Ordered \{AND, OR\}-Decomposition and Binary-Decision Diagram

Yong Lai\textsuperscript{a}, Dayou Liu\textsuperscript{a,b}

\textsuperscript{a}College of Computer Science and Technology, Jilin University, Changchun 130012, P.R. China
\textsuperscript{b}Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun 130012, P.R. China

Abstract

In the context of knowledge compilation (KC), we study the effect of augmenting Ordered Binary Decision Diagrams (OBDD) with two kinds of decomposition nodes, i.e., AND-vertices and OR-vertices which denote conjunctive and disjunctive decomposition of propositional knowledge bases, respectively. The resulting knowledge compilation language is called Ordered \{AND, OR\}-decomposition and binary-Decision Diagram (OAODD). Roughly speaking, several previous languages can be seen as special types of OAODD, including OBDD, AND/OR Binary Decision Diagram (AOBDD), OBDD with implied Literals (OBDD-L), Multi-Level Decomposition Diagrams (MLDD). On the one hand, we propose some families of algorithms which can convert some fragments of OAODD into others; on the other hand, we present a rich set of polynomial-time algorithms that perform logical operations. According to these algorithms, as well as theoretical analysis, we characterize the space efficiency and tractability of OAODD and its some fragments with respect to the evaluating criteria in the KC map. Finally, we present a compilation algorithm which can convert formulas in negative normal form into OAODD.

Keywords: knowledge compilation, target language, ordered binary decision diagram, AND-decomposition, OR-decomposition

1. Introduction

Knowledge compilation (KC) is a key approach to dealing with the computational intractability of general propositional reasoning [1–3]. According to this approach, the reasoning process is split into two phases: an off-line compilation phase, in which a propositional knowledge base is compiled into some tractable target language, and an on-line query-answering phase, in which the compiled target is used to efficiently answer the queries. The target language is one of the key aspects for any compilation approach. Over the years, dozens of target languages have been proposed, which are suitable for different applications in practice.

Due to the large number of the existing target languages, it is of non-triviality to choose a suitable target compilation language in practice. For a specific type of applications, three of the key considerations when adopting a language are the succinctness of the language, the class of queries and transformations supported in polytime, and the canonicity whether the language possesses [2, 4]. The former two factors can characterize space-efficiency and tractability, respectively, while the canonicity facilitates the search for optimal compilations. Each of the existing target languages can be viewed as a point in this three-dimensional structure. There is a trade-off between succinctness and the other two dimensions. On the one hand, Negative Normal Form (NNF) is the most succinct language, but it does not qualify as a target language since it does not satisfy any polytime querying requirement, particularly including polytime clausal entailment, which is required for any target language. On the other hand, Ordered Binary Decision Diagram (OBDD) is one of the most tractable languages and has canonicity, but it is strictly less succinct than many
other languages. One of the main tasks of research in KC community is to explore good balance points which are quite succinct, but still has canonicity and can efficiently support many operations.

The decomposability of propositional knowledge bases provides an important hint to quest good target languages, which are verified by many previous works. For a Boolean function \( \varphi \) over \( X \), if \( \varphi(X) \equiv \varphi'(\psi_1(X_1), \ldots, \psi_i(X_i)) \), where \( X_1, \ldots, X_i \) partition \( X \), we say \( \varphi \) is decomposable with respect to \( \psi' \). In this paper, we focus on AND-decomposition, OR-decomposition and NOR-decomposition; that is, \( \varphi = \psi_1(X_1) \land \ldots \land \psi_i(X_i) \). \( \varphi = \psi_1(X_1) \lor \ldots \lor \psi_i(X_i) \) and \( \varphi = \neg(\psi_1(X_1) \lor \ldots \lor \psi_i(X_i)) \). The previous works which exploited decomposability can be divided into two main research lines:

- The first one consists of the researches which relax some requirements of OBDD. Bertacco and Damiani [5] introduced NOR-only\(^1\) decomposition into OBDDs to propose Multi-Level Decomposition Diagrams (MLDDs). The MLDDs admit complement edges, but they are restricted to connect a non-NOR vertex to an NOR one. The authors pointed out that this language has canonicity over a given variable order, and proposed an algorithm called mldd\_find which can translate ROBDDs into MLDDs in polytime. Mateescu, Dechter and Marinescu [6] relaxed the orderliness of Ordered Multi-Valued Decision Diagrams (OMDDs) to by augmenting them with AND vertices, and the resulting language is called AND/OR Multi-valued Decision Diagrams (AOMDDs). In this paper, we focus on the binary cases which are called AOBDDs. Instead of linear order, the decision vertices\(^2\) in an AOBDD only need to comply with a partial order captured by a data structure pseudo tree. There is a canonical AOBDD for any Boolean formula \( \varphi \) if \( \varphi \) can be decomposed relative to a given pseudo tree, and there does not exist any AOBDD representing \( \varphi \) otherwise. AOBDD satisfies all querying and transformation requirements which involves only one AOBDD, except singleton forgetting. Moreover, given two AOBDDs respectively based on two strictly compatible pseudo trees, bounded conjunction, equivalence check and sentential entailment check can be done in polytime. Lai, Liu and Wang [7] generalized OBDD by associating some implied literals (a special type of AND-decomposition captured by literals) with each node to propose a language called OBDD-L. A special subset of OBDD-L called OBDD with as many as possible implied literals (OBDD-L\(_{\infty}\)) possesses canonicity over any given linear order of variables. Moreover, it was shown by proposing an algorithm called Inf2ROBDD that any OBDD-L\(_{\infty}\) can be converted into an equivalent OBDD in linear time in the size of result. Therefore, given any logical operation OBDD admits in polytime, OBDD-L\(_{\infty}\) can also support it in time polynomial in the size of the equivalent OBDD.

- The second one includes the works which impose some types of restrictions on NNF. Darwiche [8] imposed AND-type decomposability over NNF and then proposed a target language called decomposable negation normal form (DNNF) which supports a rich class of operations in polytime. It was shown in [2] that DNNF is strictly more succinct than OBDD. In [9], Darwiche imposed determinism over DNNF and obtained deterministic DNNF (d-DNNF), which is less succinct than DNNF but supports more querying requirements. Afterward, Pipatsrisawat and Darwiche [10] studied two subsets of DNNF and d-DNNF with respect to structured decomposition, that is, Structured DNNF (SDNNF) and Structured d-DNNF (d-SDNNF). In [4], Darwiche proposed a new canonical target called Sentential Decision Diagram (SDD) which are formulas in SDNNF only admit strongly deterministic AND-decomposition. SDD supports all binary logical operations.

Inspired by the above works (mainly by the ones on the former line), we study the effect of augmenting OBDD with two kinds of decomposition nodes, i.e., AND-vertices and OR-vertices which denote conjunctive and disjunctive decomposition of propositional knowledge bases, respectively. The resulting knowledge compilation language is called Ordered \{AND, OR\}-decomposition and binary-Decision Diagram (OAOADD). We impose two constraints on OAOADD to obtain two types of its fragments. The first one is called decomposition bounded by some integer \( i \), which means that there exists at most one child of a decomposition vertex which has more than \( i \) variables. The second one is called tree-structured decomposition, which means that for any two different children and a tree over variables, their variables are incomparable over ancestor-descendant relation. For any OAOADD and two integers \( 0 \leq i, j \leq \infty \), if each AND-vertex (OR-

---

\(^1\)It was pointed out in [5] that NOR-decomposition can be replaced with other types of decomposition (e.g., NAND-decomposition and AND/OR-decomposition) and then similar results can be obtained.

\(^2\)In [6], they are called OR nodes, where OR means OR search.
vertex) is bounded by \( i \) (\( j \)), this \( \text{OAODD} \) is denoted by \( \text{OA}_{\leq i, \leq j} \text{DD} \); in particular, this \( \text{OAODD} \) is
denoted by \( \text{OA}_{\leq i} \text{DD} \) if each decomposition is finest. For any \( \text{OAODD} \) and a tree \( \mathcal{T} \) over variables, if
each decomposition vertex respects \( \mathcal{T} \), this \( \text{OAODD} \) is denoted by \( \text{OAODD}_\mathcal{T} \). We show that \( \text{OAODD} \) and
its fragments have the following properties:

1. Given any variable order and tree \( \mathcal{T} \), Reduced \( \text{OA}_{\leq i} \text{DD} \) (ROA_{\leq i} \text{DD}) and Reduced \( \text{OAODD} \)
(ROAODD) has canonicity. We show that OBDD, OBDD-L_{\infty}, AOBDD and MLDD can be converted
into \( \text{OA}_{\leq 0, \leq 0} \text{DD} \), \( \text{OA}_{\leq 1} \text{DD} \), ROA_{\leq \infty} \text{DD}_\mathcal{T} \) and ROA\_{\leq \infty} \text{DD}, respectively, and vice versa.

2. Let \( i, j, i' \) and \( j' \) be four integers such that \( 0 \leq i \leq i' \leq \infty \) and \( 0 \leq j \leq j' \leq \infty \). We propose several
algorithms which can convert one fragment into another, including:
- an algorithm called \text{DECOMPOSE} which can convert any \( \text{OA}_{\leq i} \text{DD} \) into \( \text{ROA}_{\leq i} \text{DD} \) in
linear time in the number of variables and the size of input;
- an algorithm called \text{DECOMPOSETREE} which can convert any \( \text{OA}_{\leq i} \text{DD}_\mathcal{T} \) into \( \text{ROA}_{\leq i} \text{DD}_\mathcal{T} \)
in linear time in the number of variables and the size of input;
- an algorithm called \text{CONVERTDOWN} which can convert any \( \text{ROA}_{\leq i} \text{DD} \) into \( \text{ROA}_{\leq j} \text{DD} \)
in linear time in the number of variables and the size of output;
- an algorithm called \text{CONVERTTREE} which can convert any \( \text{ROA}_{\leq i} \text{DD} \) into \( \text{ROA}_{\leq j} \text{DD}_\mathcal{T} \)
in linear time in the number of variables and the size of output.

3. We devise a set of algorithm to perform querying and transformation operation in polytime, which
are shown in Table 1. These algorithms imply many other tractable algorithms, including deciding
satisfiability/validity of \( \text{OAODD} \), checking clausal entailment, checking implicant.

| Algorithms         | Description                                    |
|--------------------|-----------------------------------------------|
| \text{COUNT}       | counting the models of \( \text{OAODD} \) in linear time |
| \text{CONDITION}   | conditioning \( \text{OAODD} \) on a consistent term |
| \text{COMPUTECARD} | computing the minimum cardinality of \( \text{OAODD} \) |
| \text{MINIMIZE}    | minimizing \( \text{OAODD} \) |
| \text{CHECKENTAILTREE} | checking the entailment of two \( \text{OAODD}_\mathcal{T} \)s |
| \text{ENUMMODELS}  | enumerating the models of \( \text{OAODD} \) |
| \text{NEGATE}      | negating \( \text{OAODD} \) |
| \text{CONJOINTREE} | conjoining two \( \text{OAODD}_\mathcal{T} \)s |

4. Let \( i, j, i' \) and \( j' \) be four integers such that \( 0 \leq i \leq i' \leq \infty \) and \( 0 \leq j \leq j' \leq \infty \). We show that
\( \text{OA}_{\leq i, \leq j} \text{DD} \) is strictly more succinct than \( \text{OA}_{\leq i', \leq j'} \text{DD} \) if \( i+j \leq i'+j' \) and that \( \text{OA}_{\leq 0} \text{DD} \) and
\( \text{OA}_{\leq \infty} \text{DD}_\mathcal{T} \) are not more succinct than each other. These two facts implies that: ROA\_{\leq \infty} \text{DD}

is the most succinct fragment in \( \text{OAODD} \); OBDD-L_{\infty} and AOBDD is not incomparable with respect
to the succinctness relation; MLDD is strictly more succinct than OBDD, AOBDD and OBDD-L_{\infty}.

5. We propose a compilation algorithm which can convert any NNF formula into \( \text{OAODD} \).
This study is closely related to the previous works which exploit decomposability. Roughly speaking,
the algorithm \text{DECOMPOSE} generalizes the algorithm L2Inf in [7] and the algorithm \text{Linf} find in [5]; the
algorithm \text{CONVERTDOWN} generalizes the algorithm Inf2ROBDD in [7]; the algorithm \text{CONJOINTREE} generalizes
\text{Apply} in [6]. Moreover, some other operation algorithms are related to the algorithms in [7, 8, 11].

The remainder of this paper is organized as follows. In Section 2, we provide some technical and notational
preliminaries, and introduce the definitions of previous target languages. In Section 3, we introduce the
definition of \( \text{OAODD} \) and its fragments, and then analyse the relation between previous target languages
and fragments of \( \text{OAODD} \). In Section 4, we present the transformation algorithms from some fragments
of \( \text{OAODD} \) to others. Sections 5 and 6 evaluate the tractability and succinctness of \( \text{OAODD} \) and its
subsets, respectively. In Section 7, we presents the compilation algorithms for \( \text{OAODD} \) and then discuss
the optimization techniques, and we conclude in Section 8.

2. Preliminaries

In this paper, we use PS to denote a denumerable set of propositional (or Boolean) variables, $x, y, z$ to denote variables, and $X, Y, Z$ to denote subsets of PS. A formula is constructed from constants $true$, $false$ and variables in PS using negation operator $\neg$ and conjunction operator $\land$. Given a formula $\varphi$, we use $Vars(\varphi)$ to denote the set of variables appearing in $\varphi$. For every $X \subseteq PS$, PROPX denotes a language each of whose elements is a formula $\varphi$ such that $Vars(\varphi) \subseteq X$.

A world $\omega$ over variable set $X \subseteq PS$ is a truth assignment over the variables in $A$, i.e., a mapping from $X$ to $\{true, false\}$, and the set of all worlds over $X$ is denoted by $2^X$. Given any formula $\varphi$ and $\omega$ over $Vars(\varphi)$, $\omega$ satisfies/entails $\varphi$ (denoted by $\omega \models \varphi$) iff one of the following conditions holds: $\varphi = true$; $\varphi = x$ and $\omega(x) = true$; $\varphi = \neg \psi$ and $\omega \not\models \psi$; $\varphi = \psi \land \psi'$, and $\omega \models \psi$ and $\omega \models \psi'$. A model of $\varphi$ is a world over $Vars(\varphi)$ which satisfies $\varphi$. The set of models of $\varphi$ is denoted by $\Omega(\varphi)$. We call a formula satisfiable (or consistent) if it has at least one model, and we say it is unsatisfiable (or inconsistent) otherwise. We say a formula over $X$ is a tautology (or is valid) if every $\omega \in 2^X$ satisfies it. Given two formulas $\varphi$ and $\psi$ over $X$, $\varphi$ entails $\psi$ (denoted by $\varphi \models \psi'$) iff the models of the former is subsumed by those of the latter, $\varphi$ is (logically) equivalent to $\psi$ (denoted by $\varphi \equiv \psi$) iff $\varphi$ and $\psi$ imply each other.

2.1. Some other logical operations and several types of decompositions

It is well known that $\neg$ and $\land$ are complete for any propositional theory. Here we first introduce some other logical operations which can be defined using $\neg$ and $\land$. These operations will be used in the rest of the paper.

- Disjunction operator: $\varphi \lor \psi = \neg(\neg\varphi \land \neg\psi)$
- Equality operator: $\varphi \leftrightarrow \psi = (\varphi \land \psi) \lor (\neg\varphi \land \neg\psi)$
- Negative disjunction operator: $\varphi \downarrow \psi = \neg(\varphi \lor \psi)$
- Decision operator: $\varphi \downarrow x \psi = (\neg x \land \varphi) \lor (x \land \psi)$
- L-decision operator: $\varphi \downarrow x \in L \psi = (\land l \in L)(\neg x \land \varphi) \lor (x \land \psi)$

Note that the first three operators mentioned above, as well as $\land$, are easy to extend to multi-parameter cases. Next, we present two other operations mentioned in the KC map.

- Conditioning operator: Let $\varphi$ be a propositional formula, and let $\omega$ be a world over $X \subseteq Vars(\varphi)$. The conditioning of $\varphi$ on $\omega$ (denoted by $\varphi|\omega$) is a formula obtained by replacing every variable $x$ in $\varphi$ with $true$ (resp. $false$) if $x = true \in \omega$ (resp. $x = false \in \omega$).
- Forgetting operator: Let $\varphi$ be a propositional formula, and let $X$ be a subset of variables from PS. The forgetting of $X$ from $\varphi$, denoted by $\exists X.\varphi$, is a formula that does not mention any variable in $X$ and for every formula $\psi$ that does not mention any variable in $X$, we have $\varphi \models \psi$ precisely when $\exists X.\varphi \models \psi$.

Given a formula $\varphi$ and a variable in $x \in Vars(\varphi)$, we say $\varphi$ is dependant on $x$ if $\varphi|_{x=false} \not\models \varphi|_{x=true}$, and $DepVars(\varphi)$ denotes the set of all dependant variables of $\varphi$. Given any formula $\varphi$, we can get an equivalent formula which does not have independent variable by assigning each variable in $Vars(\varphi) \setminus DepVars(\varphi)$ either $true$ or $false$, and we denote the resulting formula by $[\varphi]$. Now we turn to introduce the definitions of several types of decompositions:

Definition 1 (decomposition). We say $\{\psi_1, \ldots, \psi_m\}$ is an $A$-decomposition of $\varphi$ if $\varphi \equiv \psi_1 \land \cdots \land \psi_m$ and $\{Vars(\psi_1), \ldots, Vars(\psi_m)\}$ is a partition of $Vars(\varphi)$. $O$-decompositions ($N$-decompositions) are defined in similar fashions. Given two decompositions $\Psi$ and $\Psi'$, if $\{Vars(\psi) : \psi \in \Psi\}$ is a refinement of $\{Vars(\psi) : \psi \in \Psi'\}$, we say the former is a finer decomposition than the latter. We say a decomposition is strict if it has more than one element.

