Knowledge Compilation Properties of Trees-of-BDDs, Revisited

Hélène Fargier  
IRIT-CNRS, UMR 5505  
Université de Toulouse, France  
fargier@irit.fr

Pierre Marquis  
CRIL-CNRS, UMR 8188  
Université Lille-Nord de France, Artois, France  
marquis@cril.univ-artois.fr

Abstract

Recent results have shown the interest of trees-of-BDDs [Subbarayan et al., 2007] as a suitable target language for propositional knowledge compilation from the practical side. In the present paper, the concept of tree-of-BDDs is extended to additional classes of data structures C thus leading to trees-of-C representations (ToC). We provide a number of generic results enabling one to determine the queries/transformations satisfied by ToC depending on those satisfied by C. We also present some results about the spatial efficiency of the ToC languages. Focusing on the ToOBDD< language (and other related languages), we address a number of issues that remained open in [Subbarayan et al., 2007]. We show that beyond CO and VA, the ToOBDD< fragment satisfies IM and ME but satisfies neither CD nor any query among CE, SE unless P = NP. Among other results, we prove that ToOBDD< is not comparable w.r.t. succinctness with any of CNF, DNF, DNNF unless the polynomial hierarchy collapses. This contributes to the explanation of some empirical results reported in [Subbarayan et al., 2007].

1 Introduction

This paper is concerned with “knowledge compilation” (KC), a family of approaches proposed so far for addressing the intractability of a number of AI problems of various kinds (reasoning, decision making, etc.). The key idea underlying KC is to pre-process parts of the available information (i.e., turning them into a compiled form) for improving on-line computational efficiency (see among others [Darwiche, 2001; Cadoli and Donini, 1998; Selman and Kautz, 1996; del Val, 1994]).

A important research line in KC [Gogic et al., 1995; Darwiche and Marquis, 2002] addresses the following issue: How to choose a target language for knowledge compilation? In [Darwiche and Marquis, 2002], the authors argue that the choice of a target language must be based both on the set of queries and transformations which can be achieved in polynomial time when the data are represented in the language, as well as the spatial efficiency of the language. They pointed out a KC map which can be viewed as a multi-criteria evaluation of a number of propositional fragments, including DNNF, d-DNNF, CNF, DNF, OBDD<, OBDD (the union of all OBDD< when < varies), etc. (see [Darwiche and Marquis, 2002] for details). From there, other propositional fragments have been considered so far and put in the KC map, see for instance [Wachter and Haenni, 2006; Fargier and Marquis, 2006; Subbarayan et al., 2007; Pipatsrisawat and Darwiche, 2008; Fargier and Marquis, 2008a; 2008b].

Recent experimental results have shown the practical interest of trees-of-BDDs [Subbarayan et al., 2007] as a target language for propositional knowledge compilation: it turns out that the tree-of-BDDs language renders feasible the compilation of a number of benchmarks which cannot be compiled into d-DNNF due to space limitations.

In the present paper, we elaborate on the tree-of-BDDs language. After some formal preliminaries (Section 2), we generalize the tree-of-BDDs language to the family of ToC representations where C is any complete propositional language (Section 3). We provide a number of generic results enabling one to determine the queries/transformations satisfied by ToC depending on the queries/transformations satisfied by C. We also present some results about the spatial efficiency of the ToC languages. Focusing on ToOBDD< and some related languages, we then address a number of issues that remained open in [Subbarayan et al., 2007] (Section 4): beyond CO and VA, the ToOBDD< language satisfies IM and ME but does not satisfy any query among CE, SE unless P = NP. Under similar assumptions from complexity theory, we demonstrate that ToOBDD< does not satisfy any transformation among CD, FO, \wedge BC, \vee C or ¬C. Among other succinctness results, we prove that the ToOBDD< language is not comparable w.r.t. succinctness with any of CNF, DNF or DNNF unless the polynomial hierarchy PH collapses. This contributes to the explanation of some empirical results reported in [Subbarayan et al., 2007]. We conclude the paper by a discussion of the results and some perspectives (Section 5). Proofs are omitted for space reasons but are available at http://www.fr/~marquis/fargier-marquis-ijcai09.pdf.
Representations and the KC Map

Trees-of-BDDs and their forthcoming generalization are not *stricto sensu* formulae. Hence we need to extend the notions of queries, transformations and succinctness at work in the KC map to such representations. Roughly speaking, a propositional representation language is a way to represent Boolean functions. Such a representation language often takes the form of a standard propositional language but other data structures can be used as well (e.g. Karnaugh maps, truth tables, various graphs including those binary decision diagrams,... and of course trees-of-BDDs) for the representation purpose.

Formally, given a finite set of propositional variables $PS$, we consider Boolean functions from $\{0,1\}^X \rightarrow \{0,1\}$, where $X \subseteq PS$. $Var(f) = X$ is called the scope of $f$. The support $Ω(f)$ of $f$ is the set of all assignments $ω$ of $Var(f)$ to Boolean values such that $f(ω) = 1$. For any $X \subseteq PS$, we note by $X$ the set $PS \setminus X$. The set of Boolean functions is equipped with the three standard internal laws, $∧$, $∨$ and $¬$. Given $X \subseteq PS$ we note by $∃X,f$ the Boolean function with scope $Var(f) \setminus X$ that maps 1 to an assignment $ω_{Var(f) \setminus X}$ of $Var(f) \setminus X$ iff there exists an assignment $ω$ of $Var(f)$ such that the restriction of $ω$ over $Var(f) \setminus X$ and $ω_{Var(f) \setminus X}$ coincide and $f(ω) = 1$.

**Definition 1** (representation language) ([inspired from Ilogic et al., 1995]) A (propositional) representation language over a finite set of propositional variables $PS$ is a set $C$ of data structures $α$ (also referred to as $C$ representations) together with a scope function $Var : C \rightarrow 2^X$ with $X \subseteq PS$ and an interpretation function $I(α)$ which associates to each $C$ representation $α$ a Boolean function $I(α)$ the scope of which is $Var(α)$. $C$ is also equipped with a size function from $C$ to $N$ that provides the size $|α|$ of any $C$ representation $α$.

**Definition 2** (complete language) A propositional representation language $C$ is said to be complete iff for any Boolean function $f$ with $Var(f) \subseteq PS$, there exists a $C$ representation $α$ such that $I(α) = f$.

