New Compilation Languages Based on Structured Decomposability

Knot Pipatsrisawat and **Adnan Darwiche**
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095 USA
{thammakn,darwiche}@cs.ucla.edu

Abstract

We introduce in this paper two new, complete propositional languages and study their properties in terms of (1) their support for polytime operations and (2) their ability to represent boolean functions compactly. The new languages are based on a structured version of decomposability—a property that underlies a number of tractable languages. The key characteristic of structured decomposability is its support for a polytime conjoin operation, which is known to be intractable for unstructured decomposability. We show that any CNF can be compiled into formulas in the new languages, whose size is only exponential in the treewidth of the CNF. Our study also reveals that one of the languages we identify is as powerful as OBDDs in terms of answering key inference queries, yet is more succinct than OBDDs.

Introduction

Knowledge compilation is an approach to inference which is based on converting knowledge bases into target compilation languages that support relevant inference tasks in polytime. Over the years, many target compilation languages have been proposed for different inference tasks, including OBDDs (Bryant 1986), prime implicates (Reiter and de Kleer 1987; Marquis 1995), prime implicants (Schrag 1996), DNNF (Darwiche 2001a), and AOMDD (Mateescu and Dechter 2006).

In this work, we focus our attention on the DNNF language (Darwiche 2001a), which is a subset of negation normal form (NNF) that satisfies *decomposability*. While DNNF has many potential applications, it has never been utilized in an actual system due to the lack of a practical compiler for converting CNF formulas into DNNF. In practice, a subset of DNNF which satisfies the property of *determinism* is used. This subset, which is called d-DNNF, is less succinct than DNNF, and is supported by a publicly available compiler that converts CNF to d-DNNF (Darwiche 2004). This compiler is available at *http://reasoning.cs.ucla.edu/c2d/* and has been successfully employed in many reasoning applications, including probabilistic reasoning (Wachter and Haenni 2006; Chavira, Darwiche, and Jaeger 2006), planning (Bonet and Geffner 2006; Palacios et al. 2005), diagnosis (Barrett 2005; Elliott and Williams 2006; Siddiqi and Huang 2007), and Max-SAT (Pipatsrisawat and Darwiche 2007; Ramírez and Geffner 2007). None of these applications, however, except for probabilistic reasoning, require determinism. Hence, these applications can well be handled by the more succinct DNNF language, if one had access to an efficient DNNF compiler. Yet, until today, one could not enforce the property of decomposability efficiently without also enforcing determinism as a side effect. Note also that the previous list of applications does not include classical applications from formal verification that are based on unbounded model checking (e.g., (Clarke, Grumberg, and Peled 2000)). These applications require bottom-up, incremental compilation of formulas, where pieces of the knowledge base are compiled independently and then conjoined together to produce a compilation of the whole knowledge base. Neither DNNF nor d-DNNF, however, support a conjoin operation in polytime (Darwiche and Marquis 2002), which happens to be the key property of OBDDs that enables their use in formal verification applications.

To address these challenges, we introduce the notion of *structured decomposability* in this paper, leading to structured versions of DNNF and d-DNNF that provide support for a polytime conjoin operation. As a result, we can now develop bottom-up compilers for both DNNF and d-DNNF that parallel the standard compilers for OBDD. Moreover, we now seem to have the first practical compiler that can enforce decomposability without enforcing determinism as a side effect. We study the properties of the new languages according to the metrics proposed in (Darwiche and Marquis 2002), where we show that structured d-DNNF supports as many polytime queries as OBDD, while being strictly more succinct than OBDD.

Basic Definitions

A *Negation Normal Form* (NNF) is a rooted, directed acyclic graph (DAG) in which each leaf node is labeled with a literal, true, or false and each internal node is labeled with a conjunction ($\wedge$) or disjunction ($\vee$); see Figure 1. A DAG node will be used to represent the formula rooted at the node. An NNF is *decomposable* iff every child of every AND node shares no variable with its siblings, leading to the DNNF language (Darwiche 2001a). An NNF is *deterministic* iff every child of every OR node shares no models with its siblings,

leading to the d-DNNF language (Darwiche 2001b). Lastly, OBDD is a subset of d-DNNF that satisfies two additional properties: (1) the root node is a decision node, (2) variables appear in the same order on every directed path from the root (See (Darwiche and Marquis 2002) for formal definitions).