Let $\varphi$ be a non-constant formula. It is obvious that $\{\varphi\}$ is a $A$-decomposition and a $O$-decomposition of $\varphi$. Given a decomposition $\Psi$ of $\varphi$, if $\varphi$ is not equivalent to a constant, we can use the following function to get a decomposition of $[\varphi]$:

$$[\Psi] = \{ [\psi] : \psi \in \Psi \text{ and } \psi \not\models true \};$$
At the end of this subsection, we give some useful observations without proofs:

**Observation 1.** Given any formula \( \varphi \), we have the following conclusions:
(a) For any variable \( x \in \text{DepVar}(\varphi) \), there exists some world \( \omega \) over \( \text{Vars}(\varphi) \setminus \{x\} \) such that \( \varphi|_{\omega} \equiv x \) or \( \varphi|_{\omega} \equiv \neg x \);
(b) Let \( \{\psi, \psi'\} \) be a semantical \( \land / \lor \)-decomposition of \( \varphi \). For any two variables \( x \in \text{Var}(\psi) \) and \( x' \in \text{Var}(\psi') \), there exist a world \( \omega \in 2^{\text{Vars}(\varphi) \setminus \{x, x'\}} \) such that \( \psi|_{\omega}, \psi'|_{\omega} \) is an \( \mathbb{A}/O \)-decomposition of \( \varphi|_{\omega} \);
(c) If \( \text{Vars}(\varphi) = \text{DepVars}(\varphi) \neq \emptyset \), \( \varphi \) does not simultaneously have strict \( \mathbb{A} \)-decomposition and \( \mathbb{O} \)-decomposition.

### 2.2. Some subsets of \( \text{PROP}_S \)

Some specific types of formulas used in this paper are enumerated as follows. A literal is either a variable \( x \) (positive literal) or its negation \( \neg x \) (negative literal). A clause \( C \) is a set of literals representing their disjunction. \( C \) is a Horn clause if it contains at most one positive literal. A max-clause over a set of variables \( X \) is a clause in which each variable in \( X \) appears exactly once. A term \( T \) is a set of literals representing their conjunction. A min-term over a set of variables \( X \) in which each variable in \( X \) appears exactly once. A formula in conjunctive normal form (CNF) is a set of clauses representing their conjunction. A CNF formula is a Horn theory if all its clauses are Horn clauses. A Krom CNF formula is a set of clauses whose lengths are not greater than two. A formula in disjunctive normal form (DNF) is a set of terms representing their disjunction. A formula in negation normal form (NNF) is constructed from \( \text{true}, \text{false} \) and literals using only the conjunction and disjunction operators. It is obvious that any clause, term, CNF formula and DNF formula are in NNF.

An implicate (resp. implicant) of a formula \( \varphi \) is a invalid clause (resp. a consistent term) \( \gamma \) satisfying \( \varphi \models \gamma \) (resp. \( \gamma \models \varphi \)). A prime implicate (resp. prime implicant) of \( \varphi \) is one of its logically strongest implicates (resp. one of its logically weakest implicants). A formula \( \varphi \) is in prime implicates normal form (or a Blake formula) \([12]\) iff it is a CNF formula whose clauses are the prime implicants of \( \varphi \). We use IP to denote the propositional fragment of all Blake formulas in \( \text{PROP}_S \). For any formula \( \varphi \), there is exactly one equivalent Blake formula, which is denoted by \( \text{PI}(\varphi) \) and has the following property:

**Observation 2.** Given two invalid formulas \( \varphi \) and \( \psi \) which does not share any variable, \( \text{PI}(\varphi \land \psi) = \text{PI}(\varphi) \cup \text{PI}(\psi) \).

According to the observation above, we have the two following conclusions:

**Proposition 1.** Let \( \Psi_1 \) and \( \Psi_2 \) be two \( \mathbb{A} \)-decompositions (resp. \( \mathbb{O} \)-decompositions) of any formula \( \varphi \), where \( \text{Vars}(\varphi) \neq \text{DepVars}(\varphi) \). Given a formula \( \psi \in \Psi_1 \) and \( \Psi_2 = \{\psi' \in \Psi_2 : \text{Vars}(\psi) \cap \text{Vars}(\psi') \neq \emptyset\} \), there exists another \( \mathbb{A} \)-decompositions (resp. \( \mathbb{O} \)-decompositions) \( \Psi_3 \) of \( \varphi \) such that \( \text{Vars}(\psi') : \psi' \in \Psi_3 \} = \{\text{Vars}(\psi) : \psi' \in \Psi_3 \} \setminus \{\psi\} \cup \{\text{Vars}(\psi) \cap \text{Vars}(\psi') : \psi' \in \Psi_2\} \).

**Proof.** According to Observation 2, it is known that \( \bigcup_{\psi' \in \Psi_2} \text{PI}(\psi') = \bigcup_{\psi' \in \Psi'} \text{PI}(\psi') \). Let \( \psi_1, \ldots, \psi_m \) be the formula in \( \Psi_2 \) and for \( 1 \leq i \leq m \), let \( \psi_i = \{\psi' \in \text{PI}(\psi) : \text{Vars}(\psi') \cap \text{Vars}(\psi'_{i-1}) \neq \emptyset\} \). Therefore, \( \Psi'' = \Psi \setminus \{\psi\} \cup \{\psi_1, \ldots, \psi_m\} \) is an \( \mathbb{A} \)-decomposition of \( \varphi \) which satisfies the condition in the proposition. \( \Box \)

**Proposition 2.** Given a formula \( \varphi \) which is not equivalent to a constant, the minimum disjoint partition of \( \text{PI}(\varphi) \) is the unique finest \( \mathbb{A} \)-decomposition of \( \varphi \) from the viewpoint of equivalence.

**Proof.** It is an immediate consequence of Proposition 1. \( \Box \)

Next, we introduce circular bit-shift function \([13]\), which will be used to prove succinctness relation. Several auxiliary functions are used here. The first one map a world to a term, i.e., \( w2t(\omega) = \land_{(x = \text{true}) \in \omega} x \land \land_{(x = \text{false}) \in \omega} \neg x \). When a variable set \( X \) is imposed on a linear order, each world over \( X \) can be seen as a

---

3Here "min" and "max" are defined with respect to the partial order of implication between two non-constant formulas.
The circular bit-shift function \( \varphi_{cbf} \) is defined over \( X = \{x_1, \ldots, x_k\} \), \( Y = \{y_1, \ldots, y_n\} \) and \( Z = \{z_1, \ldots, z_n\} \) as follows,

\[
\varphi_{cbf} = \bigvee_{\omega \in 2^X} \left( w2t(\omega) \land \psi_{cbf}^{w2t(\omega)} \right),
\]

where \( b2i \) is defined over the lexicographic order of \( X \), and for any \( 0 \leq i < n = 2^k \),

\[
\psi_{cbf}^i = (y_1 \leftrightarrow z_{i+1}) \land \cdots \land (y_{n-i} \leftrightarrow z_n) \land (y_{n-i+1} \leftrightarrow z_1) \land \cdots \land (y_n \leftrightarrow z_i).
\]

The circular bit-shift function can be split into a conjunction of two parts. The left part is defined as follows

\[
\varphi_{lcbf} = \bigvee_{\omega \in 2^X} \left( w2t(\omega) \land \psi_{lcbf}^{w2t(\omega)} \right),
\]

where for any \( 0 \leq i < n \),

\[
\psi_{lcbf}^i = (\neg y_1 \lor z_{i+1}) \land \cdots \land (\neg y_{n-i} \lor z_n) \land (\neg y_{n-i+1} \lor z_1) \land \cdots \land (\neg y_n \lor z_i).
\]

The right part is defined as follows

\[
\varphi_{rcbf} = \bigvee_{\omega \in 2^X} \left( w2t(\omega) \land \psi_{rcbf}^{w2t(\omega)} \right),
\]

where for any \( 0 \leq i < n \),

\[
\psi_{rcbf}^i = (y_1 \lor \neg z_{i+1}) \land \cdots \land (y_{n-i} \lor \neg z_n) \land (y_{n-i+1} \lor \neg z_1) \land \cdots \land (y_n \lor \neg z_i).
\]

Reader can verify \( \varphi_{cbf} \equiv \varphi_{lcbf} \land \varphi_{rcbf} \).

We close this subsection with a discussion of decomposability of \( \varphi_{cbf} \):

**Proposition 3.** Let \( X' \) be a subset of \( X \cup Y \cup Z \), where \( X = \{x_1, \ldots, x_k\} \), \( Y = \{y_1, \ldots, y_n\} \), \( Z = \{z_1, \ldots, z_n\} \) and \( |X'| \leq k + 2n - 2 \). Given any world \( \omega \) over \( X' \), \( \varphi_{cbf}|_{\omega} \) is not semantically O-decomposable.

**Proof.** Let \( x \) and \( x' \) be the two variables in \( X \cup Y \cup Z \). We prove the case \( |X'| = k + 2n - 2 \) by case analysis. Then according to Observation 1a, the proposition immediately holds.

1. \( x, x' \in X \): Let \( \omega \) be a world such that the variables in \( X \setminus \{x, x'\} \), \( Y \setminus \{y_1\} \) and \( Z \setminus \{z_1\} \) are assigned \( false \), and that \( y_1 \) and \( z_1 \) are assigned \( true \). Obviously, \( \psi_{cbf}^0|_{\omega} \equiv true \), \( \psi_{cbf}^1|_{\omega} \equiv false(1 \leq i < n) \) and \( i2t(0)|_{\omega} \equiv \neg x \land \neg x' \). Therefore, \( \varphi_{cbf}|_{\omega} \equiv \neg x \land \neg x' \). In other word, \( \varphi_{cbf}|_{\omega} \) is not semantically O-decomposable.

2. \( x \in X, x' \in Y \): Assume \( x' \equiv y_j \). Let \( \omega \) be a world such that the variables in \( X \setminus \{x\} \), \( Y \setminus \{y_1\} \) and \( Z \setminus \{z_1\} \) are assigned \( false \), and that \( z_j \) is assigned \( true \). Obviously, \( \psi_{cbf}^0|_{\omega} \equiv y_j \), \( \psi_{cbf}^i|_{\omega} \equiv false(1 \leq i < n) \) and \( i2t(0)|_{\omega} \equiv \neg x \). Therefore, \( \varphi_{cbf}|_{\omega} \equiv \neg x \land y_j \). In other word, \( \varphi_{cbf}|_{\omega} \) is not semantically O-decomposable.

3. \( x \in X, x' \in Z \): It is similar to the case (2).

4. \( x, x' \in Y \): Let \( \omega \) be a world such that the variables in \( X \setminus \{x, x'\} \) and \( Z \) are assigned \( false \). Obviously, \( \psi_{cbf}^0|_{\omega} \equiv \neg x \land \neg x' \), \( i2t(0)|_{\omega} \equiv false(1 \leq i < n) \) and \( i2t(0)|_{\omega} \equiv true \). Therefore, \( \varphi_{cbf}|_{\omega} \equiv \neg x \land \neg x' \). In other word, \( \varphi_{cbf}|_{\omega} \) is not semantically O-decomposable.

5. \( x \in X, x' \in Z \): Without loss of generality, we assume \( x \equiv y_i \) and \( x' \equiv z_j \) with \( i \geq j \). Let \( \omega \) be a world such that \( i2t(i-j) \) is its subset, and that the variables in \( Y \setminus \{x\} \) and \( Z \setminus \{x'\} \) are assigned \( true \). Obviously, \( \psi_{cbf}^i|_{\omega} \equiv x \lor x' \), \( i2t(k)|_{\omega} \equiv false(0 \leq k < n \land k \neq i-j) \) and \( i2t(i-j)|_{\omega} \equiv true \). Therefore, \( \varphi_{cbf}|_{\omega} \equiv x \lor x' \). In other word, \( \varphi_{cbf}|_{\omega} \) is not semantically O-decomposable.

6. \( x, x' \in Z \): It is similar to the case (4). \[\square\]
2.3. Graphical representation of propositional formulas

A practical representation of formula is a rooted Directed Acyclic Graph (DAG) where each leaf vertex is labeled with \( \top \) (standing for \textit{true}), \( \bot \) (standing for \textit{false}), a variable \( x \) or a set of literals \( L \), and each internal vertex \( v \) is labeled with an operator whose parameters are the children \( Ch(v) \) of \( v \). For any vertex \( v \), we use \( \text{sym}(v) \) to denote the symbol associated with it, that is, \( \bot, \top \), variable, literal set or operator, and \( \vartheta(v) \) to denote the formula represented by it. For the sake of convenience, given any vertex \( v \), it is also denoted by \( \langle \text{sym}(v) \rangle \) if it is a non-constant leaf, and it is also denoted by \( \langle \text{sym}(v), Ch(v) \rangle \) if it is an internal vertex; in particular, if \( v \) is a decision or \( L \)-decision vertex, we occasionally denote it by \( \langle \text{sym}(v), lo(v), hi(v) \rangle \) or \( \langle \text{sym}(v), lo(v), hi(v), L(v) \rangle \), where \( \text{sym}(v) \) is the variable associated with the operator, low child \( lo(v) \) and high child \( hi(v) \) are depicted as dashed and solid lines corresponding to the cases where \( \vartheta(v) \) is assigned \textit{false} and \textit{true}, respectively, and \( L(v) \) is the set of literals associated with the operator. Given a DAG \( G \), \( |G| \) denotes the size of \( G \), i.e., the number of edges in it, and the formula represented by \( G \) is defined as the one represented by its root. For the sake of simplicity, a DAG rooted at \( v \) is denoted by \( G_v \), \( \text{Vars}(\vartheta(v)) \) and \( \text{DepVars}(\vartheta(v)) \) are abbreviated as \( \text{Vars}(v) \) and \( \text{DepVars}(v) \) respectively, vertex labeled with \( \land \) (resp. \( \lor \) and \( \neg \)) are called \( \land \)-vertex (resp. \( \lor \)-vertex and \( \neg \)-vertex), and decision vertex is called \( \omega \)-vertex. Any \( \land \)-vertex (resp. \( \lor \)-vertex and \( \neg \)-vertex) \( v \) is a \textit{A}-decomposable (resp. \textit{O}-decomposable and \textit{N}-decomposable) if \( \{ \vartheta(w) : w \in Ch(v) \} \) is an \textit{A}-decomposition (resp. \textit{O}-decomposition and \textit{N}-decomposition) of \( \vartheta(v) \).

Next, we give the definitions of some special classes of DAGs, which are closely related with ordered \{AND, OR\}-decomposition and binary-decision diagrams.

**Definition 2 (BDD, OBDD and ROBDD).** A binary decision diagram (BDD) is a rooted DAG \( G = (V,E) \). The internal vertices in \( V \) are restricted to be decision vertices, and the leaf vertices are restricted to be \( \bot \) or \( \top \). The formula represented by \( G \) is defined as the one represented by its root. A BDD is ordered (OBDD) if it is imposed a linear order of variables \( \prec \) and for any vertex \( u \) and either internal child \( v, \text{sym}(u) \prec \text{sym}(v) \). An OBDD is reduced (ROBDD) if no two distinct vertices have the identical variable, low child and high child, and no vertex has two identical children.

In [13], it was shown that the following conclusion holds:

**Proposition 4.** Given any linear order of variables, the ROBDD representing \( \varphi_{cf} \) has an exponential number of nodes.

**Definition 3 (BDD-\( L \), OBDD-\( L \), OBDD-\( L_\infty \) and ROBDD-\( L_\infty \)).** A binary decision diagram with implied literals (BDD-\( L \)) is a rooted DAG \( G = (V,E) \). Each internal vertex in \( V \) is restricted to be \( L \)-decision vertex \( \langle \text{sym}(v), lo(v), hi(v), L(v) \rangle \), and each leaf vertices is restricted to be \( \bot \) or \( L(v) \), where \( L(v) \) is a set of implied literals representing a consistent term, and for an internal vertex, \( L(v) \) does not share any variable with \( \text{Vars}(lo(v)) \) and \( \text{Vars}(hi(v)) \). Given any non-False BDD-\( L \) node \( v \), its maximal set of implied literals \( L_{\max}(v) \) is defined as follows:

\[
L_{\max}(v) = \begin{cases} 
L(v) & v \text{ is a True node;} \\
L(v) \cup \{\text{sym}(v)\} \cup L_{\max}(hi(v)) & lo(v) = \bot; \\
L(v) \cup \{-\text{sym}(v)\} \cup L_{\max}(lo(v)) & hi(v) = \bot; \\
L(v) \cup (L_{\max}(hi(v)) \setminus L_{\max}(lo(v))) & \text{otherwise.}
\end{cases}
\]

A BDD-\( L \) is ordered (OBDD-\( L \)) if it is imposed a linear order of variables \( \prec \), for any vertex \( u \) and either internal child \( v, \text{var}(u) \prec \text{var}(v) \), and for any internal node \( v \), any variable appearing in \( L(v) \) is less than the ones appearing in \( L_{\max}(v) \setminus L(v) \). An OBDD-\( L \) has as many as possible implied literals (OBDD-\( L_\infty \)) if for any nonterminal vertex \( v, L(v) = L_{\max}(v) \). An OBDD-\( L_\infty \) is reduced (ROBDD-\( L_\infty \)) if no two distinct vertices have the identical variable, low child and high child, and no vertex has two identical children.

