is the $\wedge_{\widehat{i}}$-decomposition iff Ψ is bounded by i and each factor in Ψ is not strictly $\wedge_i$-decomposable.

3.3 Definition and Properties of Tree-Structured $\wedge$-Decomposition

Next we introduce a new type of $\wedge$-decomposition respecting some tree-structure:

Definition 3 ($\wedge_{\mathcal{T}}$-decomposition)**.** A $\wedge$-decomposition Ψ *respects* a tree $\mathcal{T}$ over variables ($\wedge_{\mathcal{T}}$-decomposition), iff any two factors $\psi, \psi' \in \Psi$ satisfy that $glb(\psi)$ and $glb(\psi')$ are incomparable over $\prec_{\mathcal{T}}$.

It can be easily proven that any two factors in $\wedge_{\mathcal{T}}$-decomposition are from two disjoint subtrees $\mathcal{T}$ and that for a strictly $\wedge_{\mathcal{T}}$-decomposable formula φ, then $glb(\varphi) \notin Vars(\varphi)$. Note that $\wedge_{\mathcal{T}}$-decomposition was implicitly mentioned by Mateescu, Dechter, and Marinescu (2008), where each AND-vertex can be seen as a $\wedge_{\mathcal{T}}$-decomposition. Returning to Example 1, Ψ_3 is a $\wedge_{\mathcal{T}}$-decomposition over the tree $\mathcal{T}$ in Figure 2b, while neither Ψ_1 nor Ψ_2 is $\wedge_{\mathcal{T}}$-decomposition. Next a $\wedge_{\mathcal{T}}$-decomposition bounded by i is denoted by a $\wedge_{\mathcal{T},i}$-decomposition. It is obvious that each $\wedge_{\mathcal{T}}$-decomposition is a $\wedge_{\mathcal{T},\infty}$-decomposition. We can also obtain the finest $\wedge_{\mathcal{T}}$-decomposition from the $\wedge_{\widehat{\infty}}$-decomposition, by a slightly more complicated method:

Proposition 4. *Let $\mathcal{T}$ be a tree over variables, let Ψ be a $\wedge_{\widehat{\infty}}$-decomposition, let $\mathcal{G}$ be a graph over Ψ with the set of arcs $\{(\psi, \psi'), (\psi', \psi) : glb(\psi) \preceq_{\mathcal{T}} glb(\psi')\}$, let $\mathcal{G}_1, \ldots, \mathcal{G}_m$ be the strongly connected components of $\mathcal{G}$, and for $1 \leq k \leq m$, let $\varphi_k = \bigwedge_{\psi \in V(\mathcal{G}_k)} \psi$. Then, $\{\varphi_1, \ldots, \varphi_m\}$ is the unique finest $\wedge_{\mathcal{T}}$-decomposition from the viewpoint of equivalence.*

Hereafter the finest $\wedge_{\mathcal{T}}$-decomposition is denoted by $\wedge_{\widehat{\mathcal{T}}}$-decomposition, and a $\wedge_{\widehat{\mathcal{T}}}$-decomposition bounded by integer i is denoted by $\wedge_{\widehat{\mathcal{T}},i}$-decomposition. Consider Example 1 again. Let $\psi_1 = x_2 \vee \neg x_2$, $\psi_2 = x_3 \leftrightarrow x_5$ and $\psi_3 = x_4 \leftrightarrow x_6$. Given the tree $\mathcal{T}$ in Figure 2b, we know that $glb(\psi_1) \preceq_{\mathcal{T}} glb(\psi_3)$, $glb(\psi_1)$ and $glb(\psi_2)$ are incomparable over $\preceq_{\mathcal{T}}$, and $glb(\psi_2)$ and $glb(\psi_3)$ are incomparable over $\preceq_{\mathcal{T}}$. Then the reader can verify that Ψ_3 is the $\wedge_{\widehat{\mathcal{T}}}$-decomposition of φ, which can be obtained from Ψ_1 by the method described in Proposition 4. Obviously, each $\wedge_{\widehat{\mathcal{T}}}$-decomposition is a $\wedge_{\widehat{\mathcal{T}},\infty}$-decomposition. However, if $i < \infty$, there exist some class of formulas that do not have any $\wedge_{\widehat{\mathcal{T}},i}$-decomposition. We state this fact and another useful fact as follows:

Corollary 2. *(a) Let $\mathcal{T}$ be a tree over variables, let Ψ be the $\wedge_{\widehat{i}}$-decomposition of a consistent formula φ, let $\mathcal{G}$ be a graph over Ψ with the set of arcs $\{(\psi, \psi'), (\psi', \psi) : glb(\psi) \preceq_{\mathcal{T}} glb(\psi')\}$, let $\mathcal{G}_1, \ldots, \mathcal{G}_m$ be the strongly connected components of $\mathcal{G}$, and for $1 \leq k \leq m$, let $\varphi_k = \bigwedge_{\psi \in V(\mathcal{G}_k)} \psi$. Then, $\{\varphi_1, \ldots, \varphi_m\}$ is the $\wedge_{\widehat{\mathcal{T}},i}$-decomposition if it is bounded by i and φ_k ($1 \leq k \leq m$) is not strictly $\wedge_{\mathcal{T}}$-decomposable, and the $\wedge_{\widehat{\mathcal{T}},i}$-decomposition of φ does not exist otherwise; and*

(b) A $\wedge_{\mathcal{T}}$-decomposition is a $\wedge_{\widehat{\mathcal{T}}}$-decomposition iff each factor is not strictly $\wedge_{\mathcal{T}}$-decomposable.

4. BDD[$\wedge$] and Its Subsets

In this section, we first provide formal definitions for binary decision diagram with conjunctive decomposition (BDD[$\wedge$]) and some of its subsets, including ROBDD[$\wedge_{\widehat{i}}$]$_{\mathcal{C}}$ and ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_{\mathcal{T}}$. We then provide some auxiliary functions and notations that will be used throughout the rest of the paper. The definition of BDD[$\wedge$] is presented as follows:

Definition 4 (BDD[$\wedge$])**.** A BDD[$\wedge$] is a rooted DAG. Each vertex v is labeled with a symbol $sym(v)$. If v is a leaf, $sym(v) = \bot$ or $\top$; otherwise, $sym(v)$ is a variable (in which case v is called a *decision vertex*) or operator $\wedge$ (called a *decomposition 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 are connected by dashed and solid arcs, respectively; for a decomposition vertex, $\{\vartheta(w) : w \in Ch(v)\}$ is a strict $\wedge$-decomposition of $\vartheta(v)$. Each vertex represents a formula defined as follows:

$$\vartheta(v) = \begin{cases} false & sym(v) = \bot; \\ true & sym(v) = \top; \\ \bigwedge_{w \in Ch(v)} \vartheta(w) & sym(v) = \wedge; \\ \vartheta(lo(v)) \diamond_{sym(v)} \vartheta(hi(v)) & \text{otherwise.} \end{cases} \tag{2}$$

The formula represented by the BDD[$\wedge$] is defined as the one represented by its root.

Given two vertices, if they are leaf vertices with the same symbol, or they are internal vertices with the same symbol and children, then we say that they are *identical* with each other; otherwise, we say that they are *distinct*. Following the tradition in the BDD literature, we record all BDD[$\wedge$] vertices in a hash table called *vertex-table*. Hereafter we denote a leaf vertex by $\langle\bot\rangle$ or $\langle\top\rangle$, a $\wedge$-decomposition vertex ($\wedge$-vertex for short) by $\langle sym(v), Ch(v)\rangle$, and a decision vertex ($\diamond$-vertex for short) by $\langle sym(v), lo(v), hi(v)\rangle$. We sometimes abuse $\langle\wedge, \emptyset\rangle$ to denote $\langle\top\rangle$; $\langle x\rangle$ to denote $\langle x, \langle\bot\rangle, \langle\top\rangle\rangle$; $\langle\neg x\rangle$ to denote $\langle x, \langle\top\rangle, \langle\bot\rangle\rangle$; $\langle\wedge, \{w\}\rangle$ to denote w; $\langle\wedge, \{\langle\top\rangle\} \cup V\rangle$ to denote $\langle\wedge, V\rangle$; and $\langle\wedge, \{\langle\bot\rangle\} \cup V\rangle$ to denote $\langle\bot\rangle$. Given a BDD[$\wedge$] $\mathcal{G}$, its size $|\mathcal{G}|$ is defined as the number of its arcs. Given a BDD[$\wedge$] vertex u and a partial order $\prec$ over variables, we denote the BDD[$\wedge$] rooted at u by $\mathcal{G}_u$, denote the set of variables appearing in $\mathcal{G}_u$ by $Vars(u)$, and abbreviate $glb_\prec(Vars(u))$ as $glb_\prec(u)$. We then state two observations about BDD[$\wedge$], which will be used to analyze the time complexities of some algorithms:

Observation 2. Given a BDD[$\wedge$] rooted at u,

(a) We can compute the variable sets for all subgraphs rooted at vertices in $\mathcal{G}_u$ in time $O(|Vars(u)| \cdot |V(\mathcal{G}_u)|)$; and

(b) Whether $\mathcal{G}_u$ has more than i variables can be determined in $O(1)$, and thus if $sym(u) = \wedge$, whether u is a $\wedge_i$-decomposition can be determined in $O(|Ch(u)|)$.

Next we define some constraints on BDD[$\wedge$], which allow us to generate subsets of BDD[$\wedge$] by restricting combinations of these constraints:

Definition 5 (constraints on BDD[$\wedge$])**.** Given an integer i, a partial order $\prec$ over variables, and a tree $\mathcal{T}$ over variables,

- A BDD[$\wedge$] is *ordered* over $\prec$ (OBDD[$\wedge$]$_\prec$) iff each $\diamond$-vertex u and its $\diamond$-descendant v satisfy $sym(u) \prec sym(v)$;
- A BDD[$\wedge$] is *reduced* (RBDD[$\wedge$]) iff no two vertices are identical and no $\diamond$-vertex has two identical children;
- A BDD[$\wedge$] is $\wedge_i$*-decomposable* (BDD[$\wedge_i$]) iff each $\wedge$-vertex is a $\wedge_i$-decomposition;
- A BDD[$\wedge$] is $\wedge_{\widehat{i}}$*-decomposable* (BDD[$\wedge_{\widehat{i}}$]) iff each $\wedge$-vertex is a $\wedge_{\widehat{i}}$-decomposition and the $\wedge_{\widehat{i}}$-decomposition of each $\diamond$-vertex v is $\{\vartheta(v)\}$;

- A BDD[$\wedge$] is $\wedge_{\mathcal{T}}$-*decomposable* (BDD[$\wedge_{\mathcal{T}}$]) iff each $\wedge$-vertex is a $\wedge_{\mathcal{T}}$-decomposition; and
- A BDD[$\wedge$] is $\wedge_{\widehat{\mathcal{T}}}$-*decomposable* (BDD[$\wedge_{\widehat{\mathcal{T}}}$]) iff each $\wedge$-vertex is a $\wedge_{\widehat{\mathcal{T}}}$-decomposition and the $\wedge_{\widehat{\mathcal{T}}}$-decomposition of each $\diamond$-vertex v is $\{\vartheta(v)\}$.

In the subsequent sections, we focus on OBDD[$\wedge$]$_{\prec}$ where $\prec$ is a tree-structured order: the ancestor-descendant relationship $\prec_{\mathcal{T}}$ on a tree $\mathcal{T}$ and particularly $\prec_{\mathcal{C}}$ over a chain $\mathcal{C}$. Unless otherwise stated, we hereafter assume that $\mathcal{C}$ and $\mathcal{T}$ always satisfy $\prec_{\mathcal{T}} \subset \prec_{\mathcal{C}}$ (that is, $\mathcal{T}$ is not a chain and $\mathcal{C}$ is a topological order of $\mathcal{T}$), and we sometimes use $\mathcal{C}$ and $\mathcal{T}$ to denote $\prec_{\mathcal{C}}$ and $\prec_{\mathcal{T}}$, respectively. We mainly focus on the subsets of OBDD[$\wedge$]$_{\prec}$ described in Table 1. Note that ROBDD[$\wedge_{\widehat{i}}$]$_{\mathcal{C}}$ does not require that the children of $\wedge$-vertex agree with $\prec_{\mathcal{C}}$. In the remaining sections, we will analyze the canonicity, expressivity, and space-time efficiency of ROBDD[$\wedge_{\widehat{i}}$]$_{\mathcal{C}}$ and ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_{\mathcal{T}}$. For convenience, we will also use OBDD[$\wedge_i$]$_{\mathcal{C}}$, OBDD[$\wedge_{\mathcal{T}}$]$_{\mathcal{T}}$ and ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ in some algorithms.

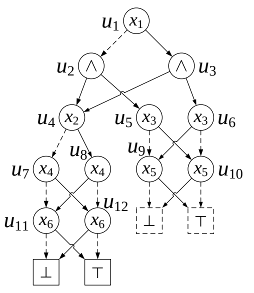
Subsets	**Descriptions**
OBDD[$\wedge_i$]$_{\mathcal{C}}$	Ordered binary decision diagram over $\prec_{\mathcal{C}}$ with conjunctive decompositions bounded by i
ROBDD[$\wedge_{\widehat{i}}$]$_{\mathcal{C}}$	Reduced ordered binary decision diagram over $\prec_{\mathcal{C}}$ with finest conjunctive decompositions bounded by i
OBDD[$\wedge_{\mathcal{T}}$]$_{\mathcal{T}}$	Ordered binary decision diagram over $\prec_{\mathcal{T}}$ with tree-structured conjunctive decompositions bounded by i
ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$	Reduced ordered binary decision diagram over $\prec_{\mathcal{T}}$ with finest conjunctive decompositions
ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_{\mathcal{T}}$	Reduced ordered binary decision diagram over $\prec_{\mathcal{T}}$ with finest tree-structured conjunctive decompositions bounded by i

Table 1: Subsets of BDD[$\wedge$] highlighted in the remainder of this paper, where i is an integer, and $\mathcal{C}$ and $\mathcal{T}$ are, respectively, a chain and a tree over variables

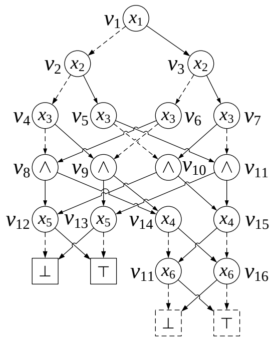
Example 2. Figures 3a and 3b depict, respectively, an ROBDD[$\wedge_{\widehat{1}}$]$_{\mathcal{C}}$ and an ROBDD[$\wedge_{\widehat{2}}$]$_{\mathcal{C}}$ representing Equation (1) with $n = i = 2$, where $\mathcal{C}$ is depicted in Figure 2a. Note that for simplicity, we sometimes draw multiple copies of vertices in this paper, denoted by dashed boxes or circles, but they represent the same vertex. Figure 3a is not an OBDD[$\wedge_1$]$_{\mathcal{C}}$ since vertex u_2 is not bounded by one. For the general form of Equation (1), the number of vertices labelled with x_{1+n} in the corresponding ROBDD[$\wedge_{\widehat{j}}$]$_{\mathcal{C}}$ ($j < i$) is equal to 2^n, while the number of vertices in the corresponding ROBDD[$\wedge_{\widehat{i}}$]$_{\mathcal{C}}$ will be $(2i + 5)n$. That is, the size of ROBDD[$\wedge_{\widehat{j}}$]$_{\mathcal{C}}$ representing Equation (1) is exponential in n, while the size of the corresponding ROBDD[$\wedge_{\widehat{i}}$]$_{\mathcal{C}}$ is only linear in n. Figure 3a is also an ROBDD[$\wedge_{\widehat{\mathcal{T}},2}$]$_{\mathcal{T}}$, where $\mathcal{T}$ is depicted in Figure 2b.

Next we provide some facts about OBDD[$\wedge$]$_{\prec}$ that will be used in the rest of the paper:

Observation 3. (a) Each OBDD[$\wedge_i$]$_{\mathcal{C}}$ is an OBDD[$\wedge_j$]$_{\mathcal{C}}$ if $i \leq j$;
(b) Each ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_{\mathcal{T}}$ is an OBDD[$\wedge_i$]$_{\mathcal{C}}$;
(c) Given an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ vertex u, $glb(u) \in Vars(u)$ iff $glb(u) = sym(u)$;



(a)



(b)

Figure 3: An ROBDD$[\wedge_{\widehat{2}}]_{\mathcal{C}}$ (a) and an ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ (b), where $\mathcal{C}$ is depicted in Figure 2a

(d) The number of vertices in a non-trivial ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$) is not more than three times of the number of its $\diamond$-vertices; and
(e) Given an ROBDD$[\wedge_{\mathcal{T}}]_{\mathcal{T}}$, if there does not exist any $\wedge$-vertex with $\wedge$-child, it is an ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$.

Some subsets in OBDD$[\wedge]_{\prec}$ are closely related to other languages presented previously in the literature, including d-DNNF and its subsets. Here we simply discuss the connections between them, in order to facilitate the theoretical comparisons in the following three sections. We will elaborate the relationships between them in Section 9. First, since each Decision-DNNF can be seen as a free BDD (Gergov & Meinel, 1994) augmented by $\wedge$-decomposition, each OBDD$[\wedge]_{\prec}$ is a strict subset of Decision-DNNF, and thus a strict subset of d-DNNF. For each $\vee$-vertex v in Decision-CSDD$_{\mathcal{V}}$, it is obvious that $\varphi_1 \equiv \neg\varphi_2$. Therefore, if φ_1 and φ_2 are literals, then v is equivalent to a decision vertex; otherwise, v is equivalent to $\varphi_2 \wedge \psi_2$, $\varphi_1 \vee \psi_2$, or $\varphi_1 \leftrightarrow \psi_1$. Note that a tree over variables can be seen as a vtree, and ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ can therefore be seen as a subset of Decision-CSDD$_{\mathcal{V}}$. We now discuss the relationship between our languages and three previous canonical languages ROBDD, ROBDD-L_∞ and AOBDD:

Theorem 1. *Given a chain $\mathcal{C}$ and a tree $\mathcal{T}$ over PV,*
(a) Each ROBDD over $\mathcal{C}$ is an ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$, and vice versa;
(b) Each OBDD-L over $\mathcal{C}$ can be transformed into OBDD$[\wedge_1]_{\mathcal{C}}$ in linear time, and the mutual transformation between an ROBDD-L_∞ over $\mathcal{C}$ and the equivalent ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ can be done in linear time; and
(c) The mutual transformation between an AOBDD over $\mathcal{T}$ and the equivalent ROBDD$[\wedge_{\widehat{\mathcal{T}},\infty}]_{\mathcal{T}}$ can be done in linear time.

Figure 4 depicts an ROBDD-L_∞ over chain $\mathcal{C}$ in Figure 2a and an AOBDD over tree $\mathcal{T}$ in Figure 2b, where the ROBDD-L_∞ is equivalent to the ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ in Figure 3b and the AOBDD is equivalent to the ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ in Figure 3a. According the above theorem, we will know that our analysis in the following three sections can indirectly reveal the relationship between our new languages and the three previous ones ROBDD, ROBDD-L_∞ and AOBDD to some extent.

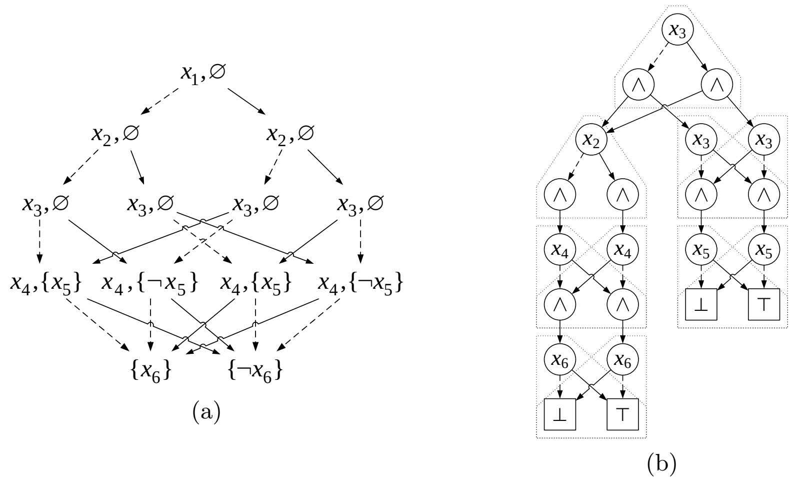


Figure 4: An ROBDD-L_∞ (a) over chain in Figure 2a, and an AOBDD (b) over tree in Figure 2b, where each meta-node is depicted by a dash box

4.1 Some Auxiliary Functions and Notations

This subsection introduces some auxiliary functions and notations that will be used in the rest of the paper. The first function is called Make, which can push an OBDD$[\wedge]_\prec$ vertex into vertex-table. In the function, we use Line 1 to guarantee that u does not have two identical children, and use Line 2 to guarantee that u is not identical with any vertex already in vertex-table. We recursively push the children of u into vertex-table on Line 3, and then push u into vertex-table on Line 4. It is obvious that this function has the following properties:

```
Function Make($u$)
  Input: a $\diamond$-vertex $u$ in OBDD$[\wedge]_\prec$
  Output: an ROBDD$[\wedge]_\prec$ vertex representing $\vartheta(u)$
1 if $u$ is a $\diamond$-vertex with $lo(u) = hi(u)$ then return Make($lo(u)$)
2 if there exists another $u'$ in vertex-table with $u = u'$ then return $u'$
3 $Ch(u) \leftarrow \{\text{Make}(v) : v \in Ch(u)\}$
4 Allocate a position in vertex-table to $u$
5 return $u$
```

Observation 4. Given an OBDD$[\wedge]_\prec$ vertex u,

(a) If all vertices in $\mathcal{G}_u$ are pushed into vertex-table by calling Make, then $\mathcal{G}_u$ is reduced.

(b) A single calling (not considering recursive callings) of Make(u) terminates in time $O(1)$ if $sym(u) \in PV$, and in time $O(|Ch(u)|)$ otherwise.

The second function is called Merge, and it is used to guarantee that a $\wedge$-decomposition is finest: Given a $\wedge$-vertex u, we replace $Ch(u)$ by $(Ch(u) \setminus V) \cup \bigcup_{v \in V} Ch(v)$, where $V = \{v \in Ch(u) : sym(v) = \wedge\}$; that is, while there exists some $\wedge$-child $v \in Ch(u)$, we repeat replacing v in $Ch(u)$ with all its children. According to Corollaries 1b and 2b, we can observe the following facts:

Observation 5. Given a $\wedge$-vertex u in BDD[$\wedge$], Merge(u) terminates in time $O(|Vars(u)|)$, and we know that:

(a) If $\mathcal{G}_u$ is $\wedge_i$-decomposable, the BDD[$\wedge$] rooted at Merge(u) is $\wedge_i$-decomposable;
(b) If $\mathcal{G}_u$ is $\wedge_i$-decomposable and $\mathcal{G}_v$ is $\wedge_{\widehat{i}}$-decomposable for each $v \in Ch(u)$, the BDD[$\wedge$] rooted at Merge(u) is $\wedge_{\widehat{i}}$-decomposable;
(c) If $\mathcal{G}_u$ is $\wedge_{\mathcal{T}}$-decomposable, the BDD[$\wedge$] rooted at Merge(u) is $\wedge_{\mathcal{T}}$-decomposable; and
(d) If $\mathcal{G}_u$ is $\wedge_{\mathcal{T}}$-decomposable and $\mathcal{G}_v$ is $\wedge_{\widehat{\mathcal{T}}}$-decomposable for each $v \in Ch(u)$, the BDD[$\wedge$] rooted at Merge(u) is $\wedge_{\widehat{\mathcal{T}}}$-decomposable.

According to Observations 3e, 4 and 5, we can immediately draw the following conclusion:

Proposition 5. *An OBDD[$\wedge_{\mathcal{T}}$]$_{\mathcal{T}}$ rooted at u can be transformed into ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ in $O(|Vars(u)| \cdot |V(\mathcal{G}_u)|)$ by calling Make for each vertex and calling Merge for each $\wedge$-vertex in a bottom-up fashion.*

The third function, which is used to remove some children from a $\wedge$-vertex, is defined as follows:

$$u \setminus V = \begin{cases} \langle \top \rangle & Ch(u) = V; \\ v & \exists v. Ch(u) \setminus V = \{v\}; \\ \langle \wedge, Ch(u) \setminus V \rangle & \text{otherwise.} \end{cases}$$

where u is a $\wedge$-vertex and V is a subset of its children.

Observation 6. Given a $\wedge$-vertex u and a subset of its children V, $u \setminus V$ can be done in time $O(|Ch(u)|)$, and we know that if $\mathcal{G}_u$ is $\wedge_i$-decomposable (resp. $\wedge_{\widehat{i}}$-decomposable, $\wedge_{\mathcal{T}}$-decomposable and $\wedge_{\widehat{\mathcal{T}}}$-decomposable), the BDD[$\wedge$] rooted at $u \setminus V$ is $\wedge_i$-decomposable (resp. $\wedge_{\widehat{i}}$-decomposable, $\wedge_{\mathcal{T}}$-decomposable and $\wedge_{\widehat{\mathcal{T}}}$-decomposable):

We close this subsection by introducing two auxiliary notions that will be used in some operation algorithms for ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_{\mathcal{T}}$. First, we use $Ch_x(v)$ to denote the subset of $Ch(v)$ whose variables stem from x; that is, $Ch_x(v) = \{w \in Ch(v) : x \preceq_{\mathcal{T}} glb(w)\}$. Then we can present the two notions as follows:

Definition 6 (meta-child and meta-vertex)**.** Let v be a $\wedge$-vertex in an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ $\mathcal{G}$. A *meta-child* of v on variable $x \in Ch_{\mathcal{T}}(glb(v))$ is defined as $mch_x(v) = \langle sym(v), Ch_x(v) \rangle$. A *meta-vertex* [3] of $\mathcal{G}$ is either a vertex in $\mathcal{G}$ or a meta-child of some meta-vertex.

According to the above definition, for the tree $\mathcal{T}$ in Figure 2c and an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ vertex $v = \langle \wedge, \{\langle x_3 \rangle, \langle x_4 \rangle, \langle x_5 \rangle\} \rangle$, $Ch_{x_2}(v) = \{\langle x_3 \rangle, \langle x_4 \rangle\}$, $mch_{x_2}(v) = \langle \wedge, \{\langle x_3 \rangle, \langle x_4 \rangle\} \rangle$, and $\mathcal{G}_v$ has 7 meta-vertices.

Observation 7. Given an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ rooted at u, the number of distinct meta-vertices is not more than $|\mathcal{G}_u| - |V(\mathcal{G}_u)| + 2$.

3. Meta-vertex is a different notion from meta-node in AOBDD.

5. Canonicity and Expressivity of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$

In this section, we show that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ is canonical and complete, while ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ is canonical and incomplete. We then compare the expressivity of ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ with that of ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$.

We first discuss the canonicity and completeness of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. The canonicity can be understood by the uniqueness of $\wedge_{\widehat{i}}$-decomposition in Proposition 3 and the fact that for two equivalent vertices $\langle x, v, w\rangle$ and $\langle x, v', w'\rangle$, we have $\vartheta(v) \equiv \vartheta(v')$ and $\vartheta(w) \equiv \vartheta(w')$. The completeness can be understood from a recursive perspective. Let φ be a non-trivial formula. If $\lfloor\varphi\rfloor$ has a strict $\wedge_{\widehat{i}}$-decomposition Ψ, then $\langle \wedge, \{v : \exists\psi \in \Psi.\psi \equiv \vartheta(v)\}\rangle$ represents φ. Otherwise, we denote $glb(\lfloor\varphi\rfloor)$ by x and assume that v and w represent $\varphi|_{x=false}$ and $\varphi|_{x=true}$, respectively. Since $x \in Vars(\lfloor\varphi\rfloor)$, $\langle x, v, w\rangle$ represents φ.

Theorem 2. *Fixing integer $0 \le i \le \infty$ and chain $\mathcal{C}$ over PV, there is exactly one ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ representing a given formula.*

Turning to ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, its canonicity can be understood in a fashion similar to that of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. However, the completeness of ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ no longer holds, according to the following observation:

Observation 8. There is no ROBDD$[\wedge_{\mathcal{T}}]_{\mathcal{T}}$ to represent an irredundant formula φ that is not strictly $\wedge_{\mathcal{T}}$-decomposable and satisfies $glb(\varphi) \notin Vars(\varphi)$.

The above observation can be understood as follows: since φ is irredundant, it cannot be represented by a leaf vertex; since φ is not strictly $\wedge_{\mathcal{T}}$-decomposable, it cannot be represented by a $\wedge$-vertex; and since $glb(\varphi) \notin Vars(\varphi)$, it cannot be represented by a $\diamond$-vertex according to Observation 3c. Let $\mathcal{T}$ be a tree that is not a chain. Then there exist two incomparable variables x and x' over $\prec_{\mathcal{T}}$. It is obvious that $\varphi = x \leftrightarrow x'$ is not strictly $\wedge_{\mathcal{T}}$-decomposable and φ satisfies $glb(\varphi) \notin Vars(\varphi)$; and therefore there is no ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{T}}$ to represent φ. This leads to the following conclusion:

Theorem 3. *Fixing tree $\mathcal{T}$ over PV, there is at most one ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ to represent a given formula, and ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ is incomplete when $\mathcal{T}$ is not a chain.*

According to the canonicity of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$), if $\mathcal{C}$ (resp. $\mathcal{T}$) has a finite number of variables, then ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$) has a finite number of formulas. Since it is not interesting to discuss the theoretical properties (e.g., succinctness and tractability) of a language with only finite formulas, we hereafter assume that $\mathcal{C}$ (resp. $\mathcal{T}$) is over an infinite number of variables.