Clearly enough, formulae from a standard propositional language are representations of Boolean functions. The size of such a formula is the number of symbols in it. Slightly abusing words, if $Σ$ is a propositional formula representing a Boolean function $g$ one often says that a representation $α$ of $g$ is a representation of $Σ$ instead of $α$ is a representation of the semantics of $Σ$.

The DAG–NNF language [Darwiche and Marquis, 2002] is also a complete graph-based representation language of Boolean functions. Distinguished formulae from DAG–NNF are the literals over $PS$, the clauses (a clause is a finite disjunction of literals or the Boolean constant $⊥$) and the terms (a term is a finite conjunction of literals or the Boolean constant $⊤$). We assume the reader to be familiar with the

DAG–NNF fragments DNNF, d–DNF, CNF, DNF, FBDD, OBDD, OBDD<, OBDD, MODS, etc.

Obviously, all the logical notions pertaining to formulae viewed up to logical equivalence can be easily extended to any representation language $C$ of Boolean functions. For instance, an assignment $ω$ of $Var(α)$ to Boolean values is said to be a model of a $C$ representation $α$ over $Var(α)$ iff $I(α)(ω) = 1$. Similarly, two representations $α$ and $β$ (possibly from different representation formalisms) are said to be equivalent, noted $α \equiv β$, when they represent the same Boolean function. A $C$ representation $α$ is consistent (resp. valid) iff $α$ does not represent the Boolean function $0$ (resp. represents the Boolean function $1$). $α$ is a logical consequence of $β$, noted $β \models α$ iff $Ω(I(β)) \subseteq Ω(I(α))$.

We are now ready to extend the notions of queries, transformations and succinctness considered in the KC map to any propositional representation language. Their importance is discussed in depth in [Darwiche and Marquis, 2002], so we refrain from recalling it here.

**Definition 3** (queries) Let $C$ denote a propositional representation language.

- $C$ satisfies $CO$ (resp. $VA$) iff there exists a polytime algorithm that maps every $C$ representation $α$ to $1$ if $α$ is consistent (resp. valid), and to $0$ otherwise.
- $C$ satisfies $CE$ iff there exists a polytime algorithm that maps every $C$ representation $α$ and every clause $δ$ to $1$ if $α \models δ$ holds, and to $0$ otherwise.
- $C$ satisfies $EQ$ (resp. $SE$) iff there exists a polytime algorithm that maps every pair of $C$ representations $α$, $β$ to $1$ if $α \equiv β$ (resp. $α \models β$) holds, and to $0$ otherwise.
- $C$ satisfies $IM$ iff there exists a polytime algorithm that maps every $C$ representation $α$ and every term $γ$ to $1$ if $γ \models α$ holds, and to $0$ otherwise.
- $C$ satisfies $CT$ iff there exists a polytime algorithm that maps every $C$ representation $α$ to a nonnegative integer that represents the number of models of $α$ over $Var(α)$ (in binary notation).
- $C$ satisfies $ME$ iff there exists a polynomial $p(.,.)$ and an algorithm that outputs all models of an arbitrary $C$ representation $α$ in time $p(|α|, m)$, where $m$ is the number of its models (over $Var(α)$).

**Definition 4** (transformations) Let $C$ denote a propositional representation language.

- $C$ satisfies $CD$ iff there exists a polytime algorithm that maps every $C$ representation $α$ and every consistent term $γ$ to a $C$ representation $β$ of the restriction of $I(α)$ to $I(γ)$, i.e., $Var(β) = Var(α) \setminus Var(γ)$ and $I(γ) = \exists Var(γ).I(α) \land I(γ)$.
- $C$ satisfies $FO$ iff there exists a polytime algorithm that maps every $C$ representation $α$ and every subset $X$ of variables from $PS$ to a $C$ representation of $∃X.I(α)$. If the property holds for each singleton $X$, we say that $C$ satisfies $SFO$.
- $C$ satisfies $\land C$ (resp. $\lor C$) iff there exists a polytime algorithm that maps every finite set of $C$ representations

\footnote{I refers to the interpretation function associated to the $C$ language, so that $I_E$ would be a more correct notation for it; nevertheless, in order to keep the notations light and since no ambiguity is possible, we refrained from indexing the functions $I$ (as well as $Var$ and the size function) by the associated representation language.}
\section*{3 The \textsc{ToC} Languages}

We start with the definition of trees-of-BDDs as given in [Subbarayan et al., 2007] (modulo the notations used):

\begin{definition}[tree-of-BDDs]
\end{definition}

\begin{definition}[succinctness]
\end{definition}

\begin{definition}[polynomial translation]
\end{definition}

Clearly, \textsc{ToB} is a complete representation language: for every Boolean function there is a \textsc{CNF} formula \( \Sigma \) representing it, and thus a tree-of-BDDs \( T \) of \( \Sigma \) such that \( I(T) = I(\Sigma) \).

The above definition can be simplified and extended, allowing the representation of other formul\( \text{ae} \) as \textsc{CNF} ones, and taking advantage of other target languages than \textsc{OBDD} for compiling the labels \( B(n) \):

\begin{definition}[\textsc{ToC}]
\end{definition}

Taking \( C = \text{\textsc{OBDD}} \), we get the \textsc{ToOBDD} language. Clearly, this definition of \textsc{ToOBDD} is close to the previous one \textsc{ToB} from [Subbarayan et al., 2007], except that a \textsc{ToOBDD} representation \( T \) is defined \textit{per se}, i.e., independently from a given \textsc{CNF} formula \( \Sigma \). Within this language, unlike with the \textsc{OBDD} one, a Boolean function may have several equivalent representations. For instance, let \( \Sigma = (\neg a \lor \neg b) \land (\neg a \land c) \lor (b \land c) \). Whatever \( \prec C \), \( I(\Sigma) \) can be represented by the \textsc{ToOBDD} representation \( T \) such that \( T \) has a single node \( n_0 \) such that \( \text{Var}(n_0) = \text{Var}(\Sigma) \) and \( B(n_0) \) is the \textsc{OBDD} representation equivalent to \( C \); observing that \( I(\Sigma) \) can also be represented by the \textsc{ToOBDD} representation \( T \) such that \( T \) has two nodes \( n_0 \) and \( n_1 \), the root of \( T \) is \( n_0 \), \( \text{Var}(n_0) = \{a, b\}, \text{Var}(n_1) = \{b, c\}, B(n_0) = \text{\textsc{OBDD}} \) formula equivalent to \( \neg a \lor b \), and \( B(n_1) \) is the \textsc{OBDD} formula equivalent to \( \neg b \lor c \). In short, \textsc{ToOBDD} does not offer the property of canonical representation.

