A Perspective on Knowledge Compilation

Adnan Darwiche
Computer Science Department
University of California
Los Angeles, CA 90095, USA
darwiche@cs.ucla.edu

Pierre Marquis
CRIL/Université d'Artois
F-62307, Lens Cedex, France
marquis@cril.univ-artois.fr

Abstract

We provide a perspective on knowledge compilation which calls for analyzing different compilation approaches according to two key dimensions: the succinctness of the target compilation language, and the class of queries and transformations that the language supports in polytime. We argue that such analysis is necessary for placing new compilation approaches within the context of existing ones. We also go beyond classical, flat target compilation languages based on CNF and DNF, and consider a richer, nested class based on directed acyclic graphs, which we show to include a relatively large number of target compilation languages.

1 Introduction

Knowledge compilation has emerged recently as a key direction of research for dealing with the computational intractability of general propositional reasoning [Darwiche, 1999a; Cadoli and Donini, 1997; Boufkhad *et al.*, 1997; Khardon and Roth, 1997; Selman and Kautz, 1996; Schrag, 1996; Marquis, 1995; del Val, 1994; Dechter and Rish, 1994; Reiter and de Kleer, 1987]. According to this direction, a propositional theory is compiled off-line into a target language, which is then used on-line to answer a large number of queries in polytime. The key motivation behind knowledge compilation is to push as much of the computational overhead into the off-line phase, which is amortized over all on-line queries. But knowledge compilation can serve other important purposes as well. For example, target compilation languages and their associated algorithms can be very simple, allowing one to develop on-line reasoning systems for simple software and hardware platforms. Moreover, the simplicity of algorithms that operate on compiled languages help in streamlining the effort of algorithmic design into a single task: that of generating the smallest compiled representations possible, as that turns out to be the main computational bottleneck in compilation approaches.

There are three key aspects of any knowledge compilation approach: the succinctness of the target language into which the propositional theory is compiled; the class of queries that can be answered in polytime based on the compiled representation; and the class of transformations that can be applied to the representation in polytime. The AI literature has thus far focused mostly on target compilation languages which are variations on DNF and CNF formulas, such as Horn theories and prime implicates. Moreover, it has focused mostly on clausal entailment queries, with very little discussion of tractable transformations on compiled theories.

The goal of this paper is to provide a broad perspective on knowledge compilation by considering a relatively large number of target compilation languages and analyzing them according to their *succinctness* and the class of *queries/transformations* that they admit in polytime.

Instead of focusing on classical, *flat* target compilation languages based on CNF and DNF, we consider a richer, *nested* class based on representing propositional sentences using directed acyclic graphs, which we refer to as `NNF`. We identify a number of target compilation languages that have been presented in the AI, formal verification, and computer science literature and show that they are special cases of `NNF`. For each such class, we list the extra conditions that need to be imposed on `NNF` to obtain the specific class, and then identify the set of queries and transformations that the class supports in polytime. We also provide cross-rankings of the different subsets of `NNF`, according to their succinctness and the polytime operations they support.

The main contribution of this paper is then a *map* for deciding the target compilation language that is most suitable for a particular application. Specifically, we propose that one starts by identifying the set of queries and transformations needed for their given application, and then choosing the most succinct language that supports these operations in polytime.

This paper is structured as follows. We start by formally defining the `NNF` language in Section 2, where we list a number of conditions on `NNF` that give rise to a variety of target compilation languages. We then study the succinctness of these languages in Section 3 and provide a cross-ranking that compares them according to this measure. We consider a number of queries and their applications in Section 4 and compare the different target compilation languages according to their tractability with respect to these queries. Section 5 is then dedicated to a class of transformations, their applications, and their tractability with respect to the different target compilation languages. We finally close in Section 6 by some concluding remarks. Proofs of theorems are omitted for space limitations but can be found in [Darwiche and Marquis, 2001].

2 The NNF Language

We consider more than a dozen languages in this paper, all of which are subsets of the NNF language, which is defined formally as follows [Darwiche, 1999a; 1999b].

Definition 2.1 *Let PS be a finite set of prop. variables. A sentence in* $\mathtt{NNF}_{PS}$ *is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with true, false,* X *or* $\neg X$, $X \in PS$*; and each internal node is labeled with* $\wedge$ *or* $\vee$ *and can have arbitrarily many children. The size of a sentence in* $\mathtt{NNF}_{PS}$ *is the number of its DAG edges. Its height is the maximum number of edges from the root to some leaf in the DAG.*

Figure 1 depicts a sentence in NNF, which represents the odd parity function (we omit reference to variables PS when no confusion is anticipated). Any propositional sentence can be represented as a sentence in NNF, so the NNF language is *complete*.

It is important here to distinguish between a *representation* language and a *target compilation* language. The former should be natural enough to enable one to encode knowledge directly, while the latter should be tractable enough to permit some polytime queries and/or transformations. We will consider a number of target compilation languages that do not qualify as representation languages from this perspective, as they are not suitable for humans to construct or interpret. We will also consider a number of representation languages that do not qualify as target compilation languages.[1]

For a language to qualify as a target compilation language, we require that it permits a polytime clausal entailment test. Therefore, NNF does not qualify as a target compilation language (unless P=NP) [Papadimitriou, 1994], but many of its subsets do. We define a number of these subsets below, each of which is obtained by imposing further conditions on NNF.

We will distinguish between two key subsets of NNF: *flat* and *nested* subsets. We first consider flat subsets, which result from imposing combinations of the following properties:

- **Flatness:** The height of each sentence is at most 2. The sentence in Figure 4 is flat, but the one in Figure 1 is not.
- **Simple-disjunction:** The children of each or-node are leaves that share no variables (the node is a *clause*).
- **Simple-conjunction:** The children of each and-node are leaves that share no variables (the node is a *term*). The sentence in Figure 4 satisfies this property.