Due to the incompleteness of ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, we compare the expressivity between ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ ($0 \le i \le \infty$). Let L_1 and L_2 be two languages. We say that L_1 is at most as *expressive* as L_2 (denoted by $\mathrm{L}_1 \le_e \mathrm{L}_2$) iff for each formula in L_1, there exists an equivalent formula in L_2. First we introduce a notion that is closely related to the expressivity of ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$:

Definition 7 (decomposition cardinality)**.** Given a tree $\mathcal{T}$ and a variable set X in which any two variables are incomparable over $\prec_{\mathcal{T}}$, we define *decomposition cardinality* of $\mathcal{T}$ on X (denoted by $dc(\mathcal{T}, X)$) as 0 if X is a singleton, and as the second highest number of vertices in the subtrees rooted at variables in X otherwise.[4] We define the decomposition cardinality of $\mathcal{T}$ (denoted by $dc(\mathcal{T})$) as the minimum decomposition cardinality on all nonempty sets in which any two variables are incomparable over $\prec_{\mathcal{T}}$.

It is obvious that the decomposition cardinality of each chain is 0. Consider the trees $\mathcal{T}$ and $\mathcal{T}'$ in Figure 2b and 2c, respectively. The reader can verify that $dc(\mathcal{T}', \{x_3, x_4, x_5\}) = 1$, $dc(\mathcal{T}) = \infty$ and $dc(\mathcal{T}') = 2$. We next present a fact about decomposition cardinality, from which we conclude the expressivity relation between ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$:

Observation 9. Given a tree $\mathcal{T}$ over variables, each ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ is an ROBDD$[\wedge_{\widehat{\mathcal{T}},dc(\mathcal{T})}]_{\mathcal{T}}$, and there exists some formula that can be represented by ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$ but not by ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ if $0 \le i < j \le dc(\mathcal{T})$.

Theorem 4. *Given a tree $\mathcal{T}$ over variables and two integers i and j, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}} \le_e$ ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$ iff $i \le j$ or $dc(\mathcal{T}) \le j$.*

Let $\mathcal{V}$ be a vtree corresponding to $\mathcal{T}$. Since ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ is a subset of Decision-CSDD$_{\mathcal{V}}$, which in turn is a subset of CSDD$_{\mathcal{V}}$, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}} \le_e$ Decision-CSDD$_{\mathcal{V}} \le_e$ CSDD$_{\mathcal{V}}$; in particular, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}} <_e$ Decision-CSDD$_{\mathcal{V}}$ if $\mathcal{T}$ is not a chain, because there exists some binary clause that can be expressed in Decision-CSDD$_{\mathcal{V}}$ but not in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$. Theorems 2–3 indicate that when moving along each solid arc in Figure 1, the expressivity of language is invariant, and that when moving along each vertical arc in Figure 1, the expressivity of language increases. Theorem 4 indicates that when moving along each dashed arc at the bottom of Figure 1, the expressivity of language does not decrease; in particular, the expressivity of language increases if $dc(\mathcal{T}) = \infty$. Note that the incompleteness of ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ does not prevent this languages from being used in practical applications. On the one hand, some practical applications do not require complete languages; for example, Horn theory is an influential incomplete language in the AI literature. On the other hand, for a practical knowledge base, we can choose an appropriate tree $\mathcal{T}$ such that this knowledge base can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ (in the worst case, we can choose a chain, and then ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ is equivalent to ROBDD), just like choosing appropriate vtrees (Oztok & Darwiche, 2015) to compile knowledge bases into Decision-SDD (this language is also incomplete for many vtrees). In particular, we mention that each naive Bayes model can be compiled into ROBDD$[\wedge_{\widehat{\mathcal{T}},2d}]_{\mathcal{T}}$ in polysize using encoding ENC1 proposed by Chavira and Darwiche (2008), where d is the second maximum size of domain of each feature.

6. Succinctness of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$

In this section, we first analyze the succinctness relationship between ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$, we then analyze the succinctness relationship between ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$, and we last discuss the succinctness relationship between these two families of languages and some existing languages. Note that the standard definition of succinctness in the KC map only applies to complete languages. Here we need to handle the case of

4. If there are two subtrees having the most variables, we define the second highest number of variables of the subtrees as the maximum number.

incomplete languages. Therefore, we next extend the standard definition of succinctness to some extent; specifically, we only compare the sizes of formulas that can be represented in both languages:

Definition 8 (succinctness)**.** Let L_1 and L_2 be two languages. L_1 is at most as *succinct* as L_2 ($L_1 \leq_s L_2$) [5], iff there exists a polynomial p such that for every sentence $\varphi_1 \in L_1$ that can be represented in L_2, there exists an equivalent sentence $\varphi_2 \in L_2$ where $|\varphi_2| \leq p(|\varphi_1|)$. Here, $|\varphi_1|$ and $|\varphi_2|$ are the sizes of φ_1 and φ_2, respectively. L_1 is strictly less succinct than L_2 ($L_1 <_s L_2$) iff $L_1 \leq_s L_2$ and $L_1 \not\geq_s L_2$. L_1 is as succinct as L_2 ($L_1 =_s L_2$) iff $L_1 \leq_s L_2$ and $L_1 \geq_s L_2$.

Definition 8 immediately leads to the conclusion that ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}} =_s$ ROBDD$[\wedge_{\widehat{\mathcal{T},j}}]_{\mathcal{T}}$. Obviously, succinctness relation is conditionally transitive: if $L_1 \leq_e L_2 \leq_e L_3$, then $L_1 \leq_s L_2$ and $L_2 \leq_s L_3$ imply $L_1 \leq_s L_3$. Moreover, if $L_1 \subseteq L_2 \leq_e L_3$, then $L_1 \not\leq_e L_3$ implies $L_2 \not\leq_e L_3$. The last two facts will be used in the second subsection.

6.1 Succinctness Relationship Between ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$

We first present an algorithm called Decompose (in Algorithm 1), which can transform an OBDD$[\wedge_i]_{\mathcal{C}}$ into the equivalent ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. With this algorithm, we can show that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \leq_s$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ if $i \leq j$. Then with some counter-examples, we can obtain the exact succinctness relationship between ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$.

The input of Decompose is an OBDD$[\wedge_i]_{\mathcal{C}}$ vertex u. Here we use a global non-negative integer i, a global chain $\mathcal{C}$ over PV, and the implicit global vertex-table. A hash table H is used to store previously computed outputs of the algorithms, in order to avoid repetitively processing any vertices. If u is a leaf vertex, the algorithm simply returns u itself (Line 2). Otherwise, we first recursively call Decompose for each child of u (Line 4). If u is a $\wedge$-vertex, we call Merge to get the $\wedge_{\widehat{i}}$-decomposition, and then call Make to insert u into vertex-table (Line 12). Otherwise, we know that u is a $\diamond$-vertex. If $\vartheta(u)$ is strictly $\wedge_i$-decomposable, then u must satisfy one of the three conditions on Lines 7–9:

Observation 10. Let u be a $\diamond$-vertex in OBDD$[\wedge_i]_{\mathcal{C}}$ with two distinct ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ children. $\vartheta(u)$ is strictly $\wedge_i$-decomposable only if one of the following conditions holds: a) $\langle \bot \rangle \in Ch(u)$ and $|Vars(u)| > 1$; b) one child is a factor of the other child; or c) both children are $\wedge$-vertices with some shared factors.

We call three functions called ExtractLeaf, ExtractPart and ExtractShare to handle the cases on Lines 7–9 in Decompose, respectively. The input parameters of these functions are $\diamond$-vertices with distinct children already in ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. We use these three functions to obtain an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex that is equivalent to their input. The main idea of the three functions is to extract possible common factors. For ExtractLeaf(u), if $i = 0$, it is obvious that u is not strictly $\wedge_i$-decomposable, and otherwise we further decompose it on Line 2 or 3. Figure 5a–5b provides two examples to show how ExtractLeaf works. ExtractLeaf has the following properties:

5. Darwiche and Marquis (2002) used "$L_1 \leq L_2$" to denote that "L_1 is at least as succinct as L_2". Here we use a different notation "$L_1 \leq_s L_2$" to denote that "the succinctness of L_1 is not stronger than that of L_2".

Algorithm 1: Decompose(u)

```
   Input: an OBDD$[\wedge_i]_{\mathcal{C}}$ vertex $u$
   Output: the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex representing $\vartheta(u)$
1  if $H(u) \neq nil$ then return $H(u)$
2  if $u$ is a leaf vertex then $H(u) \leftarrow u$
3  else
4      $Ch(u) \leftarrow \{\text{Decompose}(v) : v \in Ch(u)\}$
5      if $u$ is a $\diamond$-vertex then
6          if $lo(u) = hi(u)$ then $H(u) \leftarrow lo(u)$
7          else if $\langle\bot\rangle \in Ch(u)$ and $|Vars(u)| > 1$ then $H(u) \leftarrow$ ExtractLeaf($u$)
8          else if one child of $u$ is a factor of the other then $H(u) \leftarrow$ ExtractPart($u$)
9          else if $sym(lo(u)) = sym(hi(u)) = \wedge$ and $Ch(lo(u)) \cap Ch(hi(u)) \neq \emptyset$ then
10             $H(u) \leftarrow$ ExtractShare($u$)
11         else $H(u) \leftarrow$ Make($u$)
12     else $H(u) \leftarrow$ Make(Merge($u$))
13 end
14 return $H(u)$
```

Observation 11. Let u be a $\diamond$-vertex in OBDD$[\wedge_i]_{\mathcal{C}}$ with two ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ children satisfying that $\langle\bot\rangle \in Ch(u)$ and $|Vars(u)| > 1$. ExtractLeaf(u) will return the equivalent ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex in $O(|Vars(u)|)$, and it will introduce at most two new vertices into vertex-table and add at most two arcs to the resulting DAG.

Function ExtractLeaf(u)

```
   Input: a $\diamond$-vertex $u$ in OBDD$[\wedge_i]_{\mathcal{C}}$, where $\langle\bot\rangle \in Ch(u)$ and $|Vars(u)| > 1$
   Output: the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex representing $\vartheta(u)$
1  if $i = 0$ then return $u$
2  if $lo(u) = \langle\bot\rangle$ then $u' \leftarrow \langle\wedge, \{\langle sym(u)\rangle, hi(u)\}\rangle$
3  else $u' \leftarrow \langle\wedge, \{\langle\neg sym(u)\rangle, lo(u)\}\rangle$            // $hi(u) = \langle\top\rangle$
4  return Make(Merge($u'$))
```

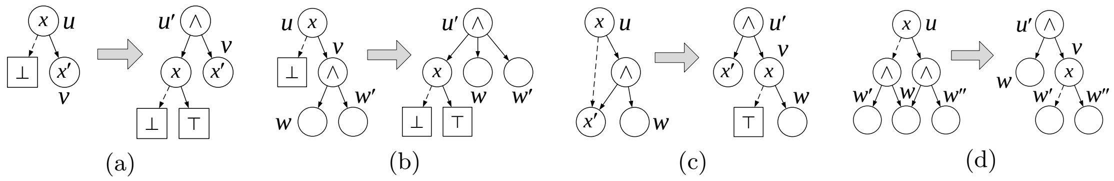


Figure 5: Four examples to show how functions ExtractLeaf, ExtractPart and ExtractShare work, where we assume that $\mathcal{G}_{u'}$ is bounded by i

For ExtractPart(u), it is obvious that u is strictly $\wedge$-decomposable, we try to $\wedge_i$-decompose it on Lines 2–3 or 5–6 by extracting the shared factor $lo(u)$ or $hi(u)$. If the result is $\wedge_i$-decomposable, it is $\wedge_{\widehat{i}}$-decomposable by Corollary 1b, and otherwise u itself is $\wedge_{\widehat{i}}$-

decomposable. Figure 5c provides an example to show how ExtractPart works. ExtractPart has the following properties:

Observation 12. Let u be a $\diamond$-vertex in OBDD$[\wedge_i]_\mathcal{C}$ with two ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$ children satisfying that one child is a factor of the other child. ExtractPart(u) will return the equivalent ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$ vertex in $O(|Vars(u)|)$, and it will introduce at most three new vertices into vertex-table and add at most one arc to the resulting DAG.

Function ExtractPart(u)

```
  Input: a $\diamond$-vertex $u$ in OBDD$[\wedge_i]_\mathcal{C}$, where one vertex in $Ch(u)$ is a $\wedge$-vertex in
         ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$, and the other is a child of the former
  Output: the ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$ vertex representing $\vartheta(u)$
1 if $lo(u) \in Ch(hi(u))$ then
2 |   $v \leftarrow \langle sym(u), \langle\top\rangle, hi(u) \setminus \{lo(u)\}\rangle$
3 |   $u' \leftarrow \langle\wedge, \{lo(u), v\}\rangle$
4 else                                         // $hi(u) \in Ch(lo(u))$
5 |   $v \leftarrow \langle sym(u), lo(u) \setminus \{hi(u)\}, \langle\top\rangle\rangle$
6 |   $u' \leftarrow \langle\wedge, \{hi(u), v\}\rangle$
7 end
8 if $u'$ is a $\wedge_i$-vertex then return Make($u'$)
9 else return $u$
```

For ExtractShare(u), it is obvious that u is strictly $\wedge$-decomposable, we try to $\wedge_i$-decompose it. First, we compute the shared factors on Line 1. It is obvious that there exists at most one factor with more than i variables. If u' on Line 4 is not bounded by i and $|V| > 1$, we can $\wedge_i$-decompose it by only extracting the factors with at most i variables. Figure 5d provides an example to show how ExtractShare works. ExtractShare has the following properties:

Observation 13. Let u be a $\diamond$-vertex in OBDD$[\wedge_i]_\mathcal{C}$ with two ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$ children satisfying that both children are $\wedge$-vertices with some shared factors. ExtractShare(u) will return the equivalent ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$ vertex in $O(|Vars(u)|)$, and it will introduce at most three new vertices into vertex-table and add at most one arc to the resulting DAG.

According to Observations 10–13 and the fact that each calling of Merge introduces at most one new vertex and adds two arcs, it is easy to prove that Algorithm Decompose has the following properties:

Proposition 6. *Given each OBDD$[\wedge_i]_\mathcal{C}$ rooted at u, the output of* Decompose*(u) is an ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$ vertex equivalent to $\vartheta(u)$, the number of vertices in the ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$ rooted at* Decompose*(u) is not more than $4 \cdot |V(\mathcal{G}_u)|$, and the size of ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$ is not more than $2 \cdot |\mathcal{G}_u|$. The time complexity of* Decompose *is bounded by $O(|Vars(u)| \cdot |V(\mathcal{G}_u)|)$.*

Given any chain $\mathcal{C}$ and integer i, the algorithm Decompose immediately provides a compilation algorithm for ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$; that is, first generate the equivalent ROBDD$[\wedge_{\widehat{0}}]_\mathcal{C}$ (resp. ROBDD$[\wedge_{\widehat{1}}]_\mathcal{C}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_\mathcal{T}$), and then transform the result into the ROBDD$[\wedge_{\widehat{i}}]_\mathcal{C}$, where ROBDD$[\wedge_{\widehat{0}}]_\mathcal{C}$ (resp. ROBDD$[\wedge_{\widehat{1}}]_\mathcal{C}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_\mathcal{T}$) can be generated by any

```
Function ExtractShare(u)
  Input: a $\diamond$-vertex $u$ in OBDD$[\wedge_i]_{\mathcal{C}}$, where both $lo(u)$ and $hi(u)$ are ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$
         vertices, $sym(lo(u)) = sym(hi(u)) = \wedge$, and $Ch(lo(u)) \cap Ch(hi(u)) \neq \emptyset$
  Output: the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex representing $\vartheta(u)$
1 $V \leftarrow Ch(lo(u)) \cap Ch(hi(u))$
2 $v \leftarrow \langle sym(u), lo(u) \setminus V, hi(u) \setminus V \rangle$
3 $u' \leftarrow \langle \wedge, V \cup \{v\} \rangle$
4 if $u'$ is a $\wedge_i$-vertex then return Make($u'$)
5 else if $|V| > 1$ then
6     $V \leftarrow \{w \in V : |Vars(\mathcal{G}_w)| \leq i\}$
7     $v \leftarrow \langle sym(u), lo(u) \setminus V, hi(u) \setminus V \rangle$
8     return Make($\langle \wedge, V \cup \{v\} \rangle$)
9 else return $u$
```

ROBDD (resp. ROBDD-L_∞ and AOBDD) compilation algorithm according to Theorem 1. Therefore, Decompose verifies the existence of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ for any formula. Lai, Liu, and Wang (2013) proposed an algorithm called L2Inf that can transform each OBDD-L into ROBDD-L_∞. Roughly speaking, L2Inf is a special case of Decompose that transforms OBDD$[\wedge_1]_{\mathcal{C}}$ into ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$.

It is known that given a CNF formula with n variables and treewidth w, there exists an AOBDD over some tree $\mathcal{T}$ whose size is bounded by $O(n2^w)$ (Mateescu, Dechter, & Marinescu, 2008). Since each AOBDD over $\mathcal{T}$ can be seen as an OBDD$[\wedge]_{\mathcal{C}}$, we can also have the same size bound of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ according to Proposition 6:

Corollary 3. *For each CNF formula with n variables and treewidth w, there exists some chain $\mathcal{C}$ over PV such that the corresponding ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ has a size $O(n2^w)$.*

Next we will show that there exists some class of formulas such that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ is exponentially more space-efficient than ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ ($i > j$). Therefore, we need some method to estimate the size of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. In the ROBDD community, Sieling and Wegener's bound (Sieling & Wegener, 1993) is typically used for estimating ROBDD size. Specifically speaking, given a chain $\mathcal{C}$ over PV, a variable x, a variable set $X = \{x' : x' \prec_{\mathcal{C}} x\}$, and a formula φ with m distinct sub-formulas that are obtained by conditioning φ on all assignments over X and depend on x, the ROBDD over $\mathcal{C}$ representing φ contains exactly m vertices labeled by x. Here we generalize Sieling and Wegener's bound to estimate the sizes of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$:

Proposition 7. *Let φ be a formula, let x be a variable in $Vars(\varphi)$, let X be the set of variables in $Vars(\varphi)$ less than $glb(\varphi)$ over chain $\mathcal{C}$ (resp. tree $\mathcal{T}$), and let Ψ be $\{\psi : \exists \omega \in 2^X . \psi$ is a factor of the $\wedge_{\widehat{i}}$-decomposition (resp. $\wedge_{\widehat{\mathcal{T}}}$ -decomposition) of $\lfloor \varphi|_\omega \rfloor\}$. In the corresponding ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ if it exists), the number of vertices labeled by x is equal to the number of distinct factors in Ψ with the appearance of x.*

According to Proposition 7 and Observation 3d, we can show that if $i > j$, Equation (1) can be represented by an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ in linear size, but the size of the equivalent

ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ is exponential in n, where $\mathcal{C}$ is depicted in Figure 2a. For a general chain $\mathcal{C}$, we arrive at a similar conclusion by simply substituting the kth variable in $\mathcal{C}$ for x_k in Equation (1). Therefore, together with Proposition 6, we can state the following succinctness results:

Theorem 5. *Given a chain $\mathcal{C}$ over variables and two integers i and j, ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \leq_s$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ iff $i \leq j$.*

The above theorem indicates that when moving along each solid arc in Figure 1, the succinctness of language increases. In particular, ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $(i \geq 2)$ is strictly more succinct than two previous languages ROBDD and ROBDD-L_∞ over $\mathcal{C}$ by Theorem 1. That is, ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ can further mitigate the size explosion problem of ROBDD from a theoretical perspective.

6.2 Succinctness Relationship Between ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$

In this subsection, we analyze the succinctness relationship between ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. The case $i = 0$ is immediate from the fact ROBDD$[\wedge_{\widehat{\mathcal{T},0}}]_{\mathcal{T}} \subseteq$ ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$. In order to obtain the succinctness results between ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $(i > 0)$, we also need to state the following fact:

Observation 14. Given a non-trivial formula φ that can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ rooted at u, $|V(\mathcal{G}_u)| \leq 3 \cdot |Vars(\varphi)| \cdot 2^{dep(\mathcal{T})}$. Therefore, if $dep(\mathcal{T}) < \infty$, then for each language L, ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}} \geq_s$ L.

It is obvious that ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}} \leq_s$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ since each ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ is an OBDD$[\wedge_i]_{\mathcal{C}}$. Moreover, given a tree $\mathcal{T}$ with an infinite path $\mathcal{C}'$, it is obvious ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}'}$ $\subseteq$ ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$. According to Theorem 5, we know that ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}'} \not\leq_s$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}'}$ when $i > 0$. Since ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}'} \subseteq$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, we immediately know ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}'} \not\leq_s$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. Then since ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}'} \subseteq$ ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$, we know ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}} \not\leq_s$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. According to Observation 14 and the above facts, we can state the following succinctness results:

Theorem 6. *ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}} =_s$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ if $i = 0$ or $dep(\mathcal{T}) < \infty$, and ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ $<_s$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ otherwise.*

The above theorem indicates that when moving along each dashed-dotted arc in Figure 1, the succinctness of language does not decrease. Moreover, we can manage the space efficiency of ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ by managing the depth of tree.

6.3 Succinctness Relationship Between Subsets in OBDD$[\wedge]_{\prec}$ and Existing Languages

We now discuss the succinctness relationship between the two families of canonical subsets in OBDD$[\wedge]_{\prec}$ and some existing languages. First, the succinctness relationship between ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$) and the ones in {ROBDD, ROBDD-L_∞, AOBDD} is immediate from Theorems 1, 5 and 6. Second, ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ is strictly less succinct than Decision-DNNF and d-DNNF. According to Proposition 7, the negation of Equation (1) corresponds to an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($\mathcal{C}$ in Figure 2a) with an exponential size, while the same formula can be represented by a Decision-DNNF formula with a linear size. Therefore, we know that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} <_s$ Decision-DNNF and then ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} <_s$ d-DNNF. This suggests that there is still plenty of room to mitigate the size explosion problem further. To some extent, this example also reflects that the space efficiency of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, as well as that of ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$, is highly dependent on the condition whether we can obtain adequate strict $\wedge$-decompositions after assigning some variables in practical knowledge bases. Third, the succinctness relationship between ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($i \geq 1$) and CSDD$_{\mathcal{V}}$ is incomparable. The reader can verify that if the variables denoting the shifting distance are less than the other variables, some class of circular bit-shift functions can be represented in ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ in polysize. However, circular bit-shift functions cannot be represented in CSDD$_{\mathcal{V}}$ in polysize (Pipatsrisawat, 2010). Finally, it is known that ROBDD $>_s$ MODS (Darwiche & Marquis, 2002), and we also know that ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}} >_s$ MODS if some variable in $\mathcal{T}$ has an infinite number of disjoint subtrees each of which has at least two vertices.

7. Operating Efficiency of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$

We now analyze the time efficiency of operating ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$. First, we present some tractable algorithms for ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$. Then, we evaluate the tractability with respect to the criteria proposed by Darwiche and Marquis (2002). Finally, we propose a new notion called *rapidity* to describe the operating efficiency, and we provide the rapidity results for ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$.

Each operation can be seen as a relation between inputs and outputs. All operations are performed on formula sequences in language L, including the set of all unary sequences on L (denoted by L itself), the set of all binary sequences on L (denoted by L $\times$ L), and the set of all sequences on L (denoted by L*). These formula sequences are referred to as the *primary input information* of operations. The remaining input information is called *supplementary input information*; particularly, $\{nil\}$ is used for convenience to denote no supplementary information. Each query operation outputs some piece of information of the corresponding formula sequence, and each transformation operation outputs an element in $\{nil\} \cup$ L, where nil is used to handle cases in which the transformation operations of incomplete languages fail for some formulas. We can write an operation as a set of triples with the form $(\varphi_1, \ldots, \varphi_n, \alpha, \beta)$, where $\varphi_1, \ldots, \varphi_n$ represent the primary information, α represents the supplementary information, and β represents the output information. Now we are ready to introduce the operations mentioned in the KC map (Darwiche & Marquis, 2002):

Definition 9 (query operations)**.** Given a language L, we focus on the following query operations:

- Consistency (resp. validity) check: CO (resp. VA) on L is a relation between L $\times$ $\{nil\}$ and $\{false, true\}$ such that for every formula $\varphi \in$ L, $(\varphi, nil, true) \in CO$ (resp. VA) iff φ is consistent (resp. valid), and $(\varphi, nil, false) \in CO$ (resp. VA) iff φ is inconsistent (resp. invalid);
- Clausal entailment check: Let L$_C$ be the set of clauses. CE on L is a relation between L $\times$ L$_C$ and $\{false, true\}$ such that for every formula $\varphi \in$ L and every clause $\delta \in$ L$_C$, $(\varphi, \delta, true) \in CE$ iff $\varphi \models \delta$ holds, and $(\varphi, \delta, false) \in CE$ iff $\varphi \not\models \delta$ holds;

- Implicant check: Let L_T be the set of terms. IM on L is a relation between $\mathrm{L} \times \mathrm{L}_T$ and $\{false, true\}$ such that for every formula $\varphi \in \mathrm{L}$ and every term $\gamma \in \mathrm{L}_T$, $(\varphi, \gamma, true) \in IM$ iff $\gamma \models \varphi$ holds, and $(\varphi, \gamma, false) \in IM$ iff $\gamma \not\models \varphi$ holds;
- Equivalence (resp. sentential entailment) check: EQ (resp. SE) on L is a relation between $\mathrm{L} \times \mathrm{L} \times \{nil\}$ and $\{false, true\}$ such that for every pair of formulas φ and φ' in L, $(\varphi, \varphi', nil, true) \in EQ$ (resp. SE) iff $\varphi \equiv \varphi'$ (resp. $\varphi \models \varphi'$) holds, and $(\varphi, \varphi', nil, false) \in EQ$ (resp. SE) iff $\varphi \not\equiv \varphi'$ (resp. $\varphi \not\models \varphi'$) holds;
- Model counting (resp. enumeration): Let $\mathcal{P}(\Omega^{all})$ be the powerset of the set of all assignments. CT (resp. ME) on L is a relation between $\mathrm{L} \times \{nil\}$ and $\mathbb{N}$ (resp. $\mathcal{P}(\Omega^{all})$) such that for every formula $\varphi \in \mathrm{L}$ and every natural number n (resp. every set of models Ω), $(\varphi, nil, n) \in CT$ (resp. $(\varphi, nil, \Omega) \in ME$) iff n (resp. Ω) is the number (resp. set) of models of φ.

Definition 10 (transformation operations)**.** Given a language L, we focus on the following transformation operations:

- Conditioning: Let Ω^{all} be the set of all assignments. CD on L is a relation between $\mathrm{L} \times \Omega^{all}$ and $\{nil\} \cup \mathrm{L}$ such that for every pair of formulas φ and φ' in L and every assignment ω, $(\varphi, \omega, \varphi')$ iff $\varphi|_\omega \equiv \varphi'$, and (φ, ω, nil) iff $\varphi|_\omega$ cannot be represented in L.
- Forgetting: Let $\mathcal{P}(PV)$ be the power set over PV. FO on L is a relation between $\mathrm{L} \times \mathcal{P}(PV)$ and $\{nil\} \cup \mathrm{L}$ such that for every pair of formulas φ and φ' in L and every variable set X, (φ, X, φ') iff $\exists X.\varphi \equiv \varphi'$, and (φ, X, nil) iff $\exists X.\varphi$ cannot be represented in L.
- Singleton forgetting: SFO on L is a relation between $\mathrm{L} \times PV$ and $\{nil\} \cup \mathrm{L}$ such that for every pair of formulas φ and φ' in L and every variable x, (φ, x, φ') iff $\exists x.\varphi \equiv \varphi'$, and (φ, x, nil) iff $\exists x.\varphi$ cannot be represented in L.
- Conjunction (resp. disjunction): $\wedge C$ (resp. $\vee C$) on L is a relation between $\mathrm{L}^* \times \{nil\}$ and $\{nil\} \cup \mathrm{L}$ such that for every finite set of formulas $\varphi_1, \ldots, \varphi_{n+1}$ in L, $(\varphi_1, \ldots, \varphi_n, nil, \varphi_{n+1}) \in \wedge C$ (resp. $\vee C$) iff $\varphi_1 \wedge \cdots \wedge \varphi_n \equiv \varphi_{n+1}$ (resp. $\varphi_1 \vee \cdots \vee \varphi_n \equiv \varphi_{n+1}$), and $(\varphi_1, \ldots, \varphi_n, nil, nil) \in \wedge C$ (resp. $\vee C$) iff $\varphi_1 \wedge \cdots \wedge \varphi_n$ (resp. $\varphi_1 \vee \cdots \vee \varphi_n$) cannot be represented in L.
- Bounded conjunction (resp. disjunction): $\wedge BC$ (resp. $\vee BC$) on L is a relation between $\mathrm{L} \times \mathrm{L} \times \{nil\}$ and $\{nil\} \cup \mathrm{L}$ such that for any three formulas φ_1, φ_2 and φ_3 in L, $(\varphi_1, \varphi_2, nil, \varphi_3) \in \wedge BC$ (resp. $\vee BC$) iff $\varphi_1 \wedge \varphi_2 \equiv \varphi_3$ (resp. $\varphi_1 \vee \varphi_2 \equiv \varphi_3$), and $(\varphi_1, \varphi_2, nil, nil) \in \wedge BC$ (resp. $\vee BC$) iff $\varphi_1 \wedge \varphi_2$ (resp. $\varphi_1 \vee \varphi_2$) cannot be represented in L.
- Negation: $\neg C$ on L is a relation between $\mathrm{L} \times \{nil\}$ and $\{nil\} \cup \mathrm{L}$ such that for every pair of formulas φ and φ' in L, $(\varphi, nil, \varphi') \in \neg C$ iff $\neg\varphi \equiv \varphi'$, and (φ, nil, nil) iff $\neg\varphi$ cannot be represented in L.

