Decomposable Negation Normal Form

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

Knowledge compilation has been emerging recently as a new direction of research for dealing with the computational intractability of general propositional reasoning. According to this approach, the reasoning process is split into two phases: an off-line compilation phase and an on-line query-answering phase. In the off-line phase, the propositional theory is compiled into some target language, which is typically a tractable one. In the on-line phase, the compiled target is used to efficiently answer a (potentially) exponential number of queries. The main motivation behind knowledge compilation is to push as much of the computational overhead as possible into the off-line phase, in order to amortize that overhead over all on-line queries. Another motivation behind compilation is to produce very simple on-line reasoning systems, which can be embedded cost-effectively into primitive computational platforms, such as those found in consumer electronics.

One of the key aspects of any compilation approach is the target language into which the propositional theory is compiled. Previous target languages included Horn theories, prime implicates/implicants and ordered binary decision diagrams (OBDDs). We propose in this paper a new target compilation language, known as decomposable negation normal form (DNNF), and present a number of its properties which make it of interest to the broad community. Specifically, we show that DNNF is universal; supports a rich set of polynomial–time logical operations; is more space-efficient than OBDDs; and is very simple as far as its structure and algorithms are concerned. Moreover, we present an algorithm for converting any propositional theory in clausal form into a DNNF and show that if the clausal form has a bounded treewidth, then its DNNF compilation has a linear size and can be computed in linear time (treewidth is a graph-theoretic parameter which measures the connectivity of the clausal form). We also propose two techniques for approximating the DNNF compilation of a theory when the size of such compilation is too large to be practical. One of the techniques generates a sound but incomplete compilation, while the other generates a complete but unsound compilation. Together, these approximations bound the exact compilation from below and above in terms of their ability to answer clausal entailment queries. Finally, we show that the class of polynomial–time DNNF operations is rich enough to support relatively complex AI applications, by proposing a specific framework for compiling model-based diagnosis systems.

Additional Key Words and Phrases: Propositional logic, satisfiability, Boolean functions, knowledge compilation, model-based diagnosis.

---

1. INTRODUCTION

Knowledge compilation has been emerging recently as a new direction of research for dealing with the computational intractability of general propositional reasoning [Marquis 1995; Selman and Kautz 1996; Cadoli and Donini 1997; Khardon and Roth 1997; del Val 2000b; del Val 1994]. According to this approach, the reasoning process is split into two phases: an off-line compilation phase and an on-line query-answering phase. In the off-line phase, the propositional theory is compiled into some target language, which is typically a tractable one. In the on-line phase, the compiled target is used to efficiently answer a (potentially) exponential number of queries. The main motivation behind knowledge compilation is to push as much of the computational overhead as possible into the off-line phase, in order to amortize that overhead over all on-line queries. Another motivation behind compilation is to produce very simple on-line reasoning systems, which can be embedded cost-effectively into primitive computational platforms, such as those found in consumer electronics.

One of the key aspects of any compilation approach is the target language into which the propositional theory is compiled. Previous compilation approaches have proposed Horn theories, prime implicates/implicants, and ordered binary decision diagrams (OBDDs) as targets for such compilation [de Kleer 1990; Forbus and de Kleer 1993; Marquis 1995; Selman and Kautz 1996; Cadoli and Donini 1997; del Val 1994; Boufkhad et al. 1997; Khardon and Roth 1997; del Val 2000b; Bryant 1986; Bryant 1992]. Horn theories and prime implicates/implicants have been quite influential in AI, while OBDDs have been quite influential in the hardware verification community. A target language is typically evaluated across three dimensions: universality (whether it can accommodate every propositional theory); degree of tractability (the class of logical operations it supports in polynomial time); and space efficiency (the size of the smallest formula in the target language needed to represent a propositional theory). Each of the existing target languages can therefore be viewed as a point in this three-dimensional structure, proving more suitable for certain applications.

The focus of this paper is on a relatively new compilation target language, known as *decomposable negation normal form (DNNF),* which we introduced recently for characterizing consistency-based diagnoses [Darwiche 1999; Darwiche 1998b]. DNNF is universal; supports a rich set of polynomial–time logical operations; is more space–efficient than OBDDs; and is very simple as far as its structure and algorithms are concerned. Therefore, DNNF represents a new, interesting point on the three-dimensional structure of universality, tractability, and space–efficiency. A key characteristic of DNNF is the rich class of polynomial–time logical operations it supports, which is sufficient to implement relatively complex AI applications, such as model-based diagnosers.[1] Moreover, the space-efficiency of DNNF establishes a very significant bottom-line for the effectiveness of DNNF-based compilations, given the current success of the OBDD community in compiling quite complex propositional theories.

[1]The focus of almost all previous compilation approaches, however, has been on logical deduction and the logical operations to support that form of reasoning [Cadoli and Donini 1997], but see [del Val 1994; del Val 2000b] for some exceptions.

Our initial investigation of DNNF was in the context of model-based diagnosis, where we used it to characterize consistency-based diagnoses [Darwiche 1998b; Hamscher et al. 1992; Reiter 1987; de Kleer et al. 1992]. Specifically, we have shown that given a device model (expressed as a structured system description [Darwiche 1998b]) and given a corresponding device observation, one can compute a DNNF which completely characterizes the device health in light of the given observation. Moreover, we presented a set of tractable operations which can be applied to the resulting DNNF to answer a variety of diagnostic queries with respect to the given observation. Some of these operations are known to be intractable when applied to general propositional theories, but were shown to be tractable when applied to the constructed DNNF representation. In addition, we were able to formally bound the time and space complexity of constructing such DNNF characterizations of health, showing polynomial time complexity for a certain class of device structures [Darwiche 1998b].

Upon closer examination, however, we found that the utility of DNNF is independent of model-based diagnosis as it represents a new tractable form of propositional theories which can be used as a target compilation language in a variety of reasoning applications (including but not limited to model-based diagnosis) [Darwiche 1999]. Specifically, DNNF is supported by a number of tractable operations (some of which were identified in [Darwiche 1998b], others in [Darwiche 1999]), which represent building blocks for answering a variety of queries arising in automated reasoning applications. Any reasoning application which can be reduced to a combination of such operations can therefore be compiled into a system consisting of two simple components: a DNNF which represents the application's knowledge base, and a DNNF–evaluator which implements the necessary DNNF operations for that application. This generality of DNNFs and their operations motivated us to develop a new algorithm which converts general propositional theories in conjunctive normal form (CNF) into equivalent decomposable negation normal form (DNNF).[2] Moreover, the study of DNNF as a general target compilation language has also led to an elegant and practical framework for compiling model-based diagnostic systems. Specifically, while our previous proposal in [Darwiche 1998b] will compute a DNNF for each diagnostic query, we present in this paper a framework in which one computes off-line a single DNNF based on the device model and then uses it on-line to answer multiple diagnostic queries. Given the structural simplicity of DNNF and the algorithmic simplicity of its operations, this leads to a diagnosis framework which allows for efficient and cost-effective implementations on a variety of (primitive) software and hardware platforms.

This paper is structured as follows. We introduce in Section 2 decomposable negation normal form and its corresponding set of tractable operations. We then present in Section 3 an algorithm for compiling a propositional theory in CNF into DNNF and provide formal bounds on its time and space complexity. We also present an approximation scheme for generating approximate DNNFs in case the exact algorithm proves to be impractical in certain situations. Section 4 is then dedicated

[2] The algorithm we presented in [Darwiche 1998b] operates on a structured system description, and computes a DNNF which characterizes the state of the system health in light of an observation about the system behavior.

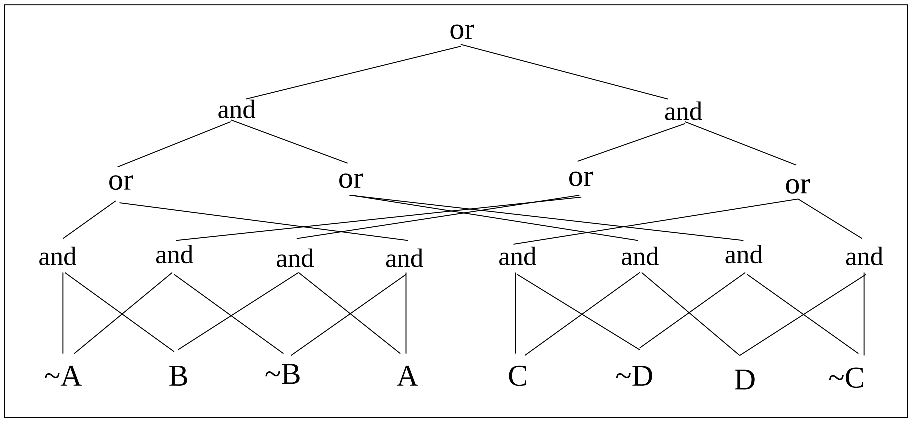

Fig. 1. A negation normal form (NNF) represented as a rooted DAG.

to the relationship between DNNFs and OBDDs, where we present a linear-time transformation from OBDDs to DNNFs and show that DNNFs are generally a more efficient representation than OBDDs. We then turn in Section 5 to one of the major application of DNNFs, where we show how a model–based diagnosis system for a given device can be compiled into two simple components: a DNNF representing the device model, and a DNNF-evaluator for answering diagnostic queries given any device observation. Finally, we close with some concluding remarks in Section 6. Proofs of all theorems are delegated to Appendix A.

2. DNNF AND ITS OPERATIONS

A propositional sentence is in negation normal form (NNF) if it is constructed from literals using only the conjoin and disjoin operators [Barwise 1977].[3] A practical representation of NNF sentences is in terms of rooted, directed acyclic graphs (DAGs), where each leaf node in the DAG is labeled with a literal, true or false; and each non-leaf (internal) node is labeled with a conjunction $\wedge$ or a disjunction $\vee$. Figure 1 depicts a rooted DAG representation of an NNF, where the children of each node are shown below it in the graph (we adopt this convention throughout the paper, therefore, eliminating the need to show the directionality of edges). The size of a DAG representation of NNF is measured by the number of edges in the DAG. All our complexity results will be based on this DAG representation.

Note that every disjunctive normal form (DNF) is an NNF, and that every conjunctive normal form (CNF) is an NNF.[4] There are NNFs, however, that are neither DNFs nor CNFs.

Our concern here is mainly with a subclass of NNF:

Definition 1. *A <u>decomposable negation normal form</u> (DNNF) is a negation normal form satisfying <u>the decomposability property</u>: for any conjunction* $\alpha = \alpha_1 \wedge \ldots \wedge$

[3]Negation normal form has also been referred to as *positive normal form*.

[4]This implies that prime implicate/implicant representations are also a strict subclass of NNF.

α_n in the form, no atom is shared between any two conjuncts of α: $\mathsf{atoms}(\alpha_i) \cap \mathsf{atoms}(\alpha_j) = \emptyset$ *for every $i \neq j$.*

The NNF in Figure 1 is decomposable. It has ten conjunctions and the conjuncts of each share no atoms. Decomposability is the property which makes DNNF tractable. We will explore this property at length later, but we first note that every DNF is also a DNNF.[5] Therefore, all properties that we shall prove of DNNF also hold for DNF. A question that may arise then is why not compile propositional theories into DNF? As it turns out, there are propositional theories that have linear DNNF representations, yet exponential DNF representations.[6] For example, consider a propositional theory Δ over n atoms, which is satisfied exactly by models in which an odd number of atoms is set to true (Δ represents the odd-parity function). The DNF representation of this theory is known to be exponential in n. However, the theory has a DNNF representation which is linear in n. Figure 1 depicts such a representation for $n = 4$.

DNNF is a highly tractable representation:

(1) Deciding the *satisfiability* of a DNNF can be done in linear time.

(2) *Conjoining* a DNNF with a set of literals can be done in linear time.

(3) *Projecting* a DNNF on some atoms can be done in linear time. Intuitively, to project a theory on a set of atoms is to compute the strongest sentence entailed by the theory on these atoms.

(4) Computing the *minimum–cardinality* of a DNNF can be done in linear time. The cardinality of a model is the number of atoms that are set to false (or true) in the model. The minimum–cardinality of a theory is the minimum–cardinality of any of its models.

(5) *Minimizing* a smooth DNNF can be done in linear time. To minimize a theory is to produce another theory whose models are exactly the minimum-cardinality models of the original theory.

(6) *Enumerating* the models of a smooth DNNF can be done in time linear in its size and quadratic in the number of its models.

Smoothness is a property we introduced recently [Darwiche 2000b]:

DEFINITION 2. *A negation normal form is <u>smooth</u> iff for every disjunction $\alpha = \alpha_1 \vee \ldots \vee \alpha_n$ in the form:* $\mathsf{atoms}(\alpha) = \mathsf{atoms}(\alpha_i)$ *for every i.*

Smoothness simplifies the statement of some DNNF operations, yet is quite easy to ensure. Any DNNF can be smoothed in $O(nm)$ time, where n is the DNNF size and m is the number of atoms it contains [Darwiche 2000b]. From here on, $\mathsf{Smooth}(\Delta)$ will denote the result of smoothing the DNNF Δ.

Therefore, as a tractable form, DNNF is distinguished by its tractability with respect to a wide range of operations, which include but are not limited to satisfiability testing. Moreover, such operations are sometimes necessary for certain

[5] We assume that in the DNF $\alpha_1 \vee \ldots \vee \alpha_n$, no atoms are shared by the literals in each α_i.

[6] The argument also applies to prime implicant representations since they are a special case of DNF representations.

applications, such as model-based diagnosis as we shall see later. Note that operations 2, 3 and 5 transform a DNNF into another DNNF, while operations 1, 4 and 6 simply return information about the corresponding DNNF. We discuss these operations in more details next.

The operations of minimum–cardinality and enumeration have been introduced in the context of model–based diagnosis, together with an implicit version of minimization [Darwiche 1998b]. For the sake of generality and completeness, we will next discuss all known tractable DNNF operations in a general propositional setting; independent of any particular application.

2.1 Literal-Conjoin and Conditioning

We present in this section two of the most fundamental operations on DNNF: the operations of conditioning and literal–conjoin.