Definition 2.2 *The language* f-NNF *is the subset of* NNF *satisfying flatness. The language* CNF *is the subset of* f-NNF *satisfying simple–disjunction. The language* DNF *is the subset of* f-NNF *satisfying simple–conjunction.*

CNF does not permit a polytime clausal entailment test and, hence, does not qualify as a target compilation language (unless P=NP). But its DNF dual does.

[1]Admittedly, the notion of a representation language is not a formal one, yet the distinction is important. For example, we believe that when proposing target compilation languages in AI, there is usually an implicit requirement that the proposed language is also a representation language. As we shall see later, however, the most powerful target compilation languages are not suitable for humans to specify or interpret directly.

The following subset of CNF, *prime implicates,* has been quite influential in computer science:

Definition 2.3 *The language* PI *is the subset of* CNF *in which each clause entailed by the sentence is subsumed by a clause that appears in the sentence; and no clause in the sentence is subsumed by another.*

A dual of PI, *prime implicants* IP, can also be defined.[2]

Definition 2.4 *The language* IP *is the subset of* DNF *in which each term entailing the sentence subsumes some term that appears in the sentence; and no term in the sentence is subsumed by another term.*

We now consider *nested* subsets of the NNF language, which do not impose any restriction on the height of a sentence. Instead, these subsets result from imposing one or more of the following conditions: *decomposability, determinism, smoothness, decision,* and *ordering.* We start by defining the first three properties. From here on, if C is a node in an NNF, then $Vars(C)$ denotes the set of all variables that label the descendants of node C.

- **Decomposability** [Darwiche, 1999a; 1999b]. An NNF satisfies this property if for each conjunction C in the NNF, the conjuncts of C do not share variables. That is, if $C_1, \ldots, C_n$ are the children of and-node C, then $Vars(C_i) \cap Vars(C_j) = \emptyset$ for $i \neq j$. Consider the and-node marked in Figure 1(a). This node has two children, the first contains variables A, B while the second contains variables C, D. This and-node is then decomposable since the two children do not share variables. Each other and-node in Figure 1(a) is also decomposable and, hence, the NNF in this figure is decomposable.
- **Determinism** [Darwiche, 2000]: An NNF satisfies this property if for each disjunction C in the NNF, each two disjuncts of C are logically contradictory. That is, if $C_1, \ldots, C_n$ are the children of or-node C, then $C_i \wedge C_j \models \mathit{false}$ for $i \neq j$. Consider the or-node marked in Figure 1(b), which has two children corresponding to sub-sentences $\neg A \wedge B$ and $\neg B \wedge A$. These two sub-sentences are logically contradictory. The or-node is then deterministic and so are the other or-nodes in Figure 1(b). Hence, the NNF in this figure is deterministic.
- **Smoothness** [Darwiche, 2000]: An NNF satisfies this property if for each disjunction C in the NNF, each disjunct of C mentions the same variables. That is, if $C_1, \ldots, C_n$ are the children of or-node C, then $Vars(C_i) = Vars(C_j)$ for $i \neq j$. Consider the marked or-node in Figure 1(c). This node has two children, each of which mentions variables A, B. This or-node is then smooth and so are the other or-nodes in Figure 1(c). Hence, the NNF in this figure is smooth.[3]

The properties of decomposability, determinism and smoothness lead to a number of interesting subsets of NNF.

[2]Horn theories (and renamable Horn theories) represent another target compilation subset of CNF, but we do not consider it here since we restrict our attention to complete languages only.

[3]Any sentence in NNF can be smoothed in polytime, while preserving decomposability and determinism. Preserving flatness, however, may blow-up the size of given NNF.

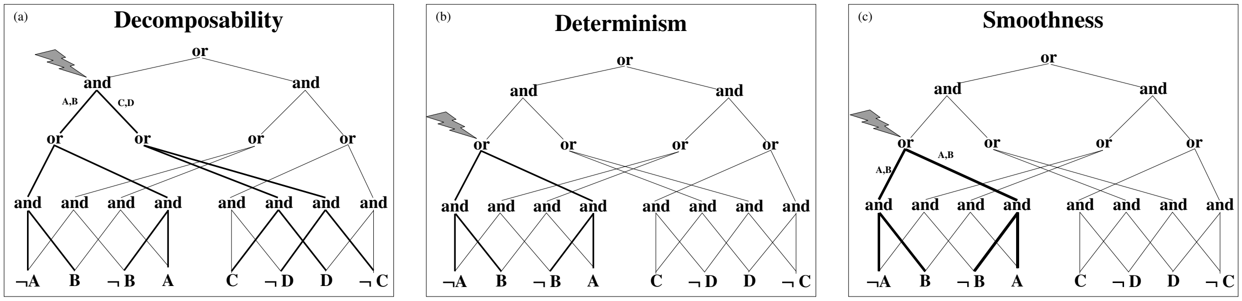


Figure 1: A sentence in NNF. Its size is 30 and height is 4.

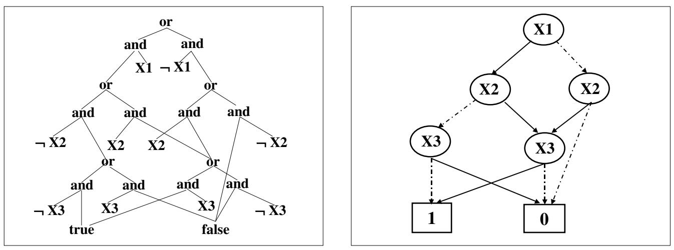


Figure 2: On the left, a sentence in the BDD language. On the right, its corresponding binary decision diagram.