We will introduce in the next section structured versions of the DNNF and d-DNNF languages, which result from requiring a structured version of the decomposability property. For this, we need the following notion.

Definition 1 (V-tree) *A v-tree for a set of variables PV is a full, rooted binary tree whose leaves are in one-to-one correspondence with the variables in PV.*

Figure 1 depicts a v-tree for variables a, b, c, d and e. For a v-tree node t, we use $vars(t)$ to denote the set of variables mentioned by the subtree rooted at t. For a non-leaf v-tree node t, we use t^l (t^r) to denote the left (right) child of t. We will also say that T is a v-tree of the formula Δ if PV includes the variables of Δ.

Structured Decomposability

In this section, we give a formal definition of a structured subset of DNNF. We will also show that this language is a complete proper subset of DNNF. For the remaining of this paper, we will assume, without loss of generality, that every conjunction has exactly two non-constant conjuncts, while a disjunction can have any number of disjuncts.[1]

Definition 2 *A DNNF Δ respects a v-tree T if for every conjunction $\alpha \wedge \beta$ in Δ, there is a node t in T such that $vars(\alpha) \subseteq vars(t^l)$ and $vars(\beta) \subseteq vars(t^r)$.*

Figure 1 depicts a v-tree T and a respecting DNNF Δ. Let α be a sub-formula of DNNF Δ. The *decomposition node (d-node)* of α is defined as the deepest node d in v-tree T such that $vars(\alpha) \subseteq vars(d)$. Figure 1 depicts examples of d-nodes.

Definition 3 (DNNF$_T$) *The set of all DNNFs that respect a given v-tree T is denoted by DNNF$_T$. Moreover, structured DNNF (SDNNF) is the set containing all DNNF$_T$ for any T.*

Given any v-tree T for the formula Δ, we can always represent Δ in DNNF$_T$. Just structure each model of Δ according to T and disjoin them together. Hence, DNNF$_T$ and SDNNF are complete languages. Moreover, the following DNNF does not respect any v-tree: $(((a \wedge b) \wedge (c \wedge \neg d)) \vee ((a \wedge c) \wedge (b \wedge d)))$. Hence, DNNF$_T$ and SDNNF are proper subsets of DNNF.

A Polytime Conjoin Operation for DNNF$_T$

Now that we have defined a structured subset of DNNF, we present the main result of our work. In particular, conjoining any two DNNF$_T$ formulas is a tractable operation because of the imposed structure. This operation is important because it allows any CNF formula to be converted into a DNNF$_T$ in an incremental (bottom-up) manner. Note that an efficient conjoining algorithm does not exist for DNNF formulas in general, unless P=NP (Darwiche and Marquis 2002).

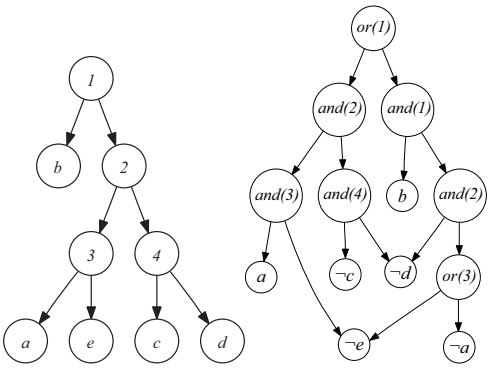

Figure 1: (left) A v-tree T with all internal nodes labeled with their indices. (right) a DNNF$_T$ formula for $(\neg a \vee \neg e) \wedge (a \vee b) \wedge (b \vee \neg c) \wedge (c \vee \neg d) \wedge (\neg b \vee \neg d)$. Each internal node is also annotated with the index of its d-node in T.