**Definition 4 (MLDD).** A Multi-Level Decomposition Diagram (MLDD) is a rooted DAG \( G = (V,E) \) which satisfies the following conditions: the internal vertices in \( V \) are restricted to be \( \downarrow \)-vertices, \( \neg \)-vertices or \( \omega \)-vertices, where any \( \downarrow \)-vertex is decomposable; the leaf vertices include \( \bot, \top \), variable vertices. \( G \) is
imposed a linear order of variables $\prec$; for a decision vertex $u$ and any descendant $v$ which is a decision vertex or variable vertex, $\text{var}(u) \prec \text{var}(v)$; no two distinct vertices are identical with each other, no decision vertex has two identical children, and the formula represented by each decision vertex is not strictly $\mathbb{N}$-decomposable.

Before introducing the definition of AOBDD, we first present the notion of tree-structured order:

**Definition 5 (tree-structured order).** Given a tree $T$ over a set of variables $X$, the tree-structured order relative to $T$ (denoted by $\prec_T$) is defined as the ancestor-descendant relationship on $T$. Given a subset $X'$ of $X$, we define the source variable of $X'$ over $T$ (denoted by $\text{source}_T(X')$) to be the minimum variable $x$ satisfying that each variable in $X \setminus \{x\}$ is a descendant of $x$. Given a variable $x \in X$, $T_x$ denotes the set of $x$ and its descendants.

For the sake of convenience, we introduce a virtual variable $x_{\text{root}}$ which is the root of any tree-structured order and does not appear in any formula. Given two strict partial orders $\prec_1$ and $\prec_2$ over a set $X$, we say the former is compatible with the latter if for all $x_1, x_2 \in X$, $x_1 \prec_2 x_2$ implies $x_1 \prec_1 x_2$.

**Definition 6 (AOBDD).** An AND/OR binary decision diagram (AOBDD) over a tree $T = \langle X, E \rangle$ is a rooted DAG $G = \langle V, E' \rangle$ which satisfies the following conditions: (1) the internal vertices in $V$ are restricted to be $\land$-vertices or $\lor$-vertices; (2) the leaf vertices only include $\bot$ and $T$; (3) for a decision vertex $v$, its children are $\land$-vertices or leaf vertices; (4) for a decision vertex $v$, $\text{var}(v)$ is the minimum variable in $\text{Vars}(v)$ up to $\prec_T$; (5) for a $\land$-vertex $v$, its children are decision vertices; (6) for a $\land$-vertex $v$, the variables in $\{\text{sym}(v) : v \in \text{Ch}(u)\}$ is incomparable up to $\prec_T$; (7) no two distinct vertices are identical with each other, and (8) no decision vertex has two identical children.

To simplify diagrams in the paper we draw multiple copies of the leaves $\bot$ and $T$, denoted by dashed boxes (and occasionally other internal vertices, denoted by dashed circles) but they represent the same node in the ROBDD, MLDD and AOBDD.

3. The definition of OAODD

In this section, we give a formal definition of OAODD and some other related definitions. First, we introduce two special types of decomposition. The first one is called bounded decomposition.

**Definition 7 (bounded decomposition).** An $\land$-decomposition $\varphi$ is bounded by a non-negative integer $0 \leq i \leq \infty$ (it is called an $A_{\leq i}$-decomposition) if there exists at most one factor of $\varphi$ which has more than $i$ variables. $\lor$-decomposition bounded by $i$ is defined in a similar fashion.

Note that any $A_{\leq 0}/O_{\leq 0}$-decomposition is a singleton set. Next, we show in a proposition that bounded decomposition has an interesting property, after an auxiliary function which will be used in the proof of that proposition is introduced as follows. Given a both invalid and consistent formula $\varphi$, the next function transforms a decomposition $\Psi$ of $[\varphi]$ to a decomposition of $\varphi$:

$$[\Psi]_i = \begin{cases} 
\{\psi \land \bigwedge_{x \in \text{Vars}(\varphi) \setminus \text{Vars}([\varphi])}(\neg x \lor x)\} & i = 0, \Psi = \{\psi\} \text{ is an } A\text{-decomposition}; \\
\{\psi \lor \bigvee_{x \in \text{Vars}(\varphi) \setminus \text{Vars}([\varphi])}(\neg x \land x)\} & i = 0, \Psi = \{\psi\} \text{ is an } O\text{-decomposition}; \\
\Psi \cup \bigcup_{x \in \text{Vars}(\varphi) \setminus \text{Vars}([\varphi])}(\neg x \lor x) & i > 0, \Psi \text{ is an } A\text{-decomposition}; \\
\Psi \cup \bigcup_{x \in \text{Vars}(\varphi) \setminus \text{Vars}([\varphi])}(\neg x \land x) & i > 0, \Psi \text{ is an } O\text{-decomposition}.
\end{cases}$$

**Proposition 5.** For any formula $\varphi$ which is not equivalent to a constant, and a non-negative integer $i$, $\varphi$ has exactly one finest $A_{\leq i}/O_{\leq i}$-decomposition from the viewpoint of equivalence.

\[\text{x_{\text{root}} will not appear in any formula.}\]
Definition 8 (tree-structured decomposition). Given a tree \( T \) over a variable set, an A/O-decomposition \( \Psi \) of \( \varphi \) respects \( T \) if for any two different formulas \( \psi \in \Psi \) and \( \psi' \in \Psi \), source\(_T\)(\( \psi \)) and source\(_T\)(\( \psi' \)) satisfies that they are incomparable over \( \preceq_T \).

Proposition 6. For any formula \( \varphi \) which is not equivalent to a constant, a non-negative integers \( i \), and a tree over \( \text{Vars}(\varphi) \), \( \varphi \) has exactly one finest \( A_{\preceq_1} \)-decomposition (\( O_{\preceq_1} \)-decomposition) respecting \( T \) from the viewpoint of equivalence.

Proof. We prove the case of \( A_{\preceq_1} \)-decomposition by induction on the size of \( \text{Vars}(\varphi) \), and the case of \( O_{\preceq_1} \)-decomposition is dual to it. The case \( |\text{Vars}(\varphi)| = 1 \) is obvious. That is, \( \{\varphi\} \) is the finest \( A_{\preceq_1} \)-decomposition of \( \varphi \). We assume that \( \varphi \) has exactly one finest \( A_{\preceq_1} \)-decomposition for \( |\text{Vars}(\varphi)| \leq n \). For the case \( |\text{Vars}(\varphi)| = n + 1 \), we show that for any two \( A_{\preceq_1} \)-decompositions \( \Psi_1 \) and \( \Psi_2 \) of \( \varphi \), there exist some \( A_{\preceq_1} \)-decomposition \( \Psi \) which is finer than or equals \( \Psi_1 \) and \( \Psi_2 \). Since the fineness relation is antisymmetrical and the number of decompositions is finite, we know the finest \( A_{\preceq_1} \)-decomposition is unique. We proceed by case analysis:

(1) \( \Psi_1 \) equals \( \Psi_2 \): It is obvious.

(2) Otherwise: Without loss of generality, we assume that \( \Psi_1 \) is not finer than \( \Psi_2 \). Given any formula \( \psi \in \Psi_1 \), either \( \{\psi' \in \Psi_2 : \text{source}_T(\psi') \prec \text{source}_T(\psi)\} \) or \( \{\psi' \in \Psi_2 : \text{source}_T(\psi') \prec \text{source}_T(\psi)\} \) is nonempty. Otherwise the variables in \( \text{Vars}(\psi) \) don’t appear in \( \text{Vars}(\Psi_2) \). Therefore, there exist some formula \( \psi_1 \in \Psi_1 \) such that \( \Psi'_2 = \{\psi' \in \Psi_2 : \text{source}_T(\psi') \prec \text{source}_T(\psi_1)\} \neq \emptyset \). Otherwise, \( \Psi_1 \) is finer than \( \Psi_2 \). Obviously, \( \text{Vars}(\psi_1) = \text{Vars}(\Psi'_2) \), and \( \psi_1 \subset \bigwedge \psi \in \Psi'_2 \). Let \( \Psi'_1 = \Psi_1 \setminus \{\psi'_1\} \). \( \Psi'_1 \cup \Psi'_2 \) is an \( A_{\preceq_1} \)-decomposition of \( \varphi \). According to the induction hypothesis, \( \Psi'_2 \setminus \Psi'_1 \) has a unique \( A_{\preceq_1} \)-decomposition, and let it be \( \Psi''_2 \). Obviously, \( \Psi'_1 \cup \Psi''_2 \) is finer than or equals \( \Psi_1 \) (\( \Psi_2 \)).

Now, we point out an observation which will be used in Section 5:

Observation 3. Given any formula \( \varphi \) and a tree \( T \), \( \varphi \) is not strictly decomposable if there exists some variable set \( X = \bigcup_{\psi \in T} Y \subset \text{Ch}_T(\text{source}_T(\varphi)) \) such that either condition holds:

(a) There exist two worlds \( \omega_1 \in 2^X \) and \( \omega_2 \in 2^X \) such that \( \varphi|_{\omega_1} \neq \varphi|_{\omega_2} \) and both \( \varphi|_{\omega_1} \) and \( \varphi|_{\omega_2} \) are equivalent to constants;
(b) There exist two worlds $\omega_1 \in 2^X$ and $\omega_2 \in 2^X$ such that $\varphi|_{\omega_1} \neq \varphi|_{\omega_2}$ and neither $\varphi|_{\omega_1}$ nor $\varphi|_{\omega_2}$ is equivalent to constant.

Next, we give the definition of \{AND, OR\}-decomposition and binary-decision diagram.

**Definition 9 (AOODD, OAODD and ROAODD).** An \{$\land$, $\lor$\}-decomposition and binary-Deci-

**Definition 10 (bounded OAODD).** We say an OAODD is $\land$-decomposable bounded by an integer

**Definition 11 (tree-structured OAODD).** Given a linear order $\prec$ and a tree $T$ over the same variable

**Proposition 7.** Given any two integers $0 \leq i, j \leq \omega$, and a linear order and a tree $T$ over a variable set, for any formula, there are exactly one ROAOODD$\leq_i$DD and at most one ROAOODD$\leq_{i\bar{\omega}}$DD representing it.

**Proof.** The completeness of ROAOODD$\leq_i$DD immediately follows the algorithm DECOMPOSE in Section 4. The uniqueness of ROAOODD$\leq_i$DD is proved by induction on the size of $Vars(\varphi)$. For the case $|Vars(\varphi)| = 0$, it is obvious that the unique ROAOODD$\leq_i$DD representing true (false) is $\top$ ($\bot$) (otherwise, there exists some D-vertex with two identical children, which violates the condition 6 in Definition 9), and that $Vars(\top) = Vars(\bot) = DepVars(\varphi) = \emptyset$. We assume that for the case $|Vars(\varphi)| \leq n$, there is a unique ROAOODD$\leq_i$DD with root $v$ such that $\varphi(v) \equiv \varphi$ and $Vars(v) = DepVars(\varphi)$. For the case $|Vars(\varphi)| = n+1$, we proceed by case analysis:

- **DepVars(\varphi) \subseteq Vars(\varphi):** According to the induction hypothesis, there is only one ROAOODD$\leq_i$DD which is equivalent to $[\varphi]$, and thus there is only one ROAOODD$\leq_i$DD which is equivalent to $\varphi$.
- **Otherwise:** We assume that there exist two ROAOODD$\leq_i$DDs rooted at $u$ and $v$ which are equivalent to $\varphi$, and then show that $u$ is identical to $v$ by case analysis:
  1. Both $u$ and $v$ are D-vertices: It is obvious that $sym(u) = sym(v)$, otherwise either $sym(u)$ or $sym(v)$ is independent. Therefore, $\varphi(lo(u)) \equiv \varphi(lo(v)) \equiv \varphi sym(u)=false$ and $\varphi(hi(u)) \equiv \varphi(hi(v)) \equiv \varphi sym(u)=true$. According to the induction hypothesis, lo$(u)$ and hi$(u)$ are identical to lo$(v)$ and hi$(v)$, respectively. Therefore, $u$ is identical to $v$.
  2. $u$ is a D-vertex and $v$ is an $\land$-vertex: This case violates the condition 1 in Definition 10, since $\{\varphi(w) : w \in Ch(u)\}$ is finer than $\{\varphi(w)\}$.
  3. $u$ is a D-vertex and $v$ is an $\lor$-vertex, or $u$ is an $\land$-vertex and $v$ is a D-vertex, or $u$ is an $\lor$-vertex and $v$ is a D-vertex: They are similar to (2).
(4) $u$ is an $\land$-vertex and $v$ is an $\lor$-vertex, or $u$ is an $\land$-vertex and $v$ is an $\lor$-vertex: These two cases violate Observation 1c.

(5) Both $u$ and $v$ are $\land$-vertices or $\lor$-vertices: According to Proposition 5, $\{\vartheta(w) : w \in Ch(u)\}$ is identical to $\{\vartheta(w) : w \in Ch(v)\}$ from the viewpoint of equivalence. Obviously, for each vertex $w \in Ch(u) \cup Ch(v)$, $\text{Vars}(w) \leq n$. Hence, according to the induction hypothesis, $Ch(u)$ identical to $Ch(v)$; that is, $u$ is identical to $v$.

We show by contradiction that for each $\land$-vertex $v$ in $\text{ROAODD}_T$, $\{\vartheta(v)\}$ is the finest $\land$-decomposition and $\lor$-decomposition up to $T$, then a similar fashion can be applied to prove the uniqueness of $\text{ROAODD}_T$. Assume that $\Psi$ is an $A/O$-decomposition of $\vartheta(v)$ with respect to $T$ such that $|\Psi| > 1$. Let $\psi$ and $\psi'$ be two different formulas in $\Psi$ such $\text{sym}(\psi) \in \text{Vars}(\psi)$. Obviously, every variable in $\text{Vars}(\psi')$ is not less than $\text{sym}(\psi)$ over $\land_T$, which contradicts the condition of Definition 11.

At the end of this section, we build the connections between some subsets in $\text{OAODD}$ and other target languages presented in the last section.

**Proposition 8.**

(1) Every $(R)\text{OBDD}$ is an $(R)\text{OA}_{\leq 0}\text{O}_{\leq 0}\text{DD}$.

(2) The mutual transformation between an OBDD-L (resp. OBDD-$L_{\infty}$ and ROBDD-$L_{\infty}$) and an equivalent $\text{OA}_{\leq 1}\text{O}_{\leq 0}\text{DD}$ (resp. $\text{OA}_{\leq 1}\text{O}_{\leq 0}\text{DD}$ and $\text{ROA}_{\leq 1}\text{O}_{\leq 0}\text{DD}$) can be done in linear time.

(3) The mutual transformation between an MLDD and the equivalent $\text{ROA}_{\leq \infty}\text{O}_{\leq \infty}\text{DD}$ can be done in linear time.

(4) Given a tree $T$ over a variable set, the mutual transformation between an $\text{AOBDD}_T$ and the equivalent $\text{ROA}_{\leq \infty}\text{O}_{\leq \infty}\text{DD}$ can be done in linear time.

**Proof.** First, we introduce some auxiliary function. The first one called $l2d$ transforms a literal $l$ into the root of the equivalent $A\text{ODD}$, that is, $l2d(l) = \langle x, \bot, \top \rangle$ if $l$ is a positive literal of variable $x$, and $l2d(l) = \langle x, \top, \bot \rangle$ otherwise. The second one called $L2D(L)$ transforms a set of literals into the set of the roots of the equivalent $A\text{ODD}$s, that is, $L2D(L) = \{l2d(l) : l \in L\}$. The last one called $d2l$ and $D2L$ are the inverse functions of $l2d$ of $L2D$. The proof is organized respectively corresponding to the items in the proposition:

(1) It is obvious, since no decomposition vertex appears in an $(R)\text{OA}_{\leq 0}\text{O}_{\leq 0}\text{DD}$ and any set with one single formula is an $\land/\lor$-decomposition.

(2) We define two functions $f$ and $f'$ to do the transformations as follows, where the former is from OBDD-L to $\text{OA}_{\leq 1}\text{O}_{\leq 0}\text{DD}$ and the latter is on the inverse direction.

$$
\begin{align*}
\text{f}(u) = & \begin{cases} 
\bot & u = \bot \\
\top & u = \langle \emptyset \rangle \\
\langle \land, L2D(L(u)) \rangle & u = \langle \{1\} \rangle \\
\langle \land, \langle \text{sym}(u), f(lo(u)), f(\text{hi}(u)) \rangle \rangle & u \text{ is an internal vertex and } L(u) = \emptyset \\
\langle \land, \langle \text{sym}(u), f(lo(u)), f(\text{hi}(u)) \rangle \rangle \cup L2D(L(u)) & u \text{ is an internal vertex and } L(u) \neq \emptyset \\
\end{cases} \\
\text{f}'(u) = & \begin{cases} 
\bot & u = \bot \\
\langle \emptyset \rangle & u = \top \\
\langle \{l2d(u)\} \rangle & |\text{Vars}(u)| = 1; \\
\langle D2U(Ch(u)) \rangle & \forall v \in Ch(u), |\text{Vars}(u)| = 1; \\
\langle \text{sym}(u), f'(lo(u)), f'(hi(u)), D2L(Ch(u) \setminus \{v\}) \rangle & v \in Ch(u) \text{ such that } |\text{Vars}(v)| > 1.
\end{cases}
\end{align*}
$$

Taking advantages of dynamic programming, the transformations can be done in linear time.
(3) Again, we define two functions $g$ and $g'$ to do the transformations as follows.

$$
g(u, c) = \begin{cases} 
\bot & u = \bot \text{ and } c = \text{true}, \text{ or } u = \top \text{ and } c = \text{false} \\
\top & u = \bot \text{ and } c = \text{false}, \text{ or } u = \top \text{ and } c = \text{true} \\
l2d(x) & u = \langle x \rangle \text{ and } c = \text{true} \\
l2d(\neg x) & u = \langle \neg \{v\} \rangle \text{ and } c = \text{false} \\
g(u, \neg c) & u = \langle \neg \{v\} \rangle \\
\langle \vee, \{f(v, \text{true}) : v \in Ch(u)\} \rangle & \text{sym}(u) = \bot \text{ and } c = \text{false} \\
\langle \wedge, \{f(v, \text{false}) : v \in Ch(u)\} \rangle & \text{sym}(u) = \bot \text{ and } c = \text{true} \\
\langle \text{sym}(u), g(lo(u), c), g(hi(u), c) \rangle & \text{otherwise}
\end{cases}
$$

$$
g'(u, c) = \begin{cases} 
\bot & u = \bot \text{ and } c = \text{true}, \text{ or } u = \top \text{ and } c = \text{false} \\
\top & u = \bot \text{ and } c = \text{false}, \text{ or } u = \top \text{ and } c = \text{true} \\
\langle \neg, \{g'(v, \text{false})\} \rangle & \text{sym}(u) = \top \text{ and } c = \text{true} \\
\langle \\{g'(v, \text{false}) : v \in Ch(u)\} \rangle & \text{sym}(u) = \top \text{ and } c = \text{false} \\
\langle \\{g'(v, \text{true}) : v \in Ch(u)\} \rangle & \text{sym}(u) = \wedge \text{ and } c = \text{true} \\
\langle \neg, \{g'(v, \text{true})\} \rangle & \text{sym}(u) = \wedge \text{ and } c = \text{false}
\end{cases}
$$

where $c$ is a constant true or false. Both the time complexity of $g$ and that of $g'$ are linear.