We need the following preliminaries first. Let $\mathbf{A}$ be a set of atomic propositions (atoms) $p_1, \ldots, p_n$. An $\mathbf{A}$-sentence is a propositional sentence constructed only from the atoms $\mathbf{A}$ and constants true/false, using the conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$) operators. An $\mathbf{A}$-*literal* is either positive literal p_i or a negative literal $\neg p_i$ for some atom $p_i \in \mathbf{A}$. We will also use $\overline{\mathbf{A}}$-sentence ($\overline{\mathbf{A}}$-literal) to mean a sentence (literal) constructed from atoms not in $\mathbf{A}$. An $\mathbf{A}$–*clause* is a disjunction $l_1 \vee \ldots \vee l_n$, where l_i is either p_i or $\neg p_i$. An $\mathbf{A}$–*instantiation* is a conjunction $l_1 \wedge \ldots \wedge l_n$, where l_i is either p_i or $\neg p_i$. There is only one $\emptyset$-clause: false and only one $\emptyset$-instantiation: true. Moreover, when we say "clause" we mean an $\mathbf{A}$-clause for some $\mathbf{A}$, and when we say "instantiation" we mean an $\mathbf{A}$-instantiation for some atoms $\mathbf{A}$. Therefore, a clause is never valid and an instantiation is never inconsistent. A *clausal form* is a finite set of clauses. A *theory* is a set of sentences. We will sometimes refer to a clausal form (theory) as a sentence, to mean the propositional sentence resulting from conjoining all members of the clausal form (theory). Finally, atoms(Δ) denotes the set of atoms appearing in propositional sentence Δ.

The operation of literal-conjoin takes a DNNF Δ, an instantiation α, and returns a DNNF which is equivalent to $\Delta \wedge \alpha$. Note that even though Δ and α may each be a DNNF, $\Delta \wedge \alpha$ may not be a DNNF since Δ and α may share atoms. Literal-conjoin is achieved using the operation of conditioning [Darwiche 1998a], which incorporates the information captured by the instantiation α into the information captured by the DNNF Δ.

DEFINITION 3. *Let Δ be a propositional sentence and let α be an $\mathbf{O}$-instantiation. The conditioning of Δ on α, denoted $\Delta \mid \alpha$, is a propositional sentence obtained by replacing every $\mathbf{O}$-literal in Δ with* true *(*false*) if it is consistent (inconsistent) with instantiation α.*

For example, conditioning the DNNF $(\neg A \wedge \neg B) \vee (B \wedge C)$ on instantiation $B \wedge D$ gives $(\neg A \wedge \text{false}) \vee (\text{true} \wedge C)$ and conditioning it on $\neg B \wedge D$ gives $(\neg A \wedge \text{true}) \vee (\text{false} \wedge C)$.

Following are some important properties of conditioning:

THEOREM 1. *Let Δ and Γ be propositional sentences, and let α be an $\mathbf{O}$-instantiation. Then*

—$\Delta \mid \alpha$ is an $\overline{\mathbf{O}}$-sentence;
—$\Delta \wedge \alpha$ entails $\Delta \mid \alpha$;
—for any $\overline{\mathbf{O}}$-sentence β, $\Delta \wedge \alpha \models \beta$ only if $\Delta \mid \alpha \models \beta$.
—Δ and Γ are equivalent only if $\Delta \mid \alpha$ and $\Gamma \mid \alpha$ are equivalent.

Therefore, by conditioning a sentence Δ on a set of literals α, we are generating a sentence that does not reference the atoms in α, yet is equivalent to $\Delta \wedge \alpha$ as far as other atoms are concerned. Lemma 1 in Appendix A provides a semantics for the operation of conditioning.

PROPOSITION 1. *The DNNF language is closed under conditioning and disjunction, but is not closed under conjunction or negation.* $\square$

This leads to the following implementation of the literal–conjoin operation:

THEOREM 2. *Let Δ be a DNNF and let α be an instantiation. Then*

$$\mathsf{Conjoin}(\Delta,\alpha) \stackrel{def}{=} (\Delta \mid \alpha) \wedge \alpha$$

is a DNNF and is equivalent to $\Delta \wedge \alpha$.

It should be clear that conditioning a DNNF Δ on an instantiation can be done in time linear in the size of Δ. The operations of conditioning and literal–conjoin are most fundamental to DNNF applications as we shall see next.

2.2 Satisfiability and Entailment

We now turn to linear DNNF operations for testing satisfiability and entailment, which lead to a large class of applications for DNNF compilations. We start with a linear test for deciding the satisfiability of NNFs.

DEFINITION 4. *For NNF Δ, the predicate $\mathsf{Sat?}(\Delta)$ is defined as follows:*

(1) $\mathsf{Sat?}(\Delta) \stackrel{def}{=} \begin{cases} \textit{true}, & \textit{if } \Delta \textit{ is a literal or } \mathsf{true}; \\ \textit{false}, & \textit{if } \Delta \textit{ is } \mathsf{false}. \end{cases}$

(2) $\mathsf{Sat?}(\Delta = \bigwedge_i \alpha_i) \stackrel{def}{=}$ *true iff* $\mathsf{Sat?}(\alpha_i)$ *is true for every i.*

(3) $\mathsf{Sat?}(\Delta = \bigvee_i \alpha_i) \stackrel{def}{=}$ *true iff* $\mathsf{Sat?}(\alpha_i)$ *is true for some i.*

It should be clear that the predicate $\mathsf{Sat?}(\Delta)$ can be evaluated in time which is linear in the size of NNF Δ. The previous satisfiability test is indeed sound and complete for DNNF:

THEOREM 3. *DNNF Δ is satisfiable iff* $\mathsf{Sat?}(\Delta)$ *is true.*

Now that we have a satisfiability test, we can also define an entailment test. Specifically, to test whether DNNF Δ entails clause β, we only need to test whether $\Delta \wedge \neg\beta$ is satisfiable. Note, however, that even though each of Δ and $\neg\beta$ may be a DNNF, their conjunction $\Delta \wedge \neg\beta$ is not guaranteed to be in DNNF. We can use the literal–conjoin operation in this case and test whether $\mathsf{Conjoin}(\Delta,\beta')$ is satisfiable in linear time, where β' is an instantiation equivalent to $\neg\beta$. But it turns out that the operation of conditioning suffices for this purpose.

THEOREM 4. *For a propositional sentence Δ and instantiation α, $\Delta \mid \alpha$ is satisfiable iff $\Delta \wedge \alpha$ is satisfiable.*

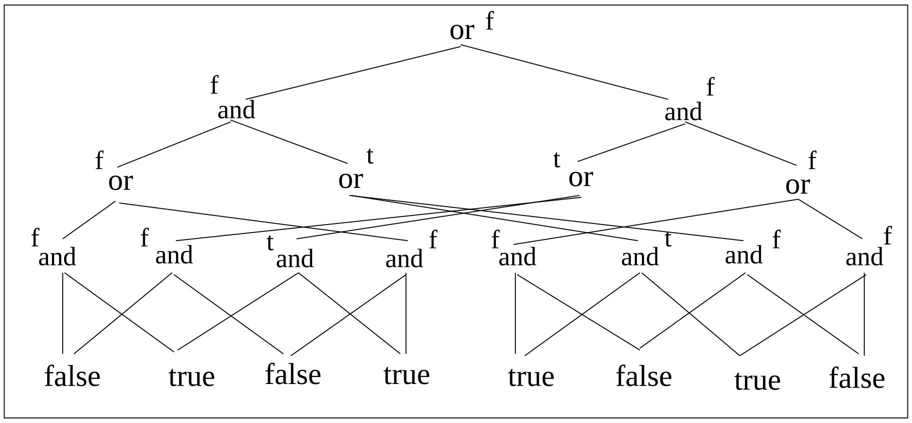

Fig. 2. A DNNF Δ, which resulted from conditioning the DNNF in Figure 1 on instantiation $A \wedge B \wedge C \wedge D$. The DNNF Δ is evaluated according to Definition 4, showing that $\mathsf{Sat?}(\Delta)$ is *false.*

Therefore, to test whether DNNF Δ entails clause β, we only need to test whether $\Delta \mid \beta'$ is satisfiable ($\beta' \equiv \neg\beta$), which is guaranteed to be a DNNF. We can now define a linear entailment test for DNNF. Actually, we will more generally define it for NNF:

Definition 5. *For NNF Δ and clause β, define $\Delta \vdash \beta$ to be true when $\mathsf{Sat?}(\Delta \mid \beta')$ is false, where β' is the instantiation equivalent to $\neg\beta$.*

This linear test is both sound and complete for DNNFs:

Theorem 5. *For DNNF Δ and clause β, $\Delta \vdash \beta$ iff $\Delta \models \beta$.*

Consider the DNNF Δ in Figure 1, the clause $\beta = \neg A \vee \neg B \vee \neg C \vee \neg D$, and suppose we want to test whether $\Delta \models \beta$. Theorem 5 suggests that we condition Δ on $\beta' = A \wedge B \wedge C \wedge D$, to yield $\Delta \mid \beta'$, and then test whether $\mathsf{Sat?}(\Delta \mid \beta')$ is false. Figure 2 depicts the conditioning of Δ on β' and the result of applying the $\mathsf{Sat?}$ test. Since $\mathsf{Sat?}(\Delta \mid \beta')$ is false, we conclude $\Delta \vdash \beta$ and also $\Delta \models \beta$.

Before we close this section, we present three important results on DNNF entailment, which will play a role later in generating approximate DNNFs.

First, the entailment test $\vdash$ is sound with respect to sentences in NNF:

Theorem 6. *For NNF Δ, $\Delta \vdash \beta$ only if $\Delta \models \beta$.*

That is, even though Δ may not be decomposable, the entailment test $\vdash$ is still sound, but not necessarily complete. Even completeness of this test, however, can be guaranteed under the following condition.

Definition 6. *NNF Δ is <u>decomposable except on atoms</u> $\mathbf{X}$ iff for any conjunction $\alpha = \alpha_1 \wedge \ldots \wedge \alpha_n$ that appears in Δ, any atom shared by the conjuncts of α belongs to $\mathbf{X}$: $\mathsf{atoms}(\alpha_i) \cap \mathsf{atoms}(\alpha_j) \subseteq \mathbf{X}$ for $i \neq j$.*

For example, the NNF $(\neg A \vee B) \wedge (\neg B \vee C)$ is decomposable except on B.

THEOREM 7. *Let Δ be an NNF which is decomposable except on atoms $\mathbf{X}$. Let β be a clause which mentions all atoms in $\mathbf{X}$. Then $\Delta \vdash \beta$ iff $\Delta \models \beta$.*

That is, even though Δ may not be decomposable, the entailment test $\vdash$ is guaranteed to be sound and complete for a certain class of queries. Consider the NNF $\Delta = (\neg A \vee B) \wedge (\neg B \vee C)$ and the queries $A \supset B$, $B \supset A$ and $A \supset C$. Since Δ is decomposable except on B, the test $\vdash$ is sound and complete with respect to the first two queries but is only sound with respect to the third query. Partial decomposability is extremely important in practice since the less decomposable a sentence is, the smaller its size will tend to be. In diagnosis applications, for example, it is possible to know beforehand what atoms will always appear in queries (such atoms are called observables). In such a case, it suffices to generate a DNNF which is decomposable except on the observables, possibly leading to significant savings in the DNNF size.

Finally, we have the following result:

THEOREM 8. *For NNF Δ, which is decomposable except on atoms $\mathbf{X}$, and for clause β, $\Delta \models \beta$ iff $\Delta \vdash \gamma_i \vee \beta$ for each $\mathbf{Y}$-clause γ_i, where $\mathbf{Y} = \mathbf{X} - \mathsf{atoms}(\beta)$.*

Therefore, if the number of atoms in $\mathbf{X}$ is bounded by a constant, then $\Delta \models \beta$ can be decided in linear time for any query β, even though Δ itself is not decomposable. One can then formally define a spectrum of decomposability, as measured by the number of shared atoms $\mathbf{X}$, with the complexity of reasoning increasing exponentially in the size of such a set.

2.3 Projection

We now turn to another key operation on DNNF, that of projection. Intuitively, to project a theory Δ on a set of atoms is to compute the strongest sentence implied by Δ on those atoms. This notion is dual to *forgetting,* as introduced in [Lin and Reiter 1994], and corresponds to the notion of *consequence,* as discussed in [Darwiche 1998b; Darwiche 1992; Darwiche and Ginsberg 1992]. Following is the formal definition of projection:

DEFINITION 7. *A projection of propositional sentence Δ on a set of atoms $\mathbf{A}$ is another sentence Γ, which satisfies the following properties:*

(1) Γ is an $\mathbf{A}$-sentence.
(2) For any $\mathbf{A}$-sentence β, $\Gamma \models \beta$ iff $\Delta \models \beta$.

Note that the projection of a sentence on a set of atoms is unique up to logical equivalence. Lemma 3 in Appendix A provides a semantics for the projection operation.

As it turns out, once a theory is converted into DNNF, projecting it on any set of atoms can be achieved in linear time:

DEFINITION 8. *For DNNF Δ and atoms $\mathbf{A}$, $\mathsf{Project}(\Delta, \mathbf{A})$ is defined as follows:*

(1) $\mathsf{Project}(\Delta, \mathbf{A}) \stackrel{def}{=} \begin{cases} \mathsf{true}, & \textit{if } \Delta \textit{ is an } \overline{\mathbf{A}}\textit{-literal;} \\ \Delta, & \textit{if } \Delta \textit{ is an } \mathbf{A}\textit{-literal, } \mathsf{true} \textit{ or } \mathsf{false}. \end{cases}$

(2) $\mathsf{Project}(\Delta = \bigwedge_i \alpha_i, \mathbf{A}) \stackrel{def}{=} \bigwedge_i \mathsf{Project}(\alpha_i, \mathbf{A})$.

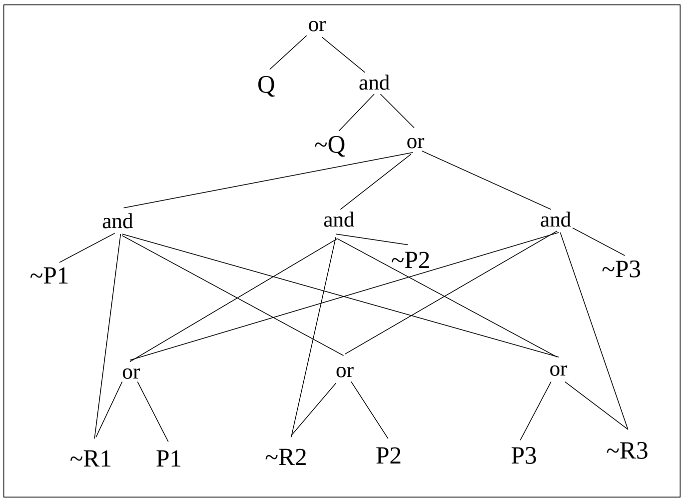

Fig. 3. A propositional sentence in DNNF.