Compiling a \textsc{CNF} formula \( \Sigma \) into a \textsc{ToC} representation \( T \) basically consists in computing first a decomposition tree of \( \Sigma \), then taking advantage of any \textsc{CNF}-to-\textsc{C} compiler so as to turn the \textsc{CNF} \( \text{\textsc{clauses}}(n) \) formul\( \text{ae} \) (for each node \( n \) of the tree) into equivalent \( C \) representations, and finally to use the well-known message-passing propagation algorithm (see the \textsc{Propagate} function in [Subbarayan et al., 2007], which applies also to \textsc{ToC} representations) from the leaves of the tree to its root then from the root to the leaves so as to ensure the global consistency property. This approach can be easily extended to deal with the compilation of any conjunctive representation into a \textsc{ToC} representation when compilers to \( C \) are available. The running intersection property enables one to replace a global computation on the resulting \textsc{ToC} represen-
tation $T$ by a number of possibly easier, local computations on the corresponding $B(n)$.

Let us now present some generic properties about $\text{ToC}$ fragments; such properties are about queries, transformations and succinctness, and are related to similar properties satisfied by the corresponding $C$ languages. We first need the following definition:

**Definition 9 (TE, CL)** Let $C$ be any propositional representation language.

- $C$ satisfies $\text{TE}$ (the term condition) iff for every term $\gamma$ over $PS$, a $C$ representation equivalent to $\gamma$ can be computed in time polynomial in $|\gamma|$.
- $C$ satisfies $\text{CL}$ (the clause condition) iff for every clause $\delta$ over $PS$, a $C$ representation equivalent to $\delta$ can be computed in time polynomial in $|\delta|$.

Clearly enough, those conditions are not very demanding and are satisfied by all complete languages considered in [Darwiche and Marquis, 2002], but $\text{MODS}$.

**Proposition 1** Let $C$ be any complete propositional representation language.

1. $C$ satisfies $\text{CO}$ iff $\text{ToC}$ satisfies $\text{CO}$.
2. $C$ satisfies $\text{VA}$ iff $\text{ToC}$ satisfies $\text{VA}$.
3. $C$ satisfies $\text{IM}$ iff $\text{ToC}$ satisfies $\text{IM}$.
4. If $C$ satisfies $\text{CD}$, then $C$ satisfies $\text{ME}$ iff $\text{ToC}$ satisfies $\text{ME}$.
5. If $C$ satisfies $\text{CL}$, then $\text{ToC}$ does not satisfy $\text{CE}$ unless $P = NP$.
6. If $C$ satisfies $\text{CL}$, then $\text{ToC}$ does not satisfy $\text{SE}$ unless $P = NP$.

Points 1. to 4. show that the $\text{ToC}$ languages typically satisfy all the queries $\text{CO}$, $\text{VA}$, $\text{IM}$ and $\text{ME}$ (just because the corresponding $C$ languages typically satisfy them and $\text{CD}$). Similarly, points 5. and 6. show that the $\text{ToC}$ languages typically do not satisfy any of $\text{CE}$ or $\text{SE}$ unless $P = NP$ (because $t$ because the corresponding $C$ languages typically satisfy $\text{CL}$). Finally, since every propositional language satisfying $\text{CO}$ and $\text{CD}$ also satisfies $\text{CE}$ (a straightforward extension of Lemma 1.4 from [Darwiche and Marquis, 2002] to any propositional representation language), we get as a corollary to points 1. and 5. that:

**Corollary 1** If $C$ satisfies $\text{CO}$ and $\text{CL}$, then $\text{ToC}$ does not satisfy $\text{CD}$ unless $P = NP$.

Considering other transformations, we obtained the following results which hold for any propositional representation language (hence specifically for the $\text{ToC}$ ones):

**Proposition 2** Let $C$ be any propositional representation language.

1. If $C$ satisfies $\text{CO}$ and $\text{TE}$ and $C$ does not satisfy $\text{CE}$ unless $P = NP$, then $C$ does not satisfy $\land \text{BC}$ unless $P = NP$.
2. If $C$ satisfies $\text{VA}$ and $\text{TE}$, then $C$ does not satisfy $\lor \text{C}$ unless $P = NP$.
3. If $C$ satisfies $\text{IM}$ and does not satisfy $\text{CE}$ unless $P = NP$, then $C$ does not satisfy $\neg \text{C}$ unless $P = NP$.

These results show that the $\text{ToC}$ languages typically satisfy only few transformations among $\text{CD}$, $\land \text{BC}$, $\lor \text{C}$ and $\neg \text{C}$. The conditions on $C$ listed in Corollary 1 and Proposition 2 are indeed not very demanding.

It is interesting to note that the algorithms $\text{Conditioning}$, $\text{Project}$, $\text{IsCE}$, $\text{IsEQ}$ reported in [Subbarayan et al., 2007] (Figure 3), for respectively computing the conditioning of a $\text{ToOBDD}_<^<$ representation by a consistent term, computing the projection of a $\text{ToOBDD}_<^<$ representation $T$ on a given set $V$ of variables (or equivalently, forgetting all variables in $T$ except those of $V$), deciding whether a clause is entailed by a $\text{ToOBDD}_<^<$ representation, deciding whether two $\text{ToOBDD}_<^<$ representations are equivalent, apply to $\text{ToC}$ representations as well (the fact that each $B(n)$ of $T$ is an $\text{OBDD}_<^<$ representation is not mandatory for ensuring the correctness of these algorithms). While these algorithms do not run in polynomial time in the general case, imposing further restrictions on $C$ can be a way to achieve tractability. Thus, it is easy to show that if $C$ has a linear-time algorithm for $\text{FO}$ and a linear-time algorithm for $\land \text{BC}$, then $\text{Project}$ is a polytime $\text{FO}$ algorithm for the $\text{ToC}$ languages. If $C$ has a linear-time algorithm for $\text{FO}$, a linear-time algorithm for $\land \text{BC}$, and a polytime algorithm for $\text{CD}$, then $\text{Conditioning}$ is a polytime $\text{CD}$ algorithm for the $\text{ToC}$ languages.

The fact that many queries/transformations are $\text{NP}$-hard in the general case does not discard $\text{ToOBDD}_<^<$ (and beyond it the $\text{ToC}$ languages) as interesting target languages for $\text{KC}$ from the practical side. Indeed, if the width of a $\text{ToC}$ representation $T$, i.e., $\max_{n \in N} \{|\text{Var}(n)|-1\}$, is (upper) bounded by a constant, then the time complexity of the $\text{Propagate}$ function becomes linear in the tree size; as a consequence, many other queries and transformations may become tractable as well; for instance if $C$ satisfies $\text{CD}$, we get that both conditioning and clausal entailment can be achieved in polynomial time in the tree size.