Given an operation OP on language L (denoted by $OP(\mathrm{L})$), its *domain* is the set $\{(\varphi_1, \ldots, \varphi_n, \alpha) : \exists \beta \in \Gamma.(\varphi_1, \ldots, \varphi_n, \alpha, \beta) \in \Gamma\}$. We say that an algorithm Alg *performs* $OP(\mathrm{L})$ iff the following two conditions hold: its set of inputs is equal to the domain of L; and for every input $(\varphi_1, \ldots, \varphi_n, \alpha)$, if Alg outputs $\beta \neq nil$ (the current input is called *valid*), then $(\varphi_1, \ldots, \varphi_n, \alpha, \beta) \in OP(\mathrm{L})$, and if Alg reports failure, then $(\varphi_1, \ldots, \varphi_n, \alpha, nil) \in OP(\mathrm{L})$.

7.1 Tractable Operation Algorithms for OBDD$[\wedge]_{\prec}$ and Its Subsets

In this subsection we discuss the tractable algorithms for OBDD$[\wedge]_{\prec}$ and its subsets. Since ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$) is a subset of d-DNNF, we can immediately use the tractable algorithms performing CT and ME on d-DNNF to perform the corresponding operations on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$). Since ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ is equivalent to AOBDD, we can immediately use the binary unweighted version of apply algorithm proposed by Mateescu, Dechter, and Marinescu (2008) to perform $\wedge BC$ on ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$. We present three algorithms to perform CD on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, CD on ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, SE on ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ and $\vee BC$ on ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$. For CO, VA and EQ on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, since each vertex has exactly one copy in vertex-table, we can perform these three types of queries in $O(1)$. For CE (resp. IM), we can perform it by first calling the CD algorithm and then calling the CO (resp. VA) algorithm on the result of the first step. These algorithms can facilitate the development of software systems in which ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ or ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ is adopted; on the other hand, they immediately reveal some of the tractability results that will be discussed in the next subsection. Note that in the following algorithms, we will employ the dynamic programming technique to improve their efficiency; that is, we will use hash tables to store the vertices that have already been processed.

7.1.1 Conditioning

We now discuss the conditioning operation, which permits reasoning under some partial observations in the real world. We present an operation algorithm called Condition (in Algorithm 2), which performs the conditioning of an OBDD$[\wedge]_{\prec}$ on an assignment. With some additional steps, we can then use this algorithm to perform CD on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$. Our algorithm is similar to the conditioning algorithm for d-DNNF, but we need to show that the operating results are in OBDD$[\wedge]_{\prec}$. This algorithm has the following property:

Proposition 8. Condition *can perform CD on OBDD*$[\wedge]_{\prec}$, Condition(u, ω) *terminates in* $O(|\mathcal{G}_u| + |\omega|)$, *and the size of its output is not greater than* $|\mathcal{G}_u|$.

Algorithm 2: Condition(u, ω)

```
  Input: an OBDD$[\wedge]_{\prec}$ vertex $u$, and an assignment $\omega$
  Output: an OBDD$[\wedge]_{\prec}$ vertex representing $\vartheta(u)|_{\omega}$
1 if $H(u) \neq nil$ then return $H(u)$
2 else if $u$ is a leaf vertex then return $H(u) \leftarrow u$
3 else if $sym(u) = false \in \omega$ then $H(u) \leftarrow$ Condition$(lo(u), \omega)$
4 else if $sym(u) = true \in \omega$ then $H(u) \leftarrow$ Condition$(hi(u), \omega)$
5 else $H(u) \leftarrow \langle sym(u), \{$Condition$(v, \omega) : v \in Ch(u)\}\rangle$
6 $H(u) \leftarrow$ Make$(H(u))$
7 return $H(u)$
```

Given an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, the conditioning operation can be performed by first calling Condition to obtain an OBDD$[\wedge_i]_{\mathcal{C}}$ and then calling Decompose to transform the resulting OBDD$[\wedge_i]_{\mathcal{C}}$ into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. Given an ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, the conditioning operation can be performed by the following steps: first, call Condition to obtain an OBDD$[\wedge_{\mathcal{T}}]_{\mathcal{T}}$; second, transform the resulting OBDD$[\wedge_{\mathcal{T}}]_{\mathcal{T}}$ into ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ according to Proposition 5. Therefore, we can draw the following conclusion:

Corollary 4. *CD on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ can be respectively performed in time $O(|Vars(u)| \cdot |V(\mathcal{G}_u)| + |\omega|)$ and $O(|\mathcal{G}_u| + |\omega|)$.*

Now we present another algorithm called ConditionMin in Algorithm 3 which is tailored for a special type of conditioning. In detail, we condition ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ on a unit assignment whose variable is the minimum variable in the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. This operation will be used in Subsection 7.3. Algorithm ConditionMin has the following property:

Proposition 9. *Given each internal ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex u and each Boolean constant b,* ConditionMin*(u) outputs the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex representing $\vartheta(u)|_{glb(u)=b}$, the size of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ rooted at the output is less than $|\mathcal{G}_u|$, and the time complexity of* ConditionMin *is bounded by $O(|Vars(u)|)$.*

Algorithm 3: ConditionMin(u, b)

```
   Input: an internal vertex $u$ in ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, and a Boolean value $b$
   Output: the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex representing $\vartheta(u)|_{glb(u)=b}$
1  if $u$ is a $\diamond$-vertex then
2      if $b = false$ then return $lo(u)$
3      else return $hi(u)$
4  else
5      Search the child $v \in Ch(u)$ with the appearance of $glb(u)$
6      if $b = false$ then $w \leftarrow lo(v)$
7      else $w \leftarrow hi(v)$
8      if $sym(w) = \wedge$ then $Ch(u) \leftarrow Ch(u) \setminus \{v\} \cup Ch(w)$
9      else $Ch(u) \leftarrow Ch(u) \setminus \{v\} \cup \{w\}$
10     return Make($u$)
11 end
```

7.1.2 Sentential Entailment of ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$

Now we turn to another tractable querying operation which checks sentential entailment relation between two ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$s. We first present an observation to show that this problem can be solved recursively by case analysis:

Observation 15. Given two internal vertices u and v in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, they fulfill at least one of the following cases:

(a) $glb(u)$ and $glb(v)$ are incomparable over $\prec_{\mathcal{T}}$: $\vartheta(u) \not\models \vartheta(v)$;
(b) $sym(v) = \wedge$: $\vartheta(u) \models \vartheta(v)$ iff $\forall w \in Ch(v).\vartheta(u) \models \vartheta(w)$;
(c) $sym(v) \in PV$ and $sym(v) \prec_{\mathcal{T}} Vars(u)$: $\vartheta(u) \models \vartheta(v)$ iff $\forall w \in Ch(v).\vartheta(u) \models \vartheta(w)$;
(d) $sym(u) = sym(v) \in PV$: $\vartheta(u) \models \vartheta(v)$ iff $\vartheta(lo(u)) \models \vartheta(lo(v))$ and $\vartheta(hi(u)) \models \vartheta(hi(v))$;
(e) $sym(u) \in PV$ and $sym(u) \prec_{\mathcal{T}} Vars(v)$: $\vartheta(u) \models \vartheta(v)$ iff $\vartheta(lo(u)) \models \vartheta(v)$ and $\vartheta(hi(u)) \models \vartheta(v)$; and

(f) $sym(u) = \wedge$ and $glb(u) \prec_{\mathcal{T}} glb(v)$: $\vartheta(u) \models \vartheta(v)$ iff $\vartheta(mch_x(u)) \models \vartheta(v)$, where $x = ch_{\mathcal{T}}(glb(u) \rightsquigarrow glb(v))$.

Based on the above observation, we present the sentential entailment algorithm called ENTAILTREE (in Algorithm 4) for ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$s. For each single calling of ENTAILTREE(w,w'), w is a meta-vertex in $\mathcal{G}_u$, and w' is a vertex in $\mathcal{G}_v$. Therefore, the number of single callings is not more than $|\mathcal{G}_u| \cdot |V(\mathcal{G}_v)|$ according to Observation 7. We can use a preprocessing routine to compute all greatest lower bounds for meta-vertices of $\mathcal{G}_u$ and vertices of $\mathcal{G}_v$ in $O(|Vars(u)| \cdot |\mathcal{G}_u|)$ and in $O(|\mathcal{G}_v|)$, respectively. Then it is easy to see that given two ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$s rooted at u and v, each single calling of ENTAILTREE can be done in $O(|Vars(u)| + |Vars(v)|)$. Therefore, we can draw the following conclusion:

Proposition 10. ENTAILTREE *can perform SE on ROBDD*$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$*, and* ENTAILTREE*(u, v) terminates in* $O((|Vars(u)| + |Vars(v)|) \cdot |\mathcal{G}_u| \cdot |V(\mathcal{G}_v)|)$.

According to Proposition 5, each OBDD$[\wedge_{\mathcal{T}}]_{\mathcal{T}}$ rooted at v can be converted into the equivalent ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ in $O(|Vars(v)| \cdot |V(\mathcal{G}_v)|)$, and thus the sentential entailment between two OBDD$[\wedge_{\mathcal{T}}]_{\mathcal{T}}$s can be checked within the same time complexity.

Algorithm 4: ENTAILTREE(u, v)

```
Input: two ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ vertices $u$ and $v$ over tree $\mathcal{T}$
Output: answer whether $\vartheta(u) \models \vartheta(v)$
1  if $H(v) \neq nil$ then return $H(v)$
2  else if $u = \bot$ or $v = \top$ then $H(u, v) \leftarrow true$
3  else if $u = \top$ or $v = \bot$ then $H(u, v) \leftarrow false$
4  else if $glb(u)$ and $glb(v)$ are incomparable then $H(u, v) \leftarrow false$
5  else if $sym(v) = \wedge$ then $H(u, v) \leftarrow \bigwedge_{w \in Ch(v)}$ ENTAILTREE$(u, w)$
6  else if $sym(v) \prec_{\mathcal{T}} Vars(u)$ then
   $H(u, v) \leftarrow$ ENTAILTREE$(u, lo(v)) \wedge$ ENTAILTREE$(u, hi(v))$
7  else if $sym(u) = sym(v)$ then
   $H(u, v) \leftarrow$ ENTAILTREE$(lo(u), lo(v)) \wedge$ ENTAILTREE$(hi(u), hi(v))$
8  else if $sym(u) = \wedge$ and $glb(u) \prec_{\mathcal{T}} glb(v)$ then
9      $x \leftarrow ch_{\mathcal{T}}(glb(u) \rightsquigarrow glb(v))$
10     $H(u, v) \leftarrow$ ENTAILTREE$(mch_x(u), v)$
11 else $H(u, v) \leftarrow$ ENTAILTREE$(lo(u), v) \wedge$ ENTAILTREE$(hi(u), v)$
12 return $H(u, v)$
```

Example 3. We show how ENTAILTREE checks the entailment relation between two ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$s as depicted in Figure 6. We first call ENTAILTREE(u_1, v_1) and the condition on Line 7 is satisfied. Then we recursively call ENTAILTREE(u_2, v_2) and ENTAILTREE(u_3, v_3). In the process of calling ENTAILTREE(u_2, v_2), the condition on Line 5 is satisfied and then we recursively call ENTAILTREE(u_2, v_4) and ENTAILTREE(u_2, v_5). In the process of calling ENTAILTREE(u_2, v_4), the condition on Line 8 is satisfied and then the output is just the output of recursively calling ENTAILTREE(u_4, v_4). In the process of calling ENTAILTREE(u_4, v_4), the condition on Line 7 is satisfied and then we recursively call ENTAILTREE$(\langle \bot \rangle, v_6)$ and ENTAILTREE(u_7, v_7). Both recursive calls return *true*, and thus

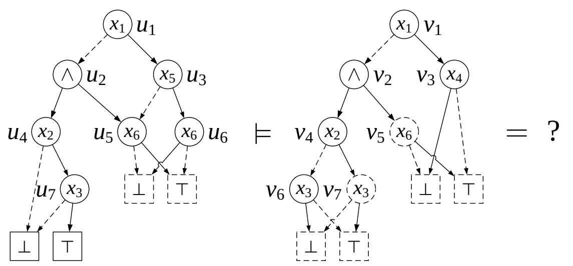


Figure 6: Example of running algorithm EntailTree on two ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$s, where $\mathcal{T}$ is depicted in Figure 2c

EntailTree(u_2, v_4) returns *true*. The process of calling EntailTree(u_2, v_5) is similar to that of calling EntailTree(u_2, v_4), and the output is *true*. Therefore, EntailTree(u_2, v_2) returns *true*. In the process of calling EntailTree(u_3, v_3), the condition on Line 4 is satisfied, and then $false$ is returned. Finally, we know the left ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ does not entail the right one.

7.1.3 Bounded Disjunction of ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$

Now we discuss the bounded disjoining algorithm for ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$. Note that due to the incompleteness of ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, we cannot perform $\vee BC$ using only $\wedge BC$ and $\neg C$; that is, $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$. For example, $x_2 \wedge x_3$, $x_2 \wedge \neg x_3$ and $(x_2 \wedge x_3) \vee (x_2 \wedge \neg x_3) \equiv x_2$ can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ ($\mathcal{T}$ is depicted in Figure 2b), but neither $\neg(x_2 \wedge x_3)$ nor $\neg(x_2 \wedge \neg x_3)$ can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$. We first point out two cases where the disjunction of two vertices in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$. For notational convenience, given two ∧-vertices u and v in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ with $glb(u) = glb(v)$, and given a variable $x \in Ch_{\mathcal{T}}(glb(u))$, we call $(mch_x(u), mch_x(v))$ a *meta-pair* between u and v; in particular, we call it a different meta-pair if $mch_x(u) \neq mch_x(v)$. For example, $\langle\langle \neg x_5\rangle, \langle x_5\rangle\rangle$ is the only different meta-pair between $\langle \wedge, \{\langle x_2\rangle, \langle \neg x_5\rangle\}\rangle$ and $\langle \wedge, \{\langle x_2\rangle, \langle x_5\rangle\}\rangle$.

Observation 16. Let u and v be two internal vertices in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ satisfying the following conditions: a) $sym(u) = \wedge$ and $glb(u) \prec_{\mathcal{T}} glb(v)$; and b) $\vartheta(u) \not\models \vartheta(v)$ and $\vartheta(v) \not\models \vartheta(u)$. We know that $\vartheta(u) \vee \vartheta(v)$ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$.

Observation 17. Let u and v be two ∧-vertices in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ satisfying the following conditions: a) $glb(u) = glb(v)$; and b) $\vartheta(u) \not\models \vartheta(v)$ and $\vartheta(v) \not\models \vartheta(u)$. We know that $\vartheta(u) \vee \vartheta(v)$ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, if there exist more than one different meta-pair between u and v.

Given three ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ ($\mathcal{T}$ depicted in Figure 2c) vertices $u = \langle \wedge, \{\langle x_2\rangle, \langle x_5\rangle\}\rangle$, $v = \langle \neg x_2\rangle$ and $v' = \langle \wedge, \{\langle \neg x_2\rangle, \langle \neg x_5\rangle\}\rangle$, we know that $\vartheta(u) \vee \vartheta(v)$ and $\vartheta(u) \vee \vartheta(v')$ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ according to Observations 16 and 17, respectively. Now we are ready to propose a polytime disjoining algorithm called DisjoinTree (in Algorithm 5). The cases in which either u or v is a leaf vertex are implicitly mentioned on Lines 2–3. The cases processed on Lines 4–12 are similar to the steps of disjoining two ROBDDs. Otherwise, according to Observations 16–17, $\vartheta(u) \vee \vartheta(v)$ can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ only if there exists exactly one different meta-pair between u and v. Then the corresponding cases are processed on Lines 13–18.

For each single calling of DisjoinTree(w, w'), w and w' are meta-vertices in $\mathcal{G}_u$ and $\mathcal{G}_v$, respectively. Therefore, we can use the algorithm EntailTree to compute the entailment relation between the meta-vertices of $\mathcal{G}_u$ and $\mathcal{G}_v$ in the preprocessing stage. By using hash table, the number of single callings of EntailTree is not more than $|\mathcal{G}_u| \cdot |\mathcal{G}_v|$, and thus the calling time is bounded by $O((|Vars(u)| + |Vars(v)|) \cdot |\mathcal{G}_u| \cdot |\mathcal{G}_v|)$. When the entailment relation on Lines 2–3 is already known, each single calling of DisjoinTree can be done in $O(|Vars(u)| + |Vars(v)|)$. According to the fact that the number of single callings of DisjoinTree is not greater than $|\mathcal{G}_u| \cdot |\mathcal{G}_v|$, we can draw the following conclusion:

Proposition 11. DisjoinTree *can perform* $\vee BC$ *on ROBDD*$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$*, and* DisjoinTree*($u$, v) terminates in* $O((|Vars(u)| + |Vars(v)|) \cdot |\mathcal{G}_u| \cdot |\mathcal{G}_v|)$.

Algorithm 5: DisjoinTree(u, v)

```
Input: two ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$s rooted at $u$ and $v$
Output: the ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ representing $\vartheta(u) \vee \vartheta(v)$ if it exists, and report failure
        otherwise
1  if $H(u,v) \neq nil$ then return $H(u,v)$
2  else if $u \models v$ then $H(u,v) \leftarrow v$
3  else if $v \models u$ then $H(u,v) \leftarrow u$
4  else if $u$ is a $\diamond$-vertex and $sym(u) \prec_{\mathcal{T}} Vars(v)$ then
5      $u_1 \leftarrow$ DisjoinTree$(lo(u), v)$; $u_2 \leftarrow$ DisjoinTree$(hi(u), v)$
6      $H(u,v) \leftarrow \langle sym(u), u_1, u_2 \rangle$
7  else if $v$ is a $\diamond$-vertex and $sym(v) \prec_{\mathcal{T}} Vars(u)$ then
8      $v_1 \leftarrow$ DisjoinTree$(u, lo(v))$; $v_2 \leftarrow$ DisjoinTree$(u, hi(v))$
9      $H(u,v) \leftarrow \langle sym(v), v_1, v_2 \rangle$
10 else if $sym(u) = sym(v) \neq \wedge$ then
11     $v_1 \leftarrow$ DisjoinTree$(lo(u), lo(v))$; $v_2 \leftarrow$ DisjoinTree$(hi(u), hi(v))$
12     $H(u,v) \leftarrow \langle sym(u), v_1, v_2 \rangle$
13 else if $sym(u) = sym(v) = \wedge$ and $glb(u) = glb(v)$ then
14     if there is exactly one different meta-pair $(w, w')$ between $u$ and $v$ then
15         $H(u,v) \leftarrow \langle \wedge, (Ch(u) \setminus \{w\}) \cup \{$DisjoinTree$(w, w')\} \rangle$
16         $H(u,v) \leftarrow$ Merge$(H(u,v))$
17     else report failure
18 else report failure
19 $H(u,v) \leftarrow$ Make$(H(u,v))$
20 return $H(u,v)$
```

Example 4. We show that how DisjoinTree disjoins two ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$s as depicted in Figure 7. We first call DisjoinTree(u_1, v_1) and the condition on Line 10 is satisfied. Then we recursively call DisjoinTree(u_2, v_2) and DisjoinTree(u_3, v_3) on Line 11. In the process of calling DisjoinTree(u_2, v_2), the condition on Line 13 is satisfied and we can confirm that there is exactly one different meta-pair (u_5, v_5) between u_2 and v_2. DisjoinTree(u_5, v_5) outputs $\langle \top \rangle$, and thus DisjoinTree(u_2, v_2) outputs $w_2 = u_4 = v_4$. In the

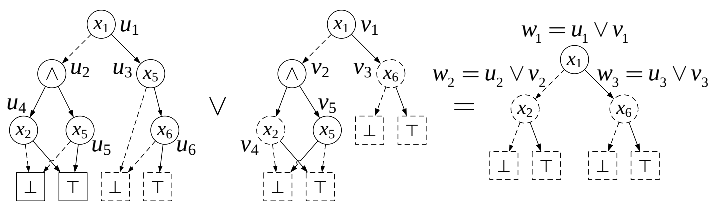


Figure 7: Example of running algorithm DisjoinTree on two ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$s, where $\mathcal{T}$ is depicted in Figure 2c

process of calling DisjoinTree(u_3, v_3), we know $u_3 \models v_3$ (i.e., the condition on Line 2 is satisfied) and thus $w_3 = v_3$ is returned. Finally, the output of DisjoinTree(u_1, v_1) is w_1.

7.2 Tractability Evaluation

We examine the tractability of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ with respect to the criteria proposed by Darwiche and Marquis (2002). That is, for an operation $OP \neq ME$, we say that L satisfies **OP** iff there exists some polytime algorithm performing OP; and for ME, we say that L satisfies **ME** iff there exists some algorithm performing ME(L) in polytime in the size of input and the number of models. Before we discuss the tractability results, we present the following conclusion, which will be used to prove some negative results of tractability:

Proposition 12. *Given any two ROBDD*$[\wedge_{\widehat{i}}]_{\mathcal{C}}$*s* ($i \geq 1$) *rooted at* u *and* v*, the problem of deciding whether* $\vartheta(u) \models \vartheta(v)$ *holds or not is co-NP-complete.*

Table 2 summarizes both query-related and transformation-related tractability results of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$. The tractability results of MODS, d-DNNF, Decision-DNNF [6], SDD$_{\mathcal{V}}$ and CSDD$_{\mathcal{V}}$ (Darwiche & Marquis, 2002; Darwiche, 2011; Van den Broeck & Darwiche, 2015) are also shown for comparison. We explain the results briefly here and include the detailed proof in the appendix. The results of ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$ are known previously since it is equivalent to ROBDD over $\mathcal{C}$. The other positive results can be understood from three aspects: a) the polytime operating algorithms in Subsection 7.1; b) the fact that if an ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ has some $\wedge$-vertex, then its negation cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$, and otherwise we can employ the negation algorithm for ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$ to perform the negation; and c) the fact that m ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$s ($dep(\mathcal{T}) < \infty$) can be conjoined in $O(m \cdot |X|^2 \cdot 2^{dep(\mathcal{T})})$ because the size of each ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ over X is not more than $|X| \cdot 2^{dep(\mathcal{T})}$. The negative result of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($i > 0$) on **SE** immediately follows from Proposition 12, which implies many other negative results. The negative result of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($i > 0$) on $\neg$**C** occurs because performing negation on some class of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s will introduce an exponential number of new vertices. The negative results of ROBDD$[\wedge_i]_{\widehat{\mathcal{T}}}$ on $\vee$**C** and **FO** are due to the facts that there exists some x with infinite descendants X and that each term on X can be represented in ROBDD$[\wedge_i]_{\widehat{\mathcal{T}}}$ in a linear size. Finally, the negative result of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($dep(\mathcal{T}) = \infty$) on $\wedge$**C** is due to the following facts: there is an infinite path $\mathcal{C}'$ in $\mathcal{T}$, and each clause on $V(\mathcal{C}')$ can be represented in ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ in a linear size.

6. The tractability results of Decision-DNNF can be proved in a similar fashion to those of d-DNNF.

L	**CO**	**VA**	**CE**	**IM**	**EQ**	**SE**	**CT**	**ME**
ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $(i > 0)$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$	$\surd$	$\surd$
ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
CSDD$_{\mathcal{V}}$/Decision-CSDD$_{\mathcal{V}}$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
MODS	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
SDD$_{\mathcal{V}}$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
d-DNNF/Decision-DNNF	$\surd$	$\surd$	$\surd$	$\surd$	?	$\circ$	$\surd$	$\surd$
L	**CD**	**FO**	**SFO**	$\wedge$**C**	$\wedge$**BC**	$\vee$**C**	$\vee$**BC**	$\neg$**C**
ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$	$\surd$	$\bullet$	$\surd$	$\bullet$	$\surd$	$\bullet$	$\surd$	$\surd$
ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $(i > 0)$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ $(dep(\mathcal{T}) < \infty)$	$\surd$	$\circ$	$\surd$	$\surd$	$\surd$	$\circ$	$\surd$	$\surd$
ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ $(dep(\mathcal{T}) = \infty)$	$\surd$	$\circ$	$\surd$	$\circ$	$\surd$	$\circ$	$\surd$	$\surd$
CSDD$_{\mathcal{V}}$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\surd$
Decision-CSDD$_{\mathcal{V}}$	$\bullet$	$\circ$	?	$\circ$	?	$\circ$	?	$\surd$
MODS	$\surd$	$\surd$	$\surd$	$\bullet$	$\surd$	$\bullet$	$\bullet$	$\bullet$
SDD$_{\mathcal{V}}$	$\surd$	$\bullet$	$\surd$	$\bullet$	$\surd$	$\bullet$	$\surd$	$\surd$
d-DNNF/Decision-DNNF	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	?

Table 2: Polytime queries and transformations of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$, Decision-CSDD$_{\mathcal{V}}$, CSDD$_{\mathcal{V}}$, Decision-DNNF, and d-DNNF, where $\mathcal{C}$, $\mathcal{T}$, and $\mathcal{V}$ are over PV, $\surd$ means "satisfies", $\bullet$ means "does not satisfy", and $\circ$ means "does not satisfy unless P = NP"

Theorem 7. *The results in Table 2 hold.*

According to the results in Table 2, we know that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $(i \geq 2)$ is as tractable as ROBDD-L_∞. In other words, compared with ROBDD-L_∞ over $\mathcal{C}$, ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ indeed improves the succinctness under the premise of maintaining the same tractability. Moreover, we know that ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ is at least as tractable as ROBDD; in particular, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ even has more tractability than ROBDD if $dep(\mathcal{T}) < \infty$. Therefore, on the basis of meeting the expressivity requirement, users may improve the time efficiency of operations on ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ by compressing the depth of $\mathcal{T}$. We also know that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ is more tractable than d-DNNF (resp. Decision-DNNF) since the latter does not support **EQ**. Finally, the tractability of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $(i > 0)$ is incomparable with that of CSDD$_{\mathcal{V}}$, because ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ supports **CD** but not **SE** or $\neg$**C**, while CSDD$_{\mathcal{V}}$ supports **SE** and $\neg$**C** but not **CD**. However, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ is more tractable than Decision-CSDD$_{\mathcal{V}}$ and CSDD$_{\mathcal{V}}$, because CSDD$_{\mathcal{V}}$ does not support **CD**, **SFO**, $\vee$**BC** or $\wedge$**BC**, and it is unknown whether or not Decision-CSDD$_{\mathcal{V}}$ supports **SFO** (resp. $\vee$**BC** and $\wedge$**BC**). We emphasize that some tractability results of ROBDD$[\wedge_{\widehat{\mathcal{T}},\infty}]_{\mathcal{T}}$ (i.e., AOBDD over $\mathcal{T}$) have been obtained by

Mateescu, Dechter, and Marinescu (2008), and Fargier and Marquis (2006), but the results about **CD**, **FO**, **SFO**, $\wedge$**C**, $\vee$**C**, $\vee$**BC** and $\neg$**C** are first reported in this paper.

7.3 New Perspective on Time Efficiency

Due to distinct succinctness, it can sometimes be insufficient to compare the time efficiency of two languages solely by comparing their tractability. We use an example to illustrate this problem. Consider an operation OP, and two languages L and L$'$ such that L does not satisfy **OP** but L$'$ satisfy **OP**. Assume that the number of basic arithmetic operations involved in performing OP on $(\varphi_1, \ldots, \varphi_n, \alpha)$ (resp. $(\varphi'_1, \ldots, \varphi'_n, \alpha)$) be 2^m (resp. n), where $\varphi_i \in \mathrm{L}$ (resp. $\varphi'_i \in \mathrm{L}'$), and $m = |\alpha| + \sum_{1 \leq i \leq n} |\varphi_i|$ (resp. $n = |\alpha| + \sum_{1 \leq i \leq n} |\varphi'_i|$). Then, performing OP on $(\varphi_1, \ldots, \varphi_n, \alpha)$ can be exponentially (in m) more time-consuming than performing OP on $(\varphi'_1, \ldots, \varphi'_n, \alpha)$ when $n = 2^{m^2}$. To overcome this problem, we define a new notion to compare time efficiency from a different perspective, supplementing the concept of tractability:

Definition 11 (rapidity)**.** An operation OP on a canonical language L_1 is *at most as rapid as* OP on another canonical language L_2 ($\mathrm{L}_1 \leq_r^{OP} \mathrm{L}_2$), iff for each algorithm Alg performing OP on L_1, there exists some polynomial p and some algorithm Alg$'$ performing OP on L_2 such that for every valid input $(\varphi_1, \ldots, \varphi_n, \alpha)$ of OP on L_1 and every valid input $(\varphi'_1, \ldots, \varphi'_n, \alpha)$ of OP on L_2 satisfying $\varphi_i \equiv \varphi'_i$ $(1 \leq i \leq n)$, Alg$'(\varphi'_1, \ldots, \varphi'_n, \alpha)$ can be done in time $p(t + |\varphi_1| + \cdots + |\varphi_n| + |\alpha|)$, where α is any element of supplementary information and t is the running time of Alg$(\varphi_1, \ldots, \varphi_n, \alpha)$.

According to the above definition, we immediately have that ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}} =_r^{OP}$ ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$. Note that rapidity relation is also conditionally transitive: If $\mathrm{L}_1 \leq_e \mathrm{L}_2 \leq_e \mathrm{L}_3$, then $\mathrm{L}_1 \leq_r^{OP} \mathrm{L}_2$ and $\mathrm{L}_2 \leq_r^{OP} \mathrm{L}_3$ imply $\mathrm{L}_1 \leq_r^{OP} \mathrm{L}_3$. Because ROBDD is strictly more succinct than MODS and satisfies **ME**, each ROBDD can be transformed into the equivalent MODS formula φ in polytime in $|\varphi|$. Therefore, for each operation OP, MODS $\leq_r^{OP}$ ROBDD. This conclusion explains to some extent why practical applications prefer ROBDD to MODS. Similarly, OP on MODS is at most as rapid as OP on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$).