(3) $\mathsf{Project}(\Delta = \bigvee_i \alpha_i, \mathbf{A}) \stackrel{def}{=} \bigvee_i \mathsf{Project}(\alpha_i, \mathbf{A})$.

That is, $\mathsf{Project}(\Delta, \mathbf{A})$ is simply the result of replacing each $\overline{\mathbf{A}}$-literal in Δ with true.

The following theorem shows that the above linear-time operation does correspond to projection:

THEOREM 9. *Let Δ be a DNNF and let $\mathbf{A}$ be a set of atoms. Then*

—$\mathsf{Project}(\Delta, \mathbf{A})$ *is a DNNF.*

—$\mathsf{Project}(\Delta, \mathbf{A})$ *is a projection of Δ on atoms $\mathbf{A}$ according to Definition 7.*

Consider the DNNF in Figure 3, which is equivalent to the theory $\Delta = \{P_1 \wedge P_2 \wedge P_3 \supset Q, R_1 \wedge \neg P_1 \supset Q, R_2 \wedge \neg P_2 \supset Q, R_3 \wedge \neg P_3 \supset Q\}$. Projecting this DNNF on atoms R_1, R_2, R_3, Q gives the DNNF in Figure 4 (which can be easily simplified to $R_1 \wedge R_2 \wedge R_3 \supset Q$). All we did is replace literals $P_1, \neg P_1, P_2, \neg P_2, P_3, \neg P_3$ with true.

Projection has three major applications:

(1) If we know that certain atoms will never appear in queries, then we can *compile them out* from the propositional theory using projection. Specifically, if $\mathbf{P}$ are all atoms in a theory Δ, and if $\mathbf{X}$ are atoms in $\mathbf{P}$ that will never appear in queries, then we can compile out atoms $\mathbf{X}$ by projecting the theory Δ on atoms $\mathbf{P} - \mathbf{X}$. The resulting theory is as good as the original one as far as answering the queries of interest in this case.

(2) In certain applications, such as planning and diagnosis, there is a set of atoms about which we want to collect specific information—for example, find an instantiation of such atoms which is consistent with the given theory. In planning, this set of atoms can represent action fluents and in diagnosis it can represent fault variables. A first step in collecting the necessary information about such a set of atoms is to project the given theory on this set. We will discuss this application of projection to model-based diagnosis in Section 5.

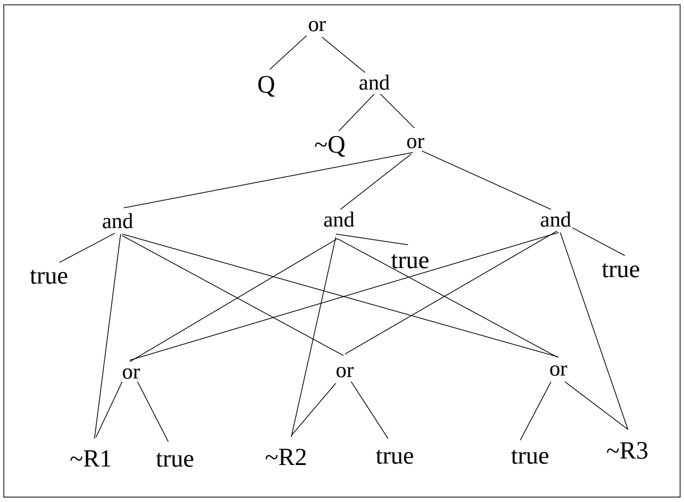

Fig. 4. Projecting a DNNF on a set of atoms.

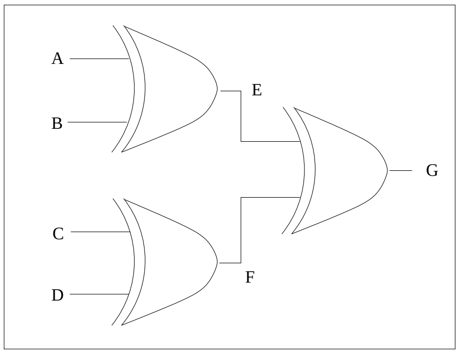

Fig. 5. An odd-parity circuit.

(3) Consider the circuit in Figure 5 which implements the odd-parity function, and let Δ be a theory representing this circuit. If we compile $\Delta \cup \{G\}$ into DNNF, and then project on atoms A, B, C and D, we obtain a DNNF representation of this Boolean function. This technique can be used to compile any circuit representation of a Boolean function into its DNNF representation.

Therefore, aside from its role as a compilation of propositional theories, DNNF constitutes yet another representation of Boolean functions, similar to DNFs, CNFs and OBDDs [Bryant 1986; Bryant 1992]. A question then arises regarding the relation between this representation and previous ones. The connection of DNNF to CNFs and DNFs was addressed in Section 2—their connection to OBDDs will be addressed in Section 4.

2.4 Computing Minimum Cardinalities

We now consider a property of propositional theories, which we call *minimum cardinality.*

Definition 9. *Let Δ be a satisfiable propositional sentence and let $\mathsf{Card}(\omega)$ be the number of atoms set to false in a truth assignment ω. The minimum cardinality of Δ is defined as $\min_{\omega \models \Delta} \mathsf{Card}(\omega)$. The minimum cardinality of an unsatisfiable sentence is defined to be ∞.*

That is, we look at all the models (satisfying truth assignments) of sentence Δ, we compute the cardinality of each of these models, and then return the minimum of these cardinalities.

The minimum cardinality of a theory has a concrete meaning in certain applications. For example, in model-based diagnosis, it can be used to compute the smallest number of faults that are conceivable given a system observation. And in planning applications, it can be used to compute the smallest number of actions needed to establish a certain goal.

Consider the DNNF in Figure 6(a), which has three models $\{okX = true, okY = false\}$, $\{okX = false, okY = true\}$ and $\{okX = false, okY = false\}$, with cardinalities 1, 1 and 2, respectively. The minimum cardinality of this DNNF is 1. From here on, we will represent a model ω by a set of assignments, $X = v$, where X is an atom and v is either *true* or *false*.

Computing the minimum cardinality of a DNNF can be done in linear time [Darwiche 1998b]:

Definition 10. *For NNF Δ, $\mathsf{MCard}(\Delta)$ is defined as follows:*

(1) $$\mathsf{MCard}(\Delta) = \begin{cases} 0, & \text{if } \Delta \text{ is a positive literal or } \mathsf{true}; \\ 1, & \text{if } \Delta \text{ is a negative literal;} \\ \infty, & \text{if } \Delta \text{ is } \mathsf{false}. \end{cases}$$

(2) $$\mathsf{MCard}(\Delta = \bigvee_i \alpha_i) = \min_i \mathsf{MCard}(\alpha_i).$$

(3) $$\mathsf{MCard}(\Delta = \bigwedge_i \alpha_i) = \sum_i \mathsf{MCard}(\alpha_i).$$

Theorem 10. $\mathsf{MCard}(\Delta)$ *is the minimum cardinality of DNNF Δ according to Definition 9.*

All of the above cases are straightforward, except possibly for the last one which requires decomposability. Figure 6 illustrates the minimum cardinality computation for two DNNFs.

2.5 Minimizing

We now turn to another tractable transformation on DNNF, which has main applications in diagnosis, planning and non-monotonic reasoning.

Definition 11. *Let Δ be a propositional sentence. A minimization of Δ is a sentence Γ such that for every truth assignment ω over the atoms of Δ, we have $\omega \models \Gamma$ iff $\omega \models \Delta$ and $\mathsf{Card}(\omega) = \mathsf{MCard}(\Delta)$.*

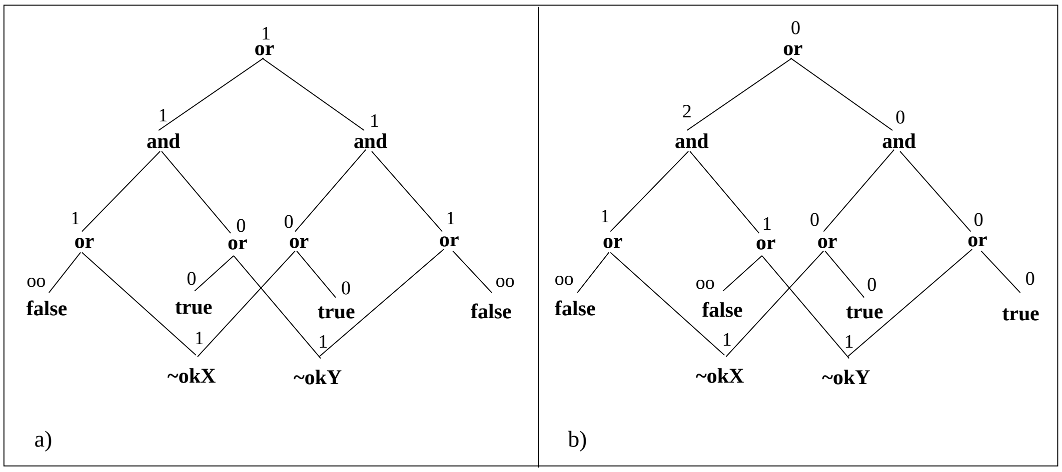

Fig. 6. Computing the minimum–cardinality of two DNNFs.

That is, the models of the minimized sentence Γ are exactly the minimum–cardinality models of Δ, assuming that models are defined over the atoms of Δ.

The operation of minimization has key applications to planning where it allows us to characterize the set of plans having a smallest number of actions, and to model-based diagnosis where it allows us to characterize diagnoses with a smallest number of faults.

Minimizing a smooth DNNF can be done in linear time:

Definition 12. *For smooth NNF Δ, define* $\mathsf{Minimize}(\Delta)$ *as follows:*

(1) $\mathsf{Minimize}(\Delta) \stackrel{def}{=} \Delta$, *if* Δ *is a literal,* true *or* false.

(2) $\mathsf{Minimize}(\Delta = \bigvee_i \alpha_i) \stackrel{def}{=} \bigvee_{\mathsf{MCard}(\alpha_i)=\mathsf{MCard}(\Delta)} \mathsf{Minimize}(\alpha_i)$.

(3) $\mathsf{Minimize}(\Delta = \bigwedge_i \alpha_i) \stackrel{def}{=} \bigwedge_i \mathsf{Minimize}(\alpha_i)$.

Theorem 11. *Let Δ be a smooth DNNF. Then*

—$\mathsf{Minimize}(\Delta)$ *is a smooth DNNF.*

—$\mathsf{Minimize}(\Delta)$ *is a minimization of Δ according to Definition 11.*

Note that applying $\mathsf{Minimize}$ to Δ affects only or-nodes as it leaves and-nodes and leaf-nodes intact. Or-nodes are affected as follows: If the minimum cardinality of an or-node (as given by Definition 10) is not equal to that of some child node, the child is disconnected from the or-node.

2.6 Enumerating Models

We now turn to the last tractable operation on DNNF that we shall discuss, that of enumerating its models $\mathsf{Models}(\Delta)$. When combined with the operation of minimization, model enumeration allows us to generate minimal-action plans in planning applications, and minimal-fault diagnoses in model-based diagnosis applications.

DEFINITION 13. *For smooth NNF Δ,* $\mathsf{Models}(\Delta)$ *is defined as follows:*

$$\text{(1) } \mathsf{Models}(\Delta) = \begin{cases} \{\{p = \mathit{true}\}\}, & \text{if } \Delta \text{ is a positive literal } p; \\ \{\{p = \mathit{false}\}\}, & \text{if } \Delta \text{ is a negative literal } \neg p; \\ \{\{\}\}, & \text{if } \Delta \text{ is } \mathsf{true}; \\ \{\}, & \text{if } \Delta \text{ is } \mathsf{false}. \end{cases}$$

$$\text{(2) } \mathsf{Models}(\Delta = \bigvee_i \alpha_i) = \bigcup_i \mathsf{Models}(\alpha_i).$$

$$\text{(3) } \mathsf{Models}(\Delta = \bigwedge_i \alpha_i) = \left\{ \bigcup_i \beta_i : \beta_i \in \mathsf{Models}(\alpha_i) \right\}.$$

THEOREM 12. *Let Δ be a smooth DNNF and let ω be a truth assignment over the atoms of Δ. Then $\omega \models \Delta$ iff $\omega \in \mathsf{Models}(\Delta)$.*

The complexity of enumerating the models of a smooth DNNF Δ is $O(mn)$, where m is the size of Δ and $n = \mid \mathsf{Models}(\Delta) \mid^2$ [Darwiche 1998b]. Therefore, if the number of models is small enough to be viewed as a constant — which is the case when one would be interested in enumerating these models — then the enumeration process takes linear time in the DNNF size.

We close this section by summarizing the tractable transformations and queries we have defined on DNNF. We have defined four tractable transformations: conditioning, literal-conjoin, projection and minimization. Each of these transformations modifies the DNNF in polynomial time, yielding a DNNF as a result. We have also defined three tractable queries: satisfiability, minimum-cardinality and model-enumeration. Each of these queries returns valuable information about the DNNF while leaving its structure intact. Note that although we have defined all of these transformations and queries for sentential DNNF, they can be easily phrased with respect to DAG DNNF, while maintaining their linear-time complexity.

3. COMPILING PROPOSITIONAL THEORIES INTO DNNF

We have identified decomposable negation normal form in the previous section, and discussed a number of powerful, tractable logical operations on it. Our goal in this section is two fold. First, to prove that every propositional theory can be expressed in DNNF. Second, to provide an algorithm for this purpose and formally bound its time and space complexity. We will indeed follow our algorithm by two approximation algorithms, which allow us to reduce the computational complexity of DNNF generation, but at the expense of compromising their soundness or completeness with respect to clausal entailment queries.

3.1 Exact Compilation

That DNNF is a universal representation of propositional theories follows from the fact that DNF is a subset of DNNF. The following theorem, however, provides a more constructive proof of this universality.

THEOREM 13. *Let Δ_1 and Δ_2 be two DNNFs and let $\mathbf{X} = \mathsf{atoms}(\Delta_1) \cap \mathsf{atoms}(\Delta_2)$. Let Δ be the sentence $\bigvee_\beta (\Delta_1 \mid \beta) \wedge (\Delta_2 \mid \beta) \wedge \beta$, where β is an $\mathbf{X}$-instantiation. Then Δ is a DNNF and Δ is equivalent to $\Delta_1 \wedge \Delta_2$.*

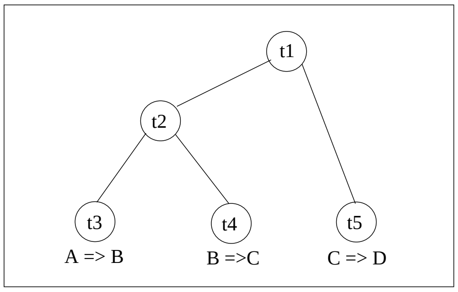

Fig. 7. A decomposition tree for the clausal form $\Delta = \{A \supset B, B \supset C, C \supset D\}$.