Our conjoining algorithm is similar in spirit to the *apply* operation for OBDD (Bryant 1986). The algorithm computes the conjunction of two formulas by recursively conjoining their sub-formulas and combining the results appropriately. To avoid making too many recursive calls, we maintain a cache which stores previously computed outputs of the algorithms. With this cache, there are only polynomially many possible recursive calls. Each recursive call performs a constant amount of computation, yielding a polynomial time complexity of the overall conjoining operation.

This algorithm is presented in Algorithm 1, which relies on several other components whose details are omitted due to space constraints. In particular, it assumes that, before the first call, the cache is properly initialized. We use nodes' id field as a cache key. It also assumes that each node has a $dnode$ field. The helper function $lca(s, t)$ computes the lowest common ancestor of nodes s and t in v-tree T. Readers are referred to (Bender and Farach-Colton 2000) for a constant-time implementation of this function.[2]

The code on Lines 4-7 simply handles the base cases, where α or β is a boolean constant. Lines 11-14 handle the cases when the inputs do not share a variable and when they are both literals. The case on Lines 16-21 is the core of the algorithm. In this case, the d-nodes are either the same or have an ancestor-descendent relationship. For the sake of brevity, we assume that α and β are appropriately swapped such that (1) if both d-nodes are the same and there is at least one OR node among α, β, then α is an OR node and (2) if both d-nodes are different, then α's d-node is always the ancestor. With this assumption, if α is an OR node (Line 16), we simply conjoin β with each of α's children. Otherwise, we decompose each input into the part that mentions variables in anc^l and the one that mentions variables in anc^r. On Line 20, we use the notation α^d to refer to the

[1] A conjunction with a constant conjunct can always be trivially simplified. To allow an arbitrary number of conjuncts, we only need to modify the definition of v-tree slightly.

[2] To achieve this, a data structured must be setup by preprocessing T prior to the first call to lca. Preprocessing takes $O(|T|)$ time and space. See (Bender and Farach-Colton 2000).

Function name: $conjoin$
input : DNNF$_T$ formulas α, β
output: A DNNF$_T$ formula equivalent to $\alpha \wedge \beta$

```
1  if CACHE[α.id, β.id] ≠ null then
2  |  return CACHE[α.id, β.id]
3  end
4  if α = false || β = false then
5  |  u ← false  //false input
6  else if α = true || β = true then
7  |  u ← (α = true?β : α)  //true input
8  else
9  |  s ← α.dnode, t ← β.dnode
10 |  anc ← lca(s, t)
11 |  if anc is neither s nor t then
   |  |  //α, β do not share a variable
12 |  |  u ← α ∧ β
13 |  else if α, β are both literal nodes then
14 |  |  u ← (α = β?α : false)
15 |  else
   |  |  //α, β share some variables
16 |  |  if α is an OR node then
   |  |  |  //Assume that α = ⋁_{i=1}^{n} α_i
17 |  |  |  u ← ⋁_{i=1}^{n} conjoin(α_i, β)
18 |  |  else
   |  |  |  //Both nodes are AND nodes
19 |  |  |  L ← anc^l, R ← anc^r
20 |  |  |  u ← conjoin(α^L, β^L) ∧ conjoin(α^R, β^R)
21 |  |  end
22 |  end
23 |  u.dnode ← anc
24 end
25 CACHE[α.id, β.id] ← u
26 return u
```

Algorithm 1: A pseudo-code of $conjoin$.

sub-formula of α that mentions only (a subset of) $vars(d)$, where d is a node of a v-tree.[3] After we decompose both inputs, we conjoin their sub-formulas appropriately (Line 20).

Note that, without structured decomposability in both input formulas, we would not be able to easily decompose both formulas and make simple recursive calls in this last case. Next, we show that $conjoin$ in fact conjoins two DNNF$_T$ formulas in time quadratic in the sizes of the formulas. Here, given a formula α, we define its size, $|\alpha|$, to be the number of nodes in its DAG representation.