(4) We first show that each AOBDD satisfies all conditions of RO$A_{\leq \infty}O_{\leq 0}DD_{\tau}$ except the condition 7 in Definition 9. Therefore, we can transform an AOBDD$_{\tau}$ into an RO$A_{\leq \infty}O_{\leq 0}DD_{\tau}$ by removing the $\wedge$-vertices which only have one child. Let $G$ be an AOBDD$_{\tau}$.

(a) $G$ obviously satisfies the conditions about $A_{\leq \infty}$-decomposition in Definition 7 and $O_{\leq 0}$-decomposition, the conditions 1–6 in Definition 9 and the condition 2 in 11.

(b) For each $\wedge$-vertex $v$, we show by contradiction that it satisfies the condition 1 in Definition 11. If there exists a finer decomposition $\Psi$ than $\{\psi(w) : w \in Ch(v)\}$, then there exists some vertex $w \in Ch(v)$ which has a strict $\wedge$-decomposition over $\tau$. According to the condition 5 in Definition 6, $w$ is a $D$-vertex. Let $\psi$ and $\psi'$ be two different formulas in $\Psi$ such $\text{sym}(v) \in Vars(\psi)$. Obviously, every variable in $Vars(\psi')$ is not less than $\text{sym}(w)$ over $\prec_{\tau}$, which contradicts the condition 2 in Definition 11.

On the inverse direction, we show that each RO$A_{\leq \infty}O_{\leq 0}DD_{\tau}$ satisfies all conditions of AOBDD except the condition 3 in Definition 6. Therefore, we can transform an RO$A_{\leq \infty}O_{\leq 0}DD_{\tau}$ into an AOBDD$_{\tau}$ by adding an $\wedge$-vertex on any arc from a $D$-vertex to another $D$-vertex. Given an RO$A_{\leq \infty}O_{\leq 0}DD_{\tau}$ $G$,

(a) $G$ obviously satisfies the conditions 1, 2, 4, and 6–8 in Definition 6.

(b) $G$ satisfies the condition 5 in Definition 6: Obviously, $\vee$-vertex does not appear in $G$. For each $\wedge$-vertex $v$, since $\{\psi(w) : w \in Ch(v)\}$ is the finest $\wedge$-decomposition with respect to $\tau$, $Ch(v)$ does not include any $\wedge$-vertex or leaf. \hfill $\square$

Obviously, the time complexity of the methods here is also linear.

4. The transformation between subsets of OAOBDD

In this section, we show that given four integers $0 \leq i' \leq i \leq \infty$, $0 \leq j' \leq j \leq \infty$, a linear order $\prec$ and two trees $\tau$ and $\tau'$, where $\prec$ and $\tau'$ is compatible with $\prec_{\tau}$, each OA$A_{\leq i}O_{\leq j}DD$ can be transformed into the equivalent RO$A_{\leq i}O_{\leq j}DD$ in linear time in the size of the input and the number of its variables, each OA$A_{\leq i}O_{\leq j}DD_{\tau}$ can be transformed into the equivalent RO$A_{\leq i}O_{\leq j}DD_{\tau'}$ in linear time in the size of the input and the number of its variables, RO$A_{\leq i}O_{\leq j}DD_{\tau}$ can be transformed into the equivalent RO$A_{\leq i}O_{\leq j}DD_{\tau'}$ over $\tau$ (if existing) in linear time in the size of output and the number of its variables, and RO$A_{\leq i}O_{\leq j}DD$ can be transformed into the equivalent RO$A_{\leq i}O_{\leq j}DD_{\tau}$ over $\tau$ (if existing) in linear time in the size of output and the number of its variables. We first introduce four functions which will be used in the following:
• \textbf{EnsureIrredundant}(u): if \( u \) is a \( \land \)-vertex with two identical children, then we return either child, and else we return \( u \) itself;

• \textbf{EnsureUnduplicated}(u): if some vertex which is identical with \( u \) has been created, then we return the previous vertex, and otherwise we return \( u \) itself;

• \textbf{EnsureValid}(u): For the case where \( u \) is a \( \lor \)-vertex, we do

  - if \( \bot \in Ch(u) \), return \( \bot \);
  - if \( Ch(u) = \{ v \} \), return \( v \);
  - if \( Ch(u) = \{ \top, v, w \} \), return \( \sym(u), Ch(u) \setminus \{ \top \} \);

  For the case where \( u \) is a \( \lor \)-vertex, we do

  - if \( \top \in Ch(u) \), return \( \top \);
  - if \( Ch(u) = \{ v \} \), return \( v \);
  - if \( Ch(u) = \{ \bot, v, w \} \), return \( \sym(u), Ch(u) \setminus \{ \bot \} \);

• \textbf{EnsureFinest}(u): while there exists some child \( v \in Ch(u) \) with \( \sym(u) = \sym(v) \), we repeat removing \( v \) from \( Ch(u) \) and adding all children of \( v \) to \( Ch(u) \).

### 4.1. Transforming \( OA_{\leq i}O_{\leq j}DD \) into \( ROA_{\leq i}O_{\leq j}DD \)

Now we present the algorithm called \textsc{Decompose} (in Algorithm 1) which can transform an \( OA_{\leq i}O_{\leq j}DD \) into the equivalent \( ROA_{\leq i}O_{\leq j}DD \). For the sake of convenience, we introduce two notations which is used in Algorithm \textsc{Decompose}.

\textbf{Definition 12.} Let \( u \) and \( v \) be two internal vertices in an OAO\( DD \). We denote the set of “common” children of \( u \) and \( v \) by \( u \cap v \), formally,

\[ u \cap v = \begin{cases} \{ u \} & v \text{ is } \lor/\neg \text{-vertex and } u \in Ch(v); \\ \{ v \} & u \text{ is } \lor/\neg \text{-vertex, and } v \in Ch(u); \\ Ch(u) \cap Ch(v) & \sym(u) = \sym(v) = \lor/\neg; \\ \emptyset & \text{otherwise.} \end{cases} \]

Let \( V \) equal \( Ch(u) \setminus Ch(v) \). We use \( u \setminus v \) to denote the result of removing \( u \cap v \) from \( u \), that is,

\[ u \setminus v = \begin{cases} \top & \sym(v) = \land \text{ and } u \in Ch(v); \\ \top & \sym(u) = \sym(v) = \land \text{ and } V = \emptyset; \\ \bot & \sym(v) = \emptyset \text{ and } u \in Ch(v); \\ \bot & \sym(u) = \sym(v) = \emptyset \text{ and } V = \emptyset; \\ w & \sym(u) = \sym(v) = \land/\lor \text{ and } V = \{ w \}; \\ \langle \sym(u), V \rangle & \sym(u) = \sym(v) = \land/\lor \text{ and } 1 < |V| < |Ch(v)|; \\ u & \text{otherwise.} \end{cases} \]

Algorithm \textsc{Decompose} has the following properties:

\textbf{Proposition 9.} Given any \( OA_{\leq i}O_{\leq j}DD \) rooted at \( u \), the output of \textsc{Decompose}(\( u \)) is equivalent to \( \varphi(\emptyset) \) and its size is more than \( 2 \cdot |u| \). The time complexity of \textsc{Decompose} is bounded by \( O(|\text{Vars}(u)| \cdot |u|) \).

Given any variable order \( \prec \) and non-negative integers \( i \) and \( j \), the algorithm \textsc{Decompose} immediately gives us a compilation algorithm \( ROA_{\leq i}O_{\leq j}DD \), i.e., first generating the equivalent \( ROA_{\leq 0}O_{\leq 0}DD \) and then transforming it into the \( ROA_{\leq i}O_{\leq j}DD \), where \( ROA_{\leq 0}O_{\leq 0}DD \) can be generated by any ROBDD compilation algorithm according to Proposition 8. Therefore, \textsc{Decompose} verifies the existence of \( ROA_{\leq i}O_{\leq j}DD \) for any formula \( \varphi \).

Roughly speaking, the algorithm L2Inf in [7] and the algorithm mldd-find in [5] are two special cases of \textsc{Decompose}, where the former transforms \( OA_{\leq i}O_{\leq 0}DD \) into \( ROA_{\leq 0}O_{\leq 0}DD \), while the latter transforms \( ROA_{\leq 0}O_{\leq 0}DD \) into \( ROA_{\leq 0}O_{\leq 0}DD \). Unlike mldd-find, \textsc{Decompose} can transform any OAO\( DD \) into \( ROA_{\leq \infty}O_{\leq \infty}DD \), particularly including \( ROA_{\leq 0}O_{\leq 0}DD \) and \( ROA_{\leq \infty}O_{\leq 0}DD \). According to Proposition 8, \( ROA_{\leq 1}O_{\leq 0}DD \) and \( ROA_{\leq \infty}O_{\leq 0}DD \) can be generated by \( ROBDD_{-\infty} \) and AO\( BBDD \) compilers,
Algorithm 1: Decompose($u, i, j$)

**Input:** an $\mathcal{OA}_{\leq} \mathcal{O}_{\leq} \mathcal{DD}$ rooted at $u$

**Output:** the $\mathcal{ROA}_{\leq} \mathcal{O}_{\leq} \mathcal{DD}$ which is equivalent to $\vartheta(u)$

1. function Extract($u$)
2. function ExtractSub($u$)
3. case $\text{lo}(u) = \bot$:
   4. $u' \leftarrow \langle \land, \{\text{var}(u), \bot, \top\}, \text{hi}(u) \rangle$
5. case $\text{lo}(u) = \top$:
   6. $u' \leftarrow \langle \lor, \{\text{var}(u), \top, \bot\}, \text{hi}(u) \rangle$
7. case $\text{hi}(u) = \bot$:
   8. $u' \leftarrow \langle \land, \{\text{var}(u), \top, \bot\}, \text{lo}(u) \rangle$
9. case $\text{hi}(u) = \top$:
   10. $u' \leftarrow \langle \lor, \{\text{var}(u), \bot, \top\}, \text{lo}(u) \rangle$

11. return EnsureFinest($u'$)

12. if $u$ has a leaf child then $u' \leftarrow \text{ExtractSub}(v)$
13. else
14. $u' \leftarrow \langle \text{var}(u), \text{lo}(u) \setminus \text{hi}(u), \text{hi}(u) \setminus \text{lo}(u) \rangle$
15. if $\text{lo}(u) \cap \text{hi}(u) \neq \emptyset$ then
16. if $u'$ has a leaf child then $u' \leftarrow \text{ExtractSub}(u')$
17. else if $\text{lo}(u')$ is a leaf then $u' \leftarrow \langle \text{sym}(\text{hi}(u)), \{\text{lo}(u) \cap \text{hi}(u)\} \cup \{u'\} \rangle$
18. else $u' \leftarrow \langle \text{sym}(\text{lo}(u)), \{\text{lo}(u) \cap \text{hi}(u)\} \cup \{u'\} \rangle$
19. end
20. end
21. if the DAG rooted at $u'$ is an $\mathcal{OA}_{\leq} \mathcal{O}_{\leq} \mathcal{DD}$ then return $u'$
22. else return $u$

23. end

24. if $H(u) \neq \text{nil}$ then return $H(u)$
25. if $u$ is a leaf vertex then $H(u) \leftarrow u$
26. else
27. $Ch(u) \leftarrow \{\text{Decompose}(w) : w \in Ch(v)\}$
28. if $v$ is a decision vertex then
29. if $\text{lo}(v) = \text{hi}(v)$ then $H(u) \leftarrow \text{lo}(u)$
30. else if $\text{Vars}(u) = \{\text{sym}(u)\}$ then $H(u) \leftarrow u$
31. else $H(u) \leftarrow \text{Extract}(v)$
32. end
33. else $H(u) \leftarrow \text{EnsureFinest}(u)$
34. end
35. $H(u) \leftarrow \text{EnsureUnduplicated}(H(u))$
36. return $H(u)$
after simple transformations. It was shown in [6, 7] that there exist more efficient compilers of ROBDD-$L_\infty$ and AOBDD than those of ROBDD, and thus Decompose gives us a more efficient ROA$_{\leq \infty}O_{\leq \infty}$DD compiler than mldd.

4.2. Transforming $OAODD_T$ into $ROAODD_T$ ($T'$ is compatible with $T$)

Next, we present the algorithm called DecomposeTree (in Algorithm 2) which can transform an $OA_{\leq i}O_{\leq j}$DD$_T$, into the equivalent $ROA_{\leq i}O_{\leq j}$DD$_{T'}$, where $i$ and $j$ are two integers, $T$ and $T'$ are two tree over variables, and $T'$ is compatible with $T$. For the sake of convenience, we introduce some notations which is used in Algorithm DecomposeTree.

**Definition 13.** Let $T$ be a tree over a variable set, let $v$ be any decomposition vertex in $OAODD_T$, and let $x$ be a child of $source_T(v)$. A meta-child of $v$ corresponding to $x$ is defined as follows:

$$mch^x_T(v) = \begin{cases} T & W = \emptyset \text{ and } \text{sym}(v) = \land; \\ \bot & W = \emptyset \text{ and } \text{sym}(v) = \lor; \\ w & W = \{w\}; \\ \langle \text{sym}(v), W \rangle & \text{otherwise}. \end{cases}$$

where $W = \{w \in \text{Ch}(v) : source_T(w) \leq x\}$. $rCh_x(v)$ denotes the set of residual children except the ones in $mch_x(v)$, that is,

$$rCh^x_T(v) = \begin{cases} \text{Ch}(v) & mch^x_T(v) = T/\bot; \\ \text{Ch}(v) \setminus \{mch^x_T(v)\} & mch^x_T(v) \in \text{Ch}(v); \\ \text{Ch}(v) \setminus \text{Ch}(mch^x_T(v)) & \text{otherwise}. \end{cases}$$

We say a meta-vertex $v$ is a meta-vertex in an $OAODD_T$ $G$ if it satisfies either of the following conditions: (1) $v$ is a vertex in $G$; and (2) $v$ is a meta-child of some meta-vertex in $G$.

Obviously, given any $OAODD_T$ rooted at $v$, the number of meta-vertices in $G_v$ is not more than $|G_v| \cdot |Vars(v)|$.

**Definition 14.** Let $u$ and $v$ be two internal vertices in an $OAODD_T$. We denote the set of “common” children of $u$ and $v$ under variable $x$ in $T$ by $u \cap^T_T v$, formally,

$$u \cap^T_T v = \{w \in u \cap v : Vars(w) \cap V(T_x) = \emptyset\}$$

Let $V$ equal $\text{Ch}(u) \setminus (u \cap^T_T v)$. We use $ u \setminus^T_T v$ to denote the result of removing $u \cap^T_T v$ from $u$, that is,

$$u \setminus^T_T v = \begin{cases} T & \text{sym}(v) = \land \text{ and } u \in \text{Ch}(v); \\ T & \text{sym}(u) = \text{sym}(v) = \land \text{ and } V = \emptyset; \\ \bot & \text{sym}(v) = \bot \text{ and } u \in \text{Ch}(v); \\ \bot & \text{sym}(u) = \text{sym}(v) = \bot \text{ and } V = \emptyset; \\ w & \text{sym}(u) = \text{sym}(v) = /\bot \text{ and } V = \{w\}; \\ \langle \text{sym}(u), V \rangle & \text{sym}(u) = \text{sym}(v) = /\bot \text{ and } 1 < |V| < |\text{Ch}(v)|; \\ u & \text{otherwise}. \end{cases}$$

Algorithm DecomposeTree has the following properties:

**Proposition 10.** Given any $OA_{\leq i}O_{\leq j}$DD rooted at $u$, the output of DecomposeTree($u$) is equivalent to $\theta(u)$ and its size is more than $2|u|$. The time complexity of Decompose is bounded by $O(|Vars(u)| \cdot |u|)$. 