As to succinctness, we got the following results:

**Proposition 3** Let $C$ be any complete propositional representation language.

1. $\text{ToC} \leq_P C$.
2. Let $C'$ be any complete propositional fragment. If $C \leq_s C'$, then $\text{ToC} \leq_s \text{ToC'}$.
3. If $C$ satisfies $\text{CL}$ and $C'$ satisfies $\text{CE}$, then $C' \leq_s^* \text{ToC}$.
4. If $C$ satisfies $\text{IM}$, then $\text{ToC} \leq_s^* \text{DNF}$.

Proposition 3 has many interesting consequences:

- From point 1., we directly get that $\text{ToC} \leq_s C$, and that $\text{ToC}$ is complete (since $C$ is). This result cannot be strengthened to $\text{ToC} \leq_s C$ in the general case (for every $C$ satisfying $\land \text{C}$, e.g. $C = \text{CNF}$, we can prove that $C \not\leq_P \text{ToC}$).

\footnote{See [Marquis, 2008] for more details on this issue.}

\footnote{The price to be paid by such a restriction is a lack of expressiveness: none of the languages of $\text{ToC}$ representations of width bounded by $c$ (where $c$ is a parameter) is complete.}
• Point 2. allows one to take advantage of previous results describing how propositional languages \( C \) are organized w.r.t. spatial efficiency in order to achieve similar results for the corresponding ToC languages.

• Point 3. implies that the DNNF language, which satisfies CE, is typically (i.e., whenever \( C \) satisfies CL) not more succinct than the corresponding ToC language; hence none of the languages \( C \) which are less succinct than DNNF (e.g., \( C = \text{DNF} \)) can be more succinct than such ToC languages; thus, we get for instance that DNNF \( \not\preceq^*_{\text{T}} \) ToDNF (which together with point 1. shows that ToDNF \( \not\preceq^*_{\text{T}} \text{DNF} \)).

• Another consequence of point 3. is that if \( C \) satisfies CL then DNNF \( \not\preceq^*_{\text{T}} \text{ToC} \) (hence \( d\text{-DNNF} \not\preceq^*_{\text{T}} \text{ToC} \)). With point 1. this shows ToDNNF to be spatially (strictly) more efficient than DNNF, while keeping CO and ME.

Finally, an interesting issue is to determine whether, at the "instance level", i.e., considering a given Boolean function to be compiled, targeting ToC in a compilation process leads always to save space w.r.t. targeting \( C \). The answer is "not always" (even in the cases when we have ToC \( \not\preceq^*_{\text{T}} C \)). We showed it by considering the notion of decomposition set:

**Definition 10 (decomposition)** Let \( f \) be a Boolean function. Let \( V_1, \ldots, V_k \) be \( k \) subsets of \( PS \). \( D = \{V_1, \ldots, V_k\} \) is a decomposition set for \( f \) iff we have \( f = \bigwedge_{i=1}^{k} \exists V_i.f \).

Clearly enough, for each ToC representation \( T \) whose set of nodes is \( N \), \( \{\forall n_i \mid n_i \in N\} \) is a decomposition set for \( T \).

**Lemma 1** Let \( f \) be a Boolean function. Let \( \delta \) be an essential prime implicate of \( f \), i.e., a prime implicate of \( f \) which is not implied by the conjunction of the other prime implicates of \( f \). Then for every decomposition set \( D \) for \( f \), there exists \( V \in D \) such that \( \forall \delta \subseteq V \).

This lemma shows that when \( f \) has an essential prime implicate containing all its variables, no ToC representation of \( f \) can be more compact than each of its \( C \) representations. This lemma also shows that when \( f \) has an essential prime implicate \( \delta \) such that \( \exists \forall \delta.f \) has no \( C \) representation of reasonable size, choosing ToC as the target language is not a way to save space.

Finally, Lemma 1 also explains why imposing a fixed decomposition tree \( T \) for defining a ToC language is not so a good idea (despite the fact it may offer a property of canonicity in some cases): either \( T \) has a node \( n \) such that \( \forall \delta.n = \{x_1, \ldots, x_p\} \) (all the variables of interest), and in this case the corresponding ToC language mainly amounts to \( C \), or \( T \) does not contain such a node, and in this case the ToC language is incomplete: the Boolean function which is the semantics of the clause \( \forall_{i=1}^{p} x_i \) cannot be represented in ToC.

4 Back to ToOBDD< Representations

Let us now fix \( C \) to \( \text{OBDD}_< \) in order to get some further results. Beyond ToOBDD< we have investigated the properties of \( U(\text{ToOBDD}_<) \) (the union of all \( \text{ToOBDD}_< \) for each total order \( < \) over \( PS \)) and of ToOBDD, as target languages for propositional knowledge compilation, along the lines of the KC map. To make the differences between these languages clearer, observe that OBDD representations \( B(n), n \in N \) where \( N \) is the set of nodes of a given ToOBDD \( T \) may rely on different variable orders \( < \), while all the \( \text{OBDD}_< \) representations in a given \( U(\text{ToOBDD}_<) \) are based on the same order. Hence, \( U(\text{ToOBDD}_<) \) is a proper subset of ToOBDD.

**Proposition 4** The results in Table 1 hold.

The fact that ToOBDD, \( U(\text{ToOBDD}_<) \), and ToOBDD< satisfy CO, VA, IM, and ME and that none of these languages satisfies any of CE, SE, CD, \( \Lambda \text{BC} \), \( \Lambda \text{VC} \), \( \Lambda \text{C} \), or \( \Lambda \text{ME} \), unless \( P = \text{NP} \) is a direct corollary of Propositions 1 and 2. Except CO and VA, all those results concern some issues left open in [Subbarayan et al., 2007]. Especially, there exist polynomial algorithms for IM and ME which are not based on the message-passing propagation algorithm (those given in [Subbarayan et al., 2007]) do not run in polynomial time in the general case. Furthermore, contrary to what was expected in [Subbarayan et al., 2007], \( \Lambda \text{C} \) is not trivial: the negation of a conjunction of \( \text{OBDD}_< \) representations is equivalent to the disjunction of their negations. We actually showed that the \( \Lambda \text{C} \) transformation on ToOBDD< cannot be achieved in polynomial time unless \( P = \text{NP} \).

