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 DNF, 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, A BC, ∨ 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.
2 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 $\{0, 1\}^X$ to $\{0, 1\}$, where $X \subseteq PS$. $Var(f) = X$ is called the scope of $f$. The support $\Omega(f)$ of $f$ is the set of all assignments $\omega$ of $Var(f)$ to Boolean values such that $f(\omega) = 1$. For any $X \subseteq PS$, we note by $\overline{X}$ the set $PS \setminus X$. The set of Boolean functions is equipped with the three standard internal laws, $\land$, $\lor$ and $\neg$. Given $X \subseteq PS$ we note by $\exists f$, the Boolean function with scope $\overline{Var(f)} \setminus X$ that maps 1 to an assignment $\omega_{\overline{Var(f)}} \setminus X$ of $\overline{Var(f)} \setminus X$ iff there exists an assignment $\omega$ of $\overline{Var(f)}$ such that the restriction of $\omega$ over $\overline{Var(f)} \setminus X$ and $\omega_{\overline{Var(f)}} \setminus X$ coincide and $f(\omega) = 1$.

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

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

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, when $\Sigma$ is a propositional formula representing a Boolean function $g$ one often says that a representation $\alpha$ of $g$ is a representation of $\Sigma$ instead of $\alpha$ is a representation of the semantics of $\Sigma$.

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

\footnote{\cite{I.Gogic et al., 1995} refers to the interpretation function associated to the $C$ language, so that $I_C$ 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.

DAG-NNF fragments DNNF, d-DNNF, CNF, DNF, FBDD, 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 $\omega$ of $\overline{Var(\alpha)}$ to Boolean values is said to be a model of a $C$ representation $\alpha$ over $\overline{Var(\alpha)}$ iff $I(\alpha)(\omega) = 1$. Similarly, two representations $\alpha$ and $\beta$ (possibly from different representation formalisms) are said to be equivalent, noted $\alpha \equiv \beta$, when they represent the same Boolean function. A $C$ representation $\alpha$ is consistent (resp. valid) iff $\alpha$ does not represent the Boolean function $0$ (resp. represents the Boolean function $1$). $\alpha$ is a logical consequence of $\beta$ noted $\beta \models \alpha$, iff $\Omega(I(\beta)) \subseteq \Omega(I(\alpha))$.

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

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

- $C$ satisfies CD iff there exists a polytime algorithm that maps every $C$ representation $\alpha$ and every consistent term $\gamma$ to a $C$ representation $\beta$ of the restriction of $I(\alpha)$ to $I(\gamma)$, i.e., $\overline{Var(\beta)} = \overline{Var(\alpha)} \setminus \overline{Var(\gamma)}$ and $I(\beta) = \exists \overline{Var(\gamma)}.(I(\alpha) \land I(\gamma))$.
- $C$ satisfies FO iff there exists a polytime algorithm that maps every $C$ representation $\alpha$ and every subset $X$ of variables from $PS$ to a $C$ representation of $\exists X. I(\alpha)$. 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...
α₁, . . . , αₙ to a C representation of I(α₁) ∧ · · · ∧ I(αₙ) (resp. I(α₁) ∨ · · · ∨ I(αₙ)).

• C satisfies ∧BC (resp. ∨BC) iff there exists a polynome

algorithm that maps every pair of C representations α

 and β to a C representation of I(α) ∧ I(β) (resp. I(α) ∨ I(β)).

• C satisfies ¬C iff there exists a polynome algorithm that

maps every C representation α to a C representation of

¬I(α).

Definition 5 (succinctness) Let C₁ and C₂ be two representation languages. C₁ is at least as succinct as C₂, noted C₁ ≤ₛ C₂, if there exists a polynomial p such that for every C₂ representation α there exists an equivalent C₁ representation β where |β| ≤ p(|α|).

~ₘ is the symmetric part of ≤ₘ defined by C₁ ~ₘ C₂ if C₁ ≤ₘ C₂ and C₂ ≤ₘ C₁. <ₘ is the asymmetric part of ≤ₘ defined by C₁ <ₘ C₂ iff C₁ ≤ₘ C₂ and C₂ ≤ₘ C₁. Finally, C₁ ≤ₘ* C₂ (resp. C₁ <ₘ* C₂) means that C₁ ≤ₘ C₂ (resp. C₁ <ₘ C₂) unless the polynomial hierarchy PH collapses (which is considered very unlikely in complexity theory).

We also consider the following restriction of the succinctness relation:

Definition 6 (polynomial translation) Let C₁ and C₂ be two representation languages. C₁ is polynomially translatable into C₂, noted C₁ ≥ₚ C₂, if there exists a polynome algorithm A such that for every C₁ representation α A(α) is a C₂ representation such that A(α) ≡ α.

Like ≥ₘ, ≥ₚ is a preorder (i.e., a reflexive and transitive relation) over propositional representation languages. It refines the spatial efficiency preorder ≥ₘ in the sense that for any C₁ and C₂, if C₁ ≥ₚ C₂, then C₁ ≥ₘ C₂ (but the converse does not hold in general). We note by ¬ₚ the symmetric part of ≥ₚ.