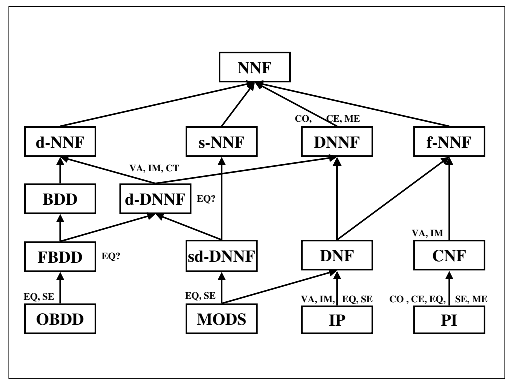


Figure 3: The set of DAG-based languages considered in this paper. An edge $\mathbf{L}_1 \rightarrow \mathbf{L}_2$ means that $\mathbf{L}_1$ is a subset of $\mathbf{L}_2$. Next to each subset, we list the polytime queries supported by the subset but not by any of its ancestors (see Section 4).

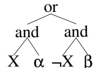
Definition 2.5 *The language* DNNF *is the subset of* NNF *satisfying decomposability;* d-NNF *is the subset satisfying determinism;* s-NNF *is the subset satisfying smoothness;* d-DNNF *is the subset satisfying decomposability and determinism; and* sd-DNNF *is the subset satisfying decomposability, determinism and smoothness.*

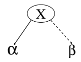
Note that DNF is a strict subset of DNNF [Darwiche, 1999a; 1999b]. The following *decision* property comes from the literature on *binary decision diagrams* [Bryant, 1986].

Definition 2.6 (Decision) *A decision node N in an* NNF *sentence is one which is labeled with true, false, or is an or-node having the form $(X \wedge \alpha) \vee (\neg X \wedge \beta)$, where X is a variable, α and β are decision nodes. In the latter case, $dVar(N)$ denotes the variable X.*

Definition 2.7 *The language* BDD *is the subset of* NNF*, where the root of each sentence is a decision node.*

The NNF sentence in Figure 2 belongs to the BDD subset.

The BDD language corresponds to *binary decision diagrams (BDDs),* as known in the formal verification literature [Bryant, 1986]. Binary decision diagrams are depicted using a more compact notation though: the labels *true* and *false* are denoted by 1 and 0, respectively; and each decision node 

is denoted by 

. The BDD sentence on the left of Figure 2 corresponds to the binary decision diagram on the right of Figure 2. Obviously enough, every NNF sentence that satisfies the decision property is also deterministic. Therefore, BDD is a subset of d-NNF.

As we show later, BDD does not qualify as a target compilation language (unless P=NP), but the following subset does.

Definition 2.8 FBDD *is the intersection of* DNNF *and* BDD*.*

That is, each sentence in FBDD is decomposable and satisfies the decision property. The FBDD language corresponds to *free binary decision diagrams,* as known in formal verification [Gergov and Meinel, 1994]. An FBDD is usually defined as a BDD that satisfies the *read-once property:* on each path from the root to a leaf, a variable can appear at most once. Read-once is equivalent to the decomposability of BDDs.[4]

A more influential subset of the BDD language is obtained by imposing the *ordering* property:

Definition 2.9 (Ordering) *Let $<$ be a total ordering on the variables PS. The language* $\mathrm{OBDD}_<$ *is the subset of* FBDD *satisfying the following property: if N and M are or-nodes, and if N is an ancestor of node M, then $dVar(N) < dVar(M)$.*

The $\mathrm{OBDD}_<$ language corresponds to the well–known *ordered binary decision diagrams* [Bryant, 1986].

The languages we have discussed so far are depicted in Figure 3, where arrows denote set inclusion. The figure contains one additional language that we have not discussed yet:

[4]FBDDs are also known as read-once branching programs.

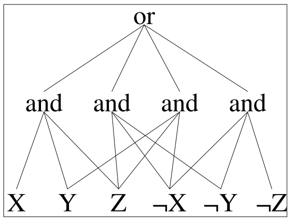


Figure 4: A sentence in language MODS.

Definition 2.10 MODS *is the intersection of* sd-DNNF *and* DNF.

Figure 4 depicts a sentence in MODS. As we show later, MODS is the most tractable NNF subset we shall consider (together with $\text{OBDD}_<$). This is not surprising since from the syntax of a sentence in MODS, one can immediately recover the sentence models.

3 On the Succinctness of Compiled Theories

We have discussed more than a dozen subsets of the NNF language. Some of these subsets are well known and have been studied extensively in the computer science literature. Others, such as DNNF [Darwiche, 1999b; 1999a] and d-DNNF [Darwiche, 2000], are relatively new. The question now is: What subset should one adopt for a particular application? As we argue in this paper, that depends on three key properties of the language: its succinctness, the class of tractable queries it supports, and the class of tractable transformations it admits.

Our goal in this and the following sections is to construct a map on which we place different subsets of the NNF language according to the above criteria. This map will then serve as a guide to system designers in choosing the target compilation language most suitable to their application. It also provides an example paradigm for studying and evaluating further target compilation languages. We start with a study of succinctness[5] in this section [Gogic *et al.*, 1995].

Definition 3.1 (Succinctness) *Let* $\mathbf{L}_1$ *and* $\mathbf{L}_2$ *be two subsets of* NNF. $\mathbf{L}_1$ *is at least as succinct as* $\mathbf{L}_2$*, denoted* $\mathbf{L}_1 \leq \mathbf{L}_2$*, iff for every sentence* $\alpha \in \mathbf{L}_2$*, there exists an equivalent sentence* $\beta \in \mathbf{L}_1$ *where the size of* β *is polynomial in the size of* α.[6]

The relation $\leq$ is clearly reflexive and transitive, hence, a pre-ordering. One can also define the relation $<$, where $\mathbf{L}_1 < \mathbf{L}_2$ iff $\mathbf{L}_1 \leq \mathbf{L}_2$ and $\mathbf{L}_2 \not\leq \mathbf{L}_1$.