As to succinctness, we proved the following results:

**Proposition 5**

1. For each \( < \), ToOBDD< \( \not\preceq^*_{\text{T}} \text{OBDD}_< \).
2. For each \( < \), DNNF \( \not\preceq^*_{\text{T}} \text{ToOBDD}_< \).
3. ToOBDD \( \not\preceq^*_{\text{T}} \text{DNF} \).
4. ToOBDD \( \not\preceq^*_{\text{T}} \text{CNF} \).

Points 1. to 3. are direct consequences of Proposition 3 and results from [Darwiche and Marquis, 2002]. A direct consequence of Proposition 5 is that \( d\text{-DNNF} \not\preceq^*_{\text{T}} \text{ToOBDD}_< \). This explains in some sense the space savings which can be offered by ToOBDD< over \( d\text{-DNNF} \) and observed empirically as reported in [Subbarayan et al., 2007]. More generally, from Proposition 3 and some results given in [Darwiche and Marquis, 2002] we get that:

**Corollary 2** Unless PH collapses, ToOBDD, \( U(\text{ToOBDD}_<) \) and ToOBDD< are incomparable w.r.t. succinctness with the languages CNF, DNF, and DNNF.

<table>
<thead>
<tr>
<th>CE</th>
<th>VA</th>
<th>CO</th>
<th>IM</th>
<th>EQ</th>
<th>SE</th>
<th>CI</th>
<th>ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>?</td>
<td>0</td>
<td>?</td>
<td>√</td>
</tr>
<tr>
<td>ToOBDD&lt;</td>
<td>0</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>?</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>ToOBDD&lt;</td>
<td>0</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>?</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>OBDD&lt;</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>?</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>OBDD&lt;</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>?</td>
<td>0</td>
<td>?</td>
</tr>
</tbody>
</table>

Table 1: \( √ \) means “satisfies”, \( ∗ \) means “does not satisfy”, \( \circ \) means “does not satisfy unless \( P = \text{NP} \)”, and \( \ast \) means “does not satisfy unless \( \text{PH collapses} \).” Results for \( \text{OBDD}_< \) and \( \text{OBDD} \) are from [Darwiche and Marquis, 2002] and are given here as a baseline.
5 Conclusion

In this paper, the concept of tree-of-BDDs has been extended to any complete propositional representation language $\mathcal{C}$ thus leading to the family of ToC languages. A number of generic results are provided, which allow to determine the queries/transformations satisfied by ToC depending on the ones satisfied by $\mathcal{C}$, as well as results about the spatial efficiency of the ToC languages. Focusing on the ToOBDD$_C$ language, we have addressed a number of issues that remained open in [Subbarayan et al., 2007]; especially, we have shown that beyond CO and VA, ToOBDD$_C$ satisfies IM and ME but does not satisfy any query among CE, SE. We have also proved that ToOBDD$_C$ does not satisfy any transformation among CD, FO, $\land BC$, $\forall C$ or $\neg C$ and that this fragment is not comparable for succinctness w.r.t. any of CNF, DNF and DNNF unless PH collapses.

From this investigation, it turns out that the ToOBDD$_C$ language (and in general the ToC languages) satisfies only few queries and transformations. Subsequently, in applications where some queries/transformations not satisfied by ToOBDD$_C$ must be achieved under some guaranteed response time, considering ToOBDD$_C$ as a target language for KC is not always the best choice. From the practical side, as reported in [Subbarayan et al., 2007] (and despite the fact that ToOBDD$_C$ $\not\preceq^*_{d}$ CNF, there are CNF formulae which can be compiled into ToOBDD$_C$ using a reasonable amount of computational resources, while it turned out impossible to generate $d$-DNNF representations for them. Such empirical results cohere with our succinctness result $d$-DNNF $\not\preceq^*_{d}$ ToOBDD$_C$. Nevertheless, our result ToOBDD $\not\preceq^*_{d}$ DNNF shows that this empirical evidence can be argued (this result implies that some DNNF representations do not have "small" ToOBDD$_C$ equivalent representations under the standard assumptions of complexity theory), so DNNF remains a very attractive language for the KC purpose.

Our results also suggest a number of ToC languages as quite promising. Consider for instance the ToFBDD$_C$ language. From our results, it comes easily that ToFBDD$_C$ satisfies CO, VA, IM, ME (hence the same queries as ToOBDD$_C$); since ToFBDD$_C$ is at least as succinct as ToOBDD$_C$, it appears as a challenging fragment. Furthermore, a compiler to FBDD is already available (see e.g. www.eecg.utoronto.ca/~jzhu/fbdduser11.ps). When none of VA or IM is expected, the ToDNNF language looks also valuable; indeed, from our results we know that ToDNNF satisfies CO and ME, while being quite compact: ToDNNF $\preceq^*_{d}$ ToOBDD$_C$ and ToDNNF $\preceq^*_{d}$ DNNF hold; beyond the spatial dimension, targeting the ToDNNF language may also reduce the on-line computation time needed for achieving queries/transformations based on Propagate function (as well as the off-line CNF-to-ToC compilation time) since DNNF satisfies FO, which is one of the two key operations of the propagation algorithm. The ToDNNF$_T$ language, based on DNNF$_T$ [Pipatsrisawat and Darwiche, 2008], also looks interesting in this respect since it satisfies both FO and $\land BC$, the other key operation of the propagation algorithm.

This is what the “theory” says in some sense about such languages. Going further requires to implement compilers and perform experiments in order to determine whether, from the practical side, representations from those languages can be computed using a reasonable amount of resources. This is an issue for further research. Another perspective for further work is to complete the missing results about queries, transformations and succinctness for the ToC languages and to extend the KC map accordingly. Especially, it would be interesting to characterize some families of propositional formulae each of DNNF and ToOBDD are “effective” on.

Appendix

Proof: [Proposition 1]

1. Proposition 3 shows that $\mathcal{C} \supseteq \mathcal{P}$ ToC. Hence, it holds that if ToC satisfies CO, then $\mathcal{C}$ satisfies CO. Conversely, a formula $\Sigma$ is inconsistent iff whatever the set of variables $V$, we have that $\exists V \Sigma$ is inconsistent. Accordingly, a ToC representation represents an inconsistent formula $\Sigma$ iff whatever $n \in N$, the $\mathcal{C}$ representation $B(n)$ is inconsistent. The fact that $\mathcal{C}$ satisfies CO allows us to conclude the proof.