Theorem 1 *Given DNNF$_T$ formulas α and β, $conjoin(\alpha, \beta)$ returns a DNNF$_T$ formula equivalent to $\alpha \wedge \beta$ in time $O(|\alpha||\beta|)$.*

The proof for this result is straightforward, as caching limits the number of possible recursive calls, each of which performs a bounded amount of computation.

We close this section by mentioning that every clause is by itself a DNNF$_T$ formula (for any T), because it contains no conjunction. As a result we can convert any CNF formula into a DNNF$_T$ formula in a bottom-up manner. All we need to do is convert each clause into its DAG form and conjoin all clauses together.

[3] If α does not mention any variable in $vars(d)$, α^d is true.

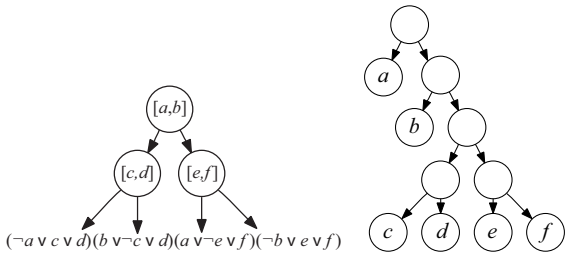

Figure 2: (left) A d-tree with internal nodes labeled with shared variables. (right) An induced v-tree.

Adding Determinism

We now consider a structured subset of d-DNNF.

Definition 4 (d-DNNF$_T$) *The language d-DNNF$_T$ contains all and only DNNF$_T$ formulas that satisfy determinism. Moreover, the language deterministic structured DNNF (d-SDNNF) is the union of all d-DNNF$_T$, for any T.*

Both d-DNNF$_T$ and d-SDNNF are complete, proper subsets of d-DNNF. The argument given earlier for DNNF$_T$ applies here in a similar manner. The following proposition shows that Algorithm 1 preserves determinism and, hence, can be use to conjoin d-DNNF$_T$ formulas in polytime.

Proposition 1 *Given two d-DNNF$_T$ formulas α, β, the output of $conjoin(\alpha, \beta)$ is a d-DNNF$_T$ formula.*

The only place where we create an OR node is on Line 17 of Algorithm 1. The children of this node cannot share a model because α is by itself deterministic.

We can convert every clause into a d-SDNNF formula in time linear in the clause size (Darwiche 2002). This enables incremental compilation of CNF into d-SDNNF.[4] The choice of v-tree has a crucial impact on the sizes of SDNNF and d-SDNNF formulas, similar to how a variable order is to OBDD. Although constructing the right v-tree for a compilation is still a subject of current research, we have the following guarantee.

Proposition 2 *Given a CNF Δ, we can always construct an equivalent d-SDNNF (hence, SDNNF) formula whose size is only exponential in the treewidth of Δ.*

In (Darwiche 2001b), an algorithm for compiling CNF into d-DNNF with the above treewidth guarantee is presented. The algorithm utilizes a data structured called *d-tree* to guide the (unstructured) decomposition. A d-tree is a binary tree whose leaves are in one-to-one correspondence with the clauses in the CNF. In general, a v-tree can be naturally induced from any d-tree. Figure 2 shows an example of a d-tree and a v-tree induced from it. Due to space constraints, we refer the readers to (Pipatsrisawat and Darwiche 2008) for a detailed discussion of this process. The algorithm in (Darwiche 2001b) can be slightly modified to make the output d-DNNF formula respect a v-tree induced from the d-tree, without affecting the treewidth guarantee.

[4] A bottom-up DNNF$_T$ compiler is an ongoing research subject.

Supported Queries and Transformations

In this section, we examine the new languages, DNNF$_T$ and d-DNNF$_T$ with respect to the criteria proposed in (Darwiche and Marquis 2002). The criteria fall into two categories: queries and transformations. A query on a formula (formulas) can be viewed as a decision problem about the formula(s) that may result in a positive (1) or negative (0) answer without modifying the formula(s). A transformation, on the other hand, is an operation that returns a modified formula. Due to space limit we will only discuss proofs of the less obvious results.