Assume that an operation OP satisfies $\mathrm{L}_1 \leq_r^{OP} \mathrm{L}_2$. Let $(\varphi_1, \ldots, \varphi_n, \alpha)$ be a valid input of OP on L_1 and $(\varphi'_1, \ldots, \varphi'_n, \alpha)$ be a valid input of OP on L_2, where $\varphi_i \equiv \varphi'_i$ for $1 \leq i \leq n$. We know the time cost of performing OP on $(\varphi'_1, \ldots, \varphi'_n, \alpha)$ increases at most polynomial times in $|\alpha| + \sum_{1 \leq i \leq n} |\varphi_i|$ than that of performing OP on $(\varphi_1, \ldots, \varphi_n, \alpha)$. In particular, if performing OP on $(\varphi_1, \ldots, \varphi_n, \alpha)$ can be done in polytime, then performing OP on $(\varphi'_1, \ldots, \varphi'_n, \alpha)$ can also be done in polytime in $|\alpha| + \sum_{1 \leq i \leq n} |\varphi_i|$. Therefore, for applications needing canonical languages, we suggest that users choose a language by the following steps rather than by the traditional viewpoint of the KC map: first, identify the set $\mathcal{L}$ of canonical languages meeting the expressivity requirement; second, identify the set $\mathcal{OP}$ of necessary operations and identify the subset $\mathcal{L}'$ of $\mathcal{L}$ meeting the tractability requirement; third, add each language $\mathrm{L} \in \mathcal{L}$ satisfying $\exists \mathrm{L}' \in \mathcal{L}' \forall OP \in \mathcal{OP}. \mathrm{L}' \leq_r^{OP} \mathrm{L}$ to $\mathcal{L}'$; and finally, choose one of the most succinct languages in $\mathcal{L}'$.

Before we present the results of rapidity, we propose two algorithms called ConvertDown and ConvertTree (in Algorithms 6–7) to transform ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$

into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ ($i \leq j$), respectively. We will show that ConvertDown and ConvertTree terminate in polytime in the sizes of outputs. Then, we will use these two algorithms to prove some rapidity results.

Algorithm 6: ConvertDown(u)

```
Input: an ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ rooted at $u$
Output: the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ representing $\vartheta(u)$, 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 $\diamond$-vertex then
4      ConvertDown($lo(v)$); ConvertDown($hi(v)$)
5      $H(u) \leftarrow \langle sym(u), H(lo(u)), H(hi(u))\rangle$
6  else
7      $V \leftarrow \{v \in Ch(u) : |Vars(v)| > i\}$
8      if $V = \emptyset$ then $H(u) \leftarrow u$
9      else if $V = \{v\}$ then $H(u) \leftarrow \langle \wedge, (Ch(u) \setminus \{v\}) \cup \{$ConvertDown$(v)\}\rangle$
10     else
11         $v_1 \leftarrow$ ConvertDown(ConditionMin($\langle \wedge, V\rangle, false$))
12         $v_2 \leftarrow$ ConvertDown(ConditionMin($\langle \wedge, V\rangle, true$))
13         $v' \leftarrow \langle \min\{sym(v) : v \in V\}, v_1, v_2\rangle$
14         $H(u) \leftarrow \langle sym(u), (Ch(u) \setminus V) \cup \{v'\}\rangle$
15     end
16 end
17 return $H(u)$
```

We first explain the algorithm ConvertDown. Here we use the method described in Corollary 1a to obtain the $\wedge_{\widehat{i}}$-decomposition from a $\wedge_{\widehat{j}}$-decomposition. Due to the canonicity of ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$, each single calling of ConvertDown will create at least one vertex in the resulting ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. Therefore, the number of single callings of ConvertDown is not greater than the number of vertices in the resulting ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. According to Proposition 9, each single calling of ConvertDown(w) terminates in $O(|Vars(w)|) = O(|Vars(v)|)$, where v is the output. Therefore, we can draw the following conclusion:

Proposition 13. *Given an ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$,* ConvertDown *can transform it into the equivalent ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($i \leq j$) in $O(|Vars(v)| \cdot |V(\mathcal{G}_v)|)$, where v is the root of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$.*

Example 5. We show how ConvertDown transforms the ROBDD$[\wedge_{\widehat{2}}]_{\mathcal{C}}$ in Figure 3a into the ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ in Figure 3b. We first call ConvertDown(u_1). Then we recursively call ConvertDown(u_2) and ConvertDown(u_3) on Line 4. The process of calling the latter is similar to the process of calling the former. In the process of calling ConvertDown(u_2), because both children of u_2 have more than one variable, V after running Line 7 is equal to $\{u_4, u_5\}$. ConditionMin($\langle \wedge, V\rangle, false$) and ConditionMin($\langle \wedge, V\rangle, true$) on Lines 11–12 return $w_1 = \langle \wedge, \{u_7, u_5\}\rangle$ and $w_2 = \langle \wedge, \{u_8, u_5\}\rangle$, respectively. Then we recursively call ConvertDown(w_1) and ConvertDown(w_2) on Lines 11–12. In the process of calling ConvertDown(w_1), since both sub-graphs rooted at u_7 and u_5 have more than

Algorithm 7: ConvertTree(u)

```
Input: an ROBDD[$\wedge_{\widehat{i}}$]$_{\mathcal{C}}$ rooted at $u$
Output: the equivalent ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_{\mathcal{T}}$ if it exists, and failure otherwise
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 $\diamond$-vertex then
4      if $glb_{\prec_{\mathcal{T}}}(u) \neq sym(u)$ then report failure
5      ConvertTree($lo(v)$); ConvertTree($hi(v)$)
6      $H(u) \leftarrow \langle sym(u), H(lo(u)), H(hi(u))\rangle$
7  else
8      $\mathcal{G} \leftarrow \langle Ch(u), \{(v,w),(w,v) : glb_{\prec_{\mathcal{T}}}(v) \prec_{\mathcal{T}} glb_{\prec_{\mathcal{T}}}(w)\}\rangle$
9      Let $\mathcal{G}_1,\ldots,\mathcal{G}_m$ be the strongly connected components of $\mathcal{G}$
10     for each $1 \leq k \leq m$ do
11         $v_k \leftarrow \langle \wedge, V(\mathcal{G}_k)\rangle$
12         if $|V(\mathcal{G}_k)| > 1$ then
13             if $glb_{\prec_{\mathcal{T}}}(v_k) \notin Vars(v_k)$ then report failure
14             $v_1 \leftarrow$ ConvertTree(ConditionMin($v_k, false$))
15             $v_2 \leftarrow$ ConvertTree(ConditionMin($v_k, true$))
16             $u_k \leftarrow \langle glb_{\prec_{\mathcal{T}}}(v_k), v_1, v_2\rangle$
17         else $u_k \leftarrow$ ConvertTree($v_k$)
18     end
19     $H(u) \leftarrow \langle \wedge, \{u_1,\ldots,u_m\}\rangle$
20     if $H(u)$ is not bounded by $i$ then report failure
21 end
22 return $H(u)$
```

one variable, V after running Line 7 is equal to $\{u_7, u_5\}$. ConditionMin($\langle\wedge, V\rangle, false$) and ConditionMin($\langle\wedge, V\rangle, true$) on Lines 11–12 return $w_3 = \langle\wedge, \{u_7, u_9\}\rangle$ and $w_4 = \langle\wedge, \{u_7, u_{10}\}\rangle$, respectively. Then we recursively call ConvertDown(w_3) and ConvertDown(w_4) on Lines 11–12. In the process of calling ConvertDown(w_3), V after running Line 7 is equal to $\{u_7\}$, and ConvertDown(u_7) returns v_{14}. Therefore, ConvertDown(w_3) returns v_8 on Line 9 (actually, the output is returned on Line 17, but we omit the last step here for the sake of simplicity). Similarly, ConvertDown(w_4) returns v_9. Then we get back to the process of calling ConvertDown(w_1), and this calling returns v_4 on Line 14. Similarly, ConvertDown(w_2) returns v_5. Then we get back to the process of calling ConvertDown(u_2), and this calling returns v_2 on Line 14. Similarly, ConvertDown(u_3) returns v_3. Finally, we get back to the process of calling ConvertDown(u_1), and this calling returns v_1 on Line 5.

We now explain the algorithm ConvertTree. If u is a $\diamond$-vertex, then $\vartheta(u)$ is not strictly $\wedge_{\widehat{\mathcal{T}},i}$-decomposable by Corollary 2a; therefore, $\vartheta(u)$ can be represented in ROBDD[$\wedge_{\widehat{\mathcal{T}},i}$]$_{\mathcal{T}}$ only when $glb_{\prec_{\mathcal{T}}}(u) = sym(u)$ by Observation 3c. Otherwise, we use the method described in Corollary 2a to obtain a $\wedge_{\widehat{\mathcal{T}},i}$-decomposition from $\wedge_{\widehat{i}}$-decomposition. Due to the canonicity of ROBDD[$\wedge_{\widehat{i}}$]$_{\mathcal{C}}$, each single calling of ConvertDown will create at least one vertex in the resulting ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$. Therefore, the number of single callings of ConvertTree is not greater than the number of vertices in the resulting ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$. For each single calling of ConvertTree(w), Lines 8–9 can be done in $O(|Vars(w)|^2)$. Therefore, each single calling of ConvertTree(w) can be done in $O(|Vars(w)|^2)$. Then, we can draw the following conclusion:

Proposition 14. *Given an ROBDD*$[\wedge_{\widehat{i}}]_{\mathcal{C}}$*,* ConvertTree *can transform it into the equivalent ROBDD*$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ *in* $O(|Vars(v)|^2 \cdot |V(\mathcal{G}_v)|)$ *if it exists, where* v *is the root of ROBDD*$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$*.*

We emphasize that the time complexity in Propositions 13–14 is polynomial in the size of output (not input), and thus it does not violate the NP-hardness of SE on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($i > 0$). ConvertDown (resp. ConvertTree), together with Decompose, provides new methods to perform queries and transformations on ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$). First, we call ConvertDown (resp. ConvertTree) to transform ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$s (resp. ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s) into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s (resp. ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$s). Next, we answer the query using the outputs of the first step, or we perform the transformation on the outputs of the first step and then transform the result into ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$) by Decompose. We emphasize that according to the above process, knowledge bases can be compiled into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ by calling the conjoining algorithm for ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$) and Decompose, which provides some basis for developing a bottom-up compiler for ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, since ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$) supports $\wedge$**BC**. Since the time complexities of Decompose and ConvertDown (resp. ConvertTree) are polynomial in the sizes of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s (resp. ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$s) and the rapidity relation is conditionally transitive, we know ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}} \leq_r^{OP}$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \leq_r^{OP}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$. Moreover, if $dep(\mathcal{T}) < \infty$, we can show that each ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ can be transformed into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ in polytime in the size of output, and thus ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}} \geq_r^{OP}$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$.

We can show that for $OP \in \{CD, \wedge BC, \vee BC, SFO\}$ and $i > j$, there exists some class of ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$s satisfying the following two conditions: a) the number of new vertices resulting from performing OP on each ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ is exponential in the sizes of the equivalent ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$; and b) we can tailor an algorithm performing OP on the class of equivalent ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s in polytime. Therefore, we know that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{OP}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$, which implies similar negative rapidity results on operations FO, $\wedge C$ and $\vee C$. We can obtain the same negative rapidity results between ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ in a similar fashion when $i > 0$ and $dep(\mathcal{T}) = \infty$. Now we are ready to present the rapidity results:

Theorem 8. *Given two integers* i *and* j*, a chain* $\mathcal{C}$ *and a tree* $\mathcal{T}$ *over variables, and an operation* OP*, we can draw the following conclusions:*

(a) ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \leq_r^{OP}$ *ROBDD*$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ *if* $i \leq j$*, and for* $OP \in \{CD, FO, SFO, \wedge C, \wedge BC, \vee BC, \vee C\}$*, ROBDD*$[\wedge_{\widehat{i}}]_{\mathcal{C}} \leq_r^{OP}$ *ROBDD*$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ *iff* $i \leq j$*;*

(b) If $i = 0$ *or* $dep(\mathcal{T}) < \infty$*, ROBDD*$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}} =_r^{OP}$ *ROBDD*$[\wedge_{\widehat{i}}]_{\mathcal{C}}$*. Otherwise, ROBDD*$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ $\leq_r^{OP}$ *ROBDD*$[\wedge_{\widehat{i}}]_{\mathcal{C}}$*; and particularly for* $OP \in \{CD, FO, SFO, \wedge C, \wedge BC, \vee BC, \vee C\}$*, ROBDD*$[\wedge_{\widehat{i}}]_{\mathcal{C}} <_r^{OP}$ *ROBDD*$[\wedge_{\widehat{i}}]_{\mathcal{C}}$*.*

From the above theorem, we can observe four interesting facts. First, when moving along each solid arc or dashed-dotted arc in Figure 1, the rapidity on any operation does not decrease, and particularly the rapidity on some operations increases. Second, for a tree with finite depth, ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ has the best succinctness, tractability and operation rapidity among the two families of languages; that is, ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ is the best choice if it can meet the expressivity requirements of some specific application, and ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ can be an option for approximate reasoning even if it cannot meet the expressivity requirements. Third, according to Proposition 12, the problem of deciding the entailment relation between two ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s ($i \geq 1$) is co-NP-complete. According to Theorem 8, however, the rapidity of SE on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ does not decrease with incremental i. If we can prove the increasing is strict, then we can partition the co-NP-complete problems from some angle. We leave as an open question whether the rapidity of SE on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ increases with incremental i. Fourth, the last interesting observation is as follows:

Corollary 5. *Given each operation OP on ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ (resp. ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$) that can be performed in polytime, OP on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($i \geq j$) can also be performed in polytime in the sizes of the equivalent ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$s (resp. ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$s).*

It was mentioned that for $OP \in \{SE, SFO, \wedge BC, \vee BC\}$, OP(ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$) can be performed in polytime but OP(ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$) ($i > 0$) cannot be performed in polytime unless P = NP. Therefore, if we only consider the tractability of OP, it may create the illusion that the time efficiency of performing OP(ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$) is pessimistically lower than that of performing OP(ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$). Actually, Corollary 5 shows that OP(ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$) can also be performed in polytime in the sizes of equivalent ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$s. In particular, for $OP \in \{SFO, \wedge BC, \vee BC\}$, there exists some class of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s such that performing OP on them is exponentially more efficient than performing OP on the equivalent ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$s according to Theorem 8. In summary, according to this new perspective, an application requiring OP prefers ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ to ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$.

8. Preliminary Experimental Results

In this section, we report some preliminary experimental results of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($0 \leq i \leq \infty$), on the one hand, to show some strength and potential of the new languages, and on the other hand, to verify several succinctness results from an experimental point of view. We developed an ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler, and then we can compile a knowledge base into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($0 \leq i < \infty$) by first employing the compiler to generate an ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ and then employing ConvertDown to transform the result into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. We first compare our ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler with previous ROBDD-L_∞, CSDD$_\mathcal{V}$ and d-DNNF compilers, and then analyze the space efficiency trend of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ with incremental i. We do not compare the space efficiency of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ with that of ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$, due to the incompleteness of ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ and the fact that ROBDD$[\wedge_{\widehat{\mathcal{T},i}}]_{\mathcal{T}}$ is a subset of CSDD$_\mathcal{V}$.

The framework of our compiler is based on an exhaustive DPLL trace (depicted in Algorithm 8; Figure 8 is an example of running this algorithm), just like the frameworks of many other top-down compilers (Huang & Darwiche, 2007). The compiler exploited some technologies that have proven efficient in other compilers (Darwiche, 2004; Huang

& Darwiche, 2007; Muise et al., 2012), such as non-chronological backtracking, dynamic decomposition, implicit binary constraint propagation, and component caching. Like the ROBDD-L_∞ compiler developed by Lai, Liu, and Wang (2013), our compiler employs a SAT solver to compute the implied literals of sub-formulas to accelerate decomposing. However, because the employment of SAT engine is normally time-consuming, we use a new heuristic function during the preprocessing stage to estimate whether calling the SAT solver would have positive effects, and then we decide whether the SAT solver should be called in the compilation stage based on the estimated result. In our experiments, the chain $\mathcal{C}$ was generated by the minfill heuristic.

Algorithm 8: Compile(φ, $\mathcal{C}$)

```
Input: a CNF formula $\varphi$ and a chain $\mathcal{C}$ over $PV$
Output: an OBDD$[\wedge]_{\mathcal{C}}$ representing $\varphi$
1 if $H(\varphi) \neq nil$ then return $H(u)$
2 $\wedge_1$-Decompose $\varphi$ into $\{l_1, \ldots, l_m, \varphi'\}$, and let $v$ represent $l_1 \wedge \cdots \wedge l_m$
3 if $\varphi'$ is false then return $\langle \bot \rangle$
4 if $\varphi'$ is true then return $v$
5 Decompose $\varphi'$ into $\{\psi_1, \ldots, \psi_n\}$
6 if $n = 1$ then
7     $w_1 \leftarrow$ Compile($\varphi'|_{glb(\varphi')=false}, \mathcal{C}$); $w_2 \leftarrow$ Compile($\varphi'|_{glb(\varphi')=true}, \mathcal{C}$)
8     return $\langle \wedge, \{v\} \cup \langle glb(\varphi'), w_1, w_2 \rangle \rangle$
9 else return $\langle \wedge, \{v\} \cup \{$Compile($\psi_i, \mathcal{C}$) $: 1 \leq i \leq n\} \rangle$
```

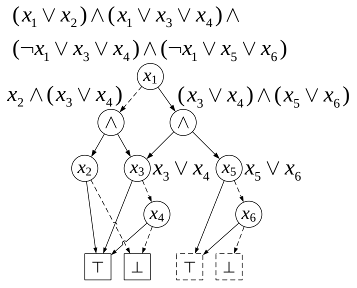

$(x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_4) \wedge$
$(\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_6)$
x_1
$x_2 \wedge (x_3 \vee x_4)$
$(x_3 \vee x_4) \wedge (x_5 \vee x_6)$
$\wedge$
$\wedge$
x_2
x_3
$x_3 \vee x_4$
x_5
$x_5 \vee x_6$
x_4
x_6
$\top$
$\bot$
$\top$
$\bot$

Figure 8: Example on how Compile runs, where $\mathcal{C}$ is depicted in Figure 2a. After x_1 is assigned *true*, we decompose the resulting formula into two parts, and detect that $x_3 \vee x_4$ has been compiled; and thus we reuse the previous result

8.1 Comparison with Previous Compilers

We compare our ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler with three compilers of two canonical languages ROBDD-L_∞ and CSDD$_{\mathcal{V}}$ and a non-canonical language d-DNNF. The three canonical languages can be seen as subsets of d-DNNF. The state-of-the-art ROBDD-L_∞, CSDD$_{\mathcal{V}}$, and d-DNNF compilers were reported by Lai, Liu, and Wang (2013), Oztok and Darwiche (2015), and Muise et al. (2012), where the latter two are called miniC2D and Dsharp, respectively. Since the ROBDD-L_∞ compiler can only run in 32-bit systems, we conducted experiments to compare ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ with the ROBDD-L_∞ and d-DNNF compilers on a computer with a 32-bit two-core 2.99GHz CPU and 3.4GB RAM. Since miniC2D can only run in 64-bit systems, we conducted experiments to compare the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler with miniC2D on a computer with a 64-bit four-core 3.6GHz CPU and 8GB RAM. Individual runs were limited to a one-hour time-out. The ROBDD-L_∞ compiler does not use any variable order heuristic to guide compilation. We employed a preprocessing program to make it use the minfill heuristic. We also set miniC2D to use the minfill heuristic to guide compilation. Our experimental results, as well as some previous results (see e.g., Muise et al., 2012), show that this heuristic can improve efficiency in most instances. The four compilers were tested on eight domains of benchmarks, including Bejing, blocksworld, emptyroom, grid, flat200, inductive-inference (II for short), iscas89 and sortnet, [7] and then we compared their compiling time and output sizes.

Table 3 shows the overall performance of the four compilers over the eight domains. Here we also report the results of another efficient d-DNNF compiler called c2d (Darwiche, 2004), which was run under the directives "-reduce" and "-dt-method 4" (Muise et al., 2012; Lai, Liu, & Wang, 2013). We have marked in boldface all numbers where the corresponding compiler performs best. Note that the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler happened to succeed in the same set of instances on both computers in our experiments. The experimental results show that the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler outperformed both the ROBDD-L_∞ compiler and miniC2D on three domains; and it succeeded in 18 (resp. 30 and 5) instances more than the ROBDD-L_∞ compiler (resp. miniC2D and c2d). The ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler and Dsharp outperformed each other on two domains. However, Dsharp succeeded in six more instances than the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler, since the latter was relatively inefficient in sortnet. The reason behind the inefficiency of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler in sortnet is that the minfill heuristic has a negative effect on this domain. Specifically, the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler succeeded in all instances in sortnet when we used the natural variable order depicted in Figure 2a. We mention that by employing the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler and Convert-Down, we can compile 214 instances into ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ under the given conditions. In other words, our ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler indirectly provides a more efficient ROBDD-L_∞ compiler. We can also compile 159 instances into ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$ (i.e., ROBDD) in a similar fashion. To our knowledge, this is the first report of many of these instances being compiled into ROBDD.

Figure 9 analyzes the detailed compiling time and resulting size performance between our ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compilers and the three other compilers for each instance across the eight domains. For convenience of comparison, we excluded the instances which are too trivial for both compilers of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ and L, where L $\in$ {ROBDD-L_∞, CSDD$_{\mathcal{V}}$, d-DNNF}; these are the cases in which both the compiling time of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ and that of L are less than 0.1s, or both resulting sizes are smaller than 100. Note that compared with ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$, ROBDD-$L_\infty$ can compress a $\wedge_1$-vertex and its children into one vertex, which normally makes that an ROBDD-L_∞ has less arcs than the equivalent ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$.

7. Bejing, blocksworld, flat200 and II come from SATLIB (http://people.cs.ubc.ca/~hoos/SATLIB/benchm .html); emptyroom, grid and sortnet are CNF formulas (https://bitbucket.org/haz/dsharp/downloads) converted from conformant planning problems described in a standard application of d-DNNF (Palacios et al., 2005); and iscas89 incorporates CNF formulas converted from the circuits iscas89 by iscas2cnf (http://vlsicad.eecs.umich.edu/BK/Slots/cache/sat.inesc.pt/~jpms/scripts/).

domain (#)	ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$	ROBDD-L_∞	CSDD$_{\mathcal{V}}$	d-DNNF	
				c2d	DSHARP
Bejing (16)	**6**	5	4	4	5
blocksworld (7)	**7**	**7**	6	6	**7**
emptyroom (28)	**28**	**28**	**28**	**28**	**28**
flat200 (100)	**100**	**100**	**100**	**100**	**100**
grid (33)	31	16	11	27	**33**
II (41)	**15**	14	9	13	13
iscas89 (35)	**24**	23	**24**	**24**	**24**
sortnet (12)	5	5	4	9	**12**
total (261)	216	198	186	211	**222**

Table 3: Comparative compiling performance between ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$, ROBDD-$L_\infty$, CSDD$_{\mathcal{V}}$ and d-DNNF, where each cell below language L refers to the number of instances compiled successfully into L under the given conditions

However, ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ can also compress all unit factors in each $\wedge$-decomposition to reduce the number of arcs. To improve comparability, we use the size of ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ to indicate the size of ROBDD-L_∞. We drew four lines $\times 0.1$, $\times 0.5$, $\times 2$ and $\times 10$ (for the sake of readability, we also drew another line $\times 1$ here). According to the ratio of the ordinate to the abscissa, these four lines partition the area of each sub-figure into five parts $(0, 0.1]$, $(0.1, 0.5]$, $(0.5, 2)$, $[2, 10)$ and $[10, \infty)$. Given two compilers of L and ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$, the compiling time (resulting size) of each instance falling into $(0, 0.1]$, $(0.1, 0.5]$, $(0.5, 2)$, $[2, 10)$ and $[10, \infty)$ indicates, respectively, that on this instance, the L compiler is an order-of-magnitude more efficient than the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler, the L compiler is at least twice but at most ten times as efficient as the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler, the L compiler is almost as efficient as the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler, the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler is at least twice but at most ten times as efficient as the L compiler, and the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler is an order-of-magnitude more efficient than the L compiler.

The experimental results in Figure 9 show that our ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler is significantly more time-efficient than the compilers of ROBDD-L_∞ and CSDD$_{\mathcal{V}}$. In fact, for 42.9% and 46.7% of non-trivial instances, the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler is at least an order-of-magnitude more time-efficient than the ROBDD-L_∞ compiler and the CSDD$_{\mathcal{V}}$ compiler, respectively. ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ is also significantly more space-efficient than both ROBDD-L_∞ and CSDD$_{\mathcal{V}}$. Choi and Darwiche (2013) demonstrated that the size of an CSDD$_{\mathcal{V}}$ can be reduced by reordering the corresponding v-tree. Similarly, improving the space efficiency of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ by reordering the corresponding chain is a future direction of research. Figure 9 also shows that from both aspects of compiling time and resulting sizes, the performance of the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler is comparable with that of DSHARP.

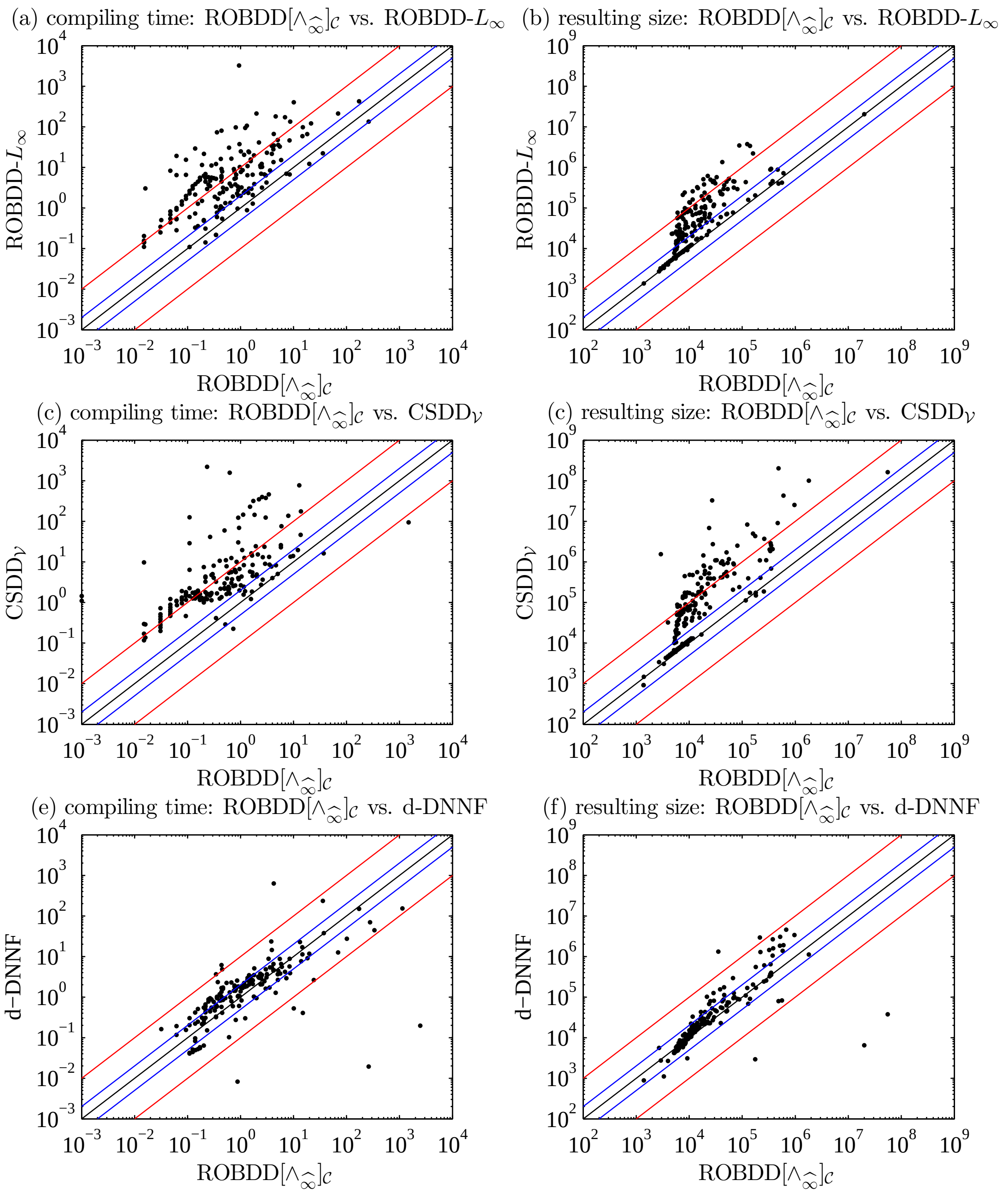


Figure 9: Compiling time and resulting sizes comparison of all non-trivial instances using ROBDD-L_∞, CSDD$_{\mathcal{V}}$, d-DNNF and ROBDD[$\wedge_{\widehat{\infty}}$]$_{\mathcal{C}}$ compilers, where (a)–(f) are six scatter plots of compiling time (in seconds) or resulting size for each instance using one previous compiler (y-axis) or our ROBDD[$\wedge_{\widehat{\infty}}$]$_{\mathcal{C}}$ compiler (x-axis)

Since ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ is a subset of d-DNNF, our compiler is naturally a d-DNNF compiler. This naturally raises the question of under what circumstances our compiler has better space-time performance than the state-of-art d-DNNF compiler mentioned previously. Note that all instances in each domain originate from the same type of applications, and thus here we simply use the domain name as the classification criterion for analysis. It is also worth mentioning that the treewidth of an instance is an important parameter to measure compiling difficulty, but we did not catch obvious pattern that the treewidth can reflect the difference between the efficiency of our compiler and that of Dsharp. We further analyze the detailed performance of these two compilers for each domain of instances in Figure 10. The experimental results show that for blocksworld, flat200 and II, the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler had better time-performance than the d-DNNF compiler, while for emptyroom, grid, iscas89 and sortnet, the latter had better time-performance than the former. In terms of space efficiency, the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler outperforms the d-DNNF compiler on Bejing, grid, flat200, II and iscas89, while the latter outperformed the former only on blocksworld and sortnet. In other words, the ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler is slightly more space-efficient than Dsharp.