15
Algorithm 2: DecomposeTree\(u,i,j\)

Input: an \(\mathcal{D}_s\mathcal{O}_{<j}\mathcal{D}^T\) rooted at \(u\) and two global tree-structured order \(\mathcal{T}\) and \(\mathcal{T}'\), where \(\mathcal{T}'\) is compatible with \(\mathcal{T}\)

Output: the \(\mathcal{R}\mathcal{A}_s\mathcal{O}_{<j}\mathcal{D}^T\) which is equivalent to \(\vartheta(u)\) if existing, report failure otherwise

\[
\begin{align*}
\text{function } & \quad \text{ExtractTree}(u) \\
\text{function } & \quad \text{ExtractTreeSub}(u) \\
& \quad \text{if } \text{sym}(u) \text{ is the minimum variable in } \text{Vars}(u) \text{ up to } \prec_\mathcal{T} \text{ then return } u \\
& \quad \text{if } \text{sym}(u) \text{ is incomparable with any variable in } \text{Vars}(u) \setminus \{\text{sym}(u)\} \text{ then return } \text{Extract}(u) \\
& \quad \text{if either } \text{lo}(u) \text{ or } \text{hi}(u) \text{ is a decision vertex then report failure} \\
& \quad \text{if } \text{lo}(u) = \top \text{ and } \text{sym}(\text{hi}(u)) = \land \text{ then report failure} \\
& \quad \text{if } \text{lo}(u) = \bot \text{ and } \text{sym}(\text{hi}(u)) = \lor \text{ then report failure} \\
& \quad \text{if } \text{sym}(\text{lo}(u)) = \lor \text{ and } \text{hi}(u) = \bot \text{ then report failure} \\
& \quad \text{if } \text{sym}(\text{lo}(u)) = \land \text{ and } \text{hi}(u) = \bot \text{ then report failure} \\
& \quad \text{if } \text{lo}(u) \text{ is a leaf then} \\
& \quad \quad \text{return } \langle \text{sym}(\text{hi}(u)), \{\text{var}(u), \text{lo}(u), \text{mch}_{\mathcal{T}}^{\text{var}(u)}(\text{hi}(u))\} \cup r\text{Ch}_{\mathcal{T}}^{\text{var}(u)}(\text{hi}(u)) \rangle \\
& \quad \text{else return } \langle \text{sym}(\text{lo}(u)), \{\text{var}(u), \text{mch}_{\mathcal{T}}^{\text{var}(u)}(\text{lo}(u)), \text{hi}(u)\} \cup r\text{Ch}_{\mathcal{T}}^{\text{var}(u)}(u) \rangle \\
& \quad \text{end} \\
& \quad \text{if } \text{u has a leaf child then return } \text{ExtractTreeSub}(u) \\
& \quad \quad u' \leftarrow \langle \text{var}(u), \text{lo}(u) \setminus \text{sym}(u), \text{hi}(u) \setminus \text{sym}(u), \text{lo}(u) \rangle \\
& \quad \text{if } \text{sym}(u) \text{ is not the minimum variable in } \text{Vars}(u') \text{ up to } \prec_\mathcal{T} \text{ then report failure} \\
& \quad \text{if } \text{lo}(u) \cap \text{sym}(u) \text{ hi}(u) \neq \emptyset \text{ then} \\
& \quad \quad \text{if } u' \text{ has a leaf child then } u' \leftarrow \text{ExtractTreeSub}(u') \\
& \quad \quad \quad \text{if } \text{lo}(u') \text{ is a leaf then } u' \leftarrow \langle \text{sym}(\text{hi}(u')), \text{lo}(u') \cap \text{sym}(u), \text{hi}(u') \rangle \cup \{u'\} \\
& \quad \quad \text{else } u' \leftarrow \langle \text{sym}(\text{lo}(u')), \text{lo}(u') \cap \text{sym}(u), \text{hi}(u') \rangle \cup \{u'\} \\
& \quad \quad \text{end} \\
& \quad \text{end} \\
& \quad \text{return } u' \\
& \text{end} \\
& \text{if } H(u) \neq \text{nil} \text{ then return } H(u) \\
& \text{if } u \text{ is a leaf vertex then } H(u) \leftarrow u \\
& \text{else} \\
& \quad \text{Ch}(u) \leftarrow \{\text{Decompose}(u) : w \in \text{Ch}(v)\} \\
& \quad \text{if } v \text{ is a decision vertex then} \\
& \quad \quad \text{if } \text{lo}(u) = \text{hi}(u) \text{ then } H(u) \leftarrow \text{lo}(u) \\
& \quad \quad \text{else if } \text{Vars}(u) = \{\text{sym}(u)\} \text{ then } H(u) \leftarrow u \\
& \quad \quad \text{else } H(u) \leftarrow \text{Extract}(v) \\
& \quad \text{else} \\
& \quad \quad \text{H}(u) \leftarrow \text{EnsureFinest}(u) \\
& \quad \quad \text{if the DAG rooted at } H(u) \text{ is not an } \mathcal{O}\mathcal{A}_s\mathcal{O}_{<j}\mathcal{D}^T \text{ then report failure} \\
& \quad \text{end} \\
& \text{end} \\
& \text{return } H(u)
\end{align*}
\]
Algorithm 3: ConvertDown(u)

**Input:** an ROA≤iO≤jDD ⪉ rooted at u, and four global integers i’ ≤ i, j’ ≤ j

**Output:** the ROA≤iO≤jDD which is equivalent to u

1. if H(u) ≠ nil then return H(u)
2. if u is a leaf then H(u) ← u
3. else if u is a o-vertex then
   4. H(u) ← (sym(u), ConvertDown(lo(u)), ConvertDown(hi(u)))
   5. else if sym(u) = ∧ then V ← {v ∈ Ch(u) : |Vars(v)| > i’}
   6. else V ← {v ∈ Ch(u) : |Vars(v)| > j’}
   7. if |V| ≤ 1 then H(u) ← u
   8. else
      9. v1 ← ConditionMin((sym(u), V), ¬x)
      10. v2 ← ConditionMin((sym(u), V), x)
      11. u’ ← {x, ConvertDown(v1), ConvertDown(v2)}
      12. if Ch(u) \ V = ∅ then H(u) ← u’
      13. else H(u) ← (sym(u), V ∪ {u’})
   14. end
   15. end
16. end
17. return H(u)

4.3. Transforming ROA≤iO≤jDD into ROA≤i’O≤j’DD (i’ ≤ i, j’ ≤ j)

Next, we present the algorithm called ConvertDown (in Algorithm 3) which can transform an ROA≤iO≤jDD into the equivalent ROA≤i’O≤j’DD, where i’ ≤ i, j’ ≤ j.

Algorithm ConvertDown has the following properties:

**Proposition 11.** Let i, j, i’ and j’ be four global integers satisfying i’ ≤ i, j’ ≤ j. Given an ROA≤iO≤jDD rooted at u, the output of ConvertDown(u) is the equivalent ROA≤i’O≤j’DD and it terminates in time bounded by O(|Vars(u)| · |v|).

4.4. Transforming ROA≤iO≤jDD into ROA≤iO≤jDDT

Next, we present the algorithm called ConvertTree (in Algorithm 4) which can transform an ROA≤iO≤jDD into the equivalent ROA≤iO≤jDDT, where i’ ≤ i, j’ ≤ j. Note that for any vertex v in OAODD, source(v) can be computed in O(|V(T)|).

Algorithm ConvertTree has the following properties:

**Proposition 12.** Given any ROA≤iO≤jDD rooted at u, if existing, the output of ConvertTree is the ROA≤iO≤jDDT which is equivalent to u. Let v be the output of ConvertTree. The time complexity of ConvertTree is bounded by O(|Vars(u)| · |v|).

5. The operations of OAODD

In this section, we first discuss a class of tractable operations of OAODD, and then evaluate the inferential power according to the criterion corresponding to the KC map.

5.1. Some tractable logical operations

5.1.1. Model counting and satisfiability and validity testing

The first operation we discuss is model counting, i.e., answering the number of models of an OAODD. The algorithm to do this is called COUNT which is presented in Algorithm 5.

It was shown that the AOBDD compiler presented in [6] is comparable to c2d, which is a d-DNNF compiler more efficient than other state-of-the-art ROBDD compilers [7, 14, 15].
Algorithm 4: ConvertTree(u)

**Input:** an ROA$_{i,j}$O$_{i,j}$DD $G$ rooted at $u$, two global integers $i$ and $j$, a global linear order $\prec$ and a global tree-like order $T$, where $\prec$ is compatible with $T$

**Output:** the equivalent ROA$_{i,j}$O$_{i,j}$DD $T$ if existing, and report 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 $\ast$-vertex then
   4. if $\text{sym}(u) \prec \text{source}_T(u)$ then report failure
   5. $H(u) \leftarrow \langle \text{sym}(u), \text{ConvertTree(lo(u))), ConvertTree(hi(u))}\rangle$
   6. else
      7. Create a graph $\mathcal{G} = \langle \text{Ch}(u), \{(v, w) : \text{source}_T(v) \prec_T \text{source}_T(w)\}\rangle$
      8. Partition $\text{Ch}(u)$ into minimal disjoint sets $V_1, \ldots, V_m$ under $\mathcal{G}$
      9. for each $1 \leq i \leq m$
         10. if $|V_i| > 1$ then
            11. $v_1 \leftarrow \text{ConditionMin}(\langle \text{sym}(v), V_i, \neg \text{source}_T(V_i)\rangle)$
            12. $v_2 \leftarrow \text{ConditionMin}(\langle \text{sym}(v), V_i, \text{source}_T(V_i)\rangle)$
            13. if $\text{source}_T(v_1) = \text{source}_T(u)$ or $\text{source}_T(v_2) = \text{source}_T(u)$ then report failure
            14. $u_i \leftarrow \langle \text{sym}(v), \text{ConvertTree}(v_1), \text{ConvertTree}(v_2)\rangle$
            15. else $u_i \leftarrow \text{ConvertTree}(v_i)$
         16. end
      17. if $m = 1$ then return $H(u) \leftarrow u_1$
      18. else return $H(u) \leftarrow \langle \text{sym}(u), \{u_1, \ldots, u_m\}\rangle$
   19. end
20. return $H(u)$

Algorithm Count has the following properties:

**Proposition 13.** Given any OAODD rooted at $u$, the time complexity of Count is bounded by $O(|\mathcal{G}_u|)$, and the output equals the number of models of $\vartheta(u)$.

It is well known that any formula $\varphi$ is satisfiable iff $|\omega(\varphi)| > 0$, and is valid iff $|\omega(\varphi)| = 2^{|\text{Vars}(\varphi)|}$. Therefore, satisfiability and validity check of any OAODD can also be done in linear time.

5.1.2. Conditioning, clausal entailment check and implicant check

Now we present another algorithm called Condition in Algorithm 6 which performs the conditioning of an OAODD on a consistent term.

Algorithm Condition has the following properties:

**Proposition 14.** Given any OAODD rooted at $u$ and a consistent term $\gamma$, the time complexity of Condition is bounded by $O(|u|)$, and the output equals the number of $\vartheta(u)|\gamma$ and whose size is not greater than $|u|$. In particular, if the input is an OAOOD, the output is still an OAOOD.

Now we present another algorithm called ConditionMin in Algorithm 7 which is tailored for special conditioning. In detail, the term is a literal and its variable is the minimum variable in the OAOOD.

Algorithm ConditionMin has the following properties:

**Proposition 15.** Given any OAOOD rooted at $u$ and a literal $l$ whose variable is the minimum one in $\text{Vars}(u)$, the time complexity of ConditionMin is bounded by $O(|\text{Vars}(u)|)$, and the output is an OAOOD which is equivalent to $\vartheta(u)|l$ and whose size is less than $|u|$. In particular, if the input is an OAOOD (resp. OAOOD), the output is an OAOOD (resp. OAOOD).
Definition 15 (minimum cardinality). Let \( \varphi \) be a satisfiable propositional formula and let \( \text{Card}(\omega) \) be the number of atoms set to false in a truth assignment \( \omega \). The minimum cardinality of \( \varphi \) is defined as \( \min_{\omega \models \varphi} \text{Card}(\omega) \). The minimum cardinality of an unsatisfiable formula are defined to be \( \infty \).

The following algorithm called \text{COMPUTECARD} in Algorithm 8 is used to answer the minimum cardinality of an \text{AOODD}.

Algorithm \text{COMPUTECARD} has the following properties:

5.1.3. Computing minimum cardinalities and minimization

We now consider a property of propositional theories, which is called minimum cardinality [8].

Algorithm 5: \text{COUNT}(v)

\begin{verbatim}
Input: an OAODD \( G \) rooted at \( v \)
Output: the number of models of \( \vartheta(v) \) over \( \text{Vars}(G) \)
1 if \( H(v) \neq \text{nil} \) then return \( H(v) \)
2 else if \( v = \bot \) then \( H(v) \leftarrow 0 \)
3 else if \( v = \top \) then \( H(v) \leftarrow 1 \)
4 else if \( v \) is a \( \circ \)-node then
5 \hspace{1em} \( a \leftarrow 2^{\text{Vars}(v) \setminus \text{Vars}(lo(v))} \); \( b \leftarrow 2^{\text{Vars}(v) \setminus \text{Vars}(hi(v))} \)
6 \hspace{1em} \( H(v) \leftarrow a \times \text{COUNT}(lo(v)) + b \times \text{COUNT}(hi(v)) \)
7 else if \( v \) is a \( \land \)-node then \( H(v) \leftarrow \prod_{w \in \text{Ch}(v)} \text{COUNT}(w) \)
8 else \( H(v) \leftarrow 2^{\text{Vars}(v)} - \prod_{w \in \text{Ch}(v)} (2^{\text{Vars}(w)} - \text{COUNT}(w)) \)
\end{verbatim}

Algorithm 6: \text{CONDITION}(v, \gamma)

\begin{verbatim}
Input: an OAODD \( G \) rooted at \( v \), and a consistent term \( \gamma \)
Output: an OAODD which is equivalent to \( \vartheta(v)|\gamma \)
1 if \( H(v) \neq \text{nil} \) then return \( H(v) \)
2 else if \( v \) is a leaf vertex then return \( H(v) \leftarrow v \)
3 else if \( \neg\text{sym}(v) \in \gamma \) then \( H(v) \leftarrow \text{CONDITION}(lo(v), \gamma) \)
4 else if \( \text{sym}(v) \in \gamma \) then \( H(v) \leftarrow \text{CONDITION}(hi(v), \gamma) \)
5 else \( H(v) \leftarrow (\text{sym}(v), \{\text{CONDITION}(w, \gamma) : w \in \text{Ch}(v)\}) \)
6 return \( H(v) \)
\end{verbatim}

Algorithm 7: \text{CONDITIONMIN}(u, l)

\begin{verbatim}
Input: an OAODD \( G \) rooted at \( u \), and the variable of literal \( l \) is the minimum variable in \( \text{Vars}(v) \)
Output: an OAODD which is equivalent to \( \vartheta(u)|l \)
1 \( v \leftarrow u \)
2 while \( \text{sym}(v) = \land/O \) do \( v \leftarrow \text{min}(\text{Ch}(v)) \)
3 if \( \text{sym}(v) = \neg l \) then \( w \leftarrow \text{lo}(v) \)
4 else \( w \leftarrow \text{hi}(u) \)
5 if \( v \neq u \) then
6 \hspace{1em} Let \( v' \) be the parent of \( v \)
7 \hspace{1em} if \( \text{sym}(v') = \text{sym}(w) \) then \( \text{Ch}(v') \leftarrow \text{Ch}(v') \setminus \{v\} \cup \text{Ch}(w) \)
8 \hspace{1em} else \( \text{Ch}(v') \leftarrow \text{Ch}(v') \setminus \{v\} \cup \{w\} \)
9 \hspace{1em} return \( u \)
10 else return \( w \)
\end{verbatim}

19
Algorithm 8: ComputeCard\( (v) \)

**Input**: an OAODD \( G \) rooted at \( v \)

**Output**: the minimum cardinality of \( \vartheta(v) \)

1. If \( H(v) \neq \text{nil} \) then return \( H(v) \)
2. Else if \( v = \bot \) then \( H(v) \leftarrow (\infty, \infty) \)
3. Else if \( v = \top \) then \( H(v) \leftarrow (0, 0) \)
4. Else if \( v \) is a decision node then
   
   5. \( H(v) \leftarrow \min\{ \text{ComputeCard}(\text{lo}(v)) + 1, \text{ComputeCard}(\text{hi}(v)) \} \)
6. Else if \( \text{sym}(v) = \wedge \) then \( H(v) \leftarrow \sum_{w \in \text{Ch}(v)} \text{ComputeCard}(w) \)
7. Else if \( \text{sym}(v) = \lor \) then \( H(v) \leftarrow \min_{w \in \text{Ch}(v)} \text{ComputeCard}(w) \)

Proposition 16. Given any OAODD rooted at \( u \), the time complexity of ComputeCard is bounded by \( O(|G_u|) \), and the output is the minimum cardinality of \( \vartheta(u) \).

We now turn to another tractable transformation on DNNF, which has main applications in diagnosis, planning and nonmonotonic reasoning [8].

Definition 16 (minimization). Let \( \varphi \) be a satisfiable propositional formula. A minimization of \( \varphi \) is a sentence \( \psi \) such that for every truth assignment \( \omega \), we have \( \omega \models \psi \) if \( \omega \models \varphi \) and \( \text{Card}(\omega) \) equals the minimum cardinality of \( \varphi \).

The following algorithm called Minimize in Algorithm 8 is used to minimize an OAODD, which has the follow properties.

Proposition 17. Given any OAODD rooted at \( u \), the time complexity of Minimize is bounded by \( O(|G_u|) \), and the output is a minimization of \( \vartheta(u) \).