We first discuss query-related properties of the new languages. We say that a language satisfies a property if and only if there exists a polytime algorithm for answering the query based on formula in the language. The following queries are considered:

- **CO** (**VA**): Is the formula consistent (valid)?
- **CE** Does the formula entail a given clause?
- **IM**: Does a given term imply the formula?
- **EQ**: Are two formulas logically equivalent?
- **SE**: Does one formula entail another formula?
- **CT**: How many models does the formula have?
- **ME**: Enumerate the models of the formula.

Readers are referred to (Darwiche and Marquis 2002) for formal definitions of queries and transformations discussed here.

Table 1 summarizes query-related properties of the new languages. Properties of DNNF, d-DNNF, and OBDD$_<$ are also shown here for comparison.[5]

Query	DNNF	DNNF$_T$	d-DNNF	d-DNNF$_T$	OBDD$_<$
CO	√	√	√	√	√
VA	○	○	√	√	√
CE	√	√	√	√	√
IM	○	○	√	√	√
EQ	○	○	?	√	√
SE	○	○	○	√	√
CT	○	○	√	√	√
ME	√	√	√	√	√

Table 1: Query table. √ indicates that the language satisfies the property. ○ indicates that the language does not satisfy the property unless P=NP. ? reflects our ignorance about whether the property holds for the language.

Proposition 3 *The results in Table 1 hold.*

The positive results for DNNF$_T$ are inherited from DNNF. The proofs for the negative results are very similar to those for DNNF, which are given in (Darwiche and Marquis 2002). Similarly, most positive results for d-DNNF$_T$ are inherited from d-DNNF. The results for **EQ** and **SE**, however, are enabled by the fact that d-DNNF$_T$ supports polytime conjunction of a bounded number of formulas. In particular, two d-DNNF$_T$ formulas Δ_1, Δ_2 are equivalent iff they have the same model count, which also equals the model count of $\Delta_1 \wedge \Delta_2$. Since the language satisfies **CT**, this test can be performed in polytime. A similar proof applies for **SE**.

According to Table 1, imposing additional structure on DNNF does not result in any additional query support. On the other hand, adding structure to d-DNNF allows both equivalence and sentential entailment to be checked in polytime, making all queries tractable.

The next set of properties considered are related to formula transformations. A property is satisfied if and only if there exists a polytime algorithm for transforming formulas in the language into appropriate formulas in the same language. We consider the following transformations:

- **CD**: The formula conditioned on a consistent term.
- **SFO** (**FO**): The result of existentially quantifying a variable (an arbitrary number of variables) from the formula.
- ∧**BC** (∧**C**): The conjunction of a bounded (unbounded) number of formulas.
- ∨**BC** (∨**C**): The disjunction of a bounded (unbounded) number of formulas.
- ¬**C**: The negation of the formula.

Table 2 presents transformation-related properties of the new languages. Again, properties of DNNF, d-DNNF, and OBDD$_<$ are shown here for comparison.

Trans.	DNNF	DNNF$_T$	d-DNNF	d-DNNF$_T$	OBDD$_<$
CD	√	√	√	√	√
FO	√	√	○	•	•
SFO	√	√	○	?	√
∧**C**	○	○	○	•	•
∧**BC**	○	√	○	√	√
∨**C**	√	√	○	•	•
∨**BC**	√	√	○	?	√
¬**C**	○	○	?	?	√

Table 2: Transformation table. √ indicates that the language satisfies the property. ○ indicates that the language does not satisfy the property unless P=NP. • means the language does not satisfy the property. ? reflects our ignorance about whether the property holds for the language.

Proposition 4 *The results in Table 2 hold.*

A proof of this proposition is available in the Appendix.