Here is a recursive algorithm $\mathsf{dnnf1}(\Delta)$, based on the above theorem, which converts any clausal form Δ into an equivalent DNNF:

(1) If Δ contains a single clause α, $\mathsf{dnnf1}(\Delta) \leftarrow \alpha$.

(2) Otherwise, $\mathsf{dnnf1}(\Delta) \leftarrow \bigvee_\beta \mathsf{dnnf1}(\Delta_1 \mid \beta) \wedge \mathsf{dnnf1}(\Delta_2 \mid \beta) \wedge \beta$, where Δ_1 and Δ_2 is a partition of the clauses in Δ, and β is an instantiation of the atoms shared by Δ_1 and Δ_2.[7]

This algorithm converts any theory in clausal form into an equivalent DNNF, but at the expense of increasing the theory size. The increase in size comes mainly from the case analysis performed on the atoms shared by the sub-theories Δ_1 and Δ_2. Consider the clausal form $\Delta = \{A \supset B, B \supset C\}$ and let $\Delta_1 = \{A \supset B\}$ and $\Delta_2 = \{B \supset C\}$. Then $\mathsf{dnnf1}(\Delta) = (\mathsf{dnnf1}(\Delta_1 \mid B) \wedge \mathsf{dnnf1}(\Delta_2 \mid B) \wedge B) \vee (\mathsf{dnnf1}(\Delta_1 \mid \neg B) \wedge \mathsf{dnnf1}(\Delta_2 \mid \neg B) \wedge \neg B)$, which simplifies to $(C \wedge B) \vee (\neg A \wedge \neg B)$.

We have two key observations about the above procedure. First, the size of resulting DNNF is sensitive to the way we split the theory Δ into two sub-theories Δ_1 and Δ_2. Second, the above procedure is not deterministic since it does not specify how to split the theory Δ into two sub-theories. To make the procedure deterministic, we will utilize a decomposition tree, which represents a recursive partitioning of the clauses in Δ.

Definition 14. *A decomposition tree T for clausal form Δ is a binary tree whose leaves correspond to the clauses in Δ. If t is the leaf node in T corresponding to clause α in Δ, then $\Delta(t) \stackrel{def}{=} \{\alpha\}$.*

Figure 7 depicts a decomposition tree for the theory Δ which contains the clauses $A \supset B$, $B \supset C$ and $C \supset D$.

For a decomposition tree to be useful computationally, we need to associate some information with each of its nodes. First, for every internal node t: t_l and t_r denote the left and right children of t, respectively, and $\Delta(t) \stackrel{def}{=} \Delta(t_l) \cup \Delta(t_r)$. Second, $\mathsf{atoms}(t)$ is defined as the set of atoms appearing in clauses $\Delta(t)$. For example, in Figure 7, $\Delta(t_2)$ contains the clauses $A \supset B$ and $B \supset C$ and $\mathsf{atoms}(t_2)$ contains the atoms A, B and C. Finally, $\mathsf{atoms}^\uparrow(t)$ is defined as the set of atoms associated

[7] To condition a clausal form Δ on β is to condition each clause in Δ on β.

Algorithm dnnf1

```
/* t is a node in decomposition tree */
/* α is an instantiation */

dnnf1(t, α)
 if t is a leaf node and Δ(t) = {φ}, then γ←φ | α
 else γ← ⋁_β dnnf1(t_l, α ∧ β) ∧ dnnf1(t_r, α ∧ β) ∧ β
       where β ranges over all instantiations
       of atoms(t_l) ∩ atoms(t_r) − atoms(α)
 return γ
```

Fig. 8. Compiling a clausal form into DNNF.

with leaf nodes that are not in the subtree rooted at t. For example, in Figure 7, $\mathsf{atoms}^{\uparrow}(t_2) = \{C, D\}$ and $\mathsf{atoms}^{\uparrow}(t_4) = \{A, B, C, D\}$.

Given a decomposition tree for clausal form Δ, Figure 8 depicts an algorithm which compiles Δ into DNNF.

THEOREM 14. *The call* $\mathsf{dnnf1}(t, \alpha)$ *in Figure 8 returns a DNNF equivalent to* $\Delta(t) \mid \alpha$.

Therefore, to convert a clausal form Δ into DNNF, we first construct a decomposition tree T for Δ and call $\mathsf{dnnf1}(t, \mathsf{true})$ with t being the root of tree T. The following is an important observation about dnnf1:

THEOREM 15. *If instantiations α and α' agree on* $\mathsf{atoms}(t)$*, then* $\mathsf{dnnf1}(t, \alpha)$ *is equivalent to* $\mathsf{dnnf1}(t, \alpha')$.

Therefore, we can improve on dnnf1 by associating a cache with each node t to store the result of $\mathsf{dnnf1}(t, \alpha)$ indexed by the subset of instantiation α pertaining to $\mathsf{atoms}(t)$, denoted $\mathsf{subinst}(\alpha, \mathsf{atoms}(t))$. When another recursive call $\mathsf{dnnf1}(t, \alpha')$ is made, we first check the cache of node t to see whether we have an entry for $\mathsf{subinst}(\alpha', \mathsf{atoms}(t))$. If we do, we return it. Otherwise, we continue with the recursion. This improvement leads to the refined algorithm in Figure 9.

We now address the complexity of dnnf2.

DEFINITION 15. *Let t be a node in a decomposition tree T. The <u>cluster</u> of node t is defined as follows:*

—*If t is a leaf node, then its cluster is* $\mathsf{atoms}(t)$.

—*If t is an internal node, then its cluster is*

$$(\mathsf{atoms}(t) \cap \mathsf{atoms}^{\uparrow}(t)) \cup (\mathsf{atoms}(t_l) \cap \mathsf{atoms}(t_r)).$$

The <u>width</u> of a decomposition tree is the size of its maximal cluster minus one.

In Figure 7, we have $\mathsf{cluster}(t_1) = \{C\}$, $\mathsf{cluster}(t_2) = \{B, C\}$, $\mathsf{cluster}(t_3) = \{A, B\}$, $\mathsf{cluster}(t_4) = \{B, C\}$ and $\mathsf{cluster}(t_5) = \{C, D\}$. Therefore, the decomposition tree has width 1.[8]

[8]We have recently proposed decomposition trees for driving inference in Bayesian networks [Darwiche 2001], where a decomposition tree is defined for a Bayesian network and is also equipped with a notion of width.

Algorithm dnnf2

```
/* t is a node in decomposition tree */
/* α is an instantiation */
/* subinst(α, atoms(t)) is the subset of instantiation α pertaining to atoms(t) */
/* cache_t(.) is initialized to nil */

dnnf2(t, α)
 ψ←subinst(α, atoms(t))
 if cache_t(ψ) ≠ nil, return cache_t(ψ)
 if t is a leaf node and Δ(t) = {φ}, then γ←φ | α
 else γ← ⋁_β dnnf2(t_l, α ∧ β) ∧ dnnf2(t_r, α ∧ β) ∧ β
        where β ranges over all instantiations
        of atoms(t_l) ∩ atoms(t_r) − atoms(α)
 cache_t(ψ)←γ
 return γ
```

Fig. 9. Compiling a theory into DNNF.

THEOREM 16. *Let T be a decomposition tree for a clausal form Δ, where w is the width of T and n is the number of clauses in Δ. The call* $\mathsf{dnnf2}(t, \mathsf{true})$ *in Figure 9, where t is the root of tree T, takes $O(nw2^w)$ time and space.*

Therefore, the complexity of compiling a propositional theory into DNNF depends crucially on the quality (width) of the decomposition tree used. The question now is how to construct good decomposition trees (ones with small width). Given an ordering of the atoms in clausal form Δ, we will now describe a simple algorithm for constructing a decomposition tree for Δ. We will then provide an upper bound on the width of constructed decomposition tree.

The algorithm is given in Figure 10. We start by creating a binary tree containing a single node t for each clause α in Δ, and associate α with it: $\Delta(t) = \{\alpha\}$. We then iterate over the atoms of Δ according to the given order π. When considering an atom X, we combine (arbitrarily) all existing binary trees that contain X into a single binary tree. After having processed all atoms, we again combine all remaining trees into a single tree which we then return.

To bound the width of decomposition trees that Algorithm el2dt constructs, we need the following definition.

DEFINITION 16. *[Dechter and Rish 1994] Let Δ be a propositional theory in clausal form. The <u>interaction graph</u> for Δ is the undirected graph G constructed as follows. The nodes of G are the atoms of Δ. There is an edge between two atoms in G iff the atoms appear in the same clause of Δ.*

Given an undirected graph G, and given an ordering π of the nodes in G, one can define the *treewidth* of ordering π with respect to G (also known as the induced width of π with respect to G [Dechter 1992; Dechter and Rish 1994]).

DEFINITION 17. *Let G be an undirected graph and let π be an ordering of the nodes in G. To eliminate a node from G is to pairwise connect all its neighbors and then remove it from G. The <u>treewidth</u> of ordering π with respect to graph G is determined as follows. If nodes are eliminated from G according to ordering π,*

Algorithm el2dt

```
/* Δ is a clausal form */
/* π is an ordering π(1), π(2), ... of the atoms in Δ */
/* compose(Γ) combines trees in Γ arbitrarily into a single binary tree: */
/* -compose({T}) = T. */
/* -compose({T1, T2}) is a binary tree with a root node having T1 and T2 as its children. */
/* -compose({T1, ..., Tn}) = compose({T1, compose({T2, ..., Tn})}), where n > 2. */

el2dt(Δ, π)
 Σ←{Tα : α ∈ Δ}, where Tα is a decomposition tree containing a single node t and Δ(t) = {α}
 for i = 1 to length of π:
   Γ←{T : T ∈ Σ, atom π(i) appears in tree T}
   Σ←(Σ \ Γ) ∪ {compose(Γ)}
 return compose(Σ)
```

Fig. 10. Constructing a decomposition tree for a clausal form.

then the treewidth of π is the maximum number of neighbors that any node has just before it is eliminated.

We have the following guarantee:

THEOREM 17. *Let Δ be a clausal form, G be its interaction graph, π be an ordering of the nodes in G (atoms in Δ), and w be the treewidth of ordering π with respect to graph G. The call* el2dt(Δ, π) *in Figure 10 returns a decomposition tree for Δ of width $\leq w$.*

We do not prove this theorem here, but direct the reader to a very similar proof in [Darwiche 2001].

The treewidth of the best ordering π for a graph G (one with the smallest treewidth) is known as the *treewidth* of G [Arnborg et al. 1987; Arnborg 1985; Robertson and Seymour 1986; Bodlaender 1993] (also known as the induced width of G [Dechter 1992; Dechter and Rish 1994]). If the treewidth of a graph G is bounded by a constant k, then one can construct an optimal ordering for graph G in linear time [Bodlaender 1996].[9] A major implication of this result is that if a clausal form Δ has an interaction graph with a bounded treewidth, then computing an optimal decomposition tree for that theory, and compiling the theory based on the computed decomposition tree can all be done in linear time.

We stress, however, that the interaction graph of a theory may not have a bounded treewidth, yet the theory may have a polynomial compilation into DNNF. Consider the theory $\Delta = \{P_1 \wedge \ldots \wedge P_n \supset Q, R_1 \wedge \neg P_1 \supset Q, \ldots, R_n \wedge \neg P_n \supset Q\}$. The interaction graph of this theory has treewidth n, yet it has a DNNF compilation of size $O(n^2)$ (shown in Figure 3 for $n = 3$).

The theory we just considered is not equivalent to any Horn theory. In fact, even its Horn approximation is known to be exponential in n [Selman and Kautz 1996]. This shows that there are theories with exponential Horn approximations,

[9]The construction algorithm, however, is only practical for small k since it has a constant factor that is a super-exponential in k.

yet polynomial DNNF representations.[10]

We close this section by observing that the complexity of the algorithm in Figure 9 depends only on the interaction graph of the theory Δ; that is, it does not depend on its logical content. This is clearly not optimal in certain cases, as we have seen above, since the logical content of a theory may play a crucial role in determining its DNNF size. Explicit algorithms that utilize the logical content of a theory are the subject of our current research and are outside the scope of this paper. Our results in Section 4, however, can be viewed as providing an implicit class of such algorithms which is obtained from algorithms for constructing OBDDs.

3.2 Approximate Compilation

What if we have a theory Δ for which the best decomposition tree has an unacceptable width? We have two key choices to address this problem. First, we can try to improve on algorithm dnnf2. Second, we can try to generate an approximate compilation, which is the direction we shall peruse in this section.

Consider the algorithm dnnf2 in Figure 9. It should be clear that the reason for possible intractability in this algorithm is the size of $\mathcal{S}(t,\alpha) \stackrel{def}{=} \mathsf{atoms}(t_l) \cap \mathsf{atoms}(t_r) - \mathsf{atoms}(\alpha)$, which contains some of the atoms shared by sub-theories $\Delta(t_l)$ and $\Delta(t_r)$. Specifically, the algorithm will consider a number of instantiations β which is exponential in the size of $\mathcal{S}(t,\alpha)$. Therefore, we can control the size of resulting compilation by reducing the number of instantiations considered. We can do this in two ways:

(1) *Ignoring atoms:* We can ignore some of the atoms in $\mathcal{S}(t,\alpha)$ by performing a case analysis on only a subset of $\mathcal{S}(t,\alpha)$. That is, we consider all instantiations β of a <u>subset</u> of $\mathcal{S}(t,\alpha)$. This leads to a variation on algorithm dnnf2 which we call dnnf_u.

(2) *Ignoring instantiations:* We can ignore some of the instantiations β. That is, we only consider <u>some</u> instantiations β of $\mathcal{S}(t,\alpha)$. This leads to a variation on algorithm dnnf2 which we call dnnf_l.

In either case, we can control the size of resulting compilation and to the degree we wish. In fact, using either technique we can ensure a linear-time compilation if we decide to ignore enough atoms or instantiations. This leaves two questions. First, what atoms or instantiations should we ignore? Second, what can we guarantee about the resulting compilations?

The choice of atoms or instantiations to ignore is typically heuristic and will not be addressed in this paper. We only address the second question here.