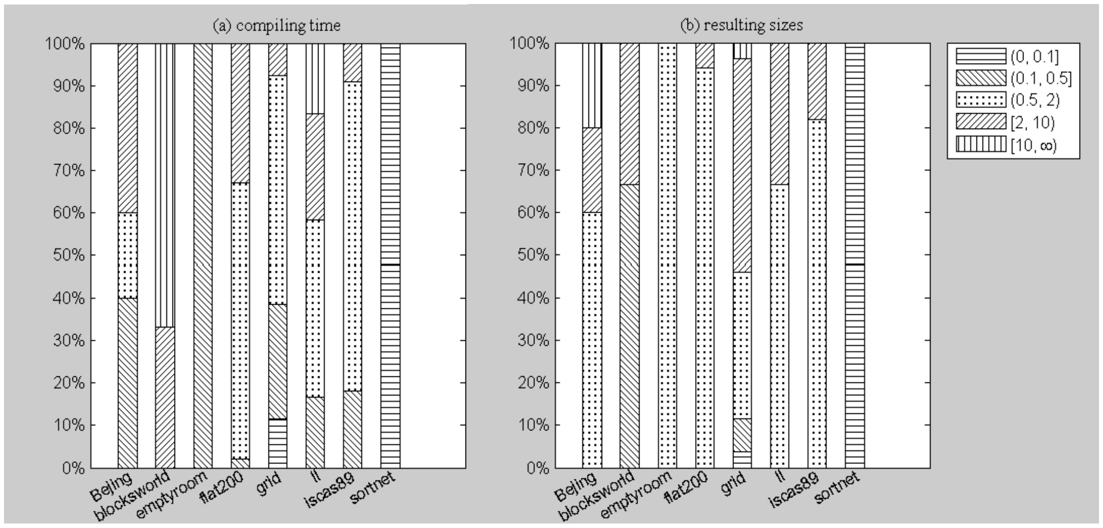


Figure 10: Detailed compiling time and resulting size performance of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ and d-DNNF compilers for instances in eight domains. The category (c, c') refers to the percentage of instances in each domain such that the ratios of compiling time (resp. resulting sizes) of d-DNNF to compiling time (resp. resulting sizes) of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ are within (c, c'). The categories $(c, c']$ and $[c, c')$ have similar meanings.

8.2 Comparing Sizes of Different ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$

We compared the sizes between ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($1 \leq i \leq 7$) and ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ to analyze the space efficiency trend of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ with incremental i. Note that we did not cover the case $i = 0$, due to two-fold reasons: a) Lai, Liu, and Wang (2013) has showed that the compilation results of ROBDD-L_{∞} are significantly smaller than those of ROBDD; and b) after being compiled into ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$, there are only two instances that cannot be transformed into ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ by employing ConvertDown, but there are 57 instances that cannot be transformed into ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$. The comparison between ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ is shown in Figure 9b, and the comparison between ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $(2 \leq i \leq 7)$ and ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ is shown in Figure 11. The experimental results reveal three interesting patterns: a) the space efficiency of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ is not worse than that of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $(1 \leq i \leq 7)$; b) with incremental i, the space efficiency of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ increases for each instance (that is, each point has a trend to converge on line $\times 1$ from Figure 11a to Figure 11e; see e.g., the points corresponding to flat200-6, s641 and s753); and c) there are many instances such that the corresponding ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s have almost the same sizes as the corresponding ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$s (actually, ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ is obviously more space-efficient than ROBDD$[\wedge_{\widehat{7}}]_{\mathcal{C}}$ only on three instances). Note that the first two observations accord with the succinctness results from Section 6.1, and that the last observation does not contradict with the succinctness results, since the strictly more succinctness is due to some special theoretical patterns but these patterns do not always work in practice.

According to the first two observations mentioned in the last paragraph, we know that ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ has the best space efficiency in practice. However, the third observation indicates that we can sometimes use ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $(1 \leq i \leq 7)$ instead in some types of applications. Consequently, we can simplify the development of compiler by abandoning or simplifying some complicated techniques (e.g., dynamically decomposing [8] on Line 5 in Algorithm 8) that were customized for our ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler, because they add extra compilation time but have little positive effect on knowledge bases generated in these applications. This raises the question of what instances are appropriate for using ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ instead of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$. Figure 12 shows that for the domains Bejing, blocksworld, emptyroom, grid, II and sortnet, we can use ROBDD$[\wedge_{\widehat{3}}]_{\mathcal{C}}$ instead of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ without obvious loss of space efficiency, and that when grid is excluded, we can even use ROBDD$[\wedge_{\widehat{2}}]_{\mathcal{C}}$ instead. This observation is particularly important for the application in which partial compilations are adopted (e.g., importance sampling for model counting (Gogate & Dechter, 2011, 2012)), since the $\wedge_{\widehat{i}}$-decomposition $(i \leq 3)$ of a CNF formula can be computed in cubic time with SAT solver as an oracle.

9. Related Work

This study is closely related to previous studies of three canonical languages that also augment BDD with certain types of decompositions.

First, Mateescu, Dechter, and Marinescu (2008) proposed a relaxation of ROBDD called AOMDD by adding tree-structured $\wedge$-decomposition and ranking $\diamond$-vertices on the same tree-structured order. They designed the apply algorithm to conjoin two AOMDDs. The only difference between the binary case of unweighted AOMDD (i.e., AOBDD) and ROBDD$[\wedge_{\widehat{\mathcal{T}},\infty}]_{\mathcal{T}}$ is that two $\diamond$-vertices in AOBDD are not connected directly, but via an intermediate $\wedge$-vertex with a unique child. We have shown that for an AOBDD, if we remove all $\wedge$-vertices with only one child, the result is an ROBDD$[\wedge_{\widehat{\mathcal{T}},\infty}]_{\mathcal{T}}$. Therefore, AOBDD over $\mathcal{T}$ is strictly both less succinct and less expressive than ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ with $\prec_{\mathcal{T}} \subset \prec_{\mathcal{C}}$.

8. For example, if we only need to detect $\wedge_1$-decomposition, then we do can discard dynamically decomposing in Algorithm 8; and if we only need to detect $\wedge_2$-decomposition by dynamically decomposing, then we do not need to separate two variables appearing in the same clause with more than two literals.

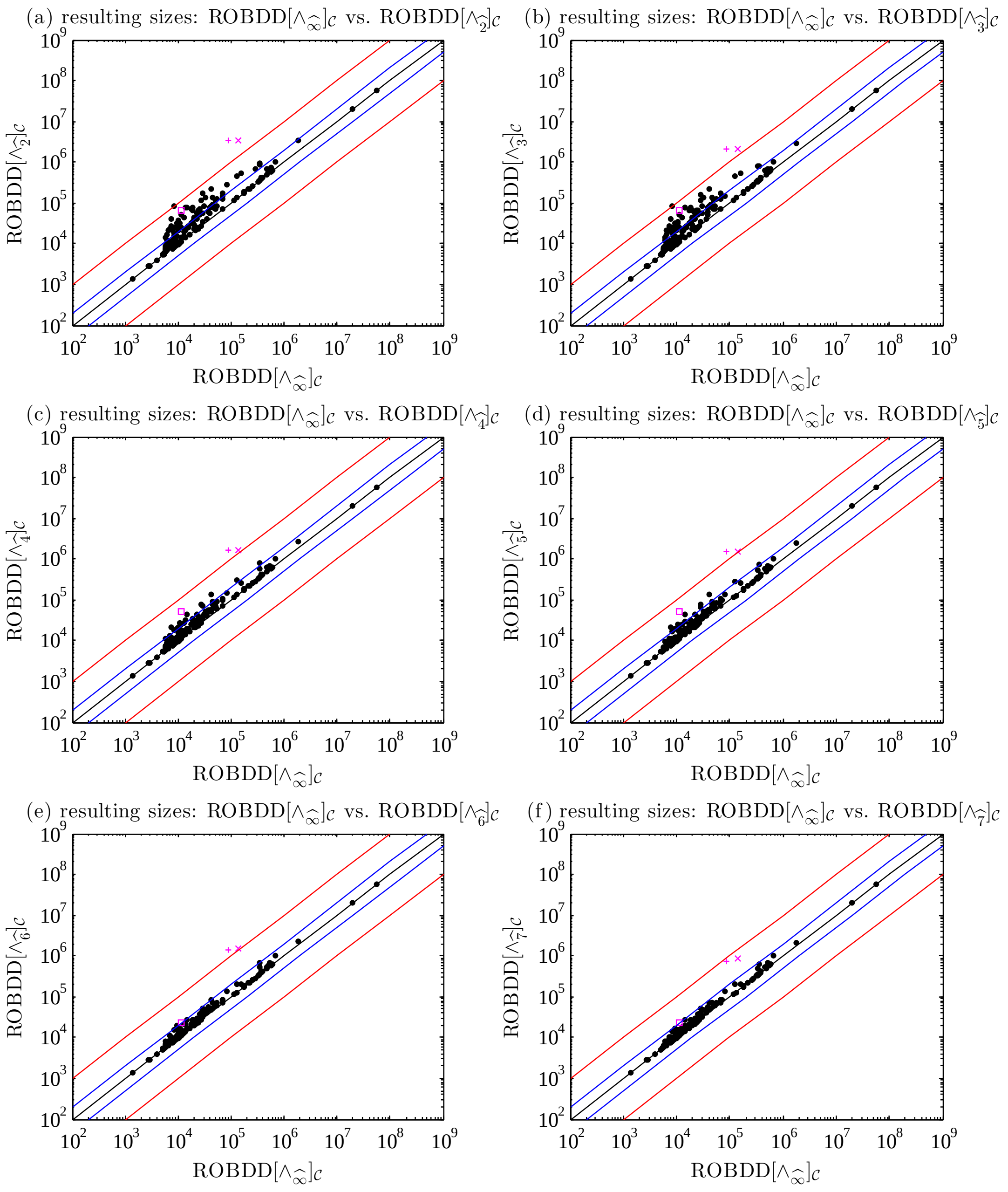


Figure 11: Trend analysis on space efficiency of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($2 \leq i \leq 7$) on all non-trivial instances, where (a)–(f) are six scatter plots of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($2 \leq i \leq 7$) size (y-axis) and ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ size (x-axis) for each instance, and the hollow box, plus sign and multiplication sign correspond to flat200-6, s641 and s753, respectively

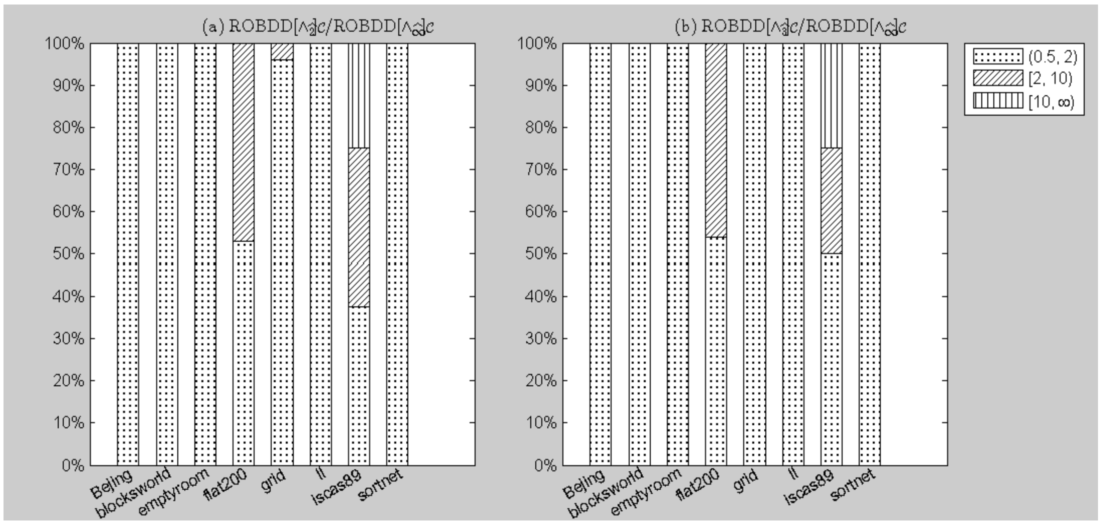


Figure 12: Detailed resulting size performance between ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ ($i = 2$ or 3) and ROBDD[$\wedge_{\hat{\infty}}$]$_\mathcal{C}$ for instances in eight domains, where each category $(0.5, 2)$ (resp. $[2, 10)$ and $[10, \infty)$) refers to the percentage of instances in each domain such that the ratios of ROBDD[$\wedge_{\hat{i}}$]$_\mathcal{C}$ sizes to ROBDD[$\wedge_{\hat{\infty}}$]$_\mathcal{C}$ sizes are in $(0.5, 2)$ (resp. $[2, 10)$ and $[10, \infty)$)

We have also shown that each operation on AOBDD over $\mathcal{T}$ is at most as rapid as the operation on ROBDD[$\wedge_{\hat{\infty}}$]$_\mathcal{C}$.

Second, Lai, Liu, and Wang (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_∞ by imposing reducedness and requiring that every internal vertex have as many as possible implied literals. They designed an algorithm called L2Inf that can transform OBDD-L into ROBDD-L_∞ in polytime in the size of input, and another algorithm called Inf2ROBDD that can transform ROBDD-L_∞ into ROBDD in polytime in the size of output. We have shown that OBDD-L over $\mathcal{C}$ can be seen as a subset of OBDD[$\wedge_1$]$_\mathcal{C}$ and ROBDD-L_∞ over $\mathcal{C}$ is equivalent to ROBDD[$\wedge_{\hat{1}}$]$_\mathcal{C}$. Therefore, L2Inf and Inf2ROBDD are two special cases of Decompose and ConvertDown, respectively. Our experimental results have shown that the compiler of ROBDD[$\wedge_{\hat{\infty}}$]$_\mathcal{C}$ is significantly more efficient than that of ROBDD-L_∞.

Last, Bertacco and Damiani (1996) added the finest negatively-disjunctive-decomposition into ROBDD to propose a representation called *Multi-Level Decomposition Diagram* (MLD-D). For completeness, $\neg$-vertices are sometimes admitted. If we introduce both conjunctive and disjunctive decompositions into ROBDD, then the resulting language is equivalent to MLDD. However, on the one hand, Bertacco and Damiani (1996) paid little theoretical attention to the space-time efficiency of MLDD; and on the other hand, they did not develop any compiler to fulfill the promise of better compiling efficiency resulting from stronger succinctness. In addition, our empirical results show that there are little disjunctive decomposition in practical benchmarks.

This study is also related to CSDD$_\mathcal{V}$ (Darwiche, 2011). Actually, each ROBDD[$\wedge_{\hat{\mathcal{T}}}$]$_\mathcal{T}$ can be seen as a Decision-CSDD$_\mathcal{V}$, and can be transformed into an CSDD$_\mathcal{V}$ in linear time, while there exists some class of CSDD$_\mathcal{V}$s that cannot be represented in ROBDD[$\wedge_{\hat{\mathcal{T}}}$]$_\mathcal{T}$.

That is, ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ can be seen as a strict subset of CSDD$_{\mathcal{V}}$, which allows us to understand the canonicity, incompleteness and tractability of ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ from the angle of CSDD$_{\mathcal{V}}$. However, according to the results obtained by Van den Broeck and Darwiche (2015), CSDD$_{\mathcal{V}}$ does not satisfy **CD**, **SFO**, $\vee$**BC** or $\wedge$**BC**, but ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ does satisfy them; furthermore, to our knowledge, it is still unknown whether Decision-CSDD$_{\mathcal{V}}$ satisfies **SFO**, $\vee$**BC** or $\wedge$**BC**. Moreover, our ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler is obviously more efficient than the state-of-the-art CSDD$_{\mathcal{V}}$ compiler. Finally, the succinctness relationship between ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($i > 0$) and CSDD$_{\mathcal{V}}$ is incomparable, and thus a future direction of generalizing this work is to exploit the bounded $\wedge$-decomposition to relax the v-tree-structured order of CSDD$_{\mathcal{V}}$, which may help identify more succinct canonical representations.

Finally, we discuss the relationship between OBDD$[\wedge]_{\prec}$ and four non-canonical languages:

- The first two are d-DNNF and Decision-DNNF mentioned previously. OBDD$[\wedge]_{\prec}$ is a strict subset of Decision-DNNF, which is a strict subset of d-DNNF. Both Decision-DNNF and d-DNNF are strictly more succinct than ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, but are slightly less tractable since they do not support **EQ**. For the rapidity of operation, given each operation ROBDD supports in polytime, ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ can also support it in time polynomial in the sizes of the equivalent ROBDDs; neither Decision-DNNF nor d-DNNF possesses this property. From a practical perspective, the efficiency of our ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ is comparable with that of the state-of-the-art d-DNNF compilers.
- The third one is called AND-OBDD (Wegener, 2000) which introduces conjunctive-nondeterministic nodes into OBDD. That is, AND-OBDD augments OBDD with $\wedge$-vertices which are not restricted to representing $\wedge$-decompositions. Therefore, OBDD$[\wedge]_{\prec}$ is a strict subset of AND-OBDD over $\prec$. AND-OBDD is a highly succinct language (e.g., each CNF formula can be represented in AND-OBDD in a linear size) at the cost of low tractability; in particular, it does not qualify as a target compilation language since **CE** is not satisfied.
- The last one is called ordered decomposable decision graph (Fargier & Marquis, 2006). Each ordered decomposable decision graph over order $\prec$ (O-DDG$_{\prec}$) is a Decision-DNNF such that for each $\vee$-vertex $(\neg x \wedge \varphi) \vee (x \wedge \psi)$ and its $\vee$-descendant $(\neg x' \wedge \varphi') \vee (x' \wedge \psi')$, $x \prec x'$. That is, O-DDG$_{\prec}$ and OBDD$[\wedge]_{\prec}$ are identical with each other. In particular, a strongly ordered DDG (SO-DDG) requires that for every pair of variables $x \prec x'$ appearing in the O-DDG, each $\vee$-vertex $(\neg x' \wedge \varphi') \vee (x' \wedge \psi')$ have some $\vee$-ancestor $(\neg x \wedge \varphi) \vee (x \wedge \psi)$. It is obvious that each OBDD$[\wedge_{\mathcal{T}}]_{\mathcal{T}}$ can be seen as SO-DDG$_{\prec_{\mathcal{T}}}$. However, on the one hand, Fargier and Marquis (2006) did not explore the canonical subsets of O-DDG$_{\prec}$, and on the other hand, they did not study whether SO-DDG$_{\prec}$ supports **SE** and the transformation tractability mentioned in the KC map. Moreover, Fargier and Marquis (2006) did not provide any compiling method for O-DDG$_{\prec}$ or its subsets.

10. Conclusions

In this paper, we proposed two families of canonical representations ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$. We analyzed their theoretical properties in terms of the existing criteria of expressivity, succinctness and tractability, as well as the new criterion rapidity. These results provide an important complement to the existing KC map. We also developed an efficient ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ compiler, which significantly advances the state-of-the-art of compiling efficiency of canonical representations, and has a compiling efficiency even comparable with that of Dsharp.

Some specific languages in the two families seem slightly more attractive than the others. First, ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$, ROBDD$[\wedge_{\widehat{1}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},\infty}]_{\mathcal{T}}$ cover the three existing languages ROBDD, ROBDD-L_∞ and AOBDD in the literature, respectively. Second, ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ is the most interesting complete language, because it has the best succinctness and rapidity. Third, ROBDD$[\wedge_{\widehat{2}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{3}}]_{\mathcal{C}}$ can sometimes be used instead of ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$ without obvious loss of space efficiency, resulting in both the simplification of compiler development and possible improvements in compiling efficiency. Fourth, ROBDD$[\wedge_{\widehat{\mathcal{T}},2}]_{\mathcal{T}}$, ROBDD$[\wedge_{\widehat{\mathcal{T}},3}]_{\mathcal{T}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},\infty}]_{\mathcal{T}}$ are interesting due to their high tractability, which allows them to serve as intermediaries to perform operations on ROBDD$[\wedge_{\widehat{2}}]_{\mathcal{C}}$, ROBDD$[\wedge_{\widehat{3}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\infty}}]_{\mathcal{C}}$, respectively. Fifth, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ ($dep(\mathcal{T}) < \infty$) has the best succinctness, tractability and operational rapidity among the two families of languages; therefore, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ is the best choice if it can meet the expressivity requirements of a specific application, and it can be an option for approximate reasoning otherwise.

The notion of rapidity sheds new light on identifying more succinct canonical representations without the worry of losing the tractability of KC languages. We believe that there is still plenty of room to identify canonical languages in d-DNNF that are more succinct than the existing ones. Intrinsically, ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ can be seen as data structures that relax the linear orderedness of ROBDD to some extent. Therefore, a future direction of generalizing this work is to exploit $\wedge_i$-decomposition to relax the vtree-structured order of CSDD$_{\mathcal{V}}$, which has the potential to identify new canonical languages with more succinctness than both ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and CSDD$_{\mathcal{V}}$.

Acknowledgements

We are grateful to the anonymous reviewers who suggested many improvements. We would also thank Umut Oztok for providing useful information about their SDD compiler, and Bo Yang for valuable discussions on the paper. This research was supported by the National Natural Science Foundation of China under grants 61402195, 61133011, 61370156 and 61503074, the China Postdoctoral Science Foundation under grant 2014M561292, and the Program for New Century Excellent Talents in University under grant NCET-13-0724.

Appendix A. Proofs

In this appendix, we present proofs for those non-obvious properties mentioned previously.

A.1 Proof of Observation 1

For the first observation, it is obvious that $\bigwedge_{\psi \in \lfloor \Psi \rfloor} \psi \equiv \bigwedge_{\psi \in \Psi} \psi \equiv \varphi \equiv \lfloor \varphi \rfloor$. For $1 \leq i \leq m$, ψ_i depends on a variable iff φ depends on it. Since each $\psi \in \lfloor \Psi \rfloor$ is non-trivial and does not share any variable with other formulas in Ψ, we know $\lfloor \Psi \rfloor$ is a $\wedge$-decomposition of $\lfloor \varphi \rfloor$.

For the second observation, we first prove $PI(\varphi) = PI(\psi_1) \cup \cdots \cup PI(\psi_m)$:

- For $1 \leq k \leq m$, we show that each $\delta \in PI(\psi_k)$ is a prime implicate of φ. If there exists some $\delta' \in PI(\varphi)$ such that $\delta' \models \delta$, we know $\psi_k \models \delta'$ since δ' does not share any variable with other factors in Ψ. Obviously, there exists some $\delta'' \in PI(\psi_k)$ such that $\delta'' \models \delta' \models \delta$, which implies that $\delta'' = \delta' = \delta$. Therefore, $\delta \in PI(\varphi)$.
- For each $\delta \in PI(\varphi)$, we show that there exists some $1 \leq i \leq m$ such that $\delta \in PI(\psi_i)$. Obviously, there exists some $1 \leq i \leq m$ such that $\psi_i \models \delta$, and otherwise $\varphi \not\models \delta$. Therefore, there exists some $\delta' \in PI(\psi_i) \subseteq PI(\varphi)$ such that $\delta' \models \delta$. That is, $\delta = \delta' \in PI(\psi_i)$.

We next prove $IP(\varphi) = \{\gamma_1 \wedge \cdots \wedge \gamma_m : \gamma_i \in IP(\psi_i)\}$. Obviously, $\gamma \models \varphi$ iff for each $1 \leq i \leq m$, $\gamma \models \psi_i$ (that is, there exists some $\gamma'' \in IP(\psi_i)$ such that $\gamma \models \gamma''$). Therefore, for each $\gamma \in IP(\varphi)$, there exists some $\gamma' \in \{\gamma_1 \wedge \cdots \wedge \gamma_m : \gamma_i \in IP(\psi_i)\}$ such that $\gamma' \models \gamma$ and $\gamma' \models \varphi$, and thus $\gamma' = \gamma$. That is, $\gamma \in \{\gamma_1 \wedge \cdots \wedge \gamma_m : \gamma_i \in IP(\psi_i)\}$. For each $\gamma \in \{\gamma_1 \wedge \cdots \wedge \gamma_m : \gamma_i \in IP(\psi_i)\}$, $\gamma \models \varphi$ and γ does not entail another $\gamma' \in \{\gamma_1 \wedge \cdots \wedge \gamma_m : \gamma_i \in IP(\psi_i)\}$. Therefore, we have $\gamma \in IP(\varphi)$.

For the last observation, the direction from left to right is obvious. For the converse direction, we know that for each $\psi \in \Psi$, there exists exactly one factor $\psi' \in \Psi'$ such that $Vars(\psi) = Vars(\psi')$. According to the second observation, if $PI(\psi) \not\subseteq PI(\psi')$, then for each implicate $\delta \in PI(\psi) \setminus PI(\psi')$, there exists another factor $\psi'' \in \Psi'$ such that $\delta \in PI(\psi'')$; that is, ψ' and ψ'' shares some variable, which is impossible. Therefore, we know that $PI(\psi) \subseteq PI(\psi')$. Similarly, we know that $PI(\psi') \subseteq PI(\psi)$. Therefore, $\psi \equiv \psi'$. Then we know that Ψ is equivalent to Ψ'. $\square$

A.2 Proof of Proposition 1

We first show that from the viewpoint of equivalence, there exists a one-to-one correspondence between the $\wedge$-decompositions of φ and the disjoint partitions of $PI(\varphi)$. Obviously, for a disjoint partition $\{\Psi_1, \ldots, \Psi_m\}$ of $PI(\varphi)$, $\{\bigwedge_{\delta \in \Psi_i} \delta : 1 \leq i \leq m\}$ is a $\wedge$-decomposition of φ. In addition, according to Observation 1b, we can obtain a disjoint partition $\{\Psi_1, \ldots, \Psi_m\}$ of $PI(\varphi)$ from each $\wedge$-decomposition of φ. Therefore, we only need to prove the existence of the finest disjoint partition of $PI(\varphi)$. We prove it by contradiction.

Assume that $PI(\varphi)$ has two different finest disjoint partitions $\{\Psi_1, \ldots, \Psi_m\}$ and $\{\Psi'_1, \ldots, \Psi'_n\}$. We know that there exists some $1 \leq i \leq m$ such that $\Psi_i \not\subseteq \Psi'_j$ for each $1 \leq j \leq n$. $\{\Psi_i \cap \Psi'_j : 1 \leq j \leq n, \Psi_i \cap \Psi'_j \neq \emptyset\}$ is a disjoint partition of Ψ_i. Therefore, $\{\Psi_1, \ldots, \Psi_m\}$ is not a finest disjoint partition since $\{\Psi_1, \ldots, \Psi_{i-1}\} \cup \{\Psi_{i+1}, \ldots, \Psi_m\} \cup \{\Psi_i \cap \Psi'_j : 1 \leq j \leq n, \Psi_i \cap \Psi'_j \neq \emptyset\}$ is finer than it.

A.3 Proof of Proposition 2

For the first conclusion, it is obvious that $\lceil \Psi \rceil_X$ is $\wedge$-decomposition of φ, and we just need to show that $\lceil \Psi \rceil_X$ is finer than each $\wedge$-decomposition Ψ' of φ. According to Observation 1a, we have that $\lfloor \Psi' \rfloor$ is a $\wedge$-decomposition of $\lfloor \varphi \rfloor$. We denote $\lceil \lfloor \Psi' \rfloor \rceil_X$ by Ψ'', and show that $\lceil \Psi \rceil_X$ is finer than Ψ' by the following two steps:

- $\lceil \Psi \rceil_X$ is finer than Ψ'': The fact that Ψ is the finest $\wedge$-decomposition of $\lfloor \varphi \rfloor$ implies that Ψ is finer than $\lfloor \Psi' \rfloor$. Obviously, $\lceil \Psi \rceil_X \setminus \Psi$ is equal to $\Psi'' \setminus \lfloor \Psi' \rfloor$. Therefore, we know $\lceil \Psi \rceil_X$ is finer than Ψ''.
- Ψ'' is finer than Ψ': We just need to show that for each $\psi \in \Psi''$, there exists some $\psi' \in \Psi'$ such that $Vars(\psi) \subseteq Vars(\psi')$. If $|Vars(\psi)| = 1$, it is obvious that there exists some

$\psi' \in \Psi'$ such that $Vars(\psi) \subseteq Vars(\psi')$. Otherwise, there exists some $\psi' \in \Psi'$ such that $\psi = \lfloor \psi' \rfloor$; that is, $Vars(\psi) \subseteq Vars(\psi')$.

Next we prove the second conclusion. We denote $\{\bigwedge_{\psi \in \lceil \Psi \rceil_X \text{ and } Vars(\psi) \subseteq Vars(\psi')} \psi : \psi' \in \Psi'\}$ by Ψ''. Since $\lceil \Psi \rceil_X$ is the finest $\wedge$-decomposition of φ, we know the following two facts: for each $\psi \in \lceil \Psi \rceil_X$, there exists exactly one $\psi' \in \Psi'$ such that $Vars(\psi) \subseteq Vars(\psi')$, which implies that Ψ'' is finer than Ψ'; and for each $\psi' \in \Psi'$ and $x \in Vars(\psi')$, there exists exactly one $\psi \in \lceil \Psi \rceil_X$ such that $x \in Vars(\psi) \subseteq Vars(\psi')$, which implies that Ψ' is finer than Ψ''. According to Observation 1c, Ψ' and Ψ'' are equivalent to each other.