2., 3. From $\mathcal{C} \supseteq \mathcal{P}$ ToC (Proposition 3), we get that if ToC satisfies VA (resp. IM), then $\mathcal{C}$ satisfies VA (resp. IM). Conversely, let $T$ be a ToC representation of a formula $\Sigma$. We have $\Sigma \equiv \Sigma(T) \equiv \bigwedge_{n \in N} B(n)$. So $\Sigma(T)$ is valid iff each $\mathcal{C}$ formula $B(n)$ for $n \in N$ is valid. Furthermore, a consistent term $\gamma$ implies $\Sigma(T)$ iff $\gamma$ implies each $\mathcal{C}$ formula $B(n)$ for $n \in N$. The fact that $\mathcal{C}$ satisfies VA (resp. IM) is enough to conclude the proof.

4. Again, from $\mathcal{C} \supseteq \mathcal{P}$ ToC (Proposition 3), we get that if ToC satisfies ME, then $\mathcal{C}$ satisfies ME. Conversely, we can design an output-polynomial time algorithm for computing the models of a ToC representation $T$ over $Var(T)$. This algorithm is a recursive algorithm taking as inputs a ToC representation $T$ and a term $\gamma$ such as $Var(\gamma) \subseteq Vars(n_0)$ (where $n_0$ is the root of $T$). Initially, $\gamma$ is the empty term and the current node $n_0$. $B(n)$ is conditioned by $\gamma$ (in polynomial time, since $\mathcal{C}$ satisfies CD). The algorithm then computes the models $\omega$ of the conditioning $B(n) | \gamma$ of $B(n)$ by $\gamma$ (in output-polynomial time, since $\mathcal{C}$ satisfies ME). If $n$ is a leaf node, then the algorithm returns this set of models. Otherwise, for each model $\omega B(n) | \gamma$, the procedure is recursively called on each of the children of $n$ (which is itself the root of ToC representations) with input $\gamma = \omega$. This returns as many sets of models $\Omega_i$ as the number of children of $n$.

5. By reduction from CNF-SAT. Let $\alpha = \bigwedge_{i=1}^{k} \delta_i$ be a CNF formula such that $Var(\alpha) = \{x_1, \ldots, x_p\}$. Let $\alpha' = \bigwedge_{i=1}^{k} \neg \text{holds}_i \lor \delta_i$ be a CNF formula such that
Proof:[ Proposition 2]
1. \( \forall \alpha \beta \gamma \). Towards a contradiction. Assume that \( \alpha \) satisfies \( \alpha \beta \gamma \). Then since it satisfies \( \alpha \beta \gamma \) TE, one can compute in polynomial time a \( \alpha \) representation of \( T \land \gamma \) from a \( \alpha \) representation \( T \) and a term \( \gamma \), and one can decide in polynomial time whether it is consistent since \( \alpha \) satisfies \( \alpha \beta \gamma \) CO. But a non-valid clause \( \delta \) is an implicate of \( \alpha' \) iff \( T \land \gamma \) is inconsistent since \( \alpha \) satisfies \( \alpha \beta \gamma \) CO. Hence by construction, every clause \( \delta \) which is a logical consequence of \( \Sigma \) holds iff \( \delta \) is consistent. This completes the proof.

By reduction from \( \text{CNF-SAT} \). Consider again the reduction given in point 5. of Proposition 1. We have that \( \alpha \) is consistent iff \( \alpha' \neq \delta \) iff \( \Sigma(T) \neq \Sigma \). This also equivalent to \( \Sigma(T) \land \neg \alpha' \). The fact that \( \alpha \) satisfies \( \alpha \beta \gamma \) CO and \( \alpha \beta \gamma \) TE completes the proof.

2. \( \forall \alpha \). When \( \alpha \) satisfies \( \alpha \beta \gamma \) TE, every term \( \gamma \) has a \( \alpha \) representation of size polynomial in the size of \( \gamma \). Since \( \text{DNF} \) does not satisfy \( \alpha \beta \gamma \) VA unless \( P = NP \), if \( \alpha \) satisfies it, it cannot be the case that \( \alpha \) satisfies \( \alpha \beta \gamma \) VC, \( \alpha \beta \gamma \) VE, \( \alpha \beta \gamma \) IM, and \( \alpha \beta \gamma \) IM concludes the proof.

Proof:[ Proposition 3]
1. Every \( \alpha \beta \gamma \) formula \( \Sigma \) can associated in linear time to its \( \alpha \beta \gamma \) representation \( T \) such that \( T \) has a single node \( n_0 \), \( V \alpha(n_0) = V \alpha(S) \) and \( B(n_0) = \alpha \) is consructed to \( \Sigma \).

2. To every \( \alpha \beta \gamma \) representation \( T' \) of a \( \alpha \beta \gamma \) formula \( \Sigma \), we can associate via a poly-size function a \( \alpha \beta \gamma \) representation \( T \) of \( \Sigma \). The tree \( T \) coincides with \( T' \), except that each node \( n \) of \( T \) is labelled by a \( \alpha \) formula \( B(n) \) equivalent to \( \alpha \beta \gamma \) formula \( B' \) labelling \( n \) in \( T' \). Indeed, since \( \alpha \beta \gamma \) CE, there exists a poly-size function \( f \) such that for each \( B(n) \) can be computed via \( f \) from \( B' \).

3. The proof is close to the one used to show that \( \alpha \beta \gamma \) does not satisfy \( \alpha \beta \gamma \) CE, unless \( P = NP \) when \( \alpha \beta \gamma \) satisfies \( \alpha \beta \gamma \) CL (point 5. of Proposition 1). Let \( p \) be any non-negative integer. Let \( \Sigma_{\max} \) be the \( \alpha \beta \gamma \) CNF formula

\[
\bigwedge_{\delta \in 3-C_n} \neg \text{holds}_i \lor \delta_i
\]