THEOREM 18. $\mathsf{dnnf}_u(t,\alpha)$ *is an NNF and is equivalent to* $\mathsf{dnnf2}(t,\alpha)$.

That is, ignoring atoms preserves equivalence to the exact compilation, but compromises the decomposability property. The more atoms we ignore, the less decomposable the approximation is. But in all cases, the compilation generated by dnnf_u is sound, although not necessarily complete, with respect to entailment queries.

COROLLARY 1. $\mathsf{dnnf}_u(t,\alpha) \vdash \beta$ *only if* $\mathsf{dnnf2}(t,\alpha) \vdash \beta$.

[10]This theory is renamable Horn [del Val 2000a]. One can modify it slightly, however, so it is no longer renamable Horn without affecting the polynomial size of its DNNF representation.

Here is the guarantee about the second approximation:

THEOREM 19. $\mathsf{dnnf}_l(t,\alpha)$ *is a DNNF and* $\mathsf{dnnf}_l(t,\alpha) \models \mathsf{dnnf2}(t,\alpha)$.

That is, ignoring instantiations preserves the decomposability property but could lead to strengthening the compilation. The more instantiations we ignore, the stronger the approximate compilation is. But in all cases, the compilation generated by dnnf_l is complete, although not necessarily sound, with respect to entailment.

COROLLARY 2. $\mathsf{dnnf}_l(t,\alpha) \not\vdash \beta$ *only if* $\mathsf{dnnf2}(t,\alpha) \not\vdash \beta$.

Therefore, if the size of a DNNF compilation Γ is too large, we can replace it with two approximations Γ_l and Γ_u. Given an entailment query β, we first test whether $\Gamma_l \vdash \beta$ and $\Gamma_u \vdash \beta$. We have three possibilities: If $\Gamma_l \not\vdash \beta$, then $\Delta \not\models \beta$. If $\Gamma_u \vdash \beta$, then $\Delta \models \beta$. If $\Gamma_l \vdash \beta$ and $\Gamma_u \not\vdash \beta$, then the approximations are not good enough to answer this query. Note that the case $\Gamma_l \not\vdash \beta$ and $\Gamma_u \vdash \beta$ is impossible.

The bounds Γ_l and Γ_u are inspired by the lower and upper Horn approximations proposed in [Selman and Kautz 1996]. In the previous approach, however, these bounds are crucial since not every theory has a Horn representation. In our case, however, the approximations are only meant to address intractability; our compilation approach would continue to be meaningful without them.

4. ON THE RELATIONSHIP BETWEEN DNNFS AND BDDS

Decomposable negation normal form and its DAG representation constitute yet another representation of Boolean functions, similar to DNFs, CNFs and BDDs (Binary Decision Diagrams) [Bryant 1986; Bryant 1992]. A question then arises regarding the relationship between this representation and previous ones. The connection of DNNF to CNFs and DNFs was addressed in Section 2, so we concentrate in this section on their connection to BDDs. There are two key results in this section:

(1) Any OBDD (and more generally FBDD) can be translated into an equivalent DAG DNNF in linear time.
(2) There are Boolean functions which admit polynomial DNNF representations, yet admit only exponential representations using FBDDs.

These results clearly have a significant implication on the scalability of DNNF compilations:

(1) Any successful algorithm for compiling FBDDs is immediately a successful algorithm for compiling DNNFs;
(2) It is reasonable to expect that algorithms for DNNF compilations will scale up better than corresponding algorithms for OBDDs/FBDDs.

We start by the formal definitions of BDDs, OBDDs, and FBDDs.

DEFINITION 18. *A binary decision diagram (BDD) over a set of binary variables* $\mathbf{X} = \{X_1, \ldots, X_n\}$ *is a directed acyclic graph with one root and at most two leaves labeled* 0 *and* 1. *Each non-leaf node* m *is labeled by a variable* $\mathsf{var}(m) \in \mathbf{X}$ *and has two outgoing edges labeled* 0 *and* 1, *where* $\mathsf{low}(m)$ *and* $\mathsf{high}(m)$ *denote the nodes pointed to by these edges, respectively. The computation path for input* $(a_1, \ldots, a_n)$,

```
Algorithm fbdd2dnnf

/* m is a node in an FBDD */
/* fbdd2dnnf(m) returns the root of a DAG DNNF */
/* cache(m) is initialized to nil */
/* true is a leaf node labeled with true */
/* false is a leaf node labeled with false */
/* leaf(L) returns a leaf node labeled with L */
/* or(n1,n2) returns a ∨-node, with n1,n2 as children */
/* and(n1,n2) returns a ∧-node, with n1,n2 as children */

fbdd2dnnf(m)
 if cache(m) ≠ nil, return cache(m)
 if m is a leaf node labeled with 1, then γ←true
  else if m is a leaf node labeled with 0, then γ←false
    else γ←or(and(fbdd2dnnf(low(m)), leaf(¬Xi)),
              and(fbdd2dnnf(high(m)), leaf(Xi)))
         where var(m) = Xi
 cache(m)←γ
 return γ
```

Fig. 11. Converting an FBDD into a DAG DNNF.

where $a_i \in \{0,1\}$*, is defined as follows. One starts at the root. At inner node* m*, where* $\mathsf{var}(m) = X_i$*, one moves to node* $\mathsf{low}(m)$ *if* $a_i = 0$ *and to node* $\mathsf{high}(m)$ *otherwise. The BDD represents the Boolean function* f *if the computation path for each input* $(a_1,\ldots,a_n)$ *leads to the leaf node labeled with* $f(a_1,\ldots,a_n)$*.*

The size of a BDD is measured by the number of nodes it contains.

DEFINITION 19. *A binary decision diagram is called a free BDD (FBBD) if on each computation path each variable is tested at most once. A free BDD is called an ordered BDD (OBDD) if on each computation path the variables are tested in the same order.*

OBDDs constitute a strict subclass of FBDDs; the two classes are known to be separated by the *hidden weighted bit* function [Bryant 1991]. Specifically, this function is known to have an exponential representation using OBDDs, regardless of the variable ordering, yet it has a polynomial representation using FBDDs [Gergov and Meinel 1994; Sieling and Wegener 1995].

OBDDs have received much consideration in the verification literature where they are used to test the equivalence between a specification of a Boolean function and its circuit implementation. OBDDs permit such a test to be performed in polynomial time, which is why they have been quite popular in this regard. They also permit satisfiability to be tested in linear time (similar to DNNFs).

As it turns out, every Boolean function which can be represented by an FBDD of size n, can also be represented by a DAG DNNF of size $O(n)$:

THEOREM 20. *Let* Γ *be an FBDD of size* n *which represents the Boolean function* f*. The FBDD* Γ *can be converted in* $O(n)$ *time into a DAG DNNF which represents the same function* f*.*

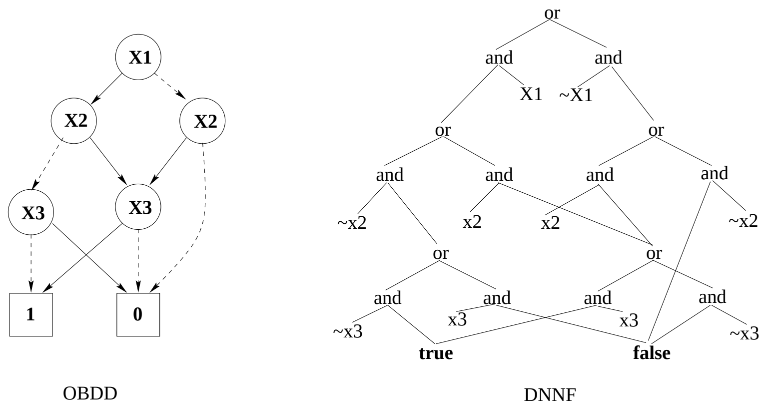

Fig. 12. An OBDD and its corresponding DNNF, which are equivalent to $(X_2 \wedge X_3) \vee (X_1 \wedge \neg X_2 \wedge \neg X_3)$. Each internal OBDD node and its two outgoing edges are transformed into 5 DNNF nodes and 6 edges.

Figure 11 depicts a recursive algorithm for converting an FBDD into a DAG DNNF. Figure 12 depicts an OBDD and its corresponding DNNF as generated by the algorithm of Figure 11.

As a representation of Boolean functions, however, the DNNF representation turns out to be more space-efficient than the FBDD representation:

THEOREM 21. *There exists a Boolean function which does not have a polynomial FBDD representation, yet has a polynomial DNNF representation.*

A function is given in [Bolling and Wegener 1998], which is shown to have an exponential FBDD representation, yet a polynomial DNF representation. Since every DNF is also a DNNF, the above theorem follows immediately. Note that this theorem does not require a DAG representation of DNNF, only a sentential one.

A question that arises then is: If DNNF are more space-efficient representations than OBDDs, why not use them instead for solving the verification problem? As it turns out, the representational win that one achieves by moving from OBDDs to DNNFs comes at a computational expense. In particular, testing the equivalence of two DNNFs cannot be achieved in polynomial time. Realize that DNFs are a special case of DNNFs and testing the equivalence of DNFs is known to be co-NP-complete. Therefore, although one obtains a more compact representation by using DNNFs, one looses the ability to test equivalence in polynomial time. And it is this ability that makes OBDDs interesting to the verification community.

Note, however, that although testing equivalence is important for verification, it is not as crucial for some AI applications. As we have illustrated earlier, DNNFs still support a number of key operations in polynomial time, such as satisfiability, minimum–cardinality and projection. Such operations can be viewed as building–block operations for some key AI applications, such as model–based diagnosis which we discuss next. Therefore, it appears that the demands imposed by verification applications are stricter than those imposed by some AI applications for which DNNFs and their tractable operations appear to be sufficient.

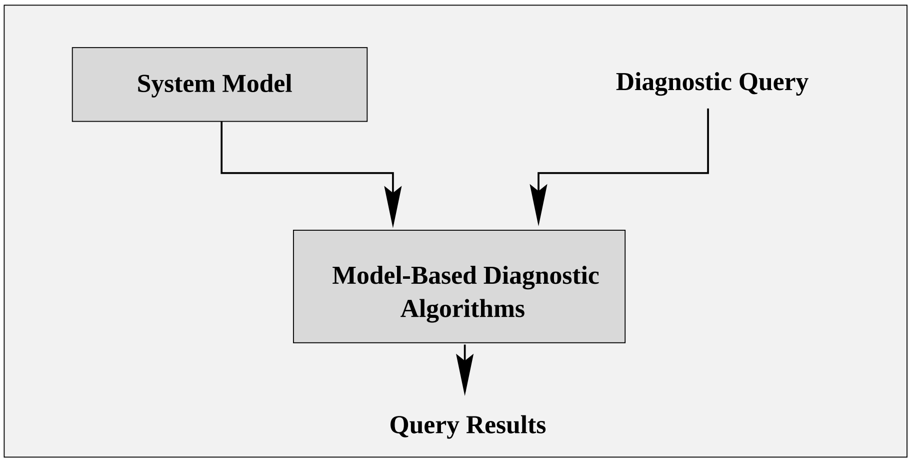

Fig. 13. The architecture of a model-based diagnosis system.

5. APPLICATION TO MODEL-BASED DIAGNOSIS: COMPILING DEVICES

Our goal in this section is to apply the DNNF theory developed in the previous sections to the problem of compiling model–based diagnosis systems. We will start with an introduction to model-based diagnosis and the need for compilation and then follow with sections that discuss: a concrete compilation example, a diagnostic compiler, a corresponding diagnostic evaluator, and the computational complexity of compilation.

5.1 Model-Based Diagnosis

The field of model-based diagnosis has been concerned with the automatic derivation of system diagnostics from formal system models [Hamscher et al. 1992]. To diagnose a system, one develops a system model and feeds it into a diagnostic engine which processes such a model to provide answers to various diagnostic queries as shown in Figure 13. Most of the initial research on model-based diagnosis has concentrated on modeling devices, on the semantics of diagnostic queries and on algorithms for answering these queries.

Recently, a new direction of research in model-based diagnosis has emerged in which the diagnostic process is further refined as given in Figure 14. Instead of operating directly on the system model to compute answers to diagnostic queries, one compiles the model into an intermediate representation based on a query schema. Diagnostic queries can then be answered by operating directly on the compiled representation.

The main motivation behind the compilation process is to divide computational work into two phases: off-line and on-line. In the off-line phase, one uses a *diagnostic compiler* to generate a compiled representation of the device. In the on-line phase, one uses a *diagnostic evaluator* to answer multiple diagnostic queries based on the same compiled representation [Darwiche 2000a]. For such a compilation scheme to be useful, however, the device compilation is expected to have a very simple structure and its associated evaluator is expected to be very simple algorithmically

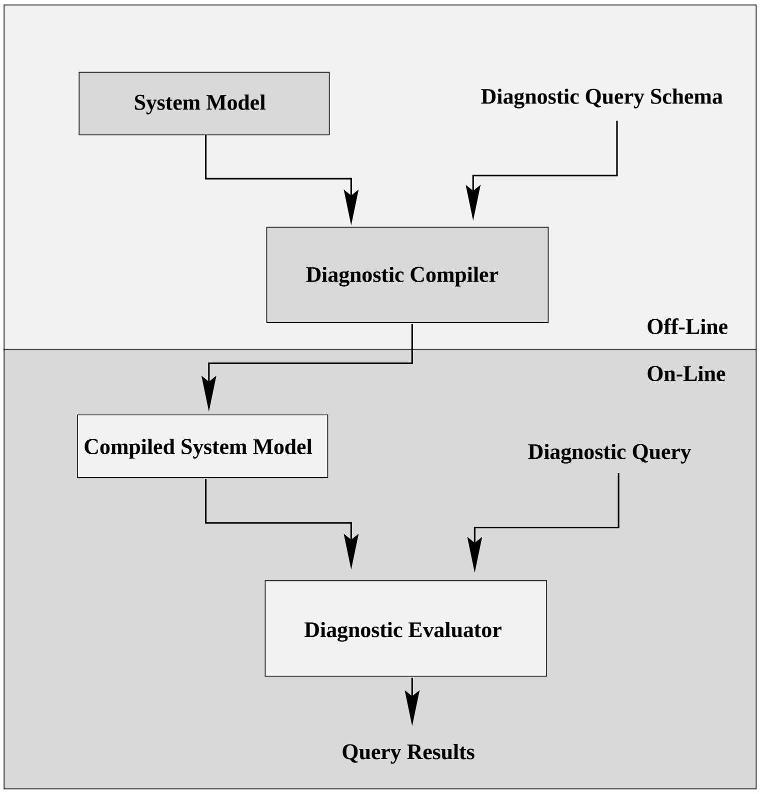

Fig. 14. The architecture of a model-based approach for compiling devices.

and to have a low computational complexity in the size of compiled representation. The main practical value of compiling a device is in:

(1) enhancing the efficiency of the on-line diagnostic system by pushing most of the computational overhead into the off-line phase;
(2) reducing the software and hardware resources needed to operate the on-line diagnostic system;
(3) simplifying the on-line diagnostic system to the point where it can be implemented cost effectively on primitive platforms with limited resources.