3 The ToC Languages

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

Definition 7 (tree-of-BDDs)

• A decomposition tree of a CNF formula Σ is a (finite) labelled tree T whose set of nodes is N. Each node n ∈ N is labelled with Var(n), a subset of Var(Σ). For each n ∈ N, let clauses(n) = {clause δ of Σ s.t. Var(δ) ⊆ Var(n)}; T satisfies two conditions: for every clause δ of Σ there exists n ∈ N such that δ ∈ clauses(n), and for every x ∈ Var(Σ), {n ∈ N | x ∈ Var(n)} forms a connected subtree of T.

• Let < be a total strict ordering over PS. A tree-of-BDDs of a CNF formula Σ given < consists of a decomposition tree T of Σ equipped with a further labelling function B such that for every n ∈ N, B(n) is the OBDD< representation of 3Var(n).I(Σ).

We have Var(T) = 3n∈N Var(n) and I(T) = 3n∈N I(B(n)). ToB denotes the set of all trees-of-BDDs given <.

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

The above definition can be simplified and extended, allowing the representation of other formulae than CNF ones, and taking advantage of other target languages than OBDD< for compiling the labels B(n):

Definition 8 (ToC) Let C be any complete propositional representation language. A ToC representation is a finite, labelled tree T, whose set of nodes is N. Each node n ∈ N is labelled with Var(n), a subset of PS and with a C representation B(n).

T must satisfy:

• the running intersection property: for each x ∈ ∪n∈N Var(n), {n ∈ N | x ∈ Var(n)} forms a connected subtree of T, and

• the global consistency property: for each n ∈ N, I(B(n)) = 3Var ar(x).3n∈N I(B(n)).

We have Var(T) = 3n∈N Var(n) and I(T) = 3n∈N I(B(n)). The size of a ToC representation T is the size of this tree, plus the sizes of the labels of the nodes of T (numbers of variables in Var(n) and sizes of B(n)).

ToC denotes the set of all ToC representations.

Taking C = OBDD<, we get the ToOBDD< language. Clearly, this definition of ToOBDD< is close to the previous one ToB from [Subbarayan et al., 2007], except that a ToOBDD< representation T is defined per se, i.e., independently from a given CNF formula Σ. Within this language, unlike with the OBDD< one, a Boolean function may have several equivalent representations. For instance, let Σ = (¬a ∧ ¬b) ∨ (¬a ∧ c) ∨ (b ∧ c). Whatever <, I(Σ) can be represented by the ToOBDD< representation T such that T has a single node n₀, such that Var(n₀) = Var(Σ) and B(n₀) is the OBDD< representation equivalent to Σ; observing that Σ ≡ (¬a ∨ b) ∧ (¬b ∨ c), I(Σ) can also be represented by the ToOBDD< representation T such that T has two nodes n₀ and n₁, the root of T is n₀, Var(n₀) = {a, b}, Var(n₁) = {b, c}, B(n₀) is the OBDD< formula equivalent to (¬a ∨ b), and B(n₁) is the OBDD< formula equivalent to (¬b ∨ c). In short, ToOBDD< does not offer the property of canonical representation.

Compiling a CNF formula Σ into a ToC representation T basically consists in computing first a decomposition tree of Σ, then taking advantage of any CNF-to-C compiler so as to turn the CNF clauses(n) formulae (for each node n of the tree) into equivalent C representations, and finally to use the well-known message-passing propagation algorithm (see the Propagate function in [Subbarayan et al., 2007]), which applies also to 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 ToC representation when compilers to C are available. The running intersection property enables one to replace a global computation on the resulting 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 $|n|$.
- $C$ 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 propositional 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 $P = \text{NP}$.
6. If $C$ satisfies $\text{CL}$, then $\text{ToC}$ does not satisfy $\text{SE}$ unless $P = \text{NP}$.

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 $P = \text{NP}$ (because they are not complete propositional languages 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 $P = \text{NP}$.

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

These results show that the $\text{ToC}$ languages typically satisfy only few transformations among $\text{CD}, \lor \text{ABC}, \lor C$ and $\neg C$. 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 $\lor C$, 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 $\lor C$, 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 NP-hard in the general case does not discard $\text{ToOBDD}_{<}$ (and beyond 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; consequently, 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 get 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 $C' \not< C$.
4. If $C$ satisfies $\text{IM}$, then $\text{ToC} \not< \text{DNF}$.

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 $\land C$, e.g., $C = \text{CNF}$, we can prove that $C \sim_{p} \text{ToC}$).

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

\[\text{The price to be paid by such a restriction is a lack of expressiveness: none of the languages of $\text{ToC}$ representations of width bounded by $c$ (where $c$ is a parameter) is complete.}\]
• 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) more succinct than the corresponding ToC language; hence none of the languages C which are less succinct than DNNF (e.g. C = DNF) can be more succinct than such ToC languages; thus, we get for instance that DNF $\not\leq^*_{\text{ToC}}$ ToDNNF (which together with point 1. shows that ToDNNF $\not<^*_{\text{ToC}}$ DNF).