Proposition 3.1 *The results in Table 1 hold.*

Figure 5 summarizes the results of Proposition 3.1 in terms of a directed acyclic graph.

A classical result in knowledge compilation states that it is not possible to compile any propositional formula α into a polysize data structure β such that: α and β entail the same set of clauses, and clausal entailment on β can be decided in time polynomial in its size, unless $\mathsf{NP} \subseteq \mathsf{P/poly}$ [Selman and Kautz, 1996; Cadoli and Donini, 1997]. This last assumption implies the collapse of the polynomial hierarchy at the second level [Karp and Lipton, 1980], which is considered very unlikely. We use this classical result from knowledge compilation in some of our proofs of Proposition 3.1, which explains why some of its parts are conditioned on the polynomial hierarchy not collapsing.

We have excluded the subsets BDD, s-NNF, d-NNF and f-NNF from Table 1 since they do not qualify as target compilation languages (see Section 4). We kept NNF and CNF though given their importance. Consider Figure 5 which depicts Table 1 graphically. With the exception of NNF and CNF, all other languages depicted in Figure 5 qualify as target compilation languages. Moreover, with the exception of language PI, DNNF is the most succinct among all target compilation languages—we know that PI is not more succinct than DNNF, but we do not know whether DNNF is more succinct than PI.

In between DNNF and MODS, there is a succinctness ordering of target compilation languages:

$$\text{DNNF} < \text{d-DNNF} < \text{FBDD} < \text{OBDD}_< < \text{MODS}.$$

DNNF is obtained by imposing decomposability on NNF; d-DNNF by adding determinism; FBDD by adding decision; and $\text{OBDD}_<$ by adding ordering. *Adding each of these properties reduces language succinctness.*

One important fact to stress here is that adding smoothness to d-DNNF does not affect its succinctness: the sd-DNNF and d-DNNF languages are equally succinct. It is also interesting to compare sd-DNNF (which is more succinct than the influential FBDD and $\text{OBDD}_<$ languages) with MODS, which is a most tractable language. Both sd-DNNF and MODS are smooth, deterministic and decomposable. MODS, however, is flat and obtains its decomposability from the stronger condition of simple-conjunction. Therefore, sd-DNNF can be viewed as the result of relaxing from MODS the flatness and

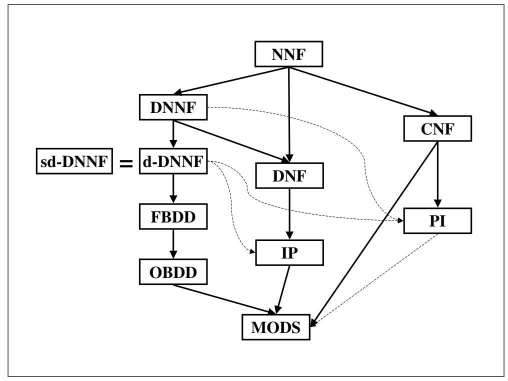


Figure 5: An edge $\mathbf{L}_1 \rightarrow \mathbf{L}_2$ indicates that $\mathbf{L}_1$ is strictly more succinct than $\mathbf{L}_2$, $\mathbf{L}_1 < \mathbf{L}_2$; $\mathbf{L}_1 = \mathbf{L}_2$ indicates that $\mathbf{L}_1$ and $\mathbf{L}_2$ are equally succinct, $\mathbf{L}_1 \leq \mathbf{L}_2$ and $\mathbf{L}_2 \leq \mathbf{L}_1$. Dotted arrows indicate unknown relationships.

[5] A more general notion of space efficiency (model preservation for polysize reductions) exists [Cadoli *et al.*, 1996], but we do not need its full generality here.

[6] We stress here that we do not require that there exists a function that computes β given α in *polytime*; we only require that a *polysize* β exists. Yet, our proofs in [Darwiche and Marquis, 2001] contain specific algorithms for computing β from α in certain cases.

L	NNF	DNNF	d-DNNF	sd-DNNF	FBDD	$\texttt{OBDD}_<$	DNF	CNF	PI	IP	MODS
NNF	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$
DNNF	$\not\leq^*$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\not\leq^*$	**?**	$\leq$	$\leq$
d-DNNF	$\not\leq^*$	$\not\leq^*$	$\leq$	$\leq$	$\leq$	$\leq$	$\not\leq^*$	$\not\leq^*$	**?**	**?**	$\leq$
sd-DNNF	$\not\leq^*$	$\not\leq^*$	$\leq$	$\leq$	$\leq$	$\leq$	$\not\leq^*$	$\not\leq^*$	**?**	**?**	$\leq$
FBDD	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$
$\texttt{OBDD}_<$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$
DNF	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\not\leq$	$\not\leq$	$\leq$	$\leq$
CNF	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\leq$	$\not\leq$	$\leq$
PI	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\not\leq$	**?**
IP	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\leq$
MODS	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$

Table 1: Succinctness of target compilation languages. $*$ means that the result holds unless the polynomial hierarchy collapses.

simple-conjunction conditions, while maintaining decomposability, determinism and smoothness. Relaxing these conditions moves the language three levels up the succinctness hierarchy, although it compromises only the polytime tests for entailment and equivalence as we show in Section 4.

4 Querying a Compiled Theory

In evaluating the suitability of a target compilation language to a particular application, the succinctness of the language must be balanced against the set of queries and transformations that it supports in polytime. We consider in this section a number of queries, each of which returns valuable information about a propositional theory, and then identify target compilation languages which provide polytime algorithms for answering such queries.[7]

The queries we consider are tests for consistency, validity, implicates (clausal entailment), implicants, equivalence, and sentential entailment. We also consider counting and enumerating theory models.[8]

¿From here on, **L** denotes a subset of language NNF.