These factors make the embeddability of on-line diagnostics much more feasible.

Suppose for example that we want to embed a diagnostic system on a vehicle. If we adopt the model-based architecture in Figure 13, then we must embed both the model and its associated algorithms on the vehicle, which could be quite demanding given the typical complexity of model-based algorithms. However, if we adopt the compilation approach sketched in Figure 14, then all we need to embed is the compiled model and its diagnostic evaluator which are assumed to be quite simple by design.

Although a well-accepted theory exists for model-based diagnosis [Reiter 1987; de Kleer et al. 1992], no such theory seems to exist for compiling devices. Most

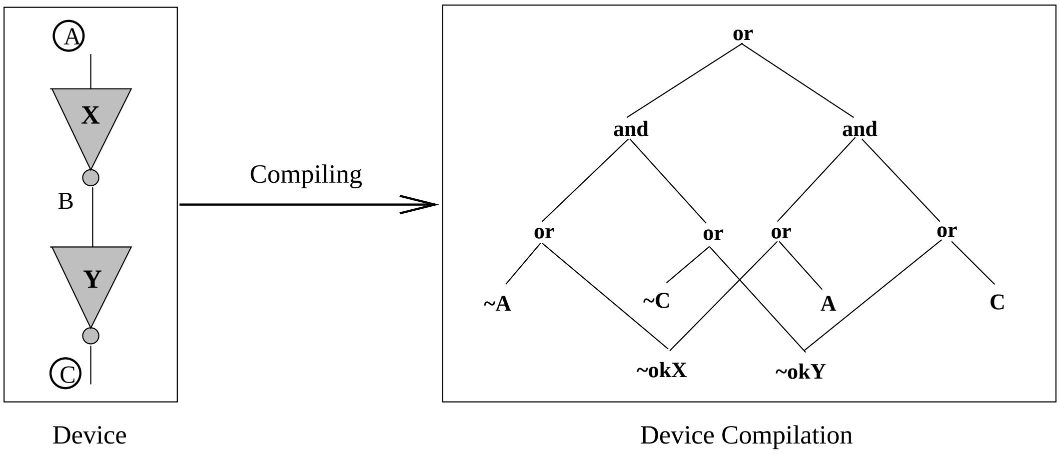

Fig. 15. On the left, a device with two observables A and C — that is, variables about which we plan to collect observations. On the right, a compiled system description that can be used to answer diagnostic queries about the collected observations.

proposals for compilation are ad-hoc techniques for enhancing the performance of various model-based diagnosis systems. An exception to this, however, is the compilation approach proposed by de Kleer [de Kleer 1990; Forbus and de Kleer 1993]. According to this approach, a device is modeled using a set of propositional sentences and is compiled in an off-line phase by computing the prime implicates of the sentences constituting the device model. Given such implicates, one can generate in the on-line phase all the prime implicates corresponding to a given device observation by simply traversing the device compilation. The resulting implicates are called *minimal conflicts* in the model-based literature and are central to answering diagnostic queries [de Kleer and Williams 1987; Williams and Nayak 1997].

This approach is marked by its elegance, but it has proven not to be practical for two reasons. First, computing conflicts is only a first step in answering diagnostic queries and it remains to be seen whether further steps can be accomplished in linear time. Second, the approach provides no formal guarantees on the time to generate, or space to store, a device compilation. In fact, it has been observed that the number of prime implicates constituting a device compilation tends to be impractically large for many real-world devices [Forbus and de Kleer 1993].

We will now present a compilation approach which addresses these two problems. Specifically, we will propose compiling the device model into a DNNF, and then show how various diagnostic queries can be answered using DNNF operations which take time polynomial in the size of device compilation. The device itself is represented using a classical system description (a set of propositional sentences [Reiter 1987; de Kleer et al. 1992]); therefore, we refer to its compilation as a *compiled system description (CSD)*. An example device and its compiled system description are shown in Figure 15.

In this approach, the compiled system description and its evaluator are so simple that they allow for efficient and cost effective implementations on a variety of software and hardware platforms. Therefore, the main practical value of our compilation technique is in synthesizing embeddable diagnostic systems, which may need to operate under stringent platforms and operational constraints.

5.2 A Compilation Example

Consider the simple device shown in Figure 15 which has one input A and one output C. The circles enclosing A and C declare these variables as being observables; that is, device variables about which we plan to collect observations. Given that each one of these variables could be either on or off, we have a total of four possible device observations: $A \wedge C$, $A \wedge \neg C$, $\neg A \wedge C$ and $\neg A \wedge \neg C$. And given that we have two components, we have four states of the device health: $okX \wedge okY$, $\neg okX \wedge okY$, $okX \wedge \neg okY$ and $\neg okX \wedge \neg okY$, where the number of faults in each state is 0, 1, 1, and 2, respectively. Our task: compute states of health that are consistent with the system model and a given system observation, and yet include only a minimum number of faults. Such states are called *minimum-cardinality diagnoses* in the model-based literature; their formal definition will be given in the following section.

If we were to solve this problem using our compilation approach, we would proceed as follows. First, we would model the device using a set of propositional sentences (clauses in this case):

$$\Delta = \left\{ \begin{array}{c} okX \wedge A \supset \neg B, \\ okX \wedge \neg A \supset B, \\ okY \wedge B \supset \neg C, \\ okY \wedge \neg B \supset C \end{array} \right\}.$$

We would then pass two items to our diagnostic compiler:

(1) the device model Δ;

(2) the set of observables A and C.

The compiler would then generate, in an off-line phase, the DNNF shown in Figure 15. This sentence alone, as we shall show, is sufficient to answer all queries of the form:

—What is the minimum number of possible faults given some observation about A and C?

—What is the minimum-cardinality diagnoses in such a case?

For example, if the observation is $A \wedge \neg C$, then the minimum number of faults is 1 and the minimum-cardinality diagnoses are $okX \wedge \neg okY$ and $\neg okX \wedge okY$.

The architecture of our proposed compilation approach is shown in Figure 16. Given a system description and a set of observables, the diagnostic compiler will output a compiled system description (CSD). Once we have a CSD and a system observation, the CSD evaluator can then be invoked to answer the necessary diagnostic queries. The CSD evaluator will be discussed in Section 5.4. In the next section, we discuss the compiler for generating CSDs.

5.3 The Diagnostic Compiler

In the rest of this paper, a *system description* is a triple $(\Delta, \mathbf{A}, \mathbf{O})$ where Δ is a set of propositional sentences, $\mathbf{A}$ and $\mathbf{O}$ are disjoint sets of atoms that appear in

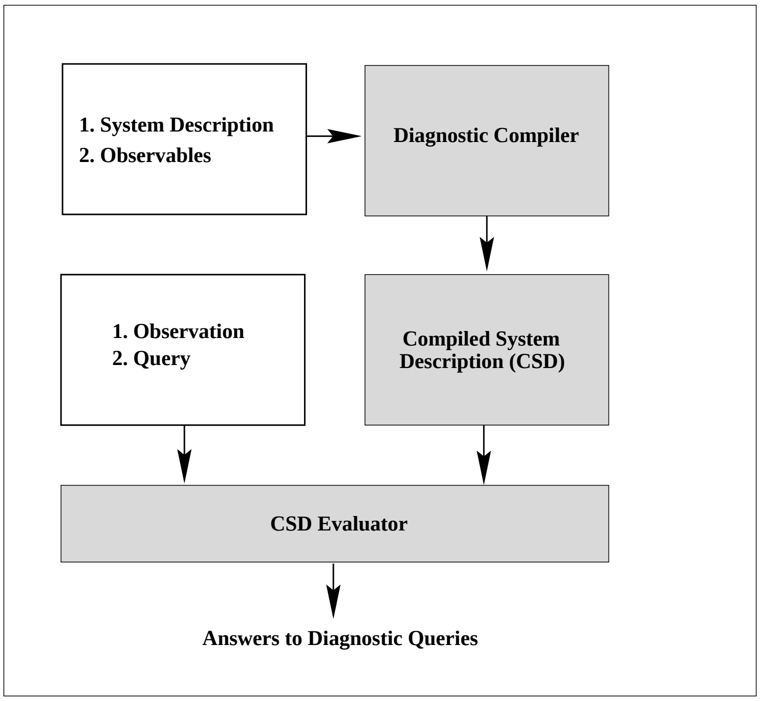

Fig. 16. The architecture of the proposed compilation approach.

Δ, called the *assumables* and *observables,* respectively. Intuitively, assumables are atoms that represent the health of components while observables are atoms that we plan to measure their truth values. For the device in Figure 15, we have

$$\Delta = \left\{ \begin{array}{c} okX \wedge A \supset \neg B, \\ okX \wedge \neg A \supset B, \\ okY \wedge B \supset \neg C, \\ okY \wedge \neg B \supset C \end{array} \right\};$$

where $\mathbf{A} = \{okX, okY\}$ and $\mathbf{O} = \{A, C\}$. The atom B is neither an assumable nor an observable.

A *system observation* is an $\mathbf{O}$-instantiation. A system observation α is said to be *normal* precisely when $\Delta \cup \mathbf{A} \cup \{\alpha\}$ is consistent; that is, precisely when the observation is consistent with the assumption that all device components are healthy. If a system observation is abnormal, then one needs to compute diagnoses. A *diagnosis* of a system description $(\Delta, \mathbf{A}, \mathbf{O})$ and observation α is an $\mathbf{A}$-instantiation consistent with $\Delta \cup \{\alpha\}$ [Reiter 1987; de Kleer et al. 1992]. The *cardinality* of a diagnosis is the number of negative literals it contains. We will denote the set of diagnoses by $\mathsf{Diagnoses}(\Delta \cup \{\alpha\})$. In general, we will use $\mathsf{Diagnoses}(\Gamma)$ to denote the set of all $\mathbf{A}$-instantiations consistent with theory Γ. We can also define a *partial system observation* as an $\mathbf{O}'$-instantiation, where $\mathbf{O}' \subseteq \mathbf{O}$. The definitions of normality and diagnoses carry over to this notion.

A main objective of diagnostic reasoning is to obtain information about the set

$\mathsf{Diagnoses}(\Delta \cup \{\alpha\})$. In this paper, we focus on two pieces of information:

(1) *The smallest number of faults possible in the system having observed α,* which is formally represented by the minimum cardinality of diagnoses in the set $\mathsf{Diagnoses}(\Delta \cup \{\alpha\})$. If this number is zero, then the observation α is guaranteed to be normal.

(2) *The minimum–cardinality diagnoses,* which are formally represented by the subset of diagnoses in $\mathsf{Diagnoses}(\Delta \cup \{\alpha\})$ that have a minimum cardinality. Intuitively, these correspond to states of the device that involve a minimal number of faults and are conceivable given that we have observed α.

This information is at the heart of model-based diagnosis and its applications. Our goal in this section is to obtain this information by operating on a device compilation in polynomial time.

We are now ready to provide the formal definition of a device compilation:

DEFINITION 20. *Given a system description $(\Delta, \mathbf{A}, \mathbf{O})$, a compiled system description, denoted $\mathsf{CSD}(\Delta, \mathbf{A}, \mathbf{O})$, is an $(\mathbf{A} \cup \mathbf{O})$-sentence satisfying the following properties:*

(1) $\mathsf{CSD}(\Delta, \mathbf{A}, \mathbf{O})$ is a smooth DNNF;

(2) for any $(\mathbf{A} \cup \mathbf{O})$-sentence β, $\Delta \models \beta$ iff $\mathsf{CSD}(\Delta, \mathbf{A}, \mathbf{O}) \models \beta$.

That is, a compiled system description is a DNNF sentence which encodes all the information that the system description contains about the assumables and observables — nothing more, nothing less!

As the following theorem shows, a compiled system description is all we need to compute the set of diagnoses and, therefore, to answer the necessary diagnostic queries:

THEOREM 22. *Let $(\Delta, \mathbf{A}, \mathbf{O})$ be a system description and α be a system observation. Then $\mathsf{Diagnoses}(\Delta \cup \{\alpha\}) = \mathsf{Diagnoses}(\mathsf{CSD}(\Delta, \mathbf{A}, \mathbf{O}) \wedge \alpha)$.*

In fact, we can obtain a compiled system description by simply compiling the system description into DNNF and, projecting the result on assumables and observables, and then smoothing the result.

THEOREM 23. *Let $(\Delta, \mathbf{A}, \mathbf{O})$ be a system description and Γ be a DNNF representation of Δ. Then $\mathsf{Smooth}(\mathsf{Project}(\Gamma, \mathbf{A} \cup \mathbf{O}))$ is a compilation of system $(\Delta, \mathbf{A}, \mathbf{O})$.*

Therefore, compiling a system description Δ is a two–step process in which we first convert it into DNNF Γ and then project Γ on assumables and observables. Note that projecting Γ on $\mathbf{A} \cup \mathbf{O}$ can be done in time which is linear in the size of Γ. The critical computational step is therefore the compilation of system description Δ into DNNF Γ. We will discuss the complexity of compilation in more detail in Section 5.5. But once we have successfully compiled a system description, we can use it to answer various diagnostic queries by applying tractable DNNF operations as described in the following section.

5.4 The Diagnostic Evaluator

Assuming that we have generated a compiled system description, our focus in this section will be on processing such a compilation in polynomial time to compute answers to diagnostic queries. Since the compiled system description is in DNNF, such queries can be answered by simply applying DNNF operations:

THEOREM 24. *Let* $(\Delta, \mathbf{A}, \mathbf{O})$ *be a system description and* Γ *be a corresponding compilation. If* α *is the current system observation, then*

—$\mathsf{MCard}(\Gamma \mid \alpha)$ *is the smallest number of possible faults in the system.*

—$\mathsf{Models}(\mathsf{Minimize}(\Gamma \mid \alpha))$ *correspond to the minimum-cardinality diagnoses of the system.*

Therefore, by simply applying the operations of conditioning, minimum–cardinality, minimization and enumeration, we can obtain answers to the key queries of concern to diagnostic reasoning.