• Another consequence of point 3. is that if C satisfies CL then DNNF $\not<^*_{\text{ToC}}$ ToC (hence d-DNNF $\not<^*_{\text{ToC}}$ C). 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 $\not<^*_{\text{ToC}}$ 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, $\{\text{Var}(n) \mid n \in N\}$ is a decomposition set for I(T). We proved that:

**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 $\text{Var}(\delta) \subseteq V$.

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 $\exists \text{Var}(\delta).f$ 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 $\text{Var}(n) = \{x_1, \ldots, x_p\}$ (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 $\bigwedge_{i=1}^p x_i$ cannot be represented in ToC.

### 4 Back to ToOBDD<\<\< Representations

Let us now fix C to OBDD<\<\< in order to get some further results. Beyond ToOBDD<\<\< we have investigated the properties of U(ToOBDD<\<\<) (the union of all 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 OBDD<\< representations in a given U(ToOBDD<\<\<) are based on the same order. Hence, U(ToOBDD<\<\<) is a proper subset of ToOBDD.

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

<table>
<thead>
<tr>
<th>C</th>
<th>VA</th>
<th>CO</th>
<th>IM</th>
<th>EQ</th>
<th>SE</th>
<th>CT</th>
<th>ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToOBDD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>U(ToOBDD&lt;&lt;&lt;)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ToOBDD&lt;&lt;&lt;</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OBDD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OBDD&lt;&lt;&lt;</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>TO</th>
<th>LDL</th>
<th>BC</th>
<th>CL</th>
<th>BL</th>
<th>BC</th>
<th>CL</th>
<th>ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToOBDD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>U(ToOBDD&lt;&lt;&lt;)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ToOBDD&lt;&lt;&lt;</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OBDD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OBDD&lt;&lt;&lt;</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 1: ✓ means “satisfies”, • means “does not satisfy”, o means “does not satisfy unless $P = NP$”, and $\not\leq$ means “does not satisfy unless PH collapses.” Results for OBDD<\<\< and OBDD are from [Darwiche and Marquis, 2002] and are given here as a baseline.

The fact that ToOBDD, U(ToOBDD<\<\<), and ToOBDD<\<\< satisfy CO, VA, IM, and ME and that none of these languages satisfies any of CE, SE, CD, $\land BC$, $\land C$, $\lor C$ or $\neg C$, unless $P = NP$ 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 polynome 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], $\neg C$ is not trivial: the negation of a conjunction of OBDD<\< representations is equivalent to the disjunction of their negations. We actually showed that the $\neg C$ transformation on ToOBDD<\< cannot be achieved in polynomial time unless $P = NP$.

As to succinctness, we proved the following results:

**Proposition 5**

1. For each $<$, ToOBDD<\<\< $\nleq^*_{\text{ToC}}$ OBDD<\<\<.
2. For each $<$, DNNF $\not\leq^*_{\text{ToC}}$ ToOBDD<\<\<.
3. ToOBDD $\not<^*_{\text{ToC}}$ DNF.
4. ToOBDD $\not\leq^*_{\text{ToC}}$ 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-DNNF $\not\leq^*_{\text{ToC}}$ ToOBDD<\<\<. This explains in some sense the space savings which can be offered by ToOBDD<\< over d-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(ToOBDD<\<\<) and ToOBDD<\<\< are incomparable w.r.t. succinctness with the languages CNF, DNF, and DNNF.
5 Conclusion

In this paper, the concept of tree-of-BDDs has been extended to any complete propositional representation language 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 C, as well as results about the spatial efficiency of the ToC languages. Focusing on the ToOBDD 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 satisfies IM and ME but does not satisfy any query among CE, SE. We have also proved that ToOBDD does not satisfy any transformation among CD, FO, ABC, or ¬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 language (and in general the ToC languages) satisfies only few queries and transformations. Subsequently, in applications where some queries/transformations not satisfied by ToOBDD must be achieved under some guaranteed response time, considering ToOBDD 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 are CNF, there are CNF formulae which can be compiled into ToOBDD 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 £ ToOBDD. Nevertheless, our result ToOBDD £ DNNF shows that this empirical evidence can be argued (this result implies that some DNNF representations do not have "small" ToOBDD 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 language. From our results, it comes easily that ToFBDD satisfies CO, VA, IM, ME (hence the same queries as ToOBDD); since ToFBDD is at least as succinct as ToOBDD, it appears as a challenging fragment. Furthermore, a compiler to FBDD is already available (see e.g. http://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 £ ToOBDD and ToDNNF £ 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 ToDNNFT language, based on DNNFT [Pipatsrisawat and Darwiche, 2008], also looks interesting in this respect since it satisfies both FO and ABC, 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.

References