Definition 4.1 (CO, VA) **L** *satisfies* **CO** *(***VA***) iff there exists a polytime algorithm that maps every formula* Σ *from* **L** *to* 1 *if* Σ *is consistent (valid), and to* 0 *otherwise.*

One of the main applications of compiling a theory is to enhance the efficiency of answering clausal entailment queries:

Definition 4.2 (CE) **L** *satisfies* **CE** *iff there exists a polytime algorithm that maps every formula* Σ *from* **L** *and every clause* γ *from* NNF *to* 1 *if* $\Sigma \models \gamma$ *holds, and to* 0 *otherwise.*

A key application for clausal entailment is in testing equivalence. Specifically, suppose we have a design expressed as a set of clauses $\Delta^d = \bigwedge_i \alpha_i$ and a specification expressed also as a set of clauses $\Delta^s = \bigwedge_j \beta_j$, and we want to test whether the design and specification are equivalent. By compiling each of Δ^d and Δ^s to targets Γ^d and Γ^s that support a polytime clausal entailment test, we can test the equivalence of Δ^d and Δ^s in polytime. That is, Δ^d and Δ^s are equivalent iff $\Gamma^d \models \beta_j$ for all j and $\Gamma^s \models \alpha_i$ for all i.

L	**CO**	**VA**	**CE**	**IM**	**EQ**	**SE**	**CT**	**ME**
NNF	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
DNNF	$\surd$	$\circ$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$	$\surd$
d-NNF	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
s-NNF	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
f-NNF	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
d-DNNF	$\surd$	$\surd$	$\surd$	$\surd$	**?**	$\circ$	$\surd$	$\surd$
sd-DNNF	$\surd$	$\surd$	$\surd$	$\surd$	**?**	$\circ$	$\surd$	$\surd$
BDD	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
FBDD	$\surd$	$\surd$	$\surd$	$\surd$	**?**	$\circ$	$\surd$	$\surd$
$\texttt{OBDD}_<$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
DNF	$\surd$	$\circ$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$	$\surd$
CNF	$\circ$	$\surd$	$\circ$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$
PI	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$	$\surd$
IP	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$	$\surd$
MODS	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$

Table 2: Subsets of the NNF language and their corresponding polytime queries. $\surd$ means "satisfies" and $\circ$ means "does not satisfy unless P = NP."

A number of the target compilation languages we shall consider, however, support a direct polytime equivalent test:

Definition 4.3 (EQ, SE) **L** *satisfies* **EQ** *(***SE***) iff there exists a polytime algorithm that maps every pair of formulas* Σ, Φ *from* **L** *to* 1 *if* $\Sigma \equiv \Phi$ *(*$\Sigma \models \Phi$*) holds, and to* 0 *otherwise.*

Note that sentential entailment (**SE**) is stronger than clausal entailment and equivalence. Therefore, if a language **L** satisfies **SE**, it also satisfies **CE** and **EQ**.

For completeness, we consider the following dual to **CE**:

Definition 4.4 (IM) **L** *satisfies* **IM** *iff there exists a polytime algorithm that maps every formula* Σ *from* **L** *and every term* γ *from* NNF *to* 1 *if* $\gamma \models \Sigma$ *holds, and to* 0 *otherwise.*

Finally, we consider counting and enumerating models:

Definition 4.5 (CT) **L** *satisfies* **CT** *iff there exists a polytime algorithm that maps every formula* Σ *from* **L** *to a nonnegative integer that represents the number of models of* Σ *(in binary notation).*

Definition 4.6 (ME) **L** *satisfies* **ME** *iff there exists a polynomial* $p(.,.)$ *and an algorithm that outputs all models of an arbitrary formula* Σ *from* **L** *in time* $p(n, m)$*, where* n *is the size of* Σ *and* m *is the number of its models (over variables occurring in* Σ*).*

Proposition 4.1 *The results in Table 2 hold.*

The results of Proposition 4.1 are summarized in Figure 3. One can draw a number of conclusions based on the results in

[7]We restrict our attention in this paper to the *existence* of polytime algorithms for answering queries, but we do not present the algorithms themselves. The interested reader is referred to [Darwiche, 1999b; 2000; 1999a; Bryant, 1986] for some of these algorithms and to the proofs of theorems given in [Darwiche and Marquis, 2001] for others.

[8]One can also consider computing the probability of a propositional sentence, assuming that all variables are probabilistically independent. For the subsets we consider, however, this can be done in polytime whenever models can be counted in polytime.

this figure. First, NNF, s-NNF, d-NNF, f-NNF, and BDD fall in one equivalence class that does not support any polytime queries and CNF satisfies only **VA** and **IM**; hence, none of them qualifies as a target compilation language in this case. But the remaining languages all support polytime tests for consistency and clausal entailment. Therefore, simply imposing either of smoothness (s-NNF), determinism (d-NNF), flatness (f-NNF), or decision (BDD) on the NNF language does not lead to tractability with respect to any of the queries we consider—neither of these properties seem to be significant in isolation. Decomposability (DNNF), however, is an exception and leads immediately to polytime tests for both consistency and clausal entailment, and to a polytime algorithm for model enumeration.