We close this section by considering two classes of observations, which do not fit under the definition of system observation as given earlier. Specifically, we have assumed that a set of atoms, called observables, are fixed upon compile-time and that a system observation is simply an instantiation of observables. We have two interesting situations which may come up in practice:

(1) Partial observations, where the value of some observable may not be known. This happens, for example, when losing a sensor which is supposed to report the value of a given observable.

(2) Disjunctive observations, where we may not have a unique (partial) observation, but a set of possible ones.

This first case is quite easy to handle as suggested by the following theorem, which generalizes Theorem 24.

THEOREM 25. *Let* $(\Delta, \mathbf{A}, \mathbf{O})$ *be a system description and let* Γ *be a corresponding compilation. If* α *is a partial system observation, then*

—$\mathsf{MCard}(\mathsf{Project}(\Gamma \mid \alpha, \mathbf{A}))$ *is the smallest number of possible faults in the system.*

—$\mathsf{Models}(\mathsf{Minimize}(\mathsf{Project}(\Gamma \mid \alpha, \mathbf{A})))$ *are the minimum-cardinality diagnoses of the system.*

That is, all we have to do is apply an extra projection on the set of assumables after we have conditioned on the partial system observation. Note that if α is an $\mathbf{O}$-instantiation, then $\mathsf{Project}(\Gamma \mid \alpha, \mathbf{A})$ is simply $\Gamma \mid \alpha$. Otherwise, if α is an $\mathbf{O}'$-instantiation, where $\mathbf{O}' \subset \mathbf{O}$, then $\mathsf{Project}(\Gamma \mid \alpha, \mathbf{A})$ is obtained by replacing every literal that mentions an atom in $\mathbf{O} \setminus \mathbf{O}'$ with true.

As for disjunctive observations, the general case can be quite difficult since the observation may then represent an arbitrary propositional sentence. However, in the case where the disjunctive observation is equivalent to the disjunction of a small set of instantiations, then one can apply the above approach to each of these instantiations and then combine the results. For example, if the set of observables are X, Y, Z and the disjunctive observation is $X \wedge (Y \vee Z)$, then we have three cases: $X \wedge Y \wedge Z$, $X \wedge Y \wedge \neg Z$ and $X \wedge \neg Y \wedge Z$. Each of these cases can be handled using the above approach. The smallest number of faults given observation $X \wedge (Y \vee Z)$ is then the minimum of cardinalities obtained with respect to each of the three cases. And the minimum–cardinality diagnoses given $X \wedge (Y \vee Z)$ are those which are minimal across the three cases. Again, as long as the number of cases is small enough, the computational complexity of the approach remains the same.

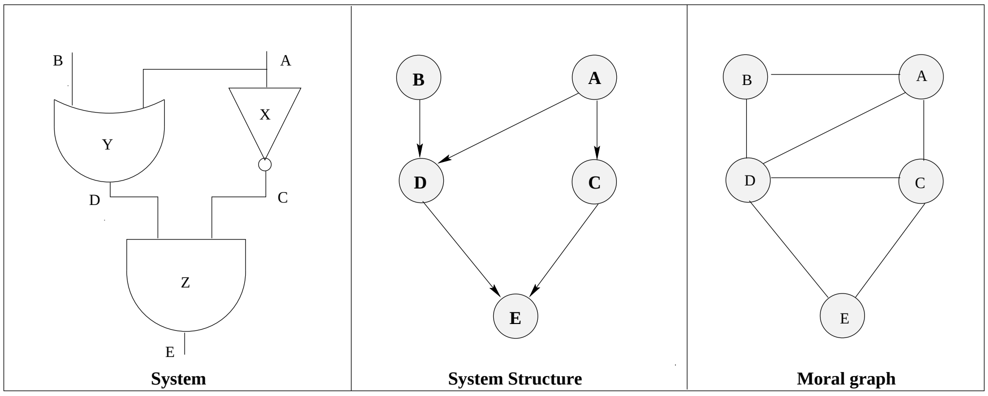

Fig. 17. A system, its structure and a corresponding moralization.

5.5 The Complexity of Compiling Devices

We have presented two key results earlier, which have direct relevance on the scalability of the presented compilation approach. The first is the structure-based guarantee which basically says that the computational complexity of compilation is linear when the CNF to be compiled has a bounded treewidth. The second result says that if the CNF can be compiled into an FBDD of a certain size, then it can be compiled immediately into a DNNF of equal size (times a constant factor). We will elaborate on each of these results next, from the perspective of compiling devices.

The structure-based guarantee is best viewed from the perspective of a *device structure,* which is a DAG that explicates the connectivity of device components—see Figure 17. Assuming that a device is made up of interconnected components, where each component has a set of input ports and a single output port, the device structure is constructed as follows:

(1) an atom is included for each input/output port in the device, where connected ports are represented by the same atom;
(2) a directed edge is extended from each input port to the output port of a given component.

If we further assume that the clauses specifying the device model are local to the device components, then we can measure the complexity of compiling the device by considering the treewidth of its structure. More formally, if we assume that the device model Δ is the union of component models Δ_c, where each Δ_c is a set of clauses satisfying the following conditions:

(1) the clauses in Δ_c only mention the input/output ports of component c;
(2) the number of assumables in Δ_c is bounded by a constant;
(3) no assumables are shared by component models;

then the time and space needed to compile the device model are linear in its structure size and exponential only in the treewidth of such structure.[11] In the graph-

[11] This follows because under these conditions, one can easily show that the connectivity graph of

No. Bits	No. Nodes in CSD	No. Arcs in CSD
1	34	59
2	68	134
3	103	212
4	138	290
5	173	368
6	208	446

Table 1. Compiling n-bit adders.

theoretic literature, treewidth is attributed to undirected graphs but is extended to directed graphs using the notion of moralization. Specifically, the moral graph of a DAG is obtained by adding an undirected edge between each pair of parents in the DAG and then dropping the directionality of all edges. The treewidth of a DAG is then the treewidth of its moral graph. Figure 17 depicts the moral graph of a DAG representing a device structure.

Many important classes of graphs have a universal bound on their treewidth. For example, trees and forests have a treewidth no greater than 1. Singly-connected DAGs, ones with no directed cycles, have a treewidth equal to k, where k is the maximum number of parents per node. Moreover, series–parallel and outerplanner graphs have treewidth no greater than 2 [Arnborg et al. 1987]. Graphs with treewidth k are also known as *partial k-trees* [Arnborg and Proskurowski 1985].

Our structure-based complexity results can therefore be summarized again as follows. If the device to be compiled has a structure of bounded treewidth, then its corresponding model has a linear-sized compilation which can be computed in linear time. An example of such a device is the n-bit adder, which structure has a treewidth independent of n. Table 1 shows the size of compiled n-bit adders as the value of n increases from 1 to 6, illustrating how the size of compilation grows linearly as a function of n. According to Table 1, an on-line system for diagnosing a 6-bit adder needs to include a propositional sentence in DNNF which has 208 nodes and 446 arcs, in addition to a CSD evaluator.

Common wisdom from the literature on structure-based reasoning calls for stressing the following: A large treewidth does not necessarily mean that the system is not practically compilable. It only means that the structure–based compilation algorithm will not be effective. It is quite possible though for other compilation methods to be practical. We cite in this connection the benchmark circuits proposed in [Beglez and Fujiwara 1985], which are known to be relatively impractical for structure–based methods as they are very connected [El Fattah and Dechter 1996]. Almost all of these circuits, however, have been successfully compiled into OBDDs and FBDDs—see [Bern et al. 1994; Meinel and Slobodova 1998] for example—which, given the results in Section 4, means that these circuits do admit practical DNNF compilations.

the clausal form Δ has the same size and treewidth as the device structure (modula a constant factor).

6. CONCLUSION

We have presented an approach for compiling propositional theories into a tractable form, known as decomposable negation normal form (DNNF), which is characterized by a number of important properties. Specifically, DNNF is universal; supports a rich set of polynomial–time operations; is more space-efficient than FBDDs; and is very simple as far as its structure and algorithms are concerned. We have also presented two key results on the compilability of propositional theories into DNNF. According to the first result, the time and space complexity of compiling a theory into DNNF is exponential only in the treewidth of its connectivity graph and linear in all other parameters. According to the second result, the complexity of compiling propositional theories into DNNF is no worse than the complexity of compiling them into FBDDs. Therefore, any successful approach for compiling theories into FBDDs is immediately a successful approach for compiling them into DNNFs. We have also presented an algorithm for approximating a DNNF compilation and provided formal guarantees on the quality of answers returned by such approximations with respect to clausal entailment queries. Finally, we have presented a comprehensive application of our proposal to the problem of compiling devices in the context of model-based diagnosis. The compilation approach is marked by its simplicity, and the complexity guarantees on its on-line and off-line phases. Specifically, we have shown that devices with bounded treewidth can be compiled in linear time and their compilations can be evaluated in polynomial time to answer a variety of diagnostic queries. Practically, our compilation approach has the following implications:

(1) By separating diagnostic computations into off-line and on-line phases, most of the computational overhead is pushed into the off-line phase which needs to be performed only once. This allows for efficient implementations of on-line diagnostic systems.

(2) This separation reduces the software and hardware demands of the on-line diagnostic system, which needs to include only a compiled system description and its evaluator. This makes the deployment of model-based diagnostic systems more cost effective.

This is a most important aspect of our compilation approach since it allows one to implement, quickly, real-world diagnostic systems on a variety of software and hardware platforms [Darwiche 2000a]. One develops a single diagnostic compiler on a platform of choice, but could then deploy on-line diagnostic systems on any software and hardware platform for which a DNNF evaluator can be made available.

ACKNOWLEDGMENTS

I would like to thank Rina Dechter, Alvaro del Val, Henry Kautz, Judea Pearl, Pierre Marquis, Bart Selman and Brian Williams for various helpful discussions and for their comments on earlier versions of this paper. This work has been partially supported by an NSF research grant number IIS-9988543.

APPENDIX

A. PROOFS

We start with some supporting definitions.

DEFINITION 21. *A world ω over atoms $\mathbf{A}$ is a truth assignment over atoms $\mathbf{A}$: a mapping from $\mathbf{A}$ into $\{true, false\}$. We will use $\omega \models \alpha$ to denote that world ω satisfies/entails sentence α in the classical sense.*

DEFINITION 22. *Let ω be a world and $\mathbf{A}$ be a set of atoms. The projection of world ω on atoms $\mathbf{A}$, denoted $\mathsf{Project}(\omega, \mathbf{A})$, is the set of all worlds that agree with world ω on atoms $\mathbf{A}$.*

We will often write $\mathsf{Project}(\omega, \overline{\mathbf{A}})$ to mean the set of all worlds that agree with ω on atoms not in $\mathbf{A}$. Note that $\omega \in \mathsf{Project}(\omega', \mathbf{A})$ iff $\omega' \in \mathsf{Project}(\omega, \mathbf{A})$. Moreover, $\omega \in \mathsf{Project}(\omega, \mathbf{A})$.

The following important lemma provides a semantics for the operation of conditioning:

LEMMA 1. *Let Δ be a propositional sentence and let α be an $\mathbf{O}$–instantiation. Then $\omega \models \Delta \mid \alpha$ iff there exists a world ω' such that $\omega' \models \Delta \wedge \alpha$ and $\omega' \in \mathsf{Project}(\omega, \overline{\mathbf{O}})$.*

Proof: Suppose that $\omega' \models \Delta \wedge \alpha$. Then $\omega' \models \Delta \mid \alpha$ by definition of conditioning. Moreover, $\omega \in \mathsf{Project}(\omega', \mathbf{A})$ only if $\omega \models \Delta \mid \alpha$ since $\Delta \mid \alpha$ in an $\overline{\mathbf{O}}$-sentence and, hence, its truth depends only on the value of atoms $\overline{\mathbf{O}}$ in ω'. Therefore, $\omega' \models \Delta \wedge \alpha$ and $\omega \in \mathsf{Project}(\omega', \overline{\mathbf{O}})$ only if $\omega \models \Delta \mid \alpha$.

Now suppose that $\omega \models \Delta \mid \alpha$. Since $\Delta \mid \alpha$ does not mention atoms $\mathbf{O}$, there must exist ω' such that $\omega \in \mathsf{Project}(\omega', \overline{\mathbf{O}})$ and $\omega' \models \alpha$. By definition of conditioning, we must also have that $\omega' \models \Delta$. Hence, $\omega \models \Delta \mid \alpha$ only if $\omega' \models \Delta \wedge \alpha$ and $\omega \in \mathsf{Project}(\omega', \overline{\mathbf{O}})$ for some world ω'. □

Proof of Theorem 1

—This follows immediately since each $\mathbf{O}$-literal in Δ will be replaced by either true or false in $\Delta \mid \alpha$.

—Follows from Lemma 4 and Definition 7: since $\Delta \mid \alpha$ is the projection of $\Delta \wedge \alpha$ on atoms $\overline{\mathbf{O}}$, and since $\Delta \mid \alpha$ is an $\overline{\mathbf{O}}$-sentence, we must have $\Delta \wedge \alpha \models \Delta \mid \alpha$.

—Follows from Lemma 4 and Definition 7: since $\Delta \mid \alpha$ is the projection of $\Delta \wedge \alpha$ on atoms $\overline{\mathbf{O}}$, we must have $\Delta \wedge \alpha \models \beta$ only if $\Delta \mid \alpha \models \beta$ for all $\overline{\mathbf{O}}$-sentences β.

—Follows directly from Lemma 1 since the models of $\Delta \mid \alpha$ depend only on the models of Δ. □

Proof of Theorem 2

That $(\Delta \mid \alpha) \wedge \alpha$ is a DNNF follows since:

(1) $\Delta \mid \alpha$ is a DNNF (Proposition 1);

(2) α and $\Delta \mid \alpha$ share no atoms (Theorem 1).

That $(\Delta \mid \alpha) \wedge \alpha$ is equivalent to $\Delta \wedge \alpha$ follows from Lemma 1. □

Proof of Theorem 3

If the sentences $\alpha_1, \ldots, \alpha_n$ do not share atoms, then each of $\alpha_1, \ldots, \alpha_n$ is satisfiable only if their conjunction $\bigwedge_{i=1}^n \alpha_i$ is satisfiable. The remaining cases follow directly. $\square$

Proof of Theorem 4

Since $\Delta \wedge \alpha \models \Delta \mid \alpha$ by Theorem 1, then $\Delta \wedge \alpha$ is satisfiable only if $\Delta \mid \alpha$ is satisfiable. Now suppose that $\Delta \mid \alpha$ is satisfiable. Lemma 1 implies that $\Delta \wedge \alpha$ must be satisfiable. $\square$

Proof of Theorem 5

Suppose that $\Delta \vdash \beta$. Then by definition, $\mathsf{Sat?}(\Delta \mid \beta')$ is false where $\beta' \equiv \neg\beta$, and $\Delta \mid \beta'$ is unsatisfiable by Theorem 3. Moreover, $\Delta \wedge \beta'$ is unsatisfiable by Theorem 4. Hence, $\Delta \models \beta$. The other direction follows similarly. $\square$

LEMMA 2. *NNF Δ is satisfiable only if* $\mathsf{Sat?}(\Delta)$ *is true.*

Proof: By induction on the structure of NNF Δ.