As shown earlier, the additional structure makes the bounded conjoining operation (∧**BC**) tractable in the new languages. This operation proves to be the only difference between DNNF and DNNF$_T$ in Table 2. However, for d-DNNF, the additional structure strengthens the negative results for several other properties, while making some others become unknown. In particular, **FO**,∧**C**, and ∨**C** cannot be satisfied by d-DNNF$_T$, whereas they were conditioned on the fact that P≠NP in the case of d-DNNF. There are two additional properties that become unknown for d-DNNF$_T$: **SFO** and ∨**BC**. We point out here, however, that the open questions in the d-DNNF$_T$ column are very closely related.

[5] OBDD$_<$ contains all OBDDs respecting the variable order $<$.

In particular, we have the following relationship between these results:

- The results for **SFO** and $\vee$**BC** have to be identical. This is because $\Delta_1 \vee \Delta_2 = \exists X\ ((X \wedge \Delta_1) \vee (\neg X \wedge \Delta_2))$, where $X \notin vars(\Delta_1) \cup vars(\Delta_2)$.
- A positive result ($\surd$) for $\neg$**C** implies positive results for **SFO** and $\vee$**BC**, because $\Delta_1 \vee \Delta_2 = \neg(\neg\Delta_1 \wedge \neg\Delta_2)$
- A negative result ($\circ$, $\bullet$) for either **SFO** or $\vee$**BC** implies the same negative result for $\neg$**C**, because of the same reason.

Succinctness

Another aspect that needs to be considered when choosing a language for a task is its succinctness (Gogic et al. 1995). Succinctness tells us how compact formulas in different languages are, relative to each other. A language L_1 is said to be *at least as succinct as* L_2, written $L_1 \leq L_2$ iff every formula α in L_2 has an L_1 equivalent, whose size is polynomial in $|\alpha|$. If $L_1 \leq L_2$ and $L_2 \not\leq L_1$, we say that L_1 is *strictly more succinct* than L_2, denoted $L_1 < L_2$ (see formal definitions in (Darwiche and Marquis 2002)). In this section, we discuss the succinctness of SDNNF and d-SDNNF relative to other closely related languages. Interested readers are referred to (Pipatsrisawat and Darwiche 2008) for proofs of the claims made here.

First of all, let us consider the structured DNNF language. Imposing structure on DNNF results in a language that is strictly less succinct (DNNF $<$ SDNNF). As it turns out, this additional structure will also cause SDNNF formulas to blow up on some d-DNNF and FBDD formulas as well (SDNNF$\not\leq$ d-DNNF, SDNNF$\not\leq$ FBDD).[6] In all these cases, the circular bit-shift function (Fortune, Hopcroft, and Schmidt 1978) provides an exponential separation between SDNNF and other languages. Interestingly, unless the polynomial hierarchy collapses (Selman and Kautz 1996; Cadoli and Donini 1997), there must be some SDNNF formulas that cannot be represented compactly as d-DNNF and FBDD as well (d-DNNF$\not\leq$SDNNF, FBDD$\not\leq$ SDNNF). These results show that SDNNF is not comparable to d-DNNF and FBDD in terms of succinctness.

We saw in the last section that d-SDNNF is as powerful as OBDD when it comes to query-related properties. As d-SDNNF does not impose as much structure as OBDD, one expects its formulas to be more compact. As it turns out, d-SDNNF is indeed strictly more succinct than OBDD (d-SDNNF$<$OBDD). The *indirect storage access* function described in (Breitbart, H. Hunt, and Rosenkrantz 1995) separates the two languages.

Related Work

(Mateescu and Dechter 2006) proposed a relaxation of OBDD called AND/OR Multi-value Decision Diagram (AOMDD). When restricted to boolean variables, AOMDD is a strict subset of d-SDNNF. In particular, AOMDD utilized a special form of v-tree called *pseudo tree*, whose structure depends on the structure of the formula to be compiled. Moreover, every AOMDD formula satisfies determinism in a restricted way; every disjunct of each OR node must disagree on the value of next variable in the ordering. The added structure allows AOMDD to efficiently support the *apply* operation, which makes operations such as negation, (bounded) conjunction and disjunction tractable (given that the formulas have compatible pseudo trees). Nevertheless, given a pseudo tree, the language of AOMDDs that respect the pseudo tree is *not complete*; not every formula may be represented this way. For example, the clause $(a \vee b \vee c)$ cannot be represented by any AOMDD that respects the pseudo tree 

, because b and c are not independent even after a is assigned. The only type of pseudo trees that induce a complete language are linear trees, in which case AOMDD reduces to just OBDD. d-SDNNF, on the other hand, does not have any of these restrictions. Any properly labeled tree can be used as a v-tree for any boolean formula, making d-SDNNF more general and more flexible.