Recall the succinctness ordering DNNF $<$ d-DNNF $<$ FBDD $<$ $\text{OBDD}_<$ $<$ MODS from Figure 5. By adding decomposability (DNNF), we obtain polytime tests for consistency and clausal entailment, in addition to a polytime model enumeration algorithm. By adding determinism to decomposability (d-DNNF), we obtain polytime tests for validity, implicant and model counting, which are quite significant. It is not clear, however, whether the combination of decomposability and determinism leads to a polytime test for equivalence. Moreover, adding the decision property on top of decomposability and determinism (FBDD) does not appear to increase tractability with respect to the given queries[9], although it does lead to reducing language succinctness as shown in Figure 5. On the other hand, adding the ordering property on top of decomposability, determinism and decision ($\text{OBDD}_<$), leads to polytime tests for sentential entailment and equivalence.

As for the succinctness ordering NNF $<$ DNNF $<$ DNF $<$ IP $<$ MODS from Figure 5, note that DNNF is obtained by imposing decomposability on NNF, while DNF is obtained by imposing flatness and simple-conjunction (which is stronger than decomposability). What is interesting is that DNF is less succinct than DNNF, yet does not support any more polytime queries; see Figure 3. However, the addition of smoothness on top of flatness and simple-conjunction (MODS) leads to five additional polytime queries, including equivalence and entailment tests.[10]

We close this section by noting that determinism appears to be necessary (but not sufficient) for polytime model counting: only deterministic languages, d-DNNF, sd-DNNF, FBDD, $\text{OBDD}_<$ and MODS, support polytime counting. Moreover, polytime counting implies a polytime test of validity, but the opposite is not true.

[9] Deciding the equivalence of two sentences in FBDD, d-DNNF, or in sd-DNNF, can be easily shown to be in coNP. However, we do not have a proof of coNP-hardness, nor do we have deterministic polytime algorithms for deciding these problems. Actually, the latter case is quite unlikely as the equivalence problem for FBDD has been intensively studied, with no such algorithm in sight. Note, however, that the equivalence of two sentences in FBDD can be decided probabilistically in polytime [Blum *et al.*, 1980].

[10] Formally, we also need determinism to obtain MODS (see Definition 2.10). But given flatness, simple-conjunction and smoothness, we can obtain determinism by simply removing duplicated terms.

5 Transforming a Compiled Theory

A query is an operation that returns information about a theory without changing it. A transformation, on the other hand, is an operation that returns a modified theory, which is then operated on using queries. Many applications require a combination of transformations and queries.

Definition 5.1 **($\wedge$C, $\vee$C)** **L** *satisfies* $\wedge$**C** *($\vee$**C**) iff there exists a polytime algorithm that maps every finite set of formulas $\Sigma_1, \ldots, \Sigma_n$ from* **L** *to a formula of* **L** *that represents $\Sigma_1 \wedge \cdots \wedge \Sigma_n$ ($\Sigma_1 \vee \cdots \vee \Sigma_n$).*

Definition 5.2 **($\neg$C)** **L** *satisfies* $\neg$**C** *iff there exists a polytime algorithm that maps every formula Σ from* **L** *to a formula of* **L** *that represents $\neg\Sigma$.*

If a language satisfies one of the above properties, we will say that it is *closed* under the corresponding operator. Closure under logical connectives is important for two key reasons. First, it has implications on how compilers are constructed for a given target language. For example, if a clause can be easily compiled into some language **L**, then closure under conjunction implies that compiling a CNF sentence into **L** is easy. Second, it has implications on the class of polytime queries supported by the target language: If a language **L** satisfies **CO** and is closed under negation and conjunction, then it must satisfy **SE** (to test whether $\Delta \models \Gamma$, all we have to do, by the Refutation Theorem, is test whether $\Delta \wedge \neg\Gamma$ is inconsistent). Similarly, if a language satisfies **VA** and is closed under negation and disjunction, it must satisfy **SE** by the Deduction Theorem.

It is important to stress here that some languages are closed under a logical operator, only if the number of operands is bounded by a constant. We will refer to this as *bounded-closure.*

Definition 5.3 **($\wedge$BC, $\vee$BC)** **L** *satisfies* $\wedge$**BC** *($\vee$**BC**) iff there exists a polytime algorithm that maps every pair of formulas Σ and Φ from* **L** *to a formula of* **L** *that represents $\Sigma \wedge \Phi$ ($\Sigma \vee \Phi$).*

We now turn to another important transformation:

Definition 5.4 (Conditioning) *[Darwiche, 1999a] Let Σ be a propositional formula, and let γ be a consistent term. The conditioning of Σ on γ, noted $\Sigma \mid \gamma$, is the formula obtained by replacing each variable X of Σ by true (resp. false) if X (resp. $\neg X$) is a positive (resp. negative) literal of γ.*

Definition 5.5 (CD) **L** *satisfies* **CD** *iff there exists a polytime algorithm that maps every formula Σ from* **L** *and every consistent term γ to a formula from* **L** *that represents $\Sigma \mid \gamma$.*

Conditioning has a number of applications, and corresponds to *restriction* in the literature on Boolean functions. The main application of conditioning is due to a theorem, which says that $\Sigma \wedge \gamma$ is consistent iff $\Sigma \mid \gamma$ is consistent [Darwiche, 1999b; 1999a]. Therefore, if a language satisfies **CO** and **CD**, then it must also satisfy **CE**. Conditioning also plays a key role in building compilers that enforce decomposability. If two sentences Δ_1 and Δ_2 are both decomposable (belong to DNNF), their conjunction $\Delta_1 \wedge \Delta_2$ is not necessarily decomposable since the sentences may share variables. Conditioning can be used to ensure decomposability in this case since $\Delta_1 \wedge \Delta_2$ is equivalent to $\bigvee_\gamma (\Delta_1 \mid \gamma) \wedge (\Delta_2 \mid \gamma) \wedge \gamma$, where γ is a term covering all variables shared by Δ_1 and Δ_2. Note that $\bigvee_\gamma (\Delta_1 \mid \gamma) \wedge (\Delta_2 \mid \gamma) \wedge \gamma$ must be decomposable since $\Delta_1 \mid \gamma$ and $\Delta_2 \mid \gamma$ do not mention variables in γ. The previous proposition is indeed a generalization to multiple variables of the well-known Shannon expansion in the literature on Boolean functions. It is also the basis for compiling CNF into DNNF [Darwiche, 1999a; 1999b].