A.4 Proof of Proposition 3

From the viewpoint of equivalence, if there does not exist any factor $\psi \in \Psi$ such that $|Vars(\psi)| > i$, then Ψ is the finest $\wedge_i$-decomposition, since each $\wedge_i$-decomposition is not finer than the finest $\wedge$-decomposition. We next assume there exists some factor in Ψ with more than i variables. If $i = 0$, the conclusion is obvious. Otherwise, we denote $\{\bigwedge_{\psi \in \Psi \text{ and } |Vars(\psi)| > i} \psi\} \cup \{\psi \in \Psi : |Vars(\psi)| \leq i\}$ by Ψ'. It is obvious that Ψ' is a $\wedge_i$-decomposition. According to Observation 1c, we show by contradiction that Ψ' is finer than every $\wedge_i$-decomposition Ψ'' of φ. We know Ψ' is the unique finest $\wedge_i$-decomposition since the fineness-relation is partially ordered. We assume that there exists some $\wedge_i$-decomposition Ψ'' of φ such that Ψ' is not finer than Ψ''. Therefore, there exists some formula $\psi' \in \Psi'$ such that each $\psi'' \in \Psi''$ satisfies $Vars(\psi') \not\subseteq Vars(\psi'')$. According to Observation 1b and Proposition 2, ψ' is strictly $\wedge$-decomposable respecting $\{Vars(\psi'') : \psi'' \in \Psi''\}$. S-ince $\{Vars(\psi'') : \psi'' \in \Psi''\}$ has at most one element with more than i variables, ψ' has a strict $\wedge_i$-decomposition Ψ'''. If $Vars(\psi') \leq i$, we know $\psi' \in \Psi$. Therefore, Ψ is not finer than $(\Psi' \setminus \{\psi'\}) \cup \Psi'''$, which contradicts with the condition that Ψ is the finest $\wedge$-decomposition. Otherwise, we know $\psi' = \bigwedge_{\psi \in \Psi \text{ and } |Vars(\psi)| > i} \psi$. Since each factor in $\{\psi \in \Psi : |Vars(\psi)| > i\}$ has more than i variables, $\{\psi \in \Psi : |Vars(\psi)| > i\}$ is not finer than Ψ'''. That is, Ψ is not finer than $(\Psi' \setminus \{\psi'\}) \cup \Psi'''$, which contradicts with the condition that Ψ is the finest $\wedge$-decomposition.

A.5 Proof of Corollary 1

By Proposition 3, the process of constructing the $\wedge_{\widehat{i}}$-decomposition from $\wedge_{\widehat{\infty}}$-decomposition is equivalent to the process of first constructing the $\wedge_{\widehat{j}}$-decomposition from $\wedge_{\widehat{\infty}}$-decomposition and then constructing the $\wedge_{\widehat{j}}$-decomposition from $\wedge_{\widehat{i}}$-decomposition. Then we immediately have the first conclusion of this corollary. For the second conclusion, we denote the $\wedge_{\widehat{i}}$-decomposition by Ψ'. For the first part of this conclusion, if Ψ is not finer than Ψ', there exists some factor $\psi \in \Psi$ such that the variables in $Vars(\psi)$ appear in at least two factors in Ψ'. Since Ψ' is bounded by i, we know ψ is strictly $\wedge_i$-decomposable. On the contrary, if there exists some factor $\psi \in \Psi$ is strictly $\wedge_i$-decomposable, then the variables in $Vars(\psi)$ appears in at least two factors in Ψ'; that is, Ψ is not finer than Ψ'. The second part in the second conclusion is immediate from the first part in the second conclusion and the uniqueness of $\wedge_{\widehat{i}}$-decomposition.

A.6 Proof of Proposition 4

We first present a lemma which will be used in the following proof:

Lemma 1. *Given a tree $\mathcal{T}$ over PV, and three variable sets X, X' and X'' with $X'' \subseteq X'$, if $glb(X)$ and $glb(X')$ are incomparable over $\prec_{\mathcal{T}}$, then $glb(X)$ and $glb(X'')$ are incomparable over $\prec_{\mathcal{T}}$.*

Proof. It is obvious that $glb(X') \preceq_{\mathcal{T}} glb(X'')$. We assume that $glb(X)$ and $glb(X'')$ are comparable over $\prec_{\mathcal{T}}$. If $glb(X'') \preceq_{\mathcal{T}} glb(X)$, then $glb(X') \preceq_{\mathcal{T}} glb(X)$, which contradicts with the condition of lemma. Otherwise, $glb(X) \prec_{\mathcal{T}} glb(X'')$. It is obvious that both $glb(X)$ and $glb(X')$ appear on the path from the root to $glb(X'')$. Therefore, $glb(X)$ and $glb(X')$ are comparable over $\prec_{\mathcal{T}}$, which contradicts with the condition of lemma. □

We then prove this proposition by contradiction. For notational convenience, we here assume that two equivalent factors are identical with each other. Assume that there exist another $\wedge_{\mathcal{T}}$-decomposition Ψ' such that $\{\varphi_1, \ldots, \varphi_m\}$ is not finer than Ψ'. Then there exists some factor $\psi \in \Psi'$ and some formula φ_k ($1 \leq k \leq m$) satisfying that $Vars(\varphi_k) \nsubseteq Vars(\psi)$ and φ_k shares some variable with ψ. We denote the $\wedge_{\widehat{\infty}}$-decompositions of ψ and φ_k by Ψ_1 and Ψ_2, respectively. It is obvious $\Psi_2 \setminus \Psi_1$ is nonempty. For each $\psi_1 \in \Psi_2 \setminus \Psi_1$, there exists some factor $\psi' \in \Psi'$ such that ψ_1 is a factor of ψ'. It is obvious that ψ' and ψ are two different factors in Ψ'. Therefore, $glb(\psi)$ and $glb(\psi')$ are incomparable over $\prec_{\mathcal{T}}$. According to Lemma 1, $glb(\psi)$ and $glb(\psi_1)$ are incomparable. Then for each factor $\psi_2 \in \Psi_2 \cap \Psi_1$, since $Vars(\psi_2) \subseteq Vars(\psi)$, $glb(\psi_2)$ and $glb(\psi_1)$ are incomparable over $\prec_{\mathcal{T}}$ by Lemma 1. Therefore, $\mathcal{G}_k$ is not a strongly connected subgraph (note that $\mathcal{G}_k$ is actually a free tree), which contradicts with the condition in the proposition.

A.7 Proof of Corollary 2

The second conclusion can be demonstrated in a similar proof to that of Corollary 1b. Next we focus on the first conclusion. If $\{\varphi_1, \ldots, \varphi_m\}$ is bounded by i and φ_k ($1 \leq k \leq m$) is not strictly $\wedge_{\mathcal{T}}$-decomposable, $\{\varphi_1, \ldots, \varphi_m\}$ is the $\wedge_{\widehat{\mathcal{T}},i}$-decomposition according to the uniqueness of $\wedge_{\widehat{\mathcal{T}}}$-decomposition. Next we prove that if the $\wedge_{\widehat{\mathcal{T}}}$-decomposition Ψ' of φ is bounded by i, then $\{\varphi_1, \ldots, \varphi_m\}$ is bounded by i and φ_k ($1 \leq k \leq m$) is not strictly $\wedge_{\mathcal{T}}$-decomposable; that is, if $\{\varphi_1, \ldots, \varphi_m\}$ is not bounded by i or φ_k ($1 \leq k \leq m$) is strictly $\wedge_{\mathcal{T}}$-decomposable, the $\wedge_{\widehat{\mathcal{T}},i}$-decomposition does not exist. It is obvious that the $\wedge_{\widehat{\infty}}$-decomposition Ψ'' is finer than Ψ, and Ψ is finer than Ψ'. For each $\psi \in \Psi'$ with at most i variables, $glb(\psi)$ is incomparable with the greatest lower bounds of other formulas in $\Psi' \setminus \{\psi\}$, and the $\wedge_{\widehat{\infty}}$-decomposition of ψ can be seen as a subset of Ψ. According to Proposition 4, the method of generating Ψ' from Ψ'' is similar to that of generating $\{\varphi_1, \ldots, \varphi_m\}$ from Ψ. Therefore, ψ can be seen as an element in $\{\varphi_1, \ldots, \varphi_m\}$. That is, each factor in Ψ' with at most i variables appears in $\{\varphi_1, \ldots, \varphi_m\}$. Since Ψ' has at most one factor with more than i variables, we know Ψ' is equivalent to $\{\varphi_1, \ldots, \varphi_m\}$. That is, $\{\varphi_1, \ldots, \varphi_m\}$ is bounded by i and φ_k ($1 \leq k \leq m$) is not strictly $\wedge_{\mathcal{T}}$-decomposable.

A.8 Proof of Observation 2

- For the first conclusion, it is obvious that we can recursively compute $Vars(u)$ as follows:

$$Vars(u) = \begin{cases} \emptyset & sym(u) = \bot \text{ or } \top; \\ \bigcup_{v \in Ch(u)} Vars(v) & sym(u) = \wedge; \\ \{sym(u)\} \cup Vars(lo(u)) \cup Vars(hi(u)) & \text{otherwise.} \end{cases}$$

  Because the subgraphs rooted at different children of a $\wedge$-vertex do not share any variable, $\bigcup_{v \in Ch(u)} Vars(v)$ can be computed in $O(|Vars(\mathcal{G}_u)|)$ when we have known $Vars(v)$ for each $v \in Ch(u)$. Therefore, we can compute the variable sets for all vertices in $\mathcal{G}_u$ in time $O(|Vars(u)| \cdot |V(\mathcal{G}_u)|)$ with the use of dynamic programming.
- For the second conclusion, the case $i = \infty$ is obvious. Next we assume $i < \infty$. We can determine whether $\mathcal{G}_u$ has more than i variables by traversing $\mathcal{G}_u$, and we show that the number of traversed vertices is $O(1)$. When traversing a $\wedge$-vertex with more than i children, we can cease traversing and answer "no" since each child introduces at least one new variable into $Vars(u)$. When the length of a traversing path is greater than i, we can cease traversing and answer "no" because each vertex on the path introduces at least one variable not appearing in the subgraph rooted at its child on the path. That is, we need to traverse at most $(i^i - 1)/(i - 1)$ vertices. Therefore, if $sym(u) = \wedge$, whether u is bounded by integer i can be determined in $O(|Ch(u)|)$.

A.9 Proof of Observation 3

- The first conclusion is immediate from the fact that each $\wedge_i$-decomposition is a $\wedge_j$-decomposition. The second one is immediate from the fact that $x \prec_{\mathcal{T}} x'$ implies $x \prec_{\mathcal{C}} x'$.
- For the third conclusion, if $glb(u) \in Vars(u)$, then u is not a $\wedge$-vertex, which implies $glb(u) = sym(u)$ according to the ordered decision; and if $glb(u) = sym(u)$, it is obvious $glb(u) \in Vars(\mathcal{G}_u)$.
- For the fourth conclusion, we only explain the case of ROBDD[$\wedge_{\widehat{i}}$]$_{\mathcal{C}}$, and the case of ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ is similar. If the root is a $\diamond$-vertex, each $\wedge$-vertex is a child of some $\diamond$-vertex and there is at least one $\diamond$-vertex without $\wedge$-child. Therefore, we immediately have the conclusion. Otherwise, the root is a $\wedge$-vertex. Each $\wedge$-vertex which is not the root is a child of some $\diamond$-vertex, and there are at least two $\diamond$-vertices without $\wedge$-child. Therefore, we immediately have the conclusion.
- For the last conclusion, we prove it by induction on the number of vertices. Let u be the root of ROBDD[$\wedge_{\mathcal{T}}$]$_{\mathcal{T}}$. The case $|V(\mathcal{G}_u)| = 1$ is obvious. We assume that this conclusion holds for $|V(\mathcal{G}_u)| \leq m$. For the case $|V(\mathcal{G}_u)| = m+1$, we proceed by case analysis. Given a $\diamond$-vertex $v \in V(\mathcal{G}_u)$, it is easy to see that $\vartheta(v)$ is not strictly $\wedge_{\mathcal{T}}$-decomposable. Therefore, $\mathcal{G}_u$ is an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ by the induction hypothesis. Otherwise, u is a $\wedge$-vertex and each child of u is a $\diamond$-vertex. Since each child of u represents a formula that is not strictly $\wedge_{\mathcal{T}}$-decomposable, $\{\vartheta(v) : v \in Ch(u)\}$ is the $\wedge_{\widehat{\mathcal{T}}}$-decomposition of $\vartheta(u)$ by Corallory 2b. Then $\mathcal{G}_u$ is an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ by the induction hypothesis.

A.10 Proof of Observation 5

Let V be the vertices in $\mathcal{G}_u$ whose ancestors are $\wedge$-vertices; that is, for each path $(v_1, \ldots, v_m)$ from the root to some vertex in V, $sym(v_k) = \wedge$ $(1 \leq k < m)$. The induced subgraph over V is a tree $\mathcal{T}$ whose leaves are $\diamond$-vertices. Since any two subgraphs rooted at two leaves in $\mathcal{T}$ do not share variables, the number of leaves is not more than $|Vars(u)|$. Therefore, $|V(\mathcal{T})| \leq 2 \cdot |Vars(u)| - 1$. Since Merge($u$) does process the vertices in V, we immediately have the conclusion about time complexity. Next prove the itemized facts:

For the first fact, we assume that v is a $\wedge$-child of $\mathcal{G}_u$. We denote the result of removing v from $Ch(u)$ and then adding all children of v to $Ch(u)$ by u'. If $\mathcal{G}_v$ has at most i variables, then each child w of v satisfies that $\mathcal{G}_w$ has less than i variables, and thus u' corresponds to a $\wedge_i$-decomposition. Otherwise, $\mathcal{G}_v$ has more than i variables, and there exists at most one child $w \in Ch(v)$ satisfying that $\mathcal{G}_w$ has more than i variables. Since each child $v' \neq v$ of u satisfies that $\mathcal{G}_{v'}$ has at most i variables, u' corresponds to a $\wedge_i$-decomposition. Therefore, Merge(u) is $\wedge_i$-decomposable.

For the second fact, we know that Merge(u) corresponds to a $\wedge_i$-decomposition by the first fact. It is obvious that each child of the output of Merge(u) is not strictly $\wedge_i$-decomposable. According to Corollary 1b, we know that Merge(u) is $\wedge_{\widehat{i}}$-decomposable.

For the third fact, we assume that v is a $\wedge$-child of $\mathcal{G}_u$. We denote the result of removing v from $Ch(u)$ and then adding all children of v to $Ch(u)$ by u'. For each child $v' \in Ch(u)$ with $v \neq v'$, each child w of v satisfies that $glb(w)$ and $glb(v')$ are incomparable over $\prec_{\mathcal{T}}$ by Lemma 1. Therefore, u' corresponds to a $\wedge_{\mathcal{T}}$-decomposition. That is, Merge(u) is $\wedge_{\mathcal{T}}$-decomposable.

For the fourth fact, we know that Merge(u) corresponds to a $\wedge_{\mathcal{T}}$-decomposition by the third fact. It is obvious that each child of the output of Merge(u) is not strictly $\wedge_{\mathcal{T}}$-decomposable. According to Corollary 2b, we know that Merge(u) is $\wedge_{\widehat{\mathcal{T}}}$-decomposable.

A.11 Proof of Observation 6

The time complexity is obvious. The cases for $\wedge_i$-decomposability and $\wedge_{\widehat{i}}$-decomposability are obvious. For the two other cases, it is obvious that the DAGs rooted at the rest children are $\wedge_{\widehat{i}}$-decomposable (resp. $\wedge_{\widehat{\mathcal{T}}}$-decomposable). According to Corollary 1b (resp. 2b), we immediately know that the BDD[$\wedge$] rooted at $u \setminus V$ is $\wedge_{\widehat{i}}$-decomposable (resp. $\wedge_{\widehat{\mathcal{T}}}$-decomposable).

A.12 Proof of Observation 7

The case when u is a leaf is obvious. Next we assume u is an internal vertex. Let V be the set of internal vertices of $\mathcal{G}_u$ and U be the set of internal meta-vertices of $\mathcal{G}_u$. Obviously, $|V| \leq |V(\mathcal{G}_u)| - 2$. For convenience, we say that w is a *meta-descendant* of $v \in V$ if it is a meta-child of v or a meta-child of meta-descendant of v. It is easy to see that $w \in U \setminus V$ is a meta-descendant of some $\wedge$-vertex $v \in V$. In the next paragraph, we show that given a $\wedge$-vertex $v \in V$, the number of meta-descendants in $U \setminus V$ of v is not more than $|Ch(v)| - 2$. Therefore, $|U| = |U \setminus V| + |V| \leq \sum_{v \in V} |Ch(v)| - 2 + 1 = |\mathcal{G}_u| - |V| \leq |\mathcal{G}_u| - |V(\mathcal{G}_u)| + 2$.

Let W be the set of v and its meta-descendants, and let $\mathcal{G}$ be a graph over W with edge set $\{(w, w') : w'$ is a meta-child of $w\}$. Obviously, $\mathcal{G}$ is a tree such that v is its root, the set of its leaves is exactly $Ch(v)$, and each internal vertex has at least two children. Therefore, the number of internal vertices in $\mathcal{G}$ is not more than $|Ch(v)| - 1$. That is, the number of meta-descendants of v in $U \setminus V$ is not more than $|Ch(v)| - 2$.

A.13 Proof of Theorem 1

First we introduce two auxiliary functions. The first one, called $L2V(L)$, transforms a set of literals into the set of roots of the equivalent BDD[$\wedge$]s; that is, $L2V(L) = \{\langle l \rangle : l \in L\}$. The second one, called $V2L$, is the inverse function of $L2V$.

The proof is organized respectively corresponding to the items in the proposition:

(a) The first conclusion is immediate from the facts that no decomposition vertex appears in OBDD[$\wedge_0$]$_{\mathcal{C}}$ and each set with one single formula is a $\wedge_{\widehat{0}}$-decomposition.

(b) Each non-false vertex v can be seen as a $\wedge_1$-decomposition, since $L(v)$ represents a consistent term and does not have any variable appearing in the subgraphs. We define two functions f and g to perform the transformations as follows, where the former is from OBDD-L (resp. ROBDD-L_∞) to OBDD[$\wedge_1$]$_{\mathcal{C}}$ (resp. ROBDD[$\wedge_{\widehat{1}}$]$_{\mathcal{C}}$) and the latter is from ROBDD[$\wedge_{\widehat{1}}$]$_{\mathcal{C}}$ to ROBDD-L_∞.

$$f(u) = \begin{cases} \langle \bot \rangle & u = \langle \bot \rangle; \\ \langle \top \rangle & u = \langle \emptyset \rangle; \\ \langle \wedge, L2V(L(u)) \rangle & u = \langle L(u) \rangle; \\ \langle \wedge, \{\langle sym(u), f(lo(u)), f(hi(u)) \rangle\} \cup L2V(L(u)) \rangle & u \text{ is an internal vertex.} \end{cases}$$

$$g(u) = \begin{cases} \langle \bot \rangle & u = \langle \bot \rangle \\ \langle \emptyset \rangle & u = \langle \top \rangle; \\ \langle sym(u), g(lo(u)), g(hi(u)), \emptyset \rangle & sym(u) \in PV; \\ \langle V2L(Ch(u)) \rangle & sym(u) = \wedge \text{ and } \forall v \in Ch(u).|Vars(u)| = 1; \\ \begin{array}{l} \langle sym(v), g(lo(v)), g(hi(v)), \\ \qquad V2L(Ch(u) \setminus \{v\}) \rangle \end{array} & sym(u) = \wedge \text{ and } \exists v \in Ch(u).|Vars(v)| > 1. \end{cases}$$

It is obvious that by use of dynamic programming, the transformations can be done in linear time.

(c) We can transform each AOBDD over $\mathcal{T}$ into ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ by the following steps: i) If there exists more than one DAG, we create a new $\wedge$-vertex whose children are exactly the roots of all DAGs; and ii) for each $\wedge$-vertex v with only one child w and each parent u, we replace the arc (u, v) with (u, w) and delete v. It is easy to see that the resulting DAG is an equivalent ROBDD[$\wedge_{\mathcal{T}}$]$_{\mathcal{T}}$ and each $\wedge$-vertex in it does not have any $\wedge$-child. According to Observation 3e, the resulting DAG is an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$. Conversely, each ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ can be transformed into AOBDD by the following steps: i) If the ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ is rooted at a $\wedge$-vertex u, we decompose it into several parts rooted at vertices in $Ch(u)$; and ii) for each $\diamond$-vertex v with a $\diamond$-child w, we introduce a new vertex $w' = \langle \wedge, \{w\} \rangle$ and replace the arc (v, w) by (v, w'). It is easy to see the following facts: i) each $\diamond$-vertex and its children can be seen as a meta-node; ii) any two roots of the parts are independent; and iii) the resulting DAG does not have redundant meta-node and two identical meta-nodes. That is, the resulting DAG is an AOBDD over $\mathcal{T}$. By use of dynamic programming, the mutual transformations can be done in linear time.

A.14 Proof of Theorem 2

The case $i = 0$ is immediate from the fact that ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$ is equivalent to ROBDD. In the following proof, we assume $i > 0$. For the existence of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, we can only consider the irredundant formulas. We prove the existence by induction on the number of variables. When $|Vars(\varphi)| = 0$, it is obvious that either $\langle\bot\rangle$ or $\langle\top\rangle$ represents it. Assume that when $|Vars(\varphi)| \leq n$, there exists some equivalent ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ rooted at u such that $Vars(u) \subseteq Vars(\varphi)$. For the case where $|Vars(\varphi)| = n+1$, we proceed by case analysis:

- φ is strictly $\wedge_i$-decomposable: Assume that $\{\psi_1, \dots, \psi_m\}$ is the $\wedge_{\widehat{i}}$-decomposition of φ. According to the induction hypothesis, we assume that each factor ψ_k is represented by an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ rooted at v_k, where $Vars(v_k) = Vars(\psi_k)$. Since each ψ_i is not strictly $\wedge_i$-decomposable, v_i is a $\diamond$-vertex and thus $u = \langle\wedge, \{v_1, \dots, v_m\}\rangle$ represents φ with $Vars(u) \subseteq Vars(\varphi)$. After removing the duplicate leaf vertices, $\mathcal{G}_u$ is an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$.
- Otherwise: Let x be $glb(\varphi)$. We have $\varphi|_{x=false} \not\equiv \varphi|_{x=true}$ since φ depends on x. According to the induction hypothesis, we assume that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertices v and w represent $\lfloor\varphi|_{x=false}\rfloor$ and $\lfloor\varphi|_{x=true}\rfloor$, respectively, and we have that $Vars(u) \subseteq Vars(\varphi)$, $x \prec_{\mathcal{C}} Vars(v)$ and $x \prec_{\mathcal{C}} Vars(w)$. It is obvious that $v \neq w$. Therefore, $u = \langle x, v, w\rangle$ represents φ. After removing the duplicate vertices, $\mathcal{G}_u$ is an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$.

For the uniqueness, it is easy to prove by induction on the size of $Vars(\varphi)$ that each trivial formula can only be represented by $\langle\bot\rangle$ or $\langle\top\rangle$. Next we only analyze the non-trivial cases. Assume that when $|Vars(\varphi)| \leq n$, there exists only one equivalent ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ rooted at v such that $Vars(v) = Vars(\lfloor\varphi\rfloor)$. For the case where $|Vars(\varphi)| \leq n+1$, φ cannot be represented by both $\diamond$-vertex and $\wedge$-vertex in ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ because of the uniqueness of the $\wedge_{\widehat{i}}$-decomposition, and then we proceed by case analysis:

- φ is represented by two $\wedge$-vertices u and v: For each $w \in Ch(u)$ or $w \in Ch(v)$, $Vars(\lfloor\vartheta(w)\rfloor) \subset Vars(\lfloor\varphi\rfloor)$ and each variable in $\vartheta(w)$ is essential, according to the induction hypothesis. That is, $Vars(u) = Vars(v) = Vars(\lfloor\varphi\rfloor)$. Since each child of $\wedge$-vertex represents a non-trivial formula, both $\{\vartheta(w) : w \in Ch(u)\}$ and $\{\vartheta(w) : w \in Ch(v)\}$ are the $\wedge_{\widehat{i}}$-decomposition of $\lfloor\varphi\rfloor$ by Proposition 3. Then we know $Ch(u) = Ch(v)$ according to the induction hypothesis. That is, u is identical with v.
- φ is represented by two $\diamond$-vertices u and v: If $sym(u) = sym(v)$, it is easy to see that u is identical with v by the induction hypothesis. Otherwise, $sym(u) \prec_{\mathcal{C}} sym(v)$ or $sym(u) \succ_{\mathcal{C}} sym(v)$. Without loss of generality, we assume $sym(u) \prec_{\mathcal{C}} sym(v)$, and we have $sym(u) \notin Vars(v)$. Therefore, $\vartheta(lo(u)) \equiv \varphi|_{sym(u)=false} \equiv \vartheta(v) \equiv \varphi|_{sym(u)=true} \equiv \vartheta(hi(u))$. By the induction hypothesis, $lo(u)$ is identical with $hi(u)$, which contradicts with the fact that a $\diamond$-vertex in ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ must have two distinct children.

A.15 Proofs of Observation 8 and Theorem 3

For convenience, we prove the observation and proposition together. Obviously, we can show the following conclusion by a proof similar to the one we prove the uniqueness of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$: there are at most one ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ vertex v to represent a given formula φ, and v satisfies $Vars(v) = Vars(\lfloor\varphi\rfloor)$. That is, we have the canonicity of ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$.

For Observation 8, since φ is irredundant, it cannot be represented by a leaf vertex; since φ is not strictly $\wedge_{\mathcal{T}}$-decomposable, it cannot be represented by a $\wedge$-vertex; and since $glb(\varphi) \notin Vars(\varphi)$, it cannot be represented by a $\diamond$-vertex according to Observation 3c. That is, φ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$. In particular, if $\mathcal{T}$ is not a chain, there exist two incomparable variables x and x' over $\prec_{\mathcal{T}}$, and thus $\varphi = x \leftrightarrow x'$ is not strictly $\wedge$-decomposable and satisfies $glb(\varphi) \notin Vars(\varphi)$; that is, φ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$.

A.16 Proof of Observation 9

The first conclusion is immediate from the fact that each $\wedge$-decomposition is bounded by $dc(\mathcal{T})$. For the second conclusion, it is obvious that there exist two incomparable variables x and x' such that $|V(\mathcal{T}_x)| \geq dc(\mathcal{T})$ and $|V(\mathcal{T}_{x'})| \geq dc(\mathcal{T})$. Therefore, there exist two variable sets X and X' from $V(\mathcal{T}_x) \setminus \{x\}$ and $V(\mathcal{T}_{x'}) \setminus \{x'\}$, respectively, with $j-1$ variables. We know neither $\varphi = x \leftrightarrow \bigwedge_{x'' \in X} x''$ nor $\varphi' = x' \leftrightarrow \bigwedge_{x'' \in X'} x''$ is strictly $\wedge$-decomposable. The reader can verify that both φ and φ' can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$, and we assume that v and v', respectively, represent them. By Corollary 1b, $\varphi \wedge \varphi'$ is strictly $\wedge_{\mathcal{T},j}$-decomposable, and $u = \langle \wedge, \{v, v'\} \rangle$ is the ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ vertex representing $\varphi \wedge \varphi'$. It is obvious that u is $\wedge_j$-decomposable but not $\wedge_i$-decomposable.

A.17 Proof of Theorem 4

We first prove the direction from right to left. For $i' = \min\{i, dc(\mathcal{T})\}$, each ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ is $\wedge_{i'}$-decomposable according to Observation 9. Therefore, each ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ is an ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$ if $i' \leq j$. That is, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}} \leq_e$ ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$ if $i \leq j$ or $dc(\mathcal{T}) \leq j$. We then prove the direction from left to right. If $i > j$ and $dc(\mathcal{T}) > j$, then we have $dc(\mathcal{T}) \geq i' > j$ and $i \geq i'$. By Observation 9, there exists some ROBDD$[\wedge_{i'}]_{\widehat{\mathcal{T}}}$ which is an ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ but not an ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$. That is, ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}} \not\leq_e$ ROBDD$[\wedge_{\widehat{\mathcal{T}},j}]_{\mathcal{T}}$ if $i > j$ and $dc(\mathcal{T}) > j$.

A.18 Proof of Observation 10

It is obvious that $\vartheta(u)$ is irredundant and $|Vars(u)| > 1$. $lo(u)$ and $hi(u)$ represent $\vartheta(u)|_{sym(u)=false}$ and $\vartheta(u)|_{sym(u)=true}$, respectively. Let Ψ be the $\wedge_{\widehat{i}}$-decomposition of $\vartheta(u)$, and we assume that $sym(u)$ appears in $\psi \in \Psi$ and ψ' is another factor in Ψ. By Corollary 1b, each factor in Ψ is not strictly $\wedge_i$-decomposable. If $\psi|_{sym(u)=false} \equiv false$ (resp. $\psi|_{sym(u)=true} \equiv false$), it is obvious that $\langle \bot \rangle \in Ch(u)$ and $|Vars(u)| > 1$. Otherwise, we have the following cases:

- $\Psi = \{\psi, \psi'\}$ and $\psi|_{sym(u)=false} \equiv true$: It is obvious that $\psi|_{sym(u)=true}$ is non-trivial and $\varphi|_{sym(u)=false} = \psi'$. Therefore, $\{\psi|_{sym(u)=true}, \psi'\}$ is a $\wedge_i$-decomposition of $\varphi|_{sym(u)=true}$. Then we know $\varphi|_{sym(u)=false} = \psi'$ is a factor of the $\wedge_{\widehat{i}}$-decomposition of $\varphi|_{sym(u)=true}$. That is, $lo(u)$ is a child of $hi(u)$.
- $\Psi = \{\psi, \psi'\}$ and $\psi|_{sym(u)=true} \equiv true$: It is similar to the first item.
- Otherwise, it is obvious that $\psi|_{sym(u)=false}$ (resp. $\psi|_{sym(u)=true}$) is non-trivial, and thus ψ' is a factor of the $\wedge_{\widehat{i}}$-decomposition of $\varphi|_{sym(u)=false}$ (resp. $\varphi|_{sym(u)=true}$). That is, both children of u are $\wedge$-vertices with some shared factors.