Algorithm 9: Minimize\( (v) \)

**Input**: an OAODD \( G \) rooted at \( v \)

**Output**: another OAODD which is a minimization of \( \vartheta(v) \)

1. If \( H(v) \neq \text{nil} \) then return \( H(v) \)
2. Create a new node \( u \) which is identical with \( v \)
3. If \( v \) is a decision node then
   
   4. If \( \text{MinCard}(\text{lo}(v)) + 1 \neq \text{MinCard}(v) \) then \( \text{lo}(u) \leftarrow \bot \)
   5. Else \( \text{lo}(u) \leftarrow \text{Minimize}(\text{lo}(v)) \)
   6. If \( \text{MinCard}(\text{hi}(v)) \neq \text{MinCard}(v) \) then \( \text{hi}(u) \leftarrow \bot \)
   7. Else \( \text{hi}(u) \leftarrow \text{Minimize}(\text{hi}(v)) \)
8. Else if \( \text{sym}(v) = \wedge \) then \( \text{Ch}(u) \leftarrow \{ \text{Minimize}(w) : w \in \text{Ch}(v) \} \)
9. Else
   
   10. \( \text{Ch}(u) \leftarrow \{ w \in \text{Ch}(v) : \text{MinCard}(w) = \text{MinCard}(v) \} \)
   11. \( \text{Ch}(u) \leftarrow \{ \text{Minimize}(w) : w \in \text{Ch}(u) \} \)
12. End
13. \( H(v) \leftarrow u \)
14. Return \( u \)

5.1.4. Enumeration models

We now turn to another tractable querying operation on OAODD that we shall discuss, that of enumerating its models. The algorithm called EnumModels is presented in Algorithm 10.

Algorithm EnumModels has the following properties:
Algorithm 10: EnumModels($u$)

Input: an OAODD $G$ rooted at $v$
Output: $\Omega(\vartheta(v))$
1 if $H(u) \neq \text{nil}$ then return $H(u)$
2 for each $w \in Ch(u)$ do EnumModels($w$)
3 if $u$ is a decision node then
4 $H(u) \leftarrow H(lo(u)) \times \{x = \text{false}\} \times 2^{Vars(hi(u))}\setminus Vars(lo(u))$
5 $H(u) \leftarrow H(u) \cup (H(hi(u)) \times \{x = \text{true}\} \times 2^{Vars(lo(u))}\setminus Vars(hi(u)))$
6 else if $\text{sym}(v) = \land$ then $H(u) \leftarrow \prod_{v \in Ch(u)} H(v)$
7 else
8 $H(u) \leftarrow \emptyset$
9 for each $v \in Ch(u)$ do
10 Let $V_1$ ($V_2$) be the set of children which are less (greater) than $v$
11 $H(u) \leftarrow H(u) \cup ((\prod_{v_1 \in V_1} 2^{Vars(v_1)} \setminus H(v_1)) \times H(v) \times (\prod_{v_2 \in V_2} 2^{Vars(v_2)}))$
12 end
13 end
14 return $H(u)$

Proposition 18. Given any OAODD rooted at $u$, the output of EnumModels equals to $\Omega(\vartheta(v))$, and the time complexity of EnumModels is bounded by $O(|\Omega(\vartheta(v))| \cdot |Vars(v)|)$.

5.1.5. Sentential entailment of ROAODD$\tau$

Now we turn to another tractable querying operation, that is, sentential entailment between two ROAODD$\tau$s.

The algorithm called CheckEntailTree presented in Algorithm 11 can check sentential entailment between two OAODD$\tau$s in polytime.

Proposition 19. Let $T$ be a tree over a variable set, and let $u$ and $v$ be two roots of OAODD$\tau$s. The output of CheckEntailTree equals true if $\vartheta(u) \models \vartheta(v)$, and equals false otherwise. The time complexity of CheckEntailTree is bounded by $O(|G_u| \cdot |G_v| \cdot |T|^2)$.

Since each OAODD rooted at $v$ can be converted into ROAODD$\tau$ in linear time, we know that the sentential entailment between two ROAODD$\tau$s can also be done in the same time complexity.

5.1.6. Negation of OAODD

We now turn to another tractable transformation operation on OAODD, that is, transforming it to its negation. The algorithm called Negate is presented in Algorithm 12.

Proposition 20. Given any OAODD rooted at $u$, the time complexity of Negate is bounded by $O(|u|)$, and the output is an OAODD which is equivalent to $\neg \vartheta(u)$ and whose size equals $|u|$. In particular, if the input is an $O_{\leq i}O_{\leq j}DD$ (resp. $O_{\leq i}O_{\leq j}DD$ and $O_{\leq i}O_{\leq j}DD$), the output is an $O_{\leq i}O_{\leq j}DD$ (resp. $O_{\leq i}O_{\leq j}DD$ and $O_{\leq i}O_{\leq j}DD$).

5.1.7. Binary conjunction and disjunction of ROAODD$\tau$

Now we turn to the conjunction and disjunction of ROAODD$\tau$. First we introduce two observations which will be used in the following algorithm:

Observation 4. Given a tree $T$ and two ROAODD$\tau$s rooted at $u$ and $v$, we have the following conclusions:
If $\text{sym}(u) = \text{sym}(v) = \lor$ and source$_T(u) = \text{source}_T(v)$, $\vartheta(u) \land \vartheta(v)$ is strictly decomposable with respect to $T$ iff one of the following conditions hold:
(a) $\vartheta(u) \models \vartheta(v)$;
Algorithm 11: CheckEntailTree$(u, v)$

**Input**: two ROAODD$_T$s rooted at $u$ and $v$ over tree $T$

**Output**: answer whether $\vartheta(u) \models \vartheta(v)$

1. if $H(v) \neq \text{nil}$ then return $H(v)$
2. else if $u = \bot$ or $v = \top$ then $H(u, v) \leftarrow \text{true}$
3. else if $u = \top$ or $v = \bot$ then $H(u, v) \leftarrow \text{false}$
4. else if $\text{source}_T(u)$ and $\text{source}_T(v)$ are incomparable then $H(u, v) \leftarrow \text{false}$
5. else if $\text{sym}(u) = \lor$ then $H(u, v) \leftarrow \bigwedge_{w \in Ch(u)} \text{CheckEntailTree}(w, v)$
6. else if $\text{sym}(v) = \land$ then $H(u, v) \leftarrow \bigwedge_{w \in Ch(v)} \text{CheckEntailTree}(u, w)$
7. else if $\text{sym}(u) = \land$ and $\text{sym}(v) = \lor$ then
8. if $\text{source}_T(u) \prec \text{source}_T(v)$ then
9. Let $x$ be the variable in $Ch(\text{source}_T(v))$ such that $\text{source}_T(u) \leq x$
10. $H(u, v) \leftarrow \text{CheckEntailTree}(u, \text{mch}_x(v))$
11. else if $\text{source}_T(v) \prec \text{source}_T(u)$ then
12. Let $x$ be the variable in $Ch(\text{source}_T(u))$ such that $\text{source}_T(v) \leq x$
13. $H(u, v) \leftarrow \text{CheckEntailTree}(\text{mch}_x(u), v)$
14. else $H(u, v) \leftarrow \bigvee_{x \in Ch(\text{source}_T(v))} \text{CheckEntailTree}(\text{mch}_x(u), \text{mch}_x(v))$
15. else if $\text{sym}(u) = \land$ then
16. if $\text{sym}(v) \preceq_T \text{source}_T(u)$ then
17. $H(u, v) \leftarrow \text{CheckEntailTree}(u, \text{lo}(v)) \land \text{CheckEntailTree}(u, \text{hi}(v))$
18. else
19. Let $x$ be the variable in $Ch(\text{source}_T(u))$ such that $\text{sym}(v) \leq x$
20. $H(u, v) \leftarrow \text{CheckEntailTree}(\text{mch}_x(u), v)$
21. end
22. else if $\text{sym}(v) = \lor$ then
23. if $\text{sym}(u) \preceq_T \text{source}_T(v)$ then
24. $H(u, v) \leftarrow \text{CheckEntailTree}(\text{lo}(u), v) \land \text{CheckEntailTree}(\text{hi}(u), v)$
25. else
26. Let $x$ be the variable in $Ch(\text{source}_T(v))$ such that $\text{sym}(u) \leq x$
27. $H(u, v) \leftarrow \text{CheckEntailTree}(u, \text{mch}_x(v))$
28. end
29. else if $\text{sym}(u) \prec \text{sym}(v)$ then
30. $H(u, v) \leftarrow \text{CheckEntailTree}(\text{lo}(u), v) \land \text{CheckEntailTree}(\text{hi}(u), v)$
31. else if $\text{sym}(u) \prec \text{sym}(v)$ then
32. $H(u, v) \leftarrow \text{CheckEntailTree}(\text{lo}(u), \text{lo}(v)) \land \text{CheckEntailTree}(\text{hi}(u), \text{hi}(v))$
33. return $H(u, v)$
Algorithm 12: NEGATE(u)

Input: an OAODD G rooted at u
Output: the root of an OAODD which is equivalent to \( \neg \vartheta(u) \)

1. if \( H(u) \neq nil \) then return \( H(u) \)
2. if \( u = \bot \) then \( H(u) \leftarrow \top \)
3. else if \( u = \top \) then \( H(u) \leftarrow \bot \)
4. else
5. Create a new node \( v \) which is identical with \( v \)
6. if \( \text{sym}(u) = \bot/\top \text{ then sym}(v) \leftarrow \top/\bot \)
7. else \( \text{sym}(v) \leftarrow \text{sym}(u) \)
8. \( Ch(v) \leftarrow \{ \text{NEGATE}(w) : w \in Ch(u) \} \)
9. end
10. return \( H(u) \)

(b) \( \vartheta(v) \models \vartheta(u) \);
(c) \( \vartheta(u) \models \neg \vartheta(v) \); or
(d) there exists some variable \( x \in \text{Ch}_T(\text{source}_T(u)) \) such that \( r\text{Ch}_x(u) = r\text{Ch}_x(v) \).

Proof. \((\Leftarrow)\) If \( \vartheta(u) \models \vartheta(v) \), then \( \vartheta(u) \land \vartheta(v) \equiv \vartheta(u) \). The following formula set

\[
\{ \vartheta(\text{mch}_x(u)) \land \bigwedge_{x \in Vars(\text{mch}_x(v)) \setminus Vars(\text{mch}_x(u))} x \neg \vartheta : x \in \text{Ch}_T(\text{source}_T(u)), \text{ and mch}_x(u) \neq \top \text{ or mch}_x(v) \neq \top \} \]

is obviously a \( \lor \)-decomposition of \( \vartheta(u) \models \neg \vartheta(v) \). The case \( \vartheta(v) \models \vartheta(u) \) is similar to the one \( \vartheta(u) \models \vartheta(v) \).

If \( \vartheta(u) \models \neg \vartheta(v) \), then \( \vartheta(u) \land \vartheta(v) \equiv \text{false} \). The following formula set

\[
\{ \bigwedge_{x \in Vars(\text{mch}_x(v)) \cup Vars(\text{mch}_x(u))} x \neg \vartheta : x \in \text{Ch}_T(\text{source}_T(u)), \text{ and mch}_x(u) \neq \top \text{ or mch}_x(v) \neq \top \} \]

is obviously a \( \lor \)-decomposition of \( \vartheta(u) \land \vartheta(v) \). If there exists some variable \( x \in \text{Ch}_T(\text{source}_T(u)) \) such that \( r\text{Ch}_x(u) = r\text{Ch}_x(v) \), it is obvious that \( \text{rCh}_x(u) \cup \{ \text{mch}_x(u) \land \text{mch}_x(v) \} \) is a \( \lor \)-decomposition of \( \vartheta(u) \land \vartheta(v) \).

\((\Rightarrow)\) We show by case analysis that \( \vartheta(u) \land \vartheta(v) \) is not decomposable with respect to \( T \) if none of the conditions satisfies.

- There exists some variable \( x \in \text{Ch}_T(\text{source}_T(u)) \) such that \( \text{mch}_T(u) \neq \top \) and \( \text{mch}_T(v) = \top \). Let \( \varphi = \bigvee_{w \in r\text{Ch}_x(u)} \vartheta(w) \). There exist three worlds \( \omega_1 \models \neg \vartheta(v) \), \( \omega_2 \models \vartheta \land \vartheta(v) \) (otherwise the condition \( c \) is satisfied) and \( \omega_3 \models \neg \vartheta \land \vartheta(v) \) (otherwise the condition \( b \) is satisfied). Therefore, \( \vartheta(u) \land \vartheta(v) \models \text{false} \), \( \vartheta(u) \land \vartheta(v) \models \text{true} \), and \( \vartheta(u) \land \vartheta(v) \models \varphi \). According to Observation 3, \( \vartheta(u) \land \vartheta(v) \) is not strictly decomposable with respect to \( T \).

- There exists some variable \( x \in \text{Ch}_T(\text{source}_T(v)) \) such that \( \text{mch}_T(u) = \top \) and \( \text{mch}_T(v) \neq \top \). It is similar to the last case.

- Otherwise, there exists some variable \( x \in \text{Ch}_T(\text{source}_T(u)) \) (let \( \varphi = \bigvee_{w \in r\text{Ch}_x(u)} \vartheta(w) \) and \( \varphi' = \bigvee_{w \in r\text{Ch}_x(v)} \vartheta(w) \)) such that: \( \text{mch}_x(u) \neq \text{mch}_x(v) \), and \( \varphi \neq \varphi' \) (otherwise the last condition is satisfied). We perform case analysis further:
  - There exist two worlds \( \omega_1 \models \neg \vartheta \land \neg \varphi' \), \( \omega_2 \models \vartheta \land \neg \varphi' \). Obviously, \( \vartheta(u) \land \vartheta(v) \models \text{false} \) and \( \vartheta(u) \land \vartheta(v) \models \text{true} \). Since \( \text{mch}_x(u) \neq \text{mch}_x(v) \), and \( \vartheta(u) \land \vartheta(v) \) are not constant, \( \vartheta(u) \land \vartheta(v) \) is not strictly decomposable with respect to \( T \) by Observation 3.
  - \( \varphi \models \varphi' \). Obviously, there exist two worlds \( \omega_1 \models \neg \vartheta \land \neg \varphi' \), \( \omega_2 \models \vartheta \land \neg \varphi' \). Obviously, \( \vartheta(u) \land \vartheta(v) \models \text{false} \) and \( \vartheta(u) \land \vartheta(v) \models \text{true} \). Since \( \text{mch}_x(u) \neq \text{mch}_x(v) \), \( \vartheta(u) \land \vartheta(v) \) is not strictly decomposable with respect to \( T \).
Observation 5. If \( \text{sym}(u) = \wedge \), \( \text{sym}(v) = \lor \) and \( \text{source}(u) \leq \text{source}(v) \), \( \vartheta(u) \land \vartheta(v) \) is strictly decomposable with respect to \( T \) iff either of the following conditions holds:

(a) \( \vartheta(u) \models \vartheta(v) \); or
(b) the number of the elements in \( \{ x \in \text{Ch}_T(\text{source}(u)) : \vartheta(\text{mch}_x(u)) \not\models \vartheta(\text{mch}_x(v)) \} \) is not more than one.

Proof. \( (\Rightarrow) \) If \( \vartheta(u) \models \vartheta(v) \), then \( \vartheta(u) \land \vartheta(v) \models \vartheta(u) \). The following formula set

\[
\{ \vartheta(\text{mch}_x(u)) \land \bigwedge_{x \in \text{Vars}(\text{mch}_x(v))) \setminus \text{Vars}(\text{mch}_x(u))} x \land \neg x : x \in \text{Ch}_T(\text{source}(u)), \text{ and } \text{mch}_x(u) \not\models \top \} \cup \{ \vartheta(\text{mch}_y(v)) \}
\]

is obviously a \( \lor \)-decomposition of \( \vartheta(u) \) for \( \vartheta(v) \). For the case where the number of the elements in \( \{ w \in \text{Ch}(v) : \vartheta(u) \not\models \neg \vartheta(w) \} \) is not more than one, we proceed by case analysis:

- \( \{ x \in \text{Ch}_T(\text{source}(u)) : \vartheta(\text{mch}_x(u)) \not\models \vartheta(\text{mch}_x(v)) \} = \emptyset \). It is obvious that \( \vartheta(u) \land \vartheta(v) \equiv \text{false} \). The following formula set

\[
\{ \vartheta(\text{mch}_x(u)) \land \bigwedge_{x \in \text{Vars}(\text{mch}_x(v))) \setminus \text{Vars}(\text{mch}_x(u))} x \land \neg x : x \in \text{Ch}_T(\text{source}(u)), \text{ and } \text{mch}_x(u) \not\models \top \} \cup \{ \vartheta(\text{mch}_y(v)) \}
\]

is obviously a \( \lor \)-decomposition of \( \vartheta(u) \land \vartheta(v) \).

- \( \{ x \in \text{Ch}_T(\text{source}(u)) : \vartheta(\text{mch}_x(u)) \not\models \vartheta(\text{mch}_x(v)) \} \neq \emptyset \). It is obvious that \( \vartheta(u) \land \vartheta(v) \equiv \text{false} \). The following formula set

\[
\{ \vartheta(\text{mch}_x(u)) \land \bigwedge_{x \in \text{Vars}(\text{mch}_x(v))) \setminus \text{Vars}(\text{mch}_x(u))} x \land \neg x : x \in \text{Ch}_T(\text{source}(u)), \text{ and } \text{mch}_x(u) \not\models \top \} \cup \{ \vartheta(\text{mch}_y(v)) \}
\]

is obviously a \( \lor \)-decomposition of \( \vartheta(u) \land \vartheta(v) \).

(\( \Rightarrow \)) We show by case analysis that \( \vartheta(u) \land \vartheta(v) \) is not decomposable with respect to \( T \) if neither conditions satisfies:

- There exists some variable \( x \in \text{Ch}_T(\text{source}(u)) \) such that \( \text{mch}_x(u) = \top \) and \( \text{mch}_x(v) \not\models \top \). Let \( \varphi = \bigvee_{w \in \text{Ch}_x(u)} \vartheta(w) \). There exist two worlds \( \omega_1 \models \neg \vartheta(u), \omega_2 \models \vartheta(u) \land \varphi \) (otherwise the condition \( a \) is satisfied), \( \omega_3 \models \vartheta(u) \land \neg \varphi \) (otherwise the condition \( b \) is satisfied). Therefore, \( \vartheta(u) \land \vartheta(v) \models \text{false} \), \( \vartheta(u) \land \vartheta(v) \models \text{false} \), \( \vartheta(u) \land \vartheta(v) \models \text{false} \), \( \vartheta(u) \land \vartheta(v) \models \text{false} \), and \( \vartheta(u) \land \vartheta(v) \models \text{false} \). According to Observation 3, \( \vartheta(u) \land \vartheta(v) \) is not strictly decomposable with respect to \( T \).