Another critical transformation we shall consider is that of *forgetting* [Lin and Reiter, 1994]:

Definition 5.6 (Forgetting) *Let Σ be a propositional formula, and let* $\mathbf{X}$ *be a subset of variables from PS. The* forgetting *of* $\mathbf{X}$ *from* Σ*, denoted* $\exists\mathbf{X}.\Sigma$*, is a formula that does not mention any variable from* $\mathbf{X}$ *and for every formula α that does not mention any variable from* $\mathbf{X}$*, we have $\Sigma \models \alpha$ precisely when* $\exists\mathbf{X}.\Sigma \models \alpha$*.*

Therefore, to forget variables from $\mathbf{X}$ is to remove any reference to $\mathbf{X}$ from Σ, while maintaining all information that Σ captures about the complement of $\mathbf{X}$. Note that $\exists\mathbf{X}.\Sigma$ is unique up to logical equivalence.

Definition 5.7 (FO, SFO) **L** *satisfies* **FO** *iff there exists a polytime algorithm that maps every formula Σ from* **L** *and every subset* $\mathbf{X}$ *of variables from PS to a formula from* **L** *equivalent to* $\exists\mathbf{X}.\Sigma$*. If the property holds for singleton* $\mathbf{X}$*, we say that* **L** *satisfies* **SFO**.

Forgetting is an important transformation as it allows us to focus/project a theory on a set of variables. For example, if we know that some variables $\mathbf{X}$ will never appear in entailment queries, we can forget these variables from the compiled theory while maintaining its ability to answer such queries correctly. Another application of forgetting is in counting/enumerating the instantiations of some variables $\mathbf{Y}$, which are consistent with a theory Δ. This query can be answered by counting/enumerating the models of $\exists\mathbf{X}.\Delta$, where $\mathbf{X}$ is the complement of $\mathbf{Y}$. Forgetting also has applications to planning, diagnosis and belief revision. For instance, in the SATPLAN framework, compiling away fluents or actions amounts to forgetting variables. In model-based diagnosis, compiling away every variable except the abnormality ones does not remove any piece of information required to compute the conflicts and the diagnoses of a system [Darwiche, 1999b]. Forgetting has also been used to design update operators with valuable properties [Herzig and Rifi, 1999].

Proposition 5.1 *The results in Table 3 hold.*

One can draw a number of observations regarding Table 3. First, all languages we consider satisfy **CD** and, hence, lend themselves to efficient application of the conditioning transformation. As for forgetting multiple variables, only DNNF, DNF, PI and MODS permit that in polytime. It is important to stress here that neither FBDD nor $\text{OBDD}_<$ permit polytime forgetting of multiple variables. This is noticeable since some of the recent applications of $\text{OBDD}_<$ to planning depends crucially on the operation of forgetting and it may be more suitable to use a language that satisfies **FO** in this case. Note, however, that $\text{OBDD}_<$ allows the forgetting of a single variable in polytime, but FBDD does not allow even that. d-DNNF is similar to FBDD as it satisfies neither **FO** nor **SFO**.

L	**CD**	**FO**	**SFO**	$\wedge$**C**	$\wedge$**BC**	$\vee$**C**	$\vee$**BC**	$\neg$**C**
NNF	√	∘	√	√	√	√	√	√
DNNF	√	√	√	∘	∘	√	√	∘
d-NNF	√	∘	?	√	√	?	?	?
s-NNF	√	∘	√	√	√	√	√	√
f-NNF	√	∘	√	•	√	•	√	√
d-DNNF	√	∘	∘	∘	∘	∘	∘	?
sd-DNNF	√	∘	∘	∘	∘	∘	∘	?
BDD	√	∘	√	√	√	√	√	√
FBDD	√	∘	∘	•	∘	•	∘	√
$\text{OBDD}_<$	√	∘	√	•	√	•	√	√
DNF	√	√	√	•	√	√	√	•
CNF	√	∘	√	√	√	•	√	•
PI	√	√	√	•	•	•	√	•
IP	√	•	∘	•	√	•	•	•
MODS	√	√	√	•	√	•	•	•

Table 3: Subsets of the NNF language and their polytime transformations. √ means "satisfies," • means "does not satisfy," while ∘ means "does not satisfy unless P=NP."

It is also interesting to observe that none of the target compilation languages is closed under conjunction. A number of them, however, are closed under bounded-conjunction, including $\text{OBDD}_<$, DNF, IP and MODS.

As for disjunction, the only target compilation languages that are closed under disjunction are DNNF and DNF. The $\text{OBDD}_<$ and PI languages, however, are closed under bounded disjunction. Again, the d-DNNF and FBDD languages are closed under neither.

The only target compilation languages that are closed under negation are FBDD and $\text{OBDD}_<$, while it is not known whether d-DNNF or sd-DNNF are closed under this operation. Note that d-DNNF and FBDD support the same set of polytime queries (equivalence checking is unknown for both) so they are indistinguishable from that viewpoint. Moreover, the only difference between the two languages in Table 3 is the closure of FBDD under negation, which does not seem to be that significant in light of no closure under either conjunction or disjunction. Note, however, that d-DNNF is more succinct than FBDD as given in Figure 5.

Finally, $\text{OBDD}_<$ is the only target compilation language that is closed under negation, bounded conjunction, and bounded disjunction. This closure actually plays an important role in compiling propositional theories into $\text{OBDD}_<$ and is the basis of state-of-the-art compilers for this purpose [Bryant, 1986].