Conclusions

We introduced two new, complete compilation languages based on structured decomposability. We showed that the conjoin operation is tractable in the new languages and that every CNF can be compiled into them with complexity exponential only in the treewidth. We studied the new languages in terms of supported operations and succinctness and showed that d-DNNF$_T$ is as powerful as OBDD$_<$ in answering key queries, yet is strictly more succinct.

Acknowledgments

This work has been partially supported by Air Force grant #FA9550-05-1-0075 and by NSF grant #IIS-0713166.

Proofs

We first present some lemmas that will simplify the proof of Proposition 4. The next lemma states that every DNF formula can be converted into a DNNF$_T$ formula in polytime.

Lemma 1 *Given a DNF formula Δ and a v-tree T of Δ, we can convert Δ into a DNNF$_T$ formula in $O(|\Delta||T|)$.*

We just need to structure each disjunct in Δ according to T.

Definition 5 *A v-tree T is linear if every internal node has at least one leaf node as its child.*

A linear v-tree naturally induces a complete variable ordering (by depth). As a result, we will say that a linear v-tree T is *compatible* with an ordering of its variables if for all $x, y \in vars(T)$, $x < y$ iff the depth of x is less than or equal to the depth of y.

Lemma 2 *Given a linear v-tree T, every d-DNNF$_T$ Δ can be converted into an OBDD$_<$ in time $O(|\Delta||T|)$, if T and $<$ are compatible.*

Proof of Proposition 4
Transformation Properties of DNNF$_T$

- **CD**. Follows from the results in (Darwiche and Marquis 2002)

[6] FBDD is the language of free binary decision diagram, which is BDD that satisfies the "test-once" property (see (Darwiche and Marquis 2002)).

- $\vee$**C** and $\vee$**BC**. These properties are satisfied as disjoining DNNF$_T$ formulas does not affect decomposability.
- **FO** and **SFO**. These properties follow from the results in (Darwiche 2001a). In particular, the projection operation defined in Definition 8 of (Darwiche 2001a) does not affect decomposability with respect to any v-tree.
- $\wedge$**BC**. Follows from Theorem 1.
- $\wedge$**C**. Unbounded conjunction, however, cannot be performed in polytime unless P=NP. This is because every clause is already a DNNF$_T$. If we could conjoin an arbitrary number of DNNF$_T$ formulas in polytime, SAT would be solvable in polytime.
- $\neg$**C**. We cannot negate an arbitrary DNNF$_T$ formula in polytime unless P=NP. If we could, we would be able to decide SAT in polytime. To do so, we convert a given CNF formula into its negation in DNF in time linear to the size of the formula. Then, we convert the DNF formula into a DNNF$_T$ formula in polytime (Lemma 1). After that, we could negate the DNNF$_T$ formula to get back a formula that is equivalent to the original CNF formula. Since DNNF$_T$ satisfies **CO**, we would be able to decide the satisfiability of the original CNF formula in polytime.

Transformation Properties of d-DNNF$_T$

- **CD**. Because conditioning preserves both decomposability and determinism (Darwiche and Marquis 2002).
- $\wedge$**BC**. Follows from Theorem 1 and Proposition 1.
- **FO**, $\wedge$**C** and $\vee$**C**. These operations cannot be be performed in polytime. This is because OBDD$_<$ $\subseteq$ d-DNNF$_T$ (given that T and $<$ are compatible). If we could perform any of these transformations in polytime, the resulting formula will still be a in d-DNNF$_T$, which can be efficiently converted into an OBDD$_<$ formula (Lemma 2). This would allow us to perform any of these transformation on OBDD$_<$ formulas in polytime. This contradicts the results in (Darwiche and Marquis 2002). □

References

Barrett, A. 2005. Model compilation for real-time planning and diagnosis with feedback. In *IJCAI-05*, 1195–1200.