A.19 Proof of Observation 11

The case i = 0 is immediate from the fact that there is not any $\wedge$-vertex in ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$. Otherwise, we have two similar cases (Lines 2–3), and we only explain the first one due to the duality. According to Equation (2),

$$\vartheta(u) \equiv (\neg sym(u) \wedge false) \vee (sym(u) \wedge \vartheta(hi(u))) \equiv sym(u) \wedge \vartheta(hi(u)).$$

Therefore, $\{sym(u), \vartheta(hi(u))\}$ is a $\wedge$-decomposition of $\vartheta(u)$. Obviously, $\langle sym(u)\rangle$ is the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex which represents $sym(u)$. Since $hi(u)$ is $\wedge_{\widehat{i}}$-decomposable, we know by Observation 5 that Merge(u') is the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex representing $\vartheta(u)$. According to Observations 4–5, the time complexity of Make and Merge is $O(|Ch(lo(u))|+|Ch(hi(u))|) = O(|Vars(u))|$. Therefore, we immediately have the time complexity of ExtractLeaf. The step on Line 2 will introduce at most two vertices (i.e., $\langle sym(u)\rangle$ and u'), and Merge(u) on Line 4 will replace u' with another new vertex if u' has a $\wedge$-vertex. That is, each single calling of ExtractLeaf(u) will introduce at most two vertices. The step on Line 2 will add at most four new arcs and reduce two old arcs, and Merge(u) on Line 4 will not add new arc. That is, each single calling of ExtractLeaf(u) will add at most two arcs.

A.20 Proof of Observation 12

In ExtractPart, we have two similar cases (Lines 2–3 and 5–6), and we only explain the first one due to the duality. According to Equation (2),

$$\begin{aligned}\vartheta(u) &\equiv [\neg sym(u) \wedge \vartheta(lo(u))] \vee [sym(u) \wedge \vartheta(lo(u)) \wedge \vartheta(hi(u) \setminus \{lo(u)\})] \\ &\equiv \vartheta(lo(u)) \wedge [\neg sym(u) \vee \vartheta(hi(u) \setminus \{lo(u)\})].\end{aligned}$$

It is obvious that neither $\vartheta(lo(u))$ nor $\neg sym(u) \vee \vartheta(hi(u) \setminus \{lo(u)\})$ is strictly $\wedge_i$-decomposable. According to Corollary 1b, $\{\vartheta(lo(u)), \neg sym(u) \vee \vartheta(hi(u) \setminus \{lo(u)\})\}$ is finer than the $\wedge_{\widehat{i}}$-decomposition of $\vartheta(u)$. Therefore, if u' on Line 8 is a $\wedge_i$-vertex, it is a $\wedge_{\widehat{i}}$-vertex; and otherwise, $\vartheta(u)$ is not strictly $\wedge_i$-decomposable, which implies that u is a vertex in ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. The time complexity of ExtractPart is obvious. The step on Line 2 will introduce at most two vertices, and the step on Line 3 will introduce at most one vertex on Line 3. That is, each single calling of Merge(u) will introduce at most three vertices. The steps on Lines 2–3 will add at most four new arcs and reduce three old arcs. That is, each single calling of ExtractPart(u) will add at most one arc.

A.21 Proof of Observation 13

According to Equation (2), we can show the output is equivalent to the input as follows:

$$\begin{aligned}\vartheta(u) &\equiv [\neg sym(u) \wedge \vartheta(lo(u) \setminus V) \wedge \bigwedge_{v \in V} \vartheta(v)] \vee [sym(u) \wedge \vartheta(hi(u) \setminus V) \wedge \bigwedge_{v \in V} \vartheta(v)] \\ &\equiv \big[[\neg sym(u) \wedge \vartheta(lo(u) \setminus V)] \vee [sym(u) \wedge \vartheta(hi(u) \setminus V)]\big] \wedge \bigwedge_{v \in V} \vartheta(v).\end{aligned}$$

According to Observation 6, $\Psi = \{[\neg sym(u) \wedge \vartheta(lo(u) \setminus V)] \vee [sym(u) \wedge \vartheta(hi(u) \setminus V)]\} \cup \{\vartheta(v) : v \in V\}$ is a $\wedge$-decomposition of $\vartheta(u)$. Each factor in this decomposition is not strictly $\wedge_i$-decomposable, and thus Ψ is finer than the $\wedge_{\widehat{i}}$-decomposition of $\vartheta(u)$ by Corollary 1b. That is, u' on Line 4 is a $\wedge_{\widehat{i}}$-vertex if u' on Line 4 is a $\wedge_i$-vertex. Otherwise, there exist exactly two factors in Ψ with more than i variables. If $|V| = 1$, we know $\vartheta(u)$ is not strictly $\wedge_i$-decomposable, which implies that u is a vertex in ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. Otherwise, we can get the $\wedge_{\widehat{i}}$-decomposition of $\vartheta(u)$ by conjoining the two factors by Corollary 1b. The time complexity of ExtractPart is obvious. The reader can verify that this function will introduce at most two vertices on Line 2 or 7, and introduce at most one vertex on Line 3 or 8. That is, each single calling of Merge(u) will introduce at most three vertices. The reader can verify that this function will add at most $|V| + 1$ new arcs and reduce $2 \cdot |V|$ old arcs on Lines 2 or 7. That is, each single calling of ExtractShare(u) will add at most one arc.

A.22 Proof of Proposition 7

We only prove the case of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ by induction on the size of $Vars(\varphi)$, and the proof for the case of ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ is similar. The case $|Vars(\varphi)| \leq 1$ is obvious. We assume that the conclusion holds when $|Vars(\varphi)| \leq n$. For the case $|Vars(\varphi)| = n+1$, we proceed by case analysis:

- φ is strictly $\wedge_i$-decomposable: We denote the factor in the $\wedge_{\widehat{i}}$-decomposition of φ with the appearance of x by ψ, and the $\wedge_{\widehat{i}}$-decomposition of $\lfloor \psi|_\omega \rfloor$ by Ψ'. It is obvious that each factor of the $\wedge_{\widehat{i}}$-decomposition of $\lfloor \varphi|_\omega \rfloor$ with the appearance of x is a factor of Ψ' with the appearance of x, and vice versa. Let $\Psi'' = \{\psi' : \exists \omega \in 2^X . \psi'$ is a factor of the $\wedge_{\widehat{i}}$ -decomposition of $\lfloor \psi|_\omega \rfloor\}$. Let u (resp. v) be the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex representing φ (resp. ψ). According to the induction hypothesis, the number of vertices in $\mathcal{G}_v$ labeled by x is equal to the number of distinct factors in Ψ'' with the appearance of x. Since each vertex labeled by x in $\mathcal{G}_u$ is exactly a vertex labeled by x in $\mathcal{G}_v$, we immediately have the conclusion.
- Otherwise, φ is not strictly $\wedge_i$-decomposable: If $x = glb(\varphi)$, the conclusion is obvious. Otherwise, we know $glb(\varphi) \prec_{\mathcal{C}} x$. Let $\Psi_1 = \{\psi : \exists \omega \in 2^X . glb(\varphi) = \mathit{false} \in \omega$ and ψ is a factor of the $\wedge_{\widehat{i}}$-decomposition of $\lfloor \varphi|_\omega \rfloor\}$, and $\Psi_2 = \{\psi : \exists \omega \in 2^X . glb(\varphi) = \mathit{true} \in \omega$ and ψ is a factor of the $\wedge_{\widehat{i}}$ -decomposition of $\lfloor \varphi|_\omega \rfloor\}$. It is obvious that each factor of Ψ with the appearance of x is a factor of Ψ_1 or Ψ_2 with the appearance of x, and vice versa. Let u (resp. v and w) be the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex representing φ (resp. $\varphi|_{glb(\varphi)=\mathit{false}}$ and $\varphi|_{glb(\varphi)=\mathit{true}}$). According to the induction hypothesis, the number of vertices labeled by x in $\mathcal{G}_v$ (resp. $\mathcal{G}_w$) is equal to the number of factors in Ψ_1 (resp. Ψ_2) with the appearance of x. According the canonicity of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, the set of vertices labeled by x in $\mathcal{G}_u$ is the union of two sets of vertices labeled by x: the ones in $\mathcal{G}_v$, and the ones in $\mathcal{G}_w$. Then we immediately have the conclusion.

A.23 Proof of Observation 14

The conclusion in the second sentence is immediate from the one in the first sentence, and next we prove the first one. For each variable $x \in Vars(\varphi)$, let X be the set of variables on the path from $glb_{\prec_{\mathcal{T}}}(\varphi)$ to the parent of x in tree $\mathcal{T}$, and let Φ be set of distinct formulas obtained by conditioning φ on assignments over X. It is obvious that $|X|$ is not more than $|dep(\mathcal{T})|$, and thus $|\Phi|$ is not more than $2^{dep(\mathcal{T})}$. According to Proposition 7 and the uniqueness of $\wedge_{\widehat{\mathcal{T}}}$-decomposition, the number of vertices labeled by x is not more than $2^{dep(\mathcal{T})}$. Therefore, if φ can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, then the resulting ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ has at most $3 \cdot |Vars(\varphi)| \cdot 2^{dep(\mathcal{T})}$ vertices by Observation 3d.

A.24 Proof of Proposition 8

The soundness of Algorithm Condition can be easily proven by induction on the number of vertices in $\mathcal{G}_v$. We can record the assignment as a vector, and thus a single calling of Condition(u) terminates in $O(|Ch(u)|)$. That is, the time complexity of Condition is $O(|\mathcal{G}_u| + |\omega|)$. Each single calling of Condition generates at most one new vertex (not considering the vertices generated by the sub-calling of Condition), and the number of arcs from the new vertex is not more than the number of arcs from the input. Therefore, the size of output of Condition(u, ω) is not greater than $|\mathcal{G}_u|$.

A.25 Proof of Proposition 9

It is easy to see that ConditionMin(u, b) is equivalent to Condition$(u, glb(u) = b)$ for ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$. Therefore, we immediately know the output of ConditionMin(u, b) is an OBDD$[\wedge]_{\prec}$ vertex representing $\vartheta(u)|_{glb(u)=b}$. Since each vertex in $Ch(u) \setminus \{v\}$ is labeled by a variable, Lines 8–9 in the algorithm mean that we replace v in $Ch(u)$ with w and then call Merge(u). Therefore, the new u is an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex by Observation 5. The size of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ rooted at the output is less than $|\mathcal{G}_u|$ since the vertex labeled by $glb(u)$ is removed. It is also easy to see the time complexity of ConditionMin.

A.26 Proof of Observation 15

We first show that u and v satisfy at least one of the five conditions. If neither the first condition nor the second one is satisfied, we know that $sym(v) \in PV$, and $sym(v) \preceq_{\mathcal{T}} glb(u)$ or $sym(v) \succ_{\mathcal{T}} glb(u)$. If $sym(v) \preceq_{\mathcal{T}} glb(u)$, we know $sym(v) = sym(u)$ or $sym(v) \prec_{\mathcal{T}} Vars(u)$; that is, either the third condition or the fourth one is satisfied. Otherwise, either the fifth condition or the sixth one is satisfied.

Next we show the soundness of each item. The first five items are easy to see. For the last item, if $\vartheta(mch_x(u)) \models \vartheta(v)$, we know $\vartheta(u) \models \vartheta(mch_x(u)) \models \vartheta(v)$. If $\vartheta(mch_x(u)) \not\models \vartheta(v)$, there exists some model ω over $Vars(mch_x(u)) \cup Vars(v)$ such that $\omega \models \vartheta(mch_x(u))$ and $\omega \not\models \vartheta(v)$. Since each subgraph rooted at some vertex in $Ch(u) \setminus Ch_x(u)$ does not share any variable with $\mathcal{G}_v$. We can extend ω to get another model ω' over $Vars(u) \cup Vars(v)$ such that $\omega' \models \vartheta(u)$ and $\omega' \not\models \vartheta(v)$. That is, $\vartheta(u) \not\models \vartheta(v)$.

A.27 Proof of Proposition 10

We first present a lemma that will be used when we analyze the time complexities of Algorithms EntailTree, DisjoinTree and ConvertTree:

Lemma 2. *Given an OBDD*$[\wedge]_{\prec}$ $\mathcal{G}$ *and a tree* $\mathcal{T}$ *with* $\prec_{\mathcal{T}} \subseteq \prec$*, we can compute all greatest lower bounds of the formulas represented by its vertices over* $\prec_{\mathcal{T}}$ *in* $O(|\mathcal{G}|)$.

Proof. Since we know the greatest lower bounds of any two variables over $\prec_{\mathcal{T}}$, we can recursively compute all greatest lower bounds of the formulas represented by its vertices. That is, if u is a $\diamond$-vertex with two constant children, $glb(u) = sym(u)$; if u is a $\diamond$-vertex with only one non-constant child v, $glb(u) = glb(\{sym(u), glb(v)\})$; if u is a $\diamond$-vertex with two non-constant children v and w, $glb(u) = glb(\{sym(u), glb(v), glb(w)\})$; and if u is a $\wedge$-vertex, $glb(u) = glb(\{glb(v) : v \in Ch(u)\})$. Moreover, $glb(X) = glb(\{x, glb(X \setminus \{x\})\})$ if $|X| > 2$ and $x \in X$. Therefore, we can obtain the lemma with the use of dynamic programming. □

According to Observation 15, the soundness of Algorithm EntailTree can be easily proven by induction on the sum of $|\mathcal{G}_u|$ and $|V(\mathcal{G}_v)|$. For each single calling of EntailTree(w,w'), w is a meta-vertex in $\mathcal{G}_u$, and w' is a vertex in $\mathcal{G}_v$. Therefore, the number of single calling is not more than $|\mathcal{G}_u| \cdot |V(\mathcal{G}_v)|$ by Observation 7. According to Lemma 2, all greatest lower bounds for vertices in $\mathcal{G}_v$ can be computed in $O(|\mathcal{G}_v|)$ in the preprocessing stage. We can also show in a similar fashion that all greatest lower bounds for meta-vertices in $\mathcal{G}_u$ can be computed in $O(|Vars(u)| \cdot |\mathcal{G}_u|)$ in the preprocessing stage. In addition, for each pair of variables x and x', we have assumed that $ch_{\mathcal{T}}(x \rightsquigarrow x')$ is already known. Therefore, it is easy to see that given two ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$s rooted at w and w', each single calling of EntailTree can be done in $O(|Vars(w)| + |Vars(w')|)$. That is, the time complexity of EntailTree is $O((|Vars(u)| + |Vars(v)|) \cdot |\mathcal{G}_u| \cdot |V(\mathcal{G}_v)|)$.

A.28 Proof of Observation 16

We first present a lemma that will be used in this proof and the proof of Observation 17:

Lemma 3. *A consistent formula φ is strictly $\wedge_{\mathcal{T}}$-decomposable, iff for any variable $x \in Ch_{\mathcal{T}}(glb(\varphi))$ and any two assignments ω and ω' over $V(\mathcal{T}_x)$ such that both $\varphi|_\omega$ and $\varphi|_{\omega'}$ are consistent, we have $\varphi|_\omega \equiv \varphi|_{\omega'}$.*

Proof. We first prove the direction from left to right. We assume that the $\wedge_{\widehat{\mathcal{T}}}$-decomposition Ψ of φ is strict. For each variable $x \in Ch_{\mathcal{T}}(glb(\varphi))$, let $\Psi' = \{\psi \in \Psi : x \preceq_{\mathcal{T}} glb(\psi)\}$. It is obvious that if $\omega \models \bigwedge_{\psi \in \Psi'} \psi$, then $\varphi|_\omega \equiv \bigwedge_{\psi \in \Psi \setminus \Psi'} \psi$, and otherwise $\varphi|_\omega \equiv false$. We have the similar conclusion for ω'. Therefore, if $\varphi|_\omega$ and $\varphi|_{\omega'}$ are consistent, then $\varphi|_\omega \equiv \varphi|_{\omega'}$.

We then prove the direction from right to left by induction on the size of $Vars(\varphi)$. The case $|Vars(\varphi)| = 2$ is obvious. We assume that the direction from right to left holds for $|Vars(\varphi)| \leq m$. Next we analyze the case $|Vars(\varphi)| = m + 1$. Let x be a variable in $Ch_{\mathcal{T}}(glb(\varphi))$ such that there exists some variable $x' \in Vars(\varphi)$ with $x \preceq_{\mathcal{T}} x'$. Let Ω be the set of assignments over $V(\mathcal{T}_x) \cap Vars(\varphi)$ such that the conditioning of φ on each assignment is consistent. Given two assignments $\omega, \omega' \in \Omega$, we have $\varphi|_\omega \equiv \varphi|_{\omega'}$. Let φ' be the formula whose variable set is $V(\mathcal{T}_x) \cap Vars(\varphi)$ and whose model set is Ω. That is, $\Psi = \{\varphi|_\omega, \varphi'\}$ is a $\wedge$-decomposition of φ. If $\varphi|_\omega$ satisfies the condition that there exists some child x'' of $glb(\varphi)$ in $\mathcal{T}$ such that all variables are from $V(\mathcal{T}_{x''})$, then Ψ is a strict $\wedge_{\mathcal{T}}$-decomposition of φ. Otherwise, it is obvious that $\varphi|_\omega$ satisfies the right condition of the conclusion; that is, for any child x'' of $glb(\varphi|_\omega)$ in $\mathcal{T}$ and any two assignments ω_1 and ω_2 over $V(\mathcal{T}_{x''})$ satisfying that if both $\varphi|_{\omega \cup \omega_1}$ and $\varphi|_{\omega \cup \omega_2}$ are consistent, we have $\varphi|_{\omega \cup \omega_1} \equiv \varphi'|_{\omega \cup \omega_2}$. Therefore, $\varphi|_\omega$ has some strict $\wedge_{\mathcal{T}}$-decomposition Ψ' by the induction hypothesis. That is, $\Psi' \cup \{\varphi'\}$ is a strict $\wedge_{\mathcal{T}}$-decomposition of φ. □

We denote $\vartheta(u) \vee \vartheta(v)$ by φ and the $\wedge_{\widehat{\mathcal{T}}}$-decomposition of $\vartheta(u)$ by Ψ. Then we proceed by case analysis:

- $\vartheta(u)$ and $\vartheta(v)$ do not share any variables: According to the dual case of Observation 1b, φ is not strictly $\wedge$-decomposable. It is obvious $glb(\varphi) = glb(u) \prec_{\mathcal{T}} glb(v)$. Since $glb(u) \notin Vars(u)$ and there is not any inessential variable in φ, we have $glb(\varphi) \notin Vars(\varphi)$. Therefore, φ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ by Observation 8.
- Otherwise, let x be $ch_{\mathcal{T}}(glb(u) \rightsquigarrow glb(v))$, and it is obvious that $Ch_x(u) \subset Ch(u)$: We denote $mch_x(u)$ and $u \setminus Ch_x(u)$ by w and u', respectively. Obviously, there exist assignments ω_1 and ω_2 over $Vars(u')$ such that $\omega_1 \models \vartheta(u')$ and $\omega_2 \not\models \vartheta(u')$. We also know $\vartheta(w) \not\models \vartheta(v)$, and otherwise $\vartheta(u) \models \vartheta(v)$. Then we have that $\varphi|_{\omega_1} \equiv \vartheta(w) \vee \vartheta(v)$ is not equivalent to $\varphi|_{\omega_2} \equiv \vartheta(v)$. Since $\vartheta(w) \vee \vartheta(v)$ and $\vartheta(v)$ are consistent, we have that $\lfloor \varphi \rfloor$ is not strictly $\wedge_{\mathcal{T}}$-decomposable by Lemma 3, and also have that φ depends on some variable $x' \in Vars(u')$. Moreover, there exist assignments ω_3 and ω_4 over $Vars(w) \cup Vars(v)$ such that $\omega_3 \models \vartheta(w)$, $\omega_3 \not\models \vartheta(v)$ and $\omega_4 \models \vartheta(v)$. Therefore, $\varphi|_{\omega_3} \equiv \vartheta(u')$ is not equivalent to $\varphi|_{\omega_4} \equiv true$, which implies that φ depends some variable $x'' \in Vars(w) \cup Vars(v)$. Since $glb(\lfloor \varphi \rfloor) \preceq_{\mathcal{T}} glb(\{x', x''\}) = glb(u)$, we know $glb(\lfloor \varphi \rfloor) \notin Vars(\varphi)$, which implies that $glb(\lfloor \varphi \rfloor) \notin Vars(\lfloor \varphi \rfloor)$. Therefore, φ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ by Observation 8.

A.29 Proof of Observation 17

We denote $\vartheta(u) \vee \vartheta(v)$ by φ. Then we show that if there exist two different meta-pairs between u and v, φ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$. According to Observation 8, we only need to show that $\lfloor \varphi \rfloor$ is not strictly $\wedge_{\mathcal{T}}$-decomposable and $glb(\lfloor \varphi \rfloor) \notin Vars(\lfloor \varphi \rfloor)$. For the second assertion, we only need to show that there exist two variables x' and x'' in $\lfloor \varphi \rfloor$ with $glb(\{x', x''\}) = glb(u)$, because $glb(\lfloor \varphi \rfloor) \preceq_{\mathcal{T}} glb(\{x, x'\})$ and $glb(u) \prec_{\mathcal{T}} Vars(\varphi)$. We proceed with case analysis:

- There exists some different meta-pair $(w, \langle \top \rangle)$ between u and v: We denote $u \setminus \{w\}$ by u'. We have $\vartheta(u') \not\models \vartheta(v)$, and otherwise $\vartheta(u) \models \vartheta(v)$. There exist two assignments ω_1 and ω_2 over $Vars(w)$ such that $\omega_1 \models \vartheta(w)$ and $\omega_2 \not\models \vartheta(w)$. We have that $\varphi|_{\omega_1} \equiv \vartheta(u') \vee \vartheta(v)$ is not equivalent to $\varphi|_{\omega_2} \equiv \vartheta(v)$, and otherwise $\vartheta(u') \models \vartheta(v)$. Since both $\varphi|_{\omega_1}$ and $\varphi|_{\omega_2}$ are consistent, we have that $\lfloor \varphi \rfloor$ is not strictly $\wedge_{\mathcal{T}}$-decomposable according to Lemma 3, and also have that φ depends on some variable $x' \in Vars(w)$. Moreover, since $\vartheta(u') \not\models \vartheta(v)$, there exist two assignments ω_3 and ω_4 over $Vars(u') \cup Vars(v)$ such that $\omega_3 \models \vartheta(u')$, $\omega_3 \not\models \vartheta(v)$ and $\omega_4 \models \vartheta(v)$. Since $\varphi|_{\omega_3} \equiv \vartheta(w)$ is not equivalent to $\varphi|_{\omega_2} \equiv true$, φ depends some variable $x'' \in Vars(u') \cup Vars(v)$. Therefore, $glb(\{x', x''\}) = glb(u)$.
- There exists some different meta-pair $(\langle \top \rangle, w)$ between u and v: It is similar to the last case.
- Otherwise, we assume that (w, w') is some different meta-pair of two internal vertices between u and v: If w (resp. w') is a $\wedge$-vertex, we denote $Ch(w)$ (resp. $Ch(w')$) by W (resp. W'), and otherwise we denote $\{w\}$ (resp. $\{w'\}$) by W (resp. W'). Then we denote $u \setminus W$ and $v \setminus W'$ by u' and v', respectively. It is obvious $\vartheta(u') \not\equiv \vartheta(v')$ since there exists another different meta-pair between u and v. We proceed with case analysis:
  - $\vartheta(w) \models \vartheta(w')$: Obviously, there exist assignments ω_1 and ω_2 over $Vars(w) \cup Vars(w')$ such that $\omega_1 \models \vartheta(w)$, $\omega_2 \not\models \vartheta(w)$ and $\omega_2 \models \vartheta(w')$. We also know $\vartheta(u') \not\models \vartheta(v')$, and otherwise $\vartheta(u) \models \vartheta(v)$. We have that $\varphi|_{\omega_1} \equiv \vartheta(u') \vee \vartheta(v')$ is not equivalent to $\varphi|_{\omega_2} \equiv \vartheta(v')$. Therefore, we have that $\lfloor \varphi \rfloor$ is not strictly $\wedge_{\mathcal{T}}$-decomposable

by Lemma 3, and that φ depends on some variable $x \in Vars(w) \cup Vars(w')$. Moreover, there exist assignments ω_3 and ω_4 over $Vars(u') \cup Vars(v')$ such that $\omega_3 \models \vartheta(u')$, $\omega_3 \not\models \vartheta(v')$ and $\omega_4 \models \vartheta(v')$. Therefore, $\varphi|_{\omega_3} \equiv \vartheta(w)$ is not equivalent to $\varphi|_{\omega_4} \equiv \vartheta(w')$. That is, φ depends on some variable $x' \in Vars(u') \cup Vars(v')$. Therefore, $glb(\{x, x'\}) = glb(u)$.

– $\vartheta(w') \models \vartheta(w)$: It is similar to the case $\vartheta(w) \models \vartheta(w')$.
– Otherwise, $\vartheta(w) \not\models \vartheta(w')$ and $\vartheta(w') \not\models \vartheta(w)$: Obviously, there exist assignments ω_1 and ω_2 over $Vars(w) \cup Vars(w')$ such that $\omega_1 \models \vartheta(w)$, $\omega_1 \not\models \vartheta(w')$, $\omega_2 \not\models \vartheta(w)$ and $\omega_2 \models \vartheta(w')$. We have that $\varphi|_{\omega_1} \equiv \vartheta(u')$ is not equivalent to $\varphi|_{\omega_2} \equiv \vartheta(v')$. Therefore, we have that $\lfloor\varphi\rfloor$ is not strictly $\wedge_{\mathcal{T}}$-decomposable by Lemma 3, and that φ depends on some variable $x \in Vars(w) \cup Vars(w')$. Since $\vartheta(u') \not\equiv \vartheta(v')$, there exists some assignment ω_3 over $Vars(u') \cup Vars(v')$ such that $\omega_3 \models \vartheta(u')$ and $\omega_3 \not\models \vartheta(v')$, or there exists some assignment ω_3 over $Vars(\varphi'_1) \cup Vars(\varphi'_2)$ such that $\omega_3 \not\models \vartheta(u')$ and $\omega_3 \models \vartheta(v')$; and these two cases are dual to each other. Without loss of generality, we only analyze the first case. Obviously, there exists some assignments ω_4 over $Vars(u') \cup Vars(v')$ such that $\omega_4 \models \vartheta(v')$. Therefore, $\varphi|_{\omega_3} \equiv \vartheta(w)$ is not equivalent to $\varphi|_{\omega_4} \models \vartheta(w')$. That is, φ depends on some variable $x' \in Vars(u') \cup Vars(v')$. Therefore, $glb(\{x, x'\}) = glb(u)$.

A.30 Proof of Proposition 11

We first prove the soundness of Algorithm DisjoinTree by induction on the sum of $|V(\mathcal{G}_u)|$ and $|V(\mathcal{G}_v)|$. For the case where either u or v is a leaf vertex, it is implicitly mentioned on Lines 2–3 and thus is obvious. We assume that for the case where $|V(\mathcal{G}_u)| + |V(\mathcal{G}_v)| \leq n$, the algorithm returns the ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ vertex representing $\vartheta(u) \vee \vartheta(v)$ iff $\vartheta(u) \vee \vartheta(v)$ can be represented in ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$. For the case where $|V(\mathcal{G}_u)| + |V(\mathcal{G}_v)| = n + 1$, we proceed by case analysis:

- The condition on Line 2 or 3 is satisfied: This case is obvious.
- The condition on Line 4, 7 or 10 is satisfied: We first show that if the algorithm does not report failure, its output is the ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ vertex representing $\vartheta(u) \vee \vartheta(v)$:
  – The condition on Line 4 is satisfied: By the induction hypothesis, u_1 (resp. u_2) on Line 5 is an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ vertex equivalent to $\vartheta(lo(u)) \vee \vartheta(v)$ (resp. $\vartheta(hi(u)) \vee \vartheta(v)$). Therefore, $\langle sym(u), u_1, u_2\rangle$ is an OBDD[$\wedge_{\mathcal{T}}$]$_{\mathcal{T}}$ vertex equivalent to $\vartheta(u) \vee \vartheta(v) \equiv [\neg sym(u) \wedge (\vartheta(lo(u)) \vee \vartheta(v))] \vee [sym(u) \wedge (\vartheta(hi(u)) \vee \vartheta(v))]$. By Proposition 5, the result is an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ after Line 19.
  – The condition on Line 7 is satisfied: This case can be proven in a similar fashion to that of the case where the condition on Line 4 is satisfied.
  – The condition on Line 10 is satisfied: By the induction hypothesis, u_1 (resp. u_2) on Line 11 is an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ vertex equivalent to $\vartheta(lo(u)) \wedge \vartheta(lo(v))$ (resp. $\vartheta(hi(u)) \wedge \vartheta(hi(v))$). Therefore, $\langle sym(u), u_1, u_2\rangle$ is an OBDD[$\wedge_{\mathcal{T}}$]$_{\mathcal{T}}$ vertex equivalent to $\vartheta(u) \vee \vartheta(v) \equiv [\neg sym(u) \wedge (\vartheta(lo(u)) \vee \vartheta(lo(v)))] \vee [sym(u) \wedge (\vartheta(hi(u)) \vee \vartheta(hi(v)))]$. By Proposition 5, the result is an ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$ after Line 19.

  According to Corollary 4, given any two formulas φ and φ' and assignment ω, if $\varphi \vee \varphi'$ can be represented in ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$, then $(\varphi \vee \varphi')|_{\omega}$ can also be represented in ROBDD[$\wedge_{\widehat{\mathcal{T}}}$]$_{\mathcal{T}}$.