6 Conclusion

The main contribution of this paper is a methodology for analyzing propositional compilation approaches according to two key dimensions: the succinctness of the target compilation language, and the class of queries and transformations it supports in polytime. The second main contribution of the paper is a comprehensive analysis, according to the proposed methodology, of more than a dozen languages for which we have produced a knowledge compilation map, which cross-ranks these languages according to their succinctness, and the polytime queries and transformations they support. This map allows system designers to make informed decisions on which target compilation language to use: after the class of queries/transformations have been decided based on the application of interest, the designer chooses the most succinct target compilation language that supports such operations in polytime. Another key contribution of this paper is the uniform treatment we have applied to diverse target compilation languages, showing how they all are subsets of the NNF language. Specifically, we have identified a number of simple, yet meaningful, properties, including decomposability, determinism, decision and flatness, and showed how combinations of these properties give rise to different target compilation languages. The studied subsets include some well known languages such as PI, which has been influential in AI; $\text{OBDD}_<$, which has been influential in formal verification; and CNF and DNF, which have been quite influential in computer science. The subsets also include some relatively new languages such as DNNF and d-DNNF, which appear to represent interesting, new balances between language succinctness and query/transformation tractability.

Acknowledgements

We wish to thank Ingo Wegener for his help with some of the issues discussed in the paper. This work has been done while the second author was a visiting researcher with the Computer Science Department at UCLA. The first author has been partly supported by NSF grant IIS-9988543 and MURI grant N00014-00-1-0617. The second author has been partly supported by IUT de Lens and the Région Nord/Pas-de-Calais.

References

[Blum *et al.*, 1980] Manuel Blum, Ashok K. Chandra, and Mark N. Wegman. Equivalence of free Boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, 10(2):80–82, 1980.

[Boufkhad *et al.*, 1997] Y. Boufkhad, E. Grégoire, P. Marquis, B. Mazure, and L. Saïs. Tractable cover compilations. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 122–127, 1997.

[Bryant, 1986] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

[Cadoli and Donini, 1997] M. Cadoli and F.M. Donini. A survey on knowledge compilation. *AI Communications*, 10:137–150, 1997 (printed in 1998).

[Cadoli *et al.*, 1996] M. Cadoli, F.M. Donini, P. Liberatore, and M. Schaerf. Comparing space efficiency of propositional knowledge representation formalisms. In *Proc. of International Conference on Knowledge Representation and Reasoning (KR'96)*, pages 364–373, 1996.

[Darwiche and Marquis, 2001] Adnan Darwiche and Pierre Marquis. A perspective on knowledge compilation. Technical Report D–116, Cognitive Systems Laboratory, UCLA, Ca 90095, 2001.

[Darwiche, 1999a] Adnan Darwiche. Compiling knowledge into decomposable negation normal form. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 284–289, 1999.

[Darwiche, 1999b] Adnan Darwiche. Decomposable negation normal form. Technical Report D–109, Cognitive Systems Laboratory, UCLA, Ca 90095, 1999. To appear in *Journal of the ACM.*

[Darwiche, 2000] Adnan Darwiche. On the tractable counting of theory models and its application to belief revision and truth maintenance. Technical Report D-113, Cognitive Systems Laboratory, UCLA, Ca 90095, 2000. To appear in *Journal of Applied Non-Classical Logics.*

[Dechter and Rish, 1994] R. Dechter and I. Rish. Directional resolution: the Davis-Putnam procedure, revisited. In *Proc. of International Conference on Knowledge Representation and Reasoning (KR'94)*, pages 134–145, 1994.

[del Val, 1994] A. del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proc. of International Conference on Knowledge Representation and Reasoning (KR'94)*, pages 551–561, 1994.

[Gergov and Meinel, 1994] J. Gergov and Ch. Meinel. Efficient analysis and manipulation of OBDD's can be extended to FBDD's. *IEEE Transactions on Computers*, 43(10):1197–1209, 1994.

[Gogic *et al.*, 1995] G. Gogic, H. Kautz, Ch. Papadimitriou, and B. Selman. The comparative linguistics of knowledge representation. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 862–869, 1995.

[Herzig and Rifi, 1999] A. Herzig and O. Rifi. Propositional belief base update and minimal change. *Artificial Intelligence*, 115(1):107–138, 1999.

[Karp and Lipton, 1980] R.M. Karp and R.J. Lipton. Some connections between non-uniform and uniform complexity classes. In *Proc. of ACM Symposium on Theory of Computing (STOC'80)*, pages 302–309, 1980.

[Khardon and Roth, 1997] R. Khardon and D. Roth. Learning to reason. *Journal of the ACM*, 44(5):697–725, 1997.

[Lin and Reiter, 1994] F. Lin and R. Reiter. Forget it! In *Proc. of the AAAI Fall Symposium on Relevance*, pages 154–159, New Orleans, 1994.

[Marquis, 1995] Pierre Marquis. Knowledge compilation using theory prime implicates. In *Proc. of International Joint Conference on Artifical Intelligence (IJCAI'95)*, pages 837–843, 1995.

[Papadimitriou, 1994] Ch. Papadimitriou. *Computational complexity*. Addison–Wesley, 1994.

[Reiter and de Kleer, 1987] Ray Reiter and Johan de Kleer. Foundations of assumption-based truth maintenance systems: Preliminary report. In *Proc. of National Conference on Artificial Intelligence (AAAI'87)*, pages 183–188, 1987.

[Schrag, 1996] R. Schrag. Compilation for critically constrained knowledge bases. In *Proc. of National Conference on Artificial Intelligence (AAAI'96)*, pages 510–515, 1996.

[Selman and Kautz, 1996] Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.