—Δ *is a literal,* true *or* false*:*
Theorem follows directly.

—Δ *is a conjunction* $\bigwedge_i \alpha_i$*:*
Suppose that Δ is satisfiable. Then each α_i must be satisfiable, and therefore each $\mathsf{Sat?}(\alpha_i)$ must be true (by the induction hypothesis). Hence, $\mathsf{Sat?}(\Delta)$ must also be true.

—Δ *is a disjunction* $\bigvee_i \alpha_i$*:*
Suppose that Δ is satisfiable. Then some α_i must be satisfiable, and therefore some $\mathsf{Sat?}(\alpha_i)$ must true (by the induction hypothesis). Hence, $\mathsf{Sat?}(\Delta)$ must be true. $\square$

Proof of Theorem 6

Suppose that $\Delta \vdash \beta$. Then by definition, $\mathsf{Sat?}(\Delta \mid \beta')$ is false where $\beta' \equiv \neg\beta$, and $\Delta \mid \beta'$ is unsatisfiable by Lemma 2. Hence, $\Delta \wedge \beta'$ is unsatisfiable by Theorem 4 and $\Delta \models \beta$. $\square$

Proof of Theorem 7

Although Δ is decomposable except on $\mathbf{X}$, $\Delta \mid \beta'$, where $\beta' \equiv \neg\beta$, is decomposable since it does not mention any atoms in $\mathbf{X}$ (by definition of conditioning). Now, by definition, $\Delta \vdash \beta$ precisely when $\mathsf{Sat?}(\Delta \mid \beta')$ is false, which is precisely when $\Delta \mid \beta'$ is unsatisfiable. This is precisely when $\Delta \wedge \beta'$ is unsatisfiable (by Theorem 4), which is precisely when $\Delta \models \beta$. $\square$

Proof of Theorem 8

Note that β is equivalent to $\bigwedge_i (\beta \vee \gamma_i)$. Therefore, $\Delta \models \beta$ precisely when $\Delta \models \bigwedge_i (\beta \vee \gamma_i)$, which is precisely when $\Delta \models \beta \vee \gamma_i$ for every γ_i. Now, since $\beta \vee \gamma_i$ mentions all atoms in $\mathbf{X}$, then $\Delta \models \beta \vee \gamma_i$ precisely when $\Delta \vdash \beta \vee \gamma_i$ (by Theorem 7). Therefore, $\Delta \models \beta$ precisely when $\Delta \vdash \beta \vee \gamma_i$ for every $\mathbf{Y}$-clause γ_i. $\square$

The following important lemma provides a semantics for the operation of projection.

LEMMA 3. *Let Δ be a propositional sentence and let Γ be its projection on atoms $\mathbf{A}$. Then $\omega \models \Gamma$ iff there exists a world ω' such that $\omega' \models \Delta$ and $\omega' \in \mathsf{Project}(\omega, \mathbf{A})$.*

Proof: First note that by Definition 7, we immediately have $\Delta \models \Gamma$ by simply taking β to be Γ. Recall also that $\omega' \in \mathsf{Project}(\omega, \mathbf{A})$ iff $\omega \in \mathsf{Project}(\omega', \mathbf{A})$.

Now, suppose that $\omega' \models \Delta$ and $\omega' \in \mathsf{Project}(\omega, \mathbf{A})$. Then $\omega' \models \Gamma$ since $\Delta \models \Gamma$. Moreover, since Γ is an $\mathbf{A}$-sentence, we must also have $\omega \models \Gamma$ since ω agrees with ω' on atoms $\mathbf{A}$. This proves $\omega' \models \Delta$ and $\omega' \in \mathsf{Project}(\omega, \mathbf{A})$ only if $\omega \models \Gamma$.

To prove the other direction, suppose that $\omega \models \Gamma$. Since Γ is an $\mathbf{A}$-sentence, then $\omega' \models \Gamma$ for any $\omega' \in \mathsf{Project}(\omega, \mathbf{A})$. Let α be an $\mathbf{A}$-sentence which models are exactly $\mathsf{Project}(\omega, \mathbf{A})$. Then $\alpha \models \Gamma$. Note that $\omega' \models \Delta \wedge \alpha$ only if $\omega' \models \Delta$ and $\omega' \in \mathsf{Project}(\omega, \mathbf{A})$. Therefore, if $\Delta \wedge \alpha$ is consistent, we are done. Now suppose that $\Delta \wedge \alpha$ is inconsistent. Then $\Delta \models \neg\alpha$. Since $\neg\alpha$ is an $\mathbf{A}$-sentence, we also have $\Gamma \models \neg\alpha$ by definition of projection Γ. This leads to $\alpha \models \neg\Gamma$ which is contradictory with $\alpha \models \Gamma$ since α is consistent. Hence, it is impossible for $\Delta \wedge \alpha$ to be inconsistent. □

The following lemma shows that conditioning is a special case of projection.

LEMMA 4. *Let Δ be a propositional sentence and let α be an $\mathbf{O}$-instantiation. Then $\Delta \mid \alpha$ is the projection of $\Delta \wedge \alpha$ on atoms $\overline{\mathbf{O}}$.*

Proof: Follows from Lemmas 1 and 3. □

Proof of Theorem 9

That $\mathsf{Project}(\Delta, \mathbf{A})$ is a DNNF follows from the fact that Δ is a DNNF and that $\mathsf{Project}(\Delta, \mathbf{A})$ results from replacing every $\overline{\mathbf{A}}$-literal in Δ with true. This also shows that $\mathsf{Project}(\Delta, \mathbf{A})$ is an $\mathbf{A}$-sentence.

We now show that $\mathsf{Project}(\Delta, \mathbf{A})$ is a projection of Δ on atoms $\mathbf{A}$ by induction on the structure of Δ.

Boundary case: If Δ is true, false or an $\mathbf{A}$-literal, then Δ and $\mathsf{Project}(\Delta, \mathbf{A}) = \Delta$ are logically equivalent. Hence, by Definition 7, $\mathsf{Project}(\Delta, \mathbf{A})$ is a projection of Δ on $\mathbf{A}$. When Δ is an $\overline{\mathbf{A}}$-literal, the only $\mathbf{A}$-sentences entailed by Δ and $\mathsf{Project}(\Delta, \mathbf{A}) = \mathsf{true}$ are the valid $\mathbf{A}$-sentences. Hence, by Definition 7, $\mathsf{Project}(\Delta, \mathbf{A})$ is a projection of Δ on $\mathbf{A}$.

Inductive step: Suppose that $\mathsf{Project}(\alpha_i, \mathbf{A})$ is a projection of α_i on atoms $\mathbf{A}$. It suffices to show two cases:

(1) $\bigvee_i \mathsf{Project}(\alpha_i, \mathbf{A})$ is a projection of $\bigvee_i \alpha_i$ on atoms $\mathbf{A}$. Given Definition 7, it suffices to show that $\bigvee_i \alpha_i$ and $\bigvee_i \mathsf{Project}(\alpha_i, \mathbf{A})$ entail the same set of $\mathbf{A}$-sentences. Let β be an $\mathbf{A}$-sentence. Then $\bigvee_i \alpha_i \models \beta$ iff $\alpha_i \models \beta$ iff $\mathsf{Project}(\alpha_i, \mathbf{A}) \models \beta$ (by induction hypothesis) iff $\bigvee_i \mathsf{Project}(\alpha_i, \mathbf{A}) \models \beta$.

(2) $\bigwedge_i \mathsf{Project}(\alpha_i, \mathbf{A})$ is a projection of $\bigwedge_i \alpha_i$ on atoms $\mathbf{A}$, assuming the α_i's share no atoms. It suffices to show that $\bigwedge_i \mathsf{Project}(\alpha_i, \mathbf{A})$ is logically equivalent to a projection Γ of $\bigwedge_i \alpha_i$ on atoms $\mathbf{A}$.

—Suppose that $\omega \models \Gamma$. By Lemma 3, there is a world ω' such that $\omega \in \mathsf{Project}(\omega', \mathbf{A})$ and $\omega' \models \bigwedge_i \alpha_i$. Hence, for each α_i, there is a world ω' such that $\omega \in \mathsf{Project}(\omega', \mathbf{A})$ and $\omega' \models \alpha_i$. By the induction hypothesis and Lemma 3, $\omega \models \mathsf{Project}(\alpha_i, \mathbf{A})$. Hence, $\omega \models \bigwedge_i \mathsf{Project}(\alpha_i, \mathbf{A})$.

—Suppose that $\omega \models \bigwedge_i \mathsf{Project}(\alpha_i, \mathbf{A})$. We then have $\omega \models \mathsf{Project}(\alpha_i, \mathbf{A})$ for each α_i. By the induction hypothesis and Lemma 3, for each α_i, there exists ω^i such that $\omega \in \mathsf{Project}(\omega^i, \mathbf{A})$ and $\omega^i \models \alpha_i$. Since each world ω^i agrees with ω on atoms $\mathbf{A}$, all ω^i's agree on atoms $\mathbf{A}$. Since the α_i's share no atoms, one can construct a world ω' that agrees with ω (and all ω^i) on atoms $\mathbf{A}$, yet entails each α_i. That is, there exists a world ω' such that $\omega \in \mathsf{Project}(\omega', \mathbf{A})$ and $\omega' \models \bigwedge_i \alpha_i$. By Lemma 3, we must then have $\omega \models \Gamma$. $\square$

Proof of Theorem 10

The proof follows by induction on the NNF structure. The base case follows directly. There are two inductive steps, one for conjunctions and another for disjunctions. The conjunction case follows because the conjuncts share no atoms, therefore, the minimum cardinality of $\bigwedge_i \alpha_i$ is the addition of minimum cardinalities of α_i. The disjunction case follows directly: the minimum cardinality of $\bigvee_i \alpha_i$ is the smallest among the minimum cardinalities of α_i. $\square$

Proof of Theorem 11

The proof is by induction on the NNF structure. The base case follows directly. There are two inductive steps, one for conjunctions and another for disjunctions. The case for conjunctions follows from decomposability: given that the cardinality of a conjunction is the summation of its conjuncts' cardinalities, we can minimize the conjunction by simultaneously minimizing each of its conjuncts. As for the case of disjunctions: to minimize a disjunction we have to minimize each disjunct and then keep only those minimized disjuncts that generate the models with smallest cardinality. Smoothness ensures that the models of each disjunct are over the same set of atoms. $\square$

Proof of Theorem 12

The proof is by induction on the NNF structure. The base case follows directly. There are two inductive steps, corresponding to conjunctions and disjunctions. The conjunctions case follows from decomposability. The disjunctions case follows from smoothness. $\square$

Proof of Theorem 13

Let $\mathbf{X}$ be the atoms shared by Δ_1 and Δ_2 and let β be an instantiation of atoms $\mathbf{X}$. We have

$$
\begin{aligned}
\Delta_1 \wedge \Delta_2 &\equiv \bigvee_{\beta} \Delta_1 \wedge \Delta_2 \wedge \beta \\
&\equiv \bigvee_{\beta} (\Delta_1 \wedge \beta) \wedge (\Delta_2 \wedge \beta) \wedge \beta \\
&\equiv \bigvee_{\beta} ((\Delta_1 \mid \beta) \wedge \beta) \wedge ((\Delta_2 \mid \beta) \wedge \beta) \wedge \beta \\
&\equiv \bigvee_{\beta} (\Delta_1 \mid \beta) \wedge (\Delta_2 \mid \beta) \wedge \beta.
\end{aligned}
$$

Note that neither $\Delta_1 \mid \beta$ nor $\Delta_2 \mid \beta$ mention atoms $\mathbf{X}$. Therefore, $(\Delta_1 \mid \beta) \wedge (\Delta_2 \mid \beta) \wedge \beta$ is decomposable, and, hence, $\bigvee_\beta (\Delta_1 \mid \beta) \wedge (\Delta_2 \mid \beta) \wedge \beta$ is in DNNF. $\Box$

In the following proofs, we will use the following definitions which apply to nodes in a decomposition tree:

—For non-leaf node t:

$$\mathsf{cutset}(t) \stackrel{def}{=} \mathsf{atoms}(t_l) \cap \mathsf{atoms}(t_r) - \mathsf{atoms}^{\uparrow}(t).$$

—For node t:

$$\mathsf{context}(t) \stackrel{def}{=} \mathsf{atoms}^{\uparrow}(t) \cap \mathsf{atoms}(t).$$

These definitions lead to

$$\mathsf{cluster}(t) = \mathsf{cutset}(t) \cup \mathsf{context}(t),$$

for non-leaf node t.

Proof of Theorem 14

We will now prove soundness by induction on the structure of a decomposition tree.

—t is a leaf node.
Then clearly $\mathsf{dnnf1}(t, \alpha)$ returns $\Delta(t) \mid \alpha$. Since $\Delta(t)$ is a clause, then $\Delta(t) \mid \alpha$ is a DNNF.

—t is non-leaf node.
Suppose that $\mathsf{dnnf1}(t_l, \alpha \wedge \beta)$ returns $\Delta(t_l) \mid \alpha \wedge \beta$ in DNNF and that $\mathsf{dnnf1}(t_r, \alpha \wedge \beta)$ returns $\Delta(t_r) \mid \alpha \wedge \beta$ in DNNF. We know that $\Delta(t) = \Delta(t_l) \cup \Delta(t_r)$. We also know that $\Delta(t) \mid \alpha = (\Delta(t_l) \mid \alpha) \cup (\Delta(t_r) \mid \alpha)$.
The common atoms between $\Delta(t_l) \mid \alpha$ and $\Delta(t_r) \mid \alpha$ are in $\mathsf{atoms}(t_l) \cap \mathsf{atoms}(t_r) - \mathsf{atoms}(\alpha)$. Therefore, if β is an instantiation of $\mathsf{atoms}(t_l) \cap \mathsf{atoms}(t_r) - \mathsf{atoms}(\alpha)$, then Theorem 13 gives:

$$\begin{aligned}\Delta(t