where \( 3 - C_n \) is the set of all 3-clausal that can be generated from \( \{x_1, \ldots, x_p\} \) and the \( \text{holds}_i \) are new variables, not among \( x_1, \ldots, x_p \). The size \( |\Sigma_{\max}^p| \) of \( \Sigma_{\max}^p \) is \( P^p \) since \( \Sigma_{\max}^p \) contains \( O(p^{3}) \)-clausal clauses. Especially, the number \( k \) of clauses of \( \Sigma_{\max}^p \) is cubic in \( p \). Now, we associate to \( \Sigma_{\max}^p \) in polynomial time the following \( \alpha \beta \gamma \) representation \( T \) of it: the set of nodes of \( T \) is \( N = \{ n_0, n_1, \ldots, n_k \} \), \( n_0 \) is the root of \( T \) and the \( k \) remaining nodes are the \( k \) children of \( n_0 \); of course, we have \( V \alpha(n_0) = \{ x_1, \ldots, x_p \} \) and for every \( i \in 1, \ldots, k \), \( V \alpha(n_i) = \{ \text{holds}_i \} \lor V \alpha(\delta_i) \). Now, by construction, every clause \( \delta \) which is a logical consequence of \( \Sigma_{\max}^p \) contains one of the \( \text{holds}_i \) as a literal. Hence forgetting every \( \text{holds}_i \) in \( \Sigma_{\max}^p \) leads to a valid formula, and forgetting all the \( \text{holds}_i \) in \( \Sigma_{\max}^p \) except one, say \( \text{holds}_{j} \), leads to a formula equivalent to the clause \( \neg \text{holds}_{j} \lor \delta_i \). We can compute in time polynomial in the size of \( \Sigma_{\max}^p \) the remaining labels \( B(n_0) = 1 \) and \( B(n_i) \) for \( i \in 1k \) as the \( \alpha \beta \gamma \) representation of the clause \( \neg \text{holds}_i \lor \delta_i \), since \( \alpha \) satisfies \( \alpha \beta \gamma \) CL.

Each 3-CNFS formula \( \alpha \beta \gamma \) built up from the set of variables \( \{ x_1, \ldots, x_p \} \) is in bijection with the subset \( S_{\alpha \beta \gamma} \) of the variables \( \alpha \beta \gamma \) s.t. \( \delta_i \) is a clause of \( \alpha \beta \gamma \) if and only if \( \text{holds}_i \in S_{\alpha \beta \gamma} \) (each variable \( \text{holds}_i \) can be viewed as the name of the clause where it appears, hence selecting a clause just amounts to select its name). Let \( \delta_{\alpha \beta \gamma} \) be the clause

\[
\bigvee_{\text{holds}_i \in S_{\alpha \beta \gamma}} \neg \text{holds}_i.
\]

It is easy to check that \( \delta_{\alpha \beta \gamma} \) is inconsistent iff

\[
\Sigma_{\max}^p \models \delta_{\alpha \beta \gamma}.
\]

Suppose now that \( \alpha \beta \gamma \) is at least at succinct as \( \alpha \beta \gamma \). Then using a poly-size compilation function \( comp \) we could associate to \( T \) an equivalent \( \alpha \beta \gamma \) formula \( comp(T) \). Since \( \alpha \beta \gamma \) satisfies \( \alpha \beta \gamma \) CE, for every clause \( \delta \), determining whether \( \delta \) can be done in deterministic polynomial time. Then we would be able to determine whether any 3-CNFS formula \( \alpha \beta \gamma \) is satisfiable using a deterministic Turing machine with a polynomial advice \( A \): if
|Var(α)| = p, then the machine loads
\[ A(n) = \text{comp}(T). \]

Once this is done, it determines whether \( \delta_\alpha \) is entailed by \( \text{comp}(T) \), which is in \( P \). Since 3-SAT is complete for \( NP \), this would imply \( NP \subseteq P/\text{poly} \), and, as a consequence, the polynomial hierarchy would collapse at the second level.

4. This result is in some sense dual to point 3. Let \( p \) be any non-negative integer. Let \( \Sigma_p^{\text{max}} \) be the \( \text{CNF} \) formula
\[ \bigwedge_{\delta_i \in 3-C_p} \neg \text{holds}_i \lor \delta_i \]
where \( 3-C_n \) is the set of all 3-literal clauses that can be generated from \( \{x_1, \ldots, x_p\} \) and the \( \text{holds}_i \) are new variables, not among \( x_1, \ldots, x_p \). The size \( |\Sigma_p^{\text{max}}| \) of \( \Sigma_p^{\text{max}} \) is in \( O(p^3) \) since \( \Sigma_p^{\text{max}} \) contains \( O(p^3) \) 4-literal clauses. Especially, the number \( k \) of clauses of \( \Sigma_p^{\text{max}} \) is cubic in \( p \). Obviously each, 3-CNF formula \( \alpha_p \) built up from the set of variables \( \{x_1, \ldots, x_p\} \) is in bijection with the subset \( S_{\alpha_p} \) of the variables \( \text{holds}_i \) s.t. \( \delta_i \) is a clause of \( \alpha_p \) if and only if \( \text{holds}_i \in S_{\alpha_p} \) (each variable \( \text{holds}_i \) can be viewed as the name of the clause where it appears, hence selecting a clause just amounts to selecting its name). Let \( \delta_{\alpha_p} \) be the clause
\[ \bigvee_{\text{holds}_i \in S_{\alpha_p}} \neg \text{holds}_i. \]

It is easy to check that \( \alpha_p \) is inconsistent if and only if \( \Sigma_p^{\text{max}} \models \neg \delta_{\alpha_p} \) if and only if \( \neg \delta_{\alpha_p} \models \neg \Sigma_p^{\text{max}} \).

Suppose now that \( \text{Toc} \) is at least as succinct as \( \text{DNF} \). Then using a polytime compilation function \( \text{comp} \) we could associate to the \( \text{DNF} \) formula \( \neg \Sigma_p^{\text{max}} \) an equivalent \( \text{Toc} \) representation \( T \). If \( \text{Toc} \) satisfies \( \text{IM} \), then from point 3, of Proposition 1, \( \text{Toc} \) satisfies \( \text{IM} \) as well. So, for every term \( \gamma \), determining whether \( \gamma \) is an implicate of the formula represented by \( T \) can be done in (deterministic) polynomial time. Then we would be able to determine whether any 3-CNF formula \( \alpha \) is satisfiable using a deterministical Turing machine with a polynomial advice \( A \): if \(|\text{Var}(\alpha)| = p \), then the machine loads
\[ A(n) = T. \]

Once this is done, it determines whether \( \gamma_\alpha = \neg \delta_\alpha \) is an implicant of \( T \), which is in \( P \). Since 3-SAT is complete for \( NP \), this would imply \( NP \subseteq P/\text{poly} \), and, as a consequence, the polynomial hierarchy would collapse at the second level.