According to the induction hypothesis, if the algorithm reports failure, $\vartheta(u) \vee \vartheta(v)$ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$.

- The conditions on Lines 13–14 are satisfied: Let u and v be two $\wedge$-vertices such that $glb(u) = glb(v)$ and there is exactly one different meta-pair (w, w') between u and v. $\{\vartheta(w'') : w'' \in Ch(u) \text{ and } w'' \neq w\} \cup \{\vartheta(w) \vee \vartheta(w')\}$ is a $\wedge_{\mathcal{T}}$-decomposition of $\vartheta(u) \vee \vartheta(v)$. That is, $\vartheta(u) \vee \vartheta(v)$ can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$, iff $\vartheta(w) \vee \vartheta(w')$ can also be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$. Then we immediately have the conclusion from the induction hypothesis.
- Otherwise: According to Observations 16–17, $\vartheta(u) \vee \vartheta(v)$ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$.

We then analyze the time complexity. For each single calling of DisjoinTree(w, w'), w and w' are meta-vertices in $\mathcal{G}_u$ and $\mathcal{G}_v$, respectively. We know that the number of single callings of DisjoinTree is not more than $|\mathcal{G}_u| \cdot |\mathcal{G}_v|$ by Observation 7. In addition, we need to compute the entailment relation between meta-vertices of $\mathcal{G}_u$ and $\mathcal{G}_v$. We can use the algorithm EntailTree to do this in the preprocessing stage. By use of hash table, it is obvious that we only call EntailTree at most once for each pair of meta-vertices respectively in $\mathcal{G}_u$ and $\mathcal{G}_v$, and thus the number of single callings of EntailTree is not more than $|\mathcal{G}_u| \cdot |\mathcal{G}_v|$; that is, the calling time is bounded by $O((|Vars(u)| + |Vars(v)|) \cdot |\mathcal{G}_u| \cdot |\mathcal{G}_v|)$. When the entailment relation on Lines 2–3 is already known, each single calling of DisjoinTree can be done in $O(|Vars(u)| + |Vars(v)|)$. Therefore, the time complexity of DisjoinTree is $O((|Vars(u)| + |Vars(v)|) \cdot |\mathcal{G}_u| \cdot |\mathcal{G}_v|)$.

A.31 Proof of Proposition 12

Membership is immediate from the fact that the problem of deciding the entailment relation of two propositional formulas is in co-NP. The hardness is proved by taking advantage of the idea that was used to prove the complexity of deciding the entailment relation of two free BDDs by Fortune, Hopcroft, and Schmidt (1978). That is, we reduce the problem of deciding the entailment relation of two ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s into 3UNSAT. For each 3-CNF formula

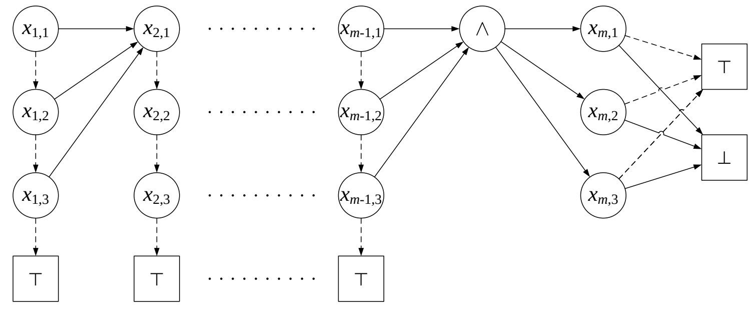
$$\varphi = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge \cdots \wedge (l_{m,1} \vee l_{m,2} \vee l_{m,3}),$$

we prove that it is unsatisfiable iff an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ entails another one, where the sizes of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s are polynomial in $|\varphi|$. For notational simplicity, we assume that $\mathcal{C}$ is depicted in Figure 2a and $x_1, \ldots, x_n$ are the variables in $Vars(\varphi)$. For other chains and variables, we can proceed in a similar fashion only with some substitutions of variables. We introduce a new variable $x_{k,c}$ ($1 \leq k \leq m, 1 \leq c \leq 3$) for each $l_{k,c}$. Let $x^-_{j,1}, \ldots, x^-_{j,a_j}$ and $x^+_{j,1}, \ldots, x^+_{j,b_j}$ be the variables in sets $\{x_{k,c} : l_{k,c} = \neg x_j\}$ and $\{x_{k,c} : l_{k,c} = x_j\}$, respectively. The reader can verify that φ is satisfiable iff the conjunction of the following two formulas is satisfiable:

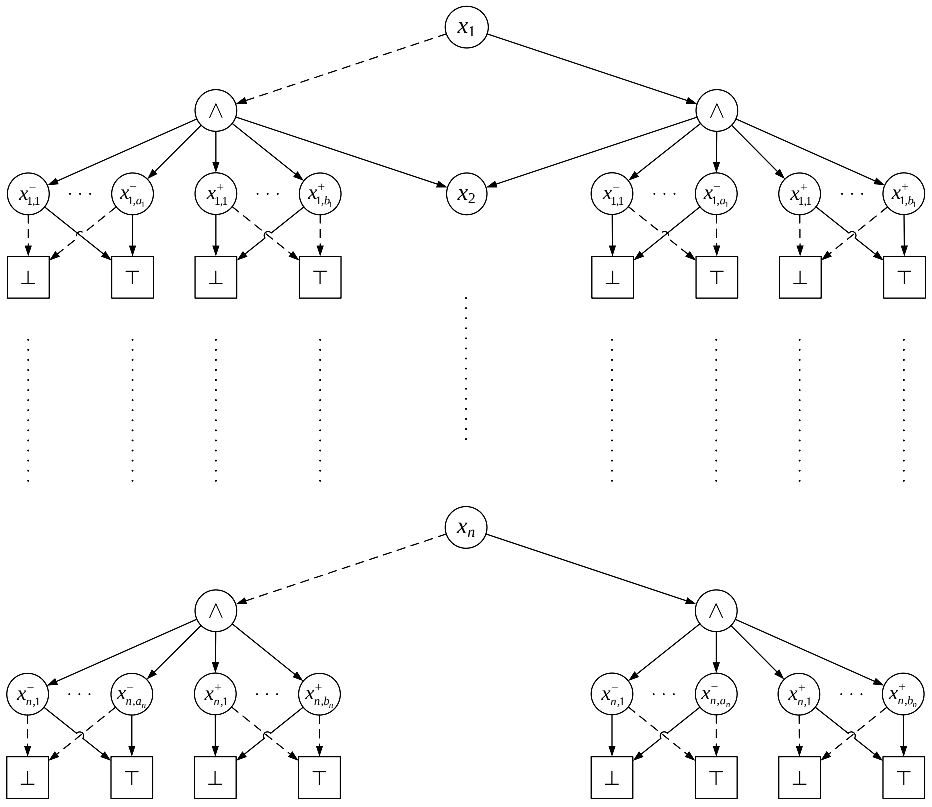
$$\varphi_1 = (x_{1,1} \vee x_{1,2} \vee x_{1,3}) \wedge \cdots \wedge (x_{m,1} \vee x_{m,2} \vee x_{m,3}),$$

$$\varphi_2 = \bigwedge_{1 \leq j \leq n} \left[\bigwedge_{1 \leq k \leq a_j} x_j \leftrightarrow \neg x^-_{j,a_k} \right] \wedge \left[\bigwedge_{1 \leq k \leq b_j} x_j \leftrightarrow x^+_{j,b_k} \right].$$

That is, φ is unsatisfiable iff $\varphi_2 \models \neg\varphi_1$, where $\neg\varphi_1$ and φ_2 are equivalent to the two ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s in Figure 5a and 5b, respectively.



(a)



(b)

Figure 5: Two ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s ($i > 0$), where $\mathcal{C}$ is depicted in Figure 2a

A.32 Proof of Theorem 7

The tractability results of ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$ is known previously since it is equivalent to ROBDD over $\mathcal{C}$. From the polytime operation algorithms in Subsection 7.2, we can immediately obtain many positive results. The negative result of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($i > 0$) on **SE** immediately follows from Proposition 12, which implies that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ does not satisfy $\wedge$**BC** (resp. $\wedge$**C**, $\vee$**BC**, $\vee$**C**, **SFO** and **FO**) unless P = NP, because by performing the latter operation and CT, we can indirectly perform SE. Next we explain the remaining results:

- ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ satisfies $\neg$**C**: We next prove by induction that given an ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ rooted at u with some $\wedge$-vertex, $\neg\vartheta(u)$ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$; in other words, for each ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ without any $\wedge$-vertex, we can employ the negation algorithm for ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$ to perform its negation. The case $sym(u) = \bot$ or $\top$ is obvious. We assume the assertion holds for $V(\mathcal{G}_u) \leq n$. Next we analyze the case $V(\mathcal{G}_u) = n + 1$. If $sym(u) = \wedge$, the assertion is obvious by Observation 8, since $\neg\vartheta(u)$ is not strictly $\wedge$-decomposable and $glb(u) \notin Vars(u)$. Otherwise, without loss of generality, we assume that there exists some $\wedge$-vertex v in $\mathcal{G}_{lo(u)}$. If $\neg\vartheta(u)$ can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$, then $\neg\vartheta(lo(u))$ can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ according to Algorithm 2, which contradicts with the induction hypothesis.
- ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ ($dep(\mathcal{T}) < \infty$) satisfies $\wedge$**C**: By Observation 14, the number of vertices in each ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$ over X is not more than $3 \cdot |X| \cdot 2^{dep(\mathcal{T})} + 1$. Therefore, conjoining two ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$s can be done in $O(|X|^3 \cdot 4^{dep(\mathcal{T})})$. For m ROBDD$[\wedge_{\widehat{\mathcal{T}}}]_{\mathcal{T}}$s, we can conjoin them by $m - 1$ callings of binary conjoining in $O(m \cdot |X|^3 \cdot 4^{dep(\mathcal{T})})$.
- ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($i > 0$) does not satisfy $\neg$**C**: For notational simplicity, we assume that $\mathcal{C}$ is depicted in Figure 2a. For other chains, we can proceed in a similar fashion only with variable substitutions. The ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $\mathcal{G}$ representing φ has a linear size. However, after conditioning $\neg\varphi$ on any two different assignments on $x_1, \ldots, x_n$ will obtain two distinct sub-formulas which are not strictly $\wedge$-decomposable. Therefore, the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ representing $\neg\varphi$ has an exponential size by Proposition 7.
- ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($dep(\mathcal{T}) = \infty$) satisfies none of $\wedge$**C**, $\vee$**C** and **FO** unless P = NP: Let $\mathcal{C}'$ be an infinite path in $\mathcal{T}$. The problem of deciding the consistency of a CNF formula over $V(\mathcal{C}')$ is NP-complete. It is obvious that each clause on $V(\mathcal{C}')$ can be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$. If ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ satisfies $\wedge$**C**, a CNF formula over $V(\mathcal{C}')$ can be transformed into ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ in polytime, which implies that the consistency of CNF formulas over $V(\mathcal{C}')$ can be checked in polytime (i.e., P = NP). $\vee$**C** is a dual case of $\wedge$**C**, and for ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$, $\vee$**C** is satisfied if **FO** is satisfied.
- ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($dep(\mathcal{T}) < \infty$) satisfies neither $\vee$**C** nor **FO** unless P = NP: Since PV is infinite, there exists some x in $\mathcal{T}$ that has an infinite number of children. We assume that the children of variable x in $\mathcal{T}$ are $X = \{x_{k_1}, \ldots, x_{k_n}, \ldots\}$. Therefore, the problem of deciding validity of DNF over X is co-NP-complete. Given a DNF formula $\varphi = \gamma_1 \vee \cdots \vee \gamma_m$, each term γ_j on X can be represented by ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $\mathcal{G}_j$. Therefore, if ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($dep(\mathcal{T}) < \infty$) satisfies $\vee$**C**, then we can decide the validity of φ by determining whether the disjunction of $\mathcal{G}_1, \ldots, \mathcal{G}_m$ is $\langle\top\rangle$. Finally, ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ ($dep(\mathcal{T}) < \infty$) satisfies $\vee$**C** iff it satisfies **FO**.

A.33 Proof of Proposition 13

We prove the soundness by induction on the size of $V(\mathcal{G}_u)$, where u is the input. The basic case $|V(\mathcal{G}_u)| = 1$ is obvious. We assume that when $|V(\mathcal{G}_u)| \leq n$, the output u' of ConvertDown(u) is the equivalent ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex with $Vars(u) = Vars(u')$. For the case $|V(\mathcal{G}_u)| = n + 1$, we proceed with case analysis:

- $sym(u) \in PV$: According to the induction hypothesis, it is easy to show $\vartheta(u') \equiv \vartheta(u)$ and $Vars(u) = Vars(u')$. Since $\vartheta(u)$ is not strictly $\wedge_j$-decomposable, $\vartheta(u')$ is not strictly

$\wedge_i$-decomposable. Since $sym(u) \prec_{\mathcal{C}} Vars(u)$, we have $sym(u') \prec_{\mathcal{C}} Vars(u')$. Therefore, u' is an ROBDD$[\wedge_{\hat{i}}]_{\mathcal{C}}$ vertex.

- $sym(u) = \wedge$: It is obvious that the vertices in ROBDD$[\wedge_{\hat{j}}]_{\mathcal{C}}$ with at most i variable are already ROBDD$[\wedge_{\hat{i}}]_{\mathcal{C}}$ vertices. Therefore, if the condition on Line 8 is satisfied, u is already an ROBDD$[\wedge_{\hat{i}}]_{\mathcal{C}}$ vertex. Otherwise, if the condition on Line 9 is satisfied, the result of ConvertDown(v) is obviously a $\diamond$-vertex, and thus u' is in ROBDD$[\wedge_{\hat{i}}]_{\mathcal{C}}$ according to the induction hypothesis. Otherwise, according to Corollary 1a, $\{\vartheta(v) : v \in Ch(u) \setminus V\} \cup \{\vartheta(\langle \wedge, V\rangle)\}$ is the $\wedge_{\hat{i}}$-decomposition of $\vartheta(u)$. Let x be $\min\{sym(v) : v \in V\}$. According to the soundness of ConditionMin and the induction hypothesis, v_1 and v_2 are the ROBDD$[\wedge_{\hat{i}}]_{\mathcal{C}}$ vertices representing $\vartheta(\langle \wedge, V\rangle)|_{x=false}$ and $\vartheta(\langle \wedge, V\rangle)|_{x=true}$, respectively. That is, $\langle x, v_1, v_2\rangle$ represents $\vartheta(\langle \wedge, V\rangle)$. Since $\vartheta(\langle \wedge, V\rangle)$ is not strictly $\wedge_i$-decomposable and x is the minimum variable, the output $\langle x, v_1, v_2\rangle$ is in ROBDD$[\wedge_{\hat{i}}]_{\mathcal{C}}$. Therefore, u' is an ROBDD$[\wedge_{\hat{i}}]_{\mathcal{C}}$ vertex representing $\vartheta(u)$.

It is obvious that each single calling of ConvertDown terminates in $O(|Vars(u)|)$ by Proposition 9. Since each variable in the input is also in the output, each single calling of ConvertDown terminates in $O(|Vars(v)|)$, where v is the output. It is obvious that each single calling of ConvertDown will generate at least one new ROBDD$[\wedge_{\hat{i}}]_{\mathcal{C}}$ vertex into vertex-table. Therefore, the number of single callings of ConvertDown is not more than $|V(\mathcal{G}_v)|$, and then the time complexity of ConvertDown is bounded by $O(|Vars(v)| \cdot |V(\mathcal{G}_v)|)$.

A.34 Proof of Proposition 14

If ConvertTree(u) does not report failure, it is easy to show that its output is the equivalent ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$ vertex, using a proof similar to the one in Subsection E.10. We next prove by induction that if ConvertTree(u) reports failure, then $\vartheta(u)$ cannot be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$. The basic case $|V(\mathcal{G}_u)| = 1$ is obvious. We assume that when $|V(\mathcal{G}_u)| \leq n$, the assertion holds. For the case $|V(\mathcal{G}_u)| = n + 1$, we proceed with case analysis:

- $sym(u) \in PV$: According to Corollary 2a, $\vartheta(u)$ is not strictly $\wedge_{\mathcal{T}}$-decomposable. Since $sym(u) \prec_{\mathcal{C}} Vars(u)$, for each $x \in Vars(u)$, $x \not\prec_{\mathcal{T}} sym(u)$. That is, $glb_{\prec_{\mathcal{T}}}(u) \in Vars(u)$ iff $glb_{\prec_{\mathcal{T}}}(u) = sym(u)$. Therefore, if $glb_{\prec_{\mathcal{T}}}(u) \neq sym(u)$, $\vartheta(u)$ cannot be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$ by Observation 8. Otherwise, by the induction hypothesis, $\vartheta(lo(u))$ (resp. $\vartheta(hi(u))$) can be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$ iff ConvertTree$(lo(u))$ (resp. ConvertTree$(hi(u))$) does not report failure. According to Algorithm 2, if $\vartheta(u)$ can be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$, then both $\vartheta(lo(u))$ and $\vartheta(hi(u))$ can be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$. That is, if ConvertTree(u) reports failure, then $\vartheta(u)$ cannot be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$.
- $sym(u) = \wedge$: Given a $\wedge$-vertex v and its children W, it is obvious that there exists some assignment ω such that $\vartheta(v)|_{\omega} \equiv \langle \wedge, W\rangle$. That is, if $\vartheta(v)$ can be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$, then $\vartheta(\langle \wedge, W\rangle)$ can also be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$ by Algorithm 2. For the case $glb_{\prec_{\mathcal{T}}}(v_k) \notin Vars(v_k)$, if $\vartheta(u)$ can be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$, then $\vartheta(v_k)$ is strictly $\wedge_{\mathcal{T}}$-decomposable, which contradicts with Corollary 2a. In other words, for the case $glb_{\prec_{\mathcal{T}}}(v_k) \notin Vars(v_k)$, $\vartheta(v_k)$ cannot be represented in ROBDD$[\wedge_{\hat{\mathcal{T}},i}]_{\mathcal{T}}$. For

the case where the recursive calling on Line 14, 15 or 17 reports failure, $\vartheta(v_k)$ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ by the the induction hypothesis, and thus $\vartheta(u)$ cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$. According to Corollary 2a, for the case where u' on Line 21 is not bounded by i, $\vartheta(u)$ also cannot be represented in ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$.

According to Lemma 2, we can compute all greatest lower bounds of all vertices in $\mathcal{G}_u$ over $\prec_{\mathcal{T}}$ in the preprocessing stage in time $O(|Vars(u)|\cdot|V(\mathcal{G}_u)|)$. When we call ConditionMin on Line 14 or 15, we can only compute the greatest lower bounds for the new vertices in vertex-table, and thus the time is bounded in $O(|Vars(u)|)$. It is obvious that Lines 8–9 can be computed in $O(|Vars(u)|^2)$. Therefore, each single calling of ConvertTree terminates in $O(|Vars(u)|^2)$. Since each variable in the input is also in the output, each single calling of ConvertDown terminates in $O(|Vars(v)|^2)$, where v is the output. Because each single calling of ConvertDown will generate at least one new ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ vertex into vertex-table, the time complexity of ConvertDown is bounded by $O(|Vars(v)|^2\cdot|V(\mathcal{G}_v)|)$.

A.35 Proof of Theorem 8

All positive rapidity results were explained in Section 7.3, except that ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}} \geq_r^{OP}$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ when $i = 0$ or $dep(\mathcal{T}) < \infty$. The case $i = 0$ is immediate from the fact ROBDD$[\wedge_{\widehat{\mathcal{T}},0}]_{\mathcal{T}} \subseteq$ ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}}$. We next explain that each ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ can be transformed into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ in polytime in the size of output, and then ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ $\geq_r^{OP}$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ when $dep(\mathcal{T}) < \infty$. Each ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$ can be transformed into ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ by Decompose. According to the canonicity of ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}}$, each single calling of Decompose will generate at least one new ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ vertex into vertex-table. Since ROBDD$[\wedge_{\widehat{\mathcal{T}},i}]_{\mathcal{T}} =_s$ ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$, the number of single callings of Decompose is polynomial in the size of output. Therefore, Decompose terminates in polytime in the size of output.

We next explain the negative rapidity results. For notational simplicity, we assume that $\mathcal{C}$ is depicted in Figure 2a. For other chains, we can proceed in a similar fashion with some substitutions of variables.

- ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{CD}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$: We consider the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ $\mathcal{G}$ and ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ $\mathcal{G}'$ representing Equation (1). Let ω be $\{x_{n+0\cdot n} = true, \ldots, x_{n+i\cdot n} = true\}$ and $\mathcal{G}''$ be the result of conditioning $\mathcal{G}'$ on ω. According to Proposition 7, the number of vertices in $\mathcal{G}''$ is exponential in $|\mathcal{G}|$. The reader can verify that more than half of the vertices in $\mathcal{G}''$ do not appear in $\mathcal{G}'$. That is, the number of new vertices in $\mathcal{G}''$ is exponential in $|\mathcal{G}|$, and thus the time of conditioning $\mathcal{G}'$ on ω is at least exponential in $|\mathcal{G}|$. However, the time of conditioning $\mathcal{G}$ on ω is polynomial in $|\mathcal{G}|$ according to Corollary 4.
- ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{\wedge BC}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{\wedge C}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$: It is well known that the conjunction of $\bigwedge_{1\leq k\leq n-1} x_{k+0\cdot n} \leftrightarrow \cdots \leftrightarrow x_{k+i\cdot n}$ and $x_{n+0\cdot n} \leftrightarrow \cdots \leftrightarrow x_{n+i\cdot n}$ is equivalent to Equation (1). We denote the two ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$s (resp. ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$s) representing the above two formulas by $\mathcal{G}_1$ and $\mathcal{G}_2$ (resp. $\mathcal{G}_3$ and $\mathcal{G}_4$). It is easy to design an algorithm to perform $\wedge BC$ or $\wedge C$ on ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ which can particularly conjoin $\mathcal{G}_1$ and $\mathcal{G}_2$ in polytime (we just need to detect whether the inputs are $\mathcal{G}_1$ and $\mathcal{G}_2$, and then immediately generate the ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ corresponding to Equation (1) if so). However, the conjunction of $\mathcal{G}_3$ and $\mathcal{G}_4$ will generate the ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ representing Equation (1)

with an exponential (in $|\mathcal{G}_1|$ and $|\mathcal{G}_2|$) number of new vertices; that is, for $\mathcal{G}_3$ and $\mathcal{G}_4$, the running time of each algorithm to perform $\wedge BC$ or $\wedge C$ on ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ is exponential in the sizes of $\mathcal{G}_1$ and $\mathcal{G}_2$.

- ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{\vee BC}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{\vee C}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$: It is well known that the disjunction of the following two formulas is equivalent to Equation (1):

$$x_{n+0\cdot n} \wedge (x_{n+1\cdot n} \leftrightarrow \cdots \leftrightarrow x_{n+i\cdot n}) \wedge \bigwedge_{1\le k\le n-1} x_{k+0\cdot n} \leftrightarrow \cdots \leftrightarrow x_{k+i\cdot n}$$

and

$$\neg x_{n+0\cdot n} \wedge \neg(x_{n+1\cdot n} \leftrightarrow \cdots \leftrightarrow x_{n+i\cdot n}) \wedge \bigwedge_{1\le k\le n-1} x_{k+0\cdot n} \leftrightarrow \cdots \leftrightarrow x_{k+i\cdot n}$$

Therefore, we can prove these two cases by proofs similar to those of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{\wedge BC}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{\wedge C}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$.

- ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{SFO}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{FO}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$: It is well known that forgetting the following formula on x_0 is equivalent to Equation (1):

$$\big[[\neg x_0 \wedge x_{n+0\cdot n} \wedge (x_{n+1\cdot n} \leftrightarrow \cdots \leftrightarrow x_{n+i\cdot n})] \vee [x_0 \wedge \neg x_{n+0\cdot n} \wedge \neg(x_{n+1\cdot n} \leftrightarrow \cdots \leftrightarrow x_{n+i\cdot n})]\big] \\ \wedge \bigwedge_{1\le k\le n-1} x_{k+0\cdot n} \leftrightarrow \cdots \leftrightarrow x_{k+i\cdot n}$$

Therefore, we can prove these two cases by proofs similar to those of ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{\wedge BC}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$ and ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{\wedge C}$ ROBDD$[\wedge_{\widehat{j}}]_{\mathcal{C}}$.

For the third conclusion, since $dep(\mathcal{T}) = \infty$, there exists some infinite path $\mathcal{C}'$ in $\mathcal{T}$. We substitute the kth variable in $\mathcal{C}'$ for each x_k in the counterexamples in the proof of the second conclusion. Then we can show by proofs similar to the ones in the second paragraph that for $i > 0$ and $OP \in \{CD, FO, SFO, \wedge C, \wedge BC, \vee BC, \vee C\}$, ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}'} \not\leq_r^{OP}$ ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}'}$. Because each ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}'}$ is an ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}}$ and each ROBDD$[\wedge_{\widehat{0}}]_{\mathcal{C}'}$ is an ROBDD$[\wedge_{\widehat{\mathcal{T},j}}]_{\mathcal{T}}$, we know that ROBDD$[\wedge_{\widehat{i}}]_{\mathcal{C}} \not\leq_r^{OP}$ ROBDD$[\wedge_{\widehat{\mathcal{T},j}}]_{\mathcal{T}}$.

References

Bertacco, V. (2003). *Achieving Scalable Hardware Verification with Symbolic Simulation.* Ph.D. thesis, Stanford University.

Bertacco, V., & Damiani, M. (1996). 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, October 7-9, 1996, Austin, TX, USA*, pp. 27–32.

Bodlaender, H. L. (1998). A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, *209*(1-2), 1–45.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691.

Cadoli, M., & Donini, F. M. (1997). A survey on knowledge compilation. *AI Communications*, *10*, 137–150.

Chavira, M., & Darwiche, A. (2008). On probabilistic inference by weighted model counting. *Artificial Intelligence*, *172*(6–7), 772–799.

Choi, A., & Darwiche, A. (2013). Dynamic minimization of sentential decision diagrams. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 187–194.

Darwiche, A. (2001a). Decomposable negation normal form. *Journal of the ACM*, *48*(4), 608–647.

Darwiche, A. (2001b). 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.

Darwiche, A. (2004). New advances in compiling cnf into decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pp. 328–332.

Darwiche, A. (2011). SDD: A new canonical representation of propositional knowledge bases. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 819–826.

Darwiche, A. (2014). Tractable knowledge representation formalisms. In Bordeaux, L., Hamadi, Y., & Kohli, P. (Eds.), *Tractability: Practical Approaches to Hard Problems*, pp. 141–172. Cambridge University Press.

Darwiche, A., & Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, *17*, 229–264.

Fargier, H., & Marquis, P. (2006). On the use of partially ordered decision graphs in knowledge compilation and quantified Boolean formulae. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, pp. 42–47.

Fortune, S., Hopcroft, J. E., & Schmidt, E. M. (1978). The complexity of equivalence and containment for free single variable program schemes. In *Proceedings of the 5th Colloquium on Automata, Languages and Programming*, pp. 227–240.

Gergov, J., & Meinel, C. (1994). Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Transactions on Computers*, *43*(10), 1197–1209.

Gogate, V., & Dechter, R. (2011). SampleSearch: Importance sampling in presence of determinism. *Artificial Intelligence*, *175*, 694–729.

Gogate, V., & Dechter, R. (2012). Importance sampling-based estimation over and/or search spaces for graphical models. *Artificial Intelligence*, *184-185*, 38–77.

Huang, J., & Darwiche, A. (2004). Using DPLL for efficient OBDD construction. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pp. 157–172.

Huang, J., & Darwiche, A. (2007). The language of search. *Journal of Artificial Intelligence Research*, *29*, 191–219.

Lai, Y., Liu, D., & Wang, S. (2013). Reduced ordered binary decision diagram with implied literals: A new knowledge compilation approach. *Knowledge and Information Systems*, *35*(3), 665–712.

Lind-Nielsen, J. (1996). BuDDy - a binary decision diagram package. Available from http://buddy.sourceforge.net.

Lv, G., Su, K., & Xu, Y. (2013). CacBDD: A BDD package with dynamic cache management. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV)*, pp. 229–234.

Mateescu, R., Dechter, R., & Marinescu, R. (2008). AND/OR Multi-Valued Decision Diagrams (AOMDDs) for Graphical Models. *Journal of Artificial Intelligence Research*, *33*, 465–519.

Muise, C. J., McIlraith, S. A., Beck, J. C., & Hsu, E. I. (2012). Dsharp: Fast d-DNNF compilation with sharpSAT. In *Proceedings of the 25th Canadian Conference on Artificial Intelligence*, pp. 356–361.

Oztok, U., & Darwiche, A. (2014). On compiling CNF into Decision-DNNF. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 42–57.

Oztok, U., & Darwiche, A. (2015). A top-down compiler for sentential decision diagrams. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3141–3148.

Palacios, H., Darwiche, A., Bonet, B., & Geffner, H. (2005). Pruning conformant plans by counting models on compiled d-DNNF representations. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 141–150.

Pipatsrisawat, T. (2010). *Reasoning with Propositional Knowledge: Frameworks for Boolean Satisfiability and Knowledge Compilation.* Ph.D. thesis, University of California, Los Angeles.

Selman, B., & Kautz, H. (1996). Knowledge compilation and theory approximation. *Journal of the ACM*, *43*, 193–224.

Sieling, D., & Wegener, I. (1993). NC-algorithms for operations on binary decision diagrams. *Parallel Processing Letters*, *3*, 3–12.

Somenzi, F. (2002). CUDD: CU decision diagram package release 2.5.0. Available from ftp://vlsi.colorado.edu/pub/.

Van den Broeck, G., & Darwiche, A. (2015). On the role of canonicity in knowledge compilation. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1641–1648.

Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams.* SIAM.