- Otherwise, there exist some variable \( x \in \text{Ch}_T(\text{source}(u)) \) (let \( \varphi = \bigvee_{w \in \text{Ch}_x(u)} \vartheta(w) \) and \( \varphi' = \bigvee_{w \in \text{Ch}_x(u)} \vartheta(w) \) such that \( \vartheta(\text{mch}_x(u)) \land \vartheta(\text{mch}_x(v)) \not\models \text{false} \) and \( \vartheta \land \varphi' \not\models \text{false} \) (otherwise the last condition is satisfied). Obviously, there exist some world \( \omega_1 \models \neg \vartheta \land \neg \varphi' \). Therefore, \( \vartheta(u) \land \vartheta(v) \models \vartheta(\text{mch}_x(u)) \). Since \( \vartheta(u) \not\models \vartheta(v) \), \( \varphi \not\models \varphi' \). Therefore, there exists some world \( \omega_2 \models \vartheta \land \neg \varphi' \). Obviously, \( \vartheta(u) \land \vartheta(v) \models \vartheta(\text{mch}_x(u)) \) and \( \vartheta(u) \land \vartheta(v) \models \vartheta(\text{mch}_x(u)) \). Since \( \vartheta(\text{mch}_x(u)) \not\models \vartheta(\text{mch}_x(u)) \land \vartheta(\text{mch}_x(v)) \) (otherwise \( \vartheta(\text{mch}_x(u)) \land \vartheta(\text{mch}_x(v)) \not\models \text{false} \), \( \vartheta(u) \land \vartheta(v) \) is not strictly decomposable with respect to \( T \) by Observation 3.
Observation 6. Let \( u \) and \( v \) be two internal vertices, respectively, where \( \text{sym}(v) = \lor \) and \( \text{source}\_T(u) \prec \text{source}\_T(v) \), let \( X = \{ x \in \text{Ch}_T(\text{source}\_T(u)) : \text{mch}_x(u) \neq \top \} \), and let \( x_0 \) be the variable in \( X \) such that \( \text{source}\_T(u) \preceq x_0 \). \( \vartheta(u) \land \vartheta(v) \) is strictly decomposable with respect to \( T \) iff
\[
\begin{align*}
& (a) \quad \vartheta(u) \models \vartheta(v) ; \quad \text{or} \\
& (b) \quad |X| = 2 \quad \text{and} \quad \vartheta(u) \models \neg \vartheta(\text{mch}_x_0(v)).
\end{align*}
\]

PROOF. \((\Leftarrow)\) If \( \vartheta(u) \models \vartheta(v) \), then \( \vartheta(u) \land \vartheta(v) \equiv \vartheta(u) \). The following formula set
\[
\left\{ \left( \bigvee_{x \in \text{Vars}(\text{mch}_x(v))} x' \lor \neg x' : x \in X \text{ and } x \neq x_0 \right) \cup \left( \vartheta(u) \land \bigwedge_{x \in \text{Vars}(\text{mch}_x(v))} x \lor \neg x \right) \right\}
\]
is obviously a \( \lor \)-decomposition of \( \vartheta(u) \models \neg \vartheta(v) \). Otherwise, let \( X = \{ x, y \} \). The following formula set
\[
\left\{ \vartheta(u) \land \bigwedge_{x' \in \text{Vars}(\text{mch}_x(v))} \left( x' \lor \neg x' \right), \vartheta(\text{mch}_x(v)) \right\}
\]
is obviously a \( \land \)-decomposition of \( \vartheta(u) \models \neg \vartheta(v) \) up to \( T \).

\((\Rightarrow)\) We show that \( \vartheta(u) \land \vartheta(v) \) is not decomposable with respect to \( T \) if neither condition satisfies. Let \( x_1 \) be a variable in \( X \setminus \{ x_0 \} \), and let \( \varphi = \bigvee_{x \in \text{Ch}_x(v)} \vartheta(u) \). There exist three worlds \( \omega_1 \models \neg \vartheta(u), \omega_2 \models \vartheta(u) \land \neg \varphi \) (otherwise the condition \( a \) is satisfied), \( \omega_3 \models \vartheta(u) \land \varphi \) (otherwise the condition \( b \) is satisfied). Therefore, \( \vartheta(u) \land \vartheta(v) \upharpoonright_{\omega_2} \equiv \text{false}, \vartheta(u) \land \vartheta(v) \upharpoonright_{\omega_3} \equiv \vartheta(\text{mch}_x(u)), \text{ and } \vartheta(u) \land \vartheta(v) \upharpoonright_{\omega_3} \equiv \text{true} \). According to Observation 3, \( \vartheta(u) \land \vartheta(v) \) is not strictly decomposable with respect to \( T \).

In order to facilitate the description of algorithm, we introduce a new notion:

Definition 17. Let \( T \) be a tree, and let internal vertices \( u \) and \( v \) be the roots of two ROAODD\(_T\), where \( \text{sym}(\mathbf{v}) = \lor \), \( \text{source}\_T(u) \lhd \text{source}\_T(v) \), and \( X = \{ y \in \text{Ch}_T(\text{source}\_T(u)) : \text{mch}_y(u) \neq \top \} \). We say that \( x \in X \) is a differential variable of \( v \) relative to \( u \) if one of the following conditions satisfies:
\[
\begin{align*}
& (a) \quad \text{sym}(u) = \lor, \quad \text{source}\_T(u) = \text{source}\_T(v), \quad \text{and} \quad r\text{Ch}_x(u) = r\text{Ch}_x(v) ; \\
& (b) \quad \text{sym}(u) = \land, \quad \text{source}\_T(u) = \text{source}\_T(v), \quad \text{and} \quad \vartheta(u) \models \bigvee_{y \in \text{Ch}_T(\text{source}\_T(u)) \setminus \{ x \}} \vartheta(\text{mch}_y(u)) ; \quad \text{and} \\
& (c) \quad \text{source}\_T(u) \prec \text{source}\_T(v), \quad \text{there is another variable} \ y \in X, \ \text{source}\_T(u) \preceq y, \ \text{and} \ \vartheta(u) \models \neg \vartheta(\text{mch}_y(v)).
\end{align*}
\]

An algorithm called \textsc{ConjoinTree} is presented in Algorithm 13, which can conjoin two ROAODD\(_T\)s in polytime, and we can propose a disjunction algorithm in a dual fashion.

Proposition 21. Given any two ROAODD\(_T\) rooted at \( u \) and \( v \), if existing, the output of \textsc{ConjoinTree} is an AOADD\(_T\), which is equivalent to \( \vartheta(u) \land \vartheta(v) \). The time complexity of \textsc{ConjoinTree} is bounded by \( O(|G_u| \cdot |G_v| \cdot |T|^2) \).

Roughly speaking, the algorithm \textsc{ConjoinTree} is a generalization of the \textsc{Apply} algorithm in [6], which can conjoin two ROA(O\(_\leq \infty\)O\(_\leq 0\)DD\(_T\)). Given two ROAODD\(_T\)s rooted at \( u \) and \( v \), the disjunction of \( G_u \) and \( G_v \) can be done as follows: first, we get the negations of \( G_u \) and \( G_v \) using \textsc{Negate}(assuming the resulting ROAODD\(_T\) rooted at \( u' \) and \( v' \)); then we conjoin \( G_{u'} \) and \( G_{v'} \) using \textsc{Negate}; finally, the disjunction of \( G_u \) and \( G_v \) is generated by negating the result in the last step. Therefore, the disjunction of two AOADD\(_T\)s can also be done in the same time complexity.

5.2. Tractability evaluation of AOADD up to the KC map

In this subsection, we evaluate the inferential power of AOADD with respect to the criterion proposed in the knowledge compilation map. The evaluating criteria fall into two categories: queries and transformations. A query is an operation that returns information about a theory without changing it, while a transformation is an operation that returns a modified theory. Next we recall all querying and transformation requirements, and the reader is referred to [2] for their importance.

Definition 18. Given any target language \( L \),
Algorithm 13: CONJOIN_TREE\(u, v)\)

**Input:** two ROAODDs rooted at \(u \) and \(v\) over tree \(T\)

**Output:** an ROAODD \(T\) which is equivalent to \(\vartheta(u) \land \vartheta(v)\) if existing, and report failure otherwise

```
1 if \(H(u, v) \neq \text{nil}\) then return \(H(u, v)\)
2 else if \(u/v = \perp\) then \(H(u, v) \leftarrow \perp\)
3 else if \(u/v = \top\) then \(H(u, v) \leftarrow v/u\)
4 else if \(\text{source} \tau(u)\) and \(\text{source} \tau(v)\) are incomparable then \(H(u, v) \leftarrow \langle \land\{u, v\}\rangle\)
5 else if \(\vartheta(u) = \vartheta(v)\) then \(H(u, v) \leftarrow u\)
6 else if \(\vartheta(u) = \neg \vartheta(u)\) then \(H(u, v) \leftarrow v\)
7 else if \(\vartheta(u) = \neg \vartheta(v)\) then \(H(u, v) \leftarrow \perp\)
8 else
9     if \(\text{source} \tau(u) \prec \text{source} \tau(v)\) then
10        if \(\text{sym}(v) = \land\) then
11           Let \(x\) be the variable in \(\text{Ch(source} \tau(u))\) such that \(\text{source} \tau(v) \preceq x\)
12           \(H(u, v) \leftarrow \langle \land\{u, \text{Ch}(\text{mch}_{x}(u)) \cup \{\text{CONJOIN_TREE}(\text{mch}_{x}(u), v)\}\}\rangle\)
13        else if \(\text{sym}(v) = \lor\) then
14           if some variable \(x\) is the differential variable of \(u\) relative to \(v\) then
15              \(H(u, v) \leftarrow \langle \land\{u, \text{mch}_{x}(v)\}\rangle\)
16           else report failure
17        else \(H(u, v) \leftarrow \langle \text{sym}(v), \text{CONJOIN_TREE}(u, \text{lo}(v)), \text{CONJOIN_TREE}(u, \text{hi}(v))\rangle\)
18     else if \(\text{source} \tau(u) \prec \text{source} \tau(v)\) then
19        if \(\text{sym}(u) = \land\) then
20           Let \(x\) be the variable in \(\text{Ch(source} \tau(v))\) such that \(\text{source} \tau(u) \preceq x\)
21           \(H(u, v) \leftarrow \langle \land\{\text{Ch}(x), \text{Ch} \left(\text{mch}_{x}(v)\right) \cup \{\text{CONJOIN_TREE}(\text{mch}_{x}(u), v)\}\}\rangle\)
22        else if \(\text{sym}(u) = \lor\) then
23           if some variable \(x\) is the differential variable of \(v\) relative to \(u\) then
24              \(H(u, v) \leftarrow \langle \land\{\text{mch}_{x}(u), v\}\rangle\)
25           else report failure
26        else \(H(u, v) \leftarrow \langle \text{sym}(v), \text{CONJOIN_TREE}(\text{lo}(u), v), \text{CONJOIN_TREE}(\text{hi}(u), v)\rangle\)
27     else
28        if \(\text{sym}(u) = \text{sym}(v) = \land\) then
29           \(H(u, v) \leftarrow \langle \land\{\text{CONJOIN_TREE}(\text{mch}_{x}(u), \text{mch}_{x}(v)) : x \in \text{Ch(source} \tau(v))\}\rangle\)
30        else if \(\text{sym}(u) = \land\) and \(\text{sym}(v) = \lor\) then
31           if some variable \(x\) is the differential variable of \(v\) relative to \(u\) then
32              \(H(u, v) \leftarrow \langle \land, \text{Ch}(x), \text{Ch} \left(\text{mch}_{x}(v)\right) \cup \{\text{CONJOIN_TREE}(\text{mch}_{x}(u), \text{mch}_{x}(v))\}\rangle\)
33           else report failure
34        else if \(\text{sym}(u) = \lor\) and \(\text{sym}(v) = \land\) then
35           if some variable \(x\) is the differential variable of \(u\) relative to \(v\) then
36              \(H(u, v) \leftarrow \langle \lor, \text{Ch}(x), \text{Ch} \left(\text{mch}_{x}(u)\right) \cup \{\text{CONJOIN_TREE}(\text{mch}_{x}(u), \text{mch}_{x}(v))\}\rangle\)
37           else report failure
38        else if \(\text{sym}(u) = \text{sym}(v) = \lor\) then
39           if some variable \(x\) is the differential variable of \(u\) and \(v\) then
40              \(H(u, v) \leftarrow \langle \lor, \text{Ch}(x) \cup \{\text{CONJOIN_TREE}(\text{mch}_{x}(u), \text{mch}_{x}(v))\}\rangle\)
41           else report failure
42        else \(H(u, v) \leftarrow \langle \text{sym}(v), \text{CONJOIN_TREE}(\text{lo}(u), \text{lo}(v)), \text{CONJOIN_TREE}(\text{hi}(u), \text{hi}(v))\rangle\)
43     end
44 end
45 return \(H(u, v)\)
```
• L satisfies \textbf{CO} (resp. \textbf{VA}) iff there exists a polytime algorithm that maps every formula \( \varphi \) in L to 1 if \( \varphi \) is consistent (resp. valid), and to 0 otherwise;
• L satisfies \textbf{CE} iff there exists a polytime algorithm that maps every formula \( \varphi \) in L and every clause \( C \) to 1 if \( \varphi \models C \) holds, and to 0 otherwise;
• L satisfies \textbf{IM} iff there exists a polytime algorithm that maps every formula \( \varphi \) in L and every term \( T \) to 1 if \( T \models \varphi \) holds, and to 0 otherwise;
• L satisfies \textbf{EQ} (resp. \textbf{SE}) iff there exists a polytime algorithm that maps every pair of formulas \( \varphi, \varphi' \) in L to 1 if \( \varphi \equiv \varphi' \) (resp. \( \varphi \models \varphi' \)) holds, and to 0 otherwise;
• L satisfies \textbf{CT} iff there exists a polytime algorithm that maps every formula \( \varphi \) in L and some the variable set \( X \) which includes all variables appearing in \( \varphi \) to a non-negative integer that represents the number of models of \( \varphi \) over \( X \) (in binary notation);
• L satisfies \textbf{ME} iff there exists a polynomial \( p(\ldots) \) and an algorithm that outputs all models of an arbitrary formula \( \varphi \) in L over some the variable set \( X \) which includes all variables appearing in \( \varphi \) in time \( p(n,m) \), where \( n \) is the size of \( \varphi \) and \( m \) is the number of its models over \( X \).

\textbf{Definition 19.} Given any target language \( L \),
• L satisfies \textbf{CD} iff there exists a polytime algorithm that maps every formula \( \varphi \) in L and every consistent term \( T \) to a formula in L that is equivalent to \( \varphi T \).
• L satisfies \textbf{FO} iff there exists a polytime algorithm that maps every formula \( \varphi \) in L and every subset \( X \) of the set of variables appearing in \( \varphi \) to a formula in L that is equivalent to \( \exists X.\varphi \), i.e., the formula that does not mention any variable in \( X \) and for every formula \( \varphi' \) that does not mention any variable in \( X \), we have \( \varphi \models \varphi' \) precisely when \( \exists X.\varphi \models \varphi' \). If the property holds for singleton \( X \), we say that \( L \) satisfies \textbf{SFO}.
• L satisfies \( \land C \) (resp. \( \lor C \)) iff there exists a polytime algorithm that maps every finite set of formulas \( \varphi_1, \ldots, \varphi_n \) in L to a formula of L that is equivalent to \( \varphi_1 \land \cdots \land \varphi_n \) (resp. \( \varphi_1 \lor \cdots \lor \varphi_n \)).
• L satisfies \( \land BC \) (resp. \( \lor BC \)) iff there exists a polytime algorithm that maps every pair of formulas \( \varphi \) and \( \varphi' \) in L to a formula of L that is equivalent to \( \varphi \land \varphi' \) (resp. \( \varphi \lor \varphi' \)).
• L satisfies \( \lnot C \) iff there exists a polytime algorithm that maps every formula \( \varphi \) in L to a formula of L that is equivalent to \( \lnot \varphi \).

For the incomplete language \( L \), it conditionally satisfies the above requirements iff there exists polytime algorithms such that: the algorithms perform the corresponding operators if the operating results can be represented in L, and report failure otherwise.

Table 2 summarizes query-related properties of \( \text{ROBDD-}L_\infty \). As \( \text{ROBDD} \), \( \text{FBDD} \) and \( \text{d-DNNF} \) are three of the most widely used target languages in practical applications, their properties are also shown here for comparison.

\textbf{Proposition 22.} Given any two \( \text{OA}_{\leq 1}\text{O}_{\leq 1}\text{DD} \)s rooted at \( u \) and \( v \), where \( i+j \geq 1 \), the problem of deciding whether \( \vartheta(u) \models \vartheta(v) \) holds is in \( \text{co-NP-complete} \).