Proof: [Lemma 1] Towards a contradiction, assume that there exists a decomposition set \( D = \{V_1, \ldots, V_k\} \) for \( \Sigma \) such that for each \( i \in 1 \ldots k \), \( \text{Var}(\delta) \not\subseteq V_i \). Since for each \( i \in 1 \ldots k \), \( PI(\exists V_i, \Sigma) = \{\delta' \in PI(\Sigma) \mid \text{Var}(\delta') \subseteq V_i\} \), we have that \( \delta \not\in \bigcup_{i=1}^k PI(\exists V_i, \Sigma) \). Since \( \bigcup_{i=1}^k PI(\exists V_i, \Sigma) \) is a subset of \( PI(\Sigma) \) not containing \( \delta \) and since the conjunction of all prime implicates of \( \Sigma \) except \( \delta \) does not imply \( \delta \), by

monotony of \( \models \), we get that \( \bigwedge_{i=1}^k PI(\exists V_i, \Sigma) \not\models \delta \). However, \( \bigwedge_{i=1}^k PI(\exists V_i, \Sigma) \models \Sigma \) whenever \( \{V_1, \ldots, V_k\} \) is a decomposition set for \( \Sigma \), so we must also have \( \bigwedge_{i=1}^k PI(\exists V_i, \Sigma) \models \delta \) since \( \delta \) is an implicate of \( \Sigma \), contradiction.

Proof: [Proposition 4]
- None of \( \text{ToOBDD}_< \) (whatever \(<\) \( \text{ToOBDD}_< \)) satisfies \( \text{FO} \) unless \( \text{PH} \) collapses.

The proof is similar to the one used to show that \( \text{OBDD}_< \) does not satisfy \( \text{FO} \) [Darwiche and Marquis, 2002]. To any \( \text{DNF} \) formula \( \Sigma = \bigvee_{i=1}^k \gamma_i \), we associate in polynomial time the \( \text{ToOBDD}_< \) representation \( T \) such that the set of nodes of \( T \) is \( \{n_0\} \) and \( B(n_0) \) is the \( \text{OBDD}_< \) formula equivalent to \( \Delta^1 \) inductively defined by:

- \( \Delta^k = \text{OBDD}_<(\gamma_k) \), i.e. the \( \text{OBDD}_< \) formula equivalent to the term \( \gamma_k \).
- \( \Delta^1 \) is the \( \text{OBDD}_< \) of the form \( (\text{new}_i \wedge \text{OBDD}_<(\gamma_i)) \vee (\neg \text{new}_i \wedge \Delta^{i+1}) \) for each \( i \in 1 \ldots k - 1 \).

Here each \( \text{new}_i \) \((i \in 1 \ldots k)\) is a fresh variable not occurring in \( \Sigma \). We assume that the \( \text{new}_i \) variables \((i \in 1 \ldots k)\) are earlier than all other variables w.r.t. \(<\). We have \( \Sigma \equiv \exists(\text{new}_1, \ldots, \text{new}_k)(\Sigma(T)) \). Obviously, \( T \) also is a \( \text{U(ToOBDD)_<} \) representation of \( \Sigma \).

If \( \text{ToOBDD}_< \) (resp. \( \text{U(ToOBDD)_<} \)) satisfies \( \text{FO} \), then we can convert in polynomial time the \( \text{DNF} \) formula \( \Sigma \) into an equivalent \( \text{U(ToOBDD)_<} \) representation \( T \). Since \( \text{U(ToOBDD)_<} \) \( \not\subseteq \), \( \text{DNF} \) unless \( \text{PH} \) collapses, this would make \( \text{PH} \) to collapse.

Proof: [Proposition 5] Points 1. to 3. are direct consequences of Proposition 3.

5. From the inclusion \( \text{ToOBDD}_< \subseteq \text{U(ToOBDD)_<} \) (whatever \(<\)), we get immediately that \( \text{U(ToOBDD)_<} \subseteq \text{ToOBDD}_< \) (whatever \(<\)).

6. Consider the formula
\[ \Sigma = (\bigwedge_{i=1}^m \bigvee_{j=1}^m \neg x_{i,j}) \wedge (\bigwedge_{j=1}^m \bigvee_{i=1}^m \neg x_{i,j}) \wedge (\bigvee_{i=1}^m \bigvee_{j=1}^m x_{i,j}) \]
over \( m^2 \) variables with \( m \) > 1. Every resolvent from a pair of clauses from \( \Sigma \) is a valid clause and no clause from \( \Sigma \) is implied by another clause from \( \Sigma \). Hence, the correctness of any resolution-based algorithm for computing prime implicates (like Tison’s one [Tison, 1967]) ensures that \( \Sigma \) is a \( \text{PI} \) formula (hence a \( \text{CNF} \) formula).

Since \( (\bigwedge_{i=1}^m \bigvee_{j=1}^m \neg x_{i,j}) \wedge (\bigwedge_{j=1}^m \bigvee_{i=1}^m \neg x_{i,j}) \) is consistent and negative (i.e., it contains only negative literals), no positive clause like \( \bigvee_{i=1}^m \bigvee_{j=1}^m x_{i,j} \) can be a logical consequence of it. Hence \( \bigwedge_{i=1}^m \bigvee_{j=1}^m x_{i,j} \) is an essential prime implicate of \( \Sigma \). From Lemma 1, we get that for every decomposition set \( D \) for \( \Sigma \), there exists \( V \in D \) such that \( \text{Var}(\bigvee_{i=1}^m \bigvee_{j=1}^m x_{i,j}) \subseteq V \).

This shows that in every \( \text{ToOBDD}_< \) representation \( T \) of \( \Sigma \) there is a node \( n \) such that \( \text{Var}(n) = \text{Var}(\Sigma) \). What remains to be shown is that the size of
the OBDD formula $B(n) \equiv \exists \text{Var}(\Sigma). \Sigma \equiv \Sigma$ is not polynomial in the size of $\Sigma$. To this end, we consider the proof of Lemma 3.5 from [Horiyama and Ibaraki, 2002], showing that every OBDD formula equivalent to $(\bigwedge_{i=1}^{m}(\bigvee_{j=1}^{m} \neg x_{i,j}) \land (\bigvee_{j=1}^{m}(\bigwedge_{i=1}^{m} \neg x_{i,j}))$ has a size in $\Omega(2^{m^2})$. Interestingly, the proof still applies to our formula $\Sigma$ (the proposed fooling set $A$ containing $2^{m^2}$ assignments also is a fooling set when $\Sigma$ is considered instead of $(\bigwedge_{i=1}^{m}(\bigvee_{j=1}^{m} \neg x_{i,j}) \land (\bigvee_{j=1}^{m}(\bigwedge_{i=1}^{m} \neg x_{i,j})))$).

**Proof:** [Corollary 2] TBD.

**References**