Bender, M. A., and Farach-Colton, M. 2000. The LCA problem revisited. In *LATIN-2000*, 88–94.

Bonet, B., and Geffner, H. 2006. Heuristics for planning with penalties and rewards using compiled knowledge. In *KR-06*, 452–462.

Breitbart, Y.; H. Hunt, I.; and Rosenkrantz, D. 1995. On the size of binary decision diagrams representing boolean functions. *Theor. Comput. Sci.* 145(1-2):45–69.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Tran. Com.* C-35:677–691.

Cadoli, M., and Donini, F. M. 1997. A survey on knowledge compilation. *AI Communications* 10(3-4):137–150.

Chavira, M.; Darwiche, A.; and Jaeger, M. 2006. Compiling relational Bayesian networks for exact inference. *International Journal of Approximate Reasoning* 42(1–2):4–20.

Clarke, E. M.; Grumberg, O.; and Peled, D. A. 2000. *Model checking*. MIT Press.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *JAIR* 17:229–264.

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

Darwiche, A. 2001b. On the tractability of counting theory models and its application to belief revision and truth maintenance. *JANCL* 11(1-2):11–34.

Darwiche, A. 2002. A compiler for deterministic, decomposable negation normal form. In *AAAI-02*, 627–634.

Darwiche, A. 2004. New advances in compiling CNF to decomposable negational normal form. In *Proceedings of European Conference on Artificial Intelligence*.

Elliott, P., and Williams, B. 2006. Dnnf-based belief state estimation. In *Proceedings of AAAI-06*, 36–41.

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

Gogic, G.; Kautz, H. A.; Papidimitriou, C.; and Selman, B. 1995. The comparative linguistics of knowledge representation. In Mellish, C., ed., *Proceedings of IJCAI-95*, 862–869. San Francisco: Morgan Kaufmann.

Marquis, P. 1995. Knowledge compilation using theory prime implicates. In *Proceedings of IJCAI 95*, 837–843.

Mateescu, R., and Dechter, R. 2006. Compiling constraint networks into and/or multi-valued decision diagrams (AOMDDs). In *CP-06*, 329–343.

Palacios, H.; Bonet, B.; Darwiche, A.; and Geffner, H. 2005. Pruning conformant plans by counting models on compiled d-dnnf representations. In *Proceedings of ICAPS 05*, 141–150. AAAI Press.

Pipatsrisawat, K., and Darwiche, A. 2007. Clone: Solving weighted max-sat in a reduced search space. In *Proceedings of 20th Australian Joint Conf. on Artificial Intel.*

Pipatsrisawat, K., and Darwiche, A. 2008. New compilation languages based on structured decomposability. Technical Report D–157, Automated Reasoning Group, Comp. Sci. Department, UCLA.

Ramírez, M., and Geffner, H. 2007. Structural relaxations by variable renaming and their compilation for solving mincostsat. In *CP-07*.

Reiter, R., and de Kleer, J. 1987. Foundations of assumption-based truth maintenance systems: Preliminary report. In *AAAI-87*, 183–189.

Schrag, R. C. 1996. Compilation of critically constrained knowledge bases. In *Proceedings of AAAI'96*, 510–515.

Selman, B., and Kautz, H. 1996. Knowledge compilation and theory approximation. *J. of the ACM* 43(2):193–224.

Siddiqi, S., and Huang, J. 2007. Hierarchical diagnosis of multiple faults. In *Proceedings of IJCAI-07*.

Wachter, M., and Haenni, R. 2006. Logical compilation of bayesian networks. Technical Report iam-06-006, University of Bern, Switzerland.