\textbf{Proof.} Membership is obvious, as the problem of deciding the entailment of two propositional formulas is in \( \text{co-NP} \). The hardness is proved by taking advantage of the idea that was used to prove the complexity of entailment of two \( \text{FBDD} \)s in [13]. That is, for any 3-CNF formula

\[ \varphi = C_1 \land \cdots \land C_m = (l_{1,1} \lor l_{1,2} \lor l_{1,3}) \land \cdots \land (l_{m,1} \lor l_{m,2} \lor l_{m,3}) \],

we prove it is unsatisfiable iff an \( \text{OA}_{\leq 1}\text{O}_{\leq 1}\text{DD} \) entails another one. According to the algorithm \textsc{Decompose}, it is sufficient to prove the result by showing the proposition holds for cases \( i = 0, j = 1 \) and \( i = 1, j = 0 \). Given two \( \text{OA}_{\leq 0}\text{O}_{\leq 1}\text{DD} \)s rooted at \( u \) and \( v \), \( \vartheta(u) \models \vartheta(v) \) iff \( \vartheta(\text{Negate}(v)) \models \vartheta(\text{Negate}(u)) \). Since the algorithm \text{Negate} has a linear complexity, it is only needed to prove the result for the case \( i = 0, j = 1 \).

Introduce a new \( y_{i,j} \) (\( 1 \leq i \leq m, 1 \leq j \leq 3 \)) for each \( l_{i,j} \). Assume that \( x_k \) negatively (positively) appears \( a_k \) (\( b_k \)) times in \( \varphi \). If \( l_{i,j} = \lnot x_k \) (\( x_k \)) and there are \( a-1 \) (\( b-1 \)) negative (resp. positive) appearances of \( x_k \) before \( l_{i,j} \), we introduce another denotation \( x_{k,a} \) (\( x_{k,a}^a \)) for \( y_{i,j} \). Over the order \( x_1 \lhd \cdots \lhd x_n \lhd y_{1,1} \lhd y_{1,2} \lhd y_{1,3} \lhd \cdots \lhd y_{m,1} \lhd y_{m,2} \lhd y_{m,3} \), \( \varphi \) is unsatisfiable iff the \( \text{OA}_{\leq 0}\text{O}_{\leq 1}\text{DD} \) in Figure 1 entails the one in Figure 2.
6. The succinctness of subsets of OAODD

In this section, we analyse the succinctness relation of subsets of OAODD through the two following propositions:

**Definition 20 (succinctness).** Let $L_1$ and $L_2$ be two target languages. $L_1$ is at least as succinct as $L_2$, if and only if there exists a polynomial $p$ such that for every sentence $\alpha \in L_2$, there exists an equivalent sentence $\beta \in L_1$ where $|\beta| \leq p(|\alpha|)$. Here, $|\alpha|$ and $|\beta|$ are the sizes of $\alpha$ and $\beta$, respectively. $L_1$ is strictly more succinct than $L_2$ if and only if $L_1$ is at least as succinct as $L_2$, while $L_2$ is not at least as succinct as $L_1$.

**Proposition 23.** For $i+j < i'+j'$, $i \leq i'$, $j \leq j'$, $OA_{\leq i}O_{\leq j}DD$ is strictly more succinct than $OA_{\leq i'}O_{\leq j'}DD$.

**Proof.** We proceed by case analysis:
Figure 2: An RO_{\leq_0}O_{\leq_1}DD

(1) $i < i'$ and $j = j'$: Let us consider the formula $\varphi = \varphi_{cbf}[y_1/(y_1 \leftrightarrow \cdots \leftrightarrow y_{1,i+1}), \cdots, y_n/(y_{n,1} \leftrightarrow \cdots \leftrightarrow y_{n,i+1})]$. Any equivalent OA_{\leq_i}O_{\leq_j}DD has an exponential number of nodes, while the number of nodes in the corresponding OA_{\leq_i}O_{\leq_j}DD over $x_1 \prec \cdots \prec x_k \prec y_1 \prec \cdots \prec y_{1,i+1} \prec \cdots \prec y_{1,i+1} \prec \cdots \prec y_{n,1} \prec \cdots \prec y_{n,i+1} \prec \cdots \prec z_1 \prec \cdots \prec z_n$ is only linear.

(2) $i = i'$ and $j < j'$: Consider the formula $\neg \varphi_{cbf}[y_1/(y_{1,1} \leftrightarrow \cdots \leftrightarrow y_{1,j+1}), \cdots, y_n/(y_{n,1} \leftrightarrow \cdots \leftrightarrow y_{n,j+1})]$. The proof is similar to (1);

(3) $i < i'$ and $j < j'$: OA_{\leq_i}O_{\leq_j}DD is strictly less succinct than OA_{\leq_i'}O_{\leq_j'}DD, which is strictly less succinct than OA_{\leq_i'}O_{\leq_j'}DD.

\boxed{}

Proposition 24. OA_{\leq_\infty}O_{\leq_0}DD_T is not at least as succinct as OA_{\leq_1}O_{\leq_0}DD.

Proof. Let us consider the formula $\varphi_{cbs}$ over $X = \{x_1, \cdots, x_k\}$, $Y = \{y_1, \cdots, y_n\}$, $Z = \{z_1, \cdots, z_n\}$ and $n = 2^k$. Let $T$ be a tree over $X \cup Y \cup Z$ such that there exists an OA_{\leq_\infty}O_{\leq_0}DD_T over $T$ which is equivalent to $\varphi_{cbs}$. We show by contradiction that for any $y_i$ and $z_j$ ($1 \leq i,j \leq n$), either $y_i \prec_T z_j$ or $z_j \prec_T y_i$ holds.
Table 3: The polytime query of ROBDD-$L_\infty$. √ means “satisfies”, ✓ means “conditionally satisfies”, • means “does not satisfy”, and ◦ means “does not satisfy unless P = NP”.

| L          | CD | FO | SFO | ∧C | ∧BC | ∨C | ∨BC | ¬C |
|------------|----|----|-----|----|-----|----|-----|----|
| $O\mathcal{A}_{\leq 0}O\mathcal{S}_0\mathcal{D}D$ | ✓  | •  | ✓   | •  | ✓   | •  | ✓   | ✓  |
| $O\mathcal{A}_{\leq 0}O_{\leq 1}\mathcal{D}D$ (i > 0) | ✓  | ◦  | ◦   | ◦  | ◦   | ◦  | ◦   | ✓  |
| $O\mathcal{A}_{\leq 0}O_{\leq 1}\mathcal{D}D$ (i ≠ j) | ✓  | ◦  | ◦   | ◦  | ◦   | ◦  | ◦   | •  |
| $O\mathcal{A}_{\leq 0}O_{\leq \infty}\mathcal{D}D_T$ | ✓  | ◦  | ✓   | ◦  | ✓   | ◦  | ✓   | ✓  |
| $O\mathcal{A}_{\leq 0}O_{\leq \infty}\mathcal{D}D_T$ | ✓  | ◦  | ✓   | ◦  | ✓   | ◦  | ✓   | ✓  |

Otherwise, let $Y'$ ($Z'$) be the set of ancestors of $y_i$ and $z_j$ in $Y$ ($Z$), and then for every min-term $\gamma$ over $X \cup Y' \cup Z'$, $\phi_{\text{cnf}}(\gamma) \equiv \varphi_1 \land \varphi_2$, where $\text{Vars}(\varphi_1) \cap \text{Vars}(\varphi_2) = \emptyset$, $y_i \in \text{Vars}(\varphi_1)$ and $z_j \in \text{Vars}(\varphi_2)$. The term $\gamma = \varphi_2(i - j) \land \gamma_1 \land \gamma_2$ is a disproof here since $y_i \leftrightarrow z_j \models \phi_{\text{cnf}}(\gamma)$, where $\varphi_2(i - j)$ is a min-term over $X$, and $\gamma_1$ and $\gamma_2$ are any two min-terms over $Y'$ and $Z'$, respectively. Therefore, either $\gamma_Y$ or $\gamma_Z$ is a linear order. Without loss of generality, we assume that $\gamma_Y$ is a linear order. Let $z_j$ be the minimum variable which is greater than every variable in $Y$, and let $Y'$ ($Z'$) be the set of variables in $Y$ ($Z$) which are less than $z_j$. $\gamma_{Y' \cup Z'}$ is linear and $|Y' \cup Z'| \geq n$. By a similar proof to that of Proposition 1, it is easy to prove this proposition. □

7. Compiling NNFs into OAOdds

7.1. Compilation algorithm

We propose a search-based compilation algorithm called Compile (Algorithm 14) for OAOdds. On line 3–6 in the algorithm, we compute the $A_{\leq 1}$-decomposition of $\vartheta(v)$. If the previous $A_{\leq 1}$-decomposition is not strict, then we compute $O_{\leq 1}$-decomposition on line 8–11. $A_{\leq 1}$-decomposition and $O_{\leq 1}$-decomposition can be computed using SAT solver. If neither $A_{\leq 1}$-decomposition nor $O_{\leq 1}$ is strict, we try to decompose $\vartheta(v)$ by partitioning $Ch(u)$ into minimal subsets $V_1, \ldots, V_m$ such that any $V_1$ and $V_i$ ($1 \leq i \neq j \leq m$) do not share variable.

7.2. Optimization Techniques

In the algorithm Compile, it is needed to compute unit implicants and implicates of $\vartheta(v)$ on lines 4 and 9, respectively. We show how unit implicates can be computed using SAT solver (e.g., MiniSAT [16]), and the computation of unit implicates is a dual case, since $l \models \vartheta(v)$ iff $\neg \vartheta(v) \models \neg l$ and an NNF formula can be negated in linear time. For the sake of simplicity, we assume the NNF formula does not include $\perp$ and $\top$ in the following.

We first introduce a literal notation $l_v$ for each vertex $v$, where $l_v = \text{sym}(v)$ if $v$ is a leaf vertex label with variable, and $l_v = \neg \text{sym}(v)$ if $\text{sym}(v) = \neg$ and $Ch(v) = \{w\}$, and $l_v = x_v$ ($x_v$ is a fresh variable for each vertex $v$) otherwise. Then we define a function from NNF to CNF:

$$nnf2cnf(v) = \begin{cases} \{l_w : w \in Ch(v)\} \cup \bigcup_{w \in V(G_v) \setminus \{v\}} v2cls(w) & \text{sym}(v) = \land; \\ \{V_{w \in Ch(v)} l_w\} \cup \bigcup_{w \in V(G_v) \setminus \{v\}} v2cls(w) & \text{sym}(v) = \lor; \\ \text{true} & v = \top; \\ \text{false} & v = \perp; \\ l_v & \text{otherwise.} \end{cases}$$
Algorithm 14: Compile(u)

Input: an NNF formula rooted at \( u \) over a global linear variable order \( \prec \)

Output: an OA\( \infty \)O\( \infty \)DD over \( \prec \) which is equivalent to \( \vartheta(v) \)

1. if \( u = \bot/\top \) then return \( v \)
2. Let \( \gamma \) be the sets of unit implicants of \( \vartheta(v) \)
3. if \( \gamma \neq \emptyset \) then
   4. \( V \leftarrow \{\langle x, \bot, \top \rangle : x \in \gamma \} \cup \{\langle x, \top, \bot \rangle : \neg x \in \gamma \} \)
   5. return \( \langle A, \{\text{Compile(Condition}(v, \gamma))\} \cup V \rangle \)
4. end
7. Let \( \delta \) be the set of unit implicates of \( \vartheta(v) \)
8. if \( \delta \neq \emptyset \) then
   9. \( V \leftarrow \{\langle x, \bot, \top \rangle : x \in \delta \} \cup \{\langle x, \top, \bot \rangle : \neg x \in \delta \} \)
10. return \( \langle O, \{\text{Compile(Condition}(v, \neg \delta))\} \cup V \rangle \)
11. end
12. Group Ch(\( u \)) in minimal subsets \( V_1, \ldots, V_m \) which don’t share variable
13. if \( i = 1 \) then
14. \( v_1 \leftarrow \text{Condition}(v, \neg x) \); \( v_2 \leftarrow \text{Condition}(v, x) \)
15. \( \text{return } \langle x, \text{Compile}(v_1), \text{Compile}(v_2) \rangle \)
16. else return \( \langle \text{sym}(v), \{\text{Compile}(\langle \text{sym}(v), V_i \rangle) : 1 \leq i \leq m\} \rangle \)

where

\[
\text{v2cls}(v) = \begin{cases} 
\{\neg l_v \lor l_w : w \in \text{Ch}(v)\} \cup \{l_v \lor \bigvee_{w \in \text{Ch}(v)} \neg l_w\} & \text{sym}(v) = \land; \\
\{l_v \lor \neg l_w : w \in \text{Ch}(v)\} \cup \{\neg l_v \lor \bigvee_{w \in \text{Ch}(v)} l_w\} & \text{sym}(v) = \lor; \\
\emptyset & \text{otherwise.}
\end{cases}
\]

It is obvious the function above has the following property:

**Proposition 25.** Given an NNF formula rooted at \( v \), a world \( \omega \in 2^{\text{Vars}(v)} \) and a literal \( l \) over \( \text{Vars}(v) \), \( \vartheta(v)|_\omega \models l \) iff \( \text{nnf2cnf}(v)|_\omega \cup \{\neg l\} \) is unsatisfiable.

Therefore, we can use two optimization techniques mentioned in [7] to speedup the computation of unit implicates and implicants, i.e., adopting efficient SAT solving techniques and exploiting Horn lower approximation.

8. Conclusions

In this paper, we study the effect of augmenting OBDD with AND-decomposability and OR-decomposability in the KC context and the resulting KC language is called OAODD. By imposing two constraints, we obtained two types of fragments in OAODD. The first one is called OAODD with bounded decomposition, denoted by OA\( \infty \)O\( \infty \)DD, and the second one is called OAODD with tree-structured decomposition, denoted by OAODD\( _\prec \)\( _\prec \), where \( i \) and \( j \) are two integers, and \( \prec \) is a tree. We devised four algorithms to convert one fragment of OAODD into another. We presented a rich set of polynomial-time algorithms that perform logical operations. According to these algorithms, as well as theoretical analysis, we characterized the space efficiency and tractability of OAODD its some fragments with respect to the evaluating criteria in the knowledge compilation map. Finally, a compilation algorithm which can convert formulas in negative normal form into OAODD was proposed.

A major contribution of this paper is to propose a unified KC framework OAODDD into which several previous languages are included into this framework, including OBDD, AOBDD, OBDD-L and MLDD. In the framework, ROA\( \infty \)O\( \infty \)DD is the most succinct and tractable fragment: given any logical operation

31
$\text{OAODD}$ supports in polytime, $\text{ROA}_{\leq \infty} \text{O}_{\leq \infty} \text{DD}$ can also support it in time polynomial in the sizes of the equivalent $\text{OAODD}$; any $\text{OAODD}$ can be converted into $\text{ROA}_{\leq \infty} \text{O}_{\leq \infty} \text{DD}$ in polytime. Therefore, for the complete compilation, $\text{ROA}_{\leq \infty} \text{O}_{\leq \infty} \text{DD}$ is the first choice. However, a formula has too many possible decompositions, and thus it is hard to compute them from the syntactic point of view. Therefore, in the applications where the incomplete compilation is needed (e.g., importance sampling for model counting [17, 18]), it is possible to choose other fragment whose decomposition is easy to be captured from the viewpoint of syntax.

References

[1] B. Selman, H. Kautz, Knowledge compilation and theory approximation, Journal of the ACM 43 (1996) 193–224.
[2] A. Darwiche, P. Marquis, A knowledge compilation map, Journal of Artificial Intelligence Research 17 (2002) 229–264.
[3] M. Cadoli, F. M. Donini, A survey on knowledge compilation, AI Communications 10 (1997) 137–150.
[4] A. Darwiche, SDD: A new canonical representation of propositional knowledge bases, in: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, 2011, pp. 819–826.
[5] V. Bertacco, M. Damiani, Boolean function representation based on disjoint-support decompositions, in: Proceedings of the 14th International Conference on Computer Design (ICCD’96), VLSI in Computers and Processors, October 7-9, 1996, Austin, TX, USA, 1996, pp. 27–32.
[6] R. Mateescu, R. Dechter, R. Marinescu, AND/OR Multi-Valued Decision Diagrams (AOMDDs) for Graphical Models, Journal of Artificial Intelligence Research 33 (2008) 465–519.
[7] Y. Lai, D. Liu, S. Wang, Reduced ordered binary decision diagram with implied literals: A new knowledge compilation approach, Knowledge and Information Systems(in print).
[8] A. Darwiche, Decomposable negation normal form, Journal of the ACM 48 (4) (2001) 608–647.
[9] A. Darwiche, On the tractability of counting theory models and its application to truth maintenance and belief revision, Journal of Applied Non-Classical Logics 11 (1–2) (2001) 11–34.
[10] K. Pipatsrisawat, A. Darwiche, New compilation languages based on structured decomposability, in: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, 2008, pp. 517–522.
[11] R. E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Transactions on Computers 35 (8) (1986) 677–691.
[12] A. Blake, Canonical expressions in boolean algebra, Ph.D. thesis, University of Chicago, Chicago, IL (1937).
[13] S. Fortune, J. E. Hopcroft, E. M. Schmidt, The complexity of equivalence and containment for free single variable program schemes, in: Proceedings of the 5th Colloquium on Automata, Languages and Programming, 1978, pp. 227–240.
[14] A. Darwiche, New advances in compiling cnf into decomposable negation normal form, in: Proceedings of Sixteenth European Conference on Artificial Intelligence, 2004, pp. 328–332.
[15] J. Huang, A. Darwiche, The language of search, Journal of Artificial Intelligence Research 29 (2007) 191–219.
[16] N. Eén, N. Sörensson, An extensible sat-solver, in: Proceedings of 6th International Conference on Theory and Applications of Satisfiability Testing, 2003, pp. 502–518.
[17] V. Gogate, R. Dechter, Samplesearch: Importance sampling in presence of determinism, Artificial Intelligence 175 (2011) 694–729.
[18] V. Gogate, R. Dechter, Importance sampling-based estimation over and/or search spaces for graphical models, Artificial Intelligence 184-185 (2012) 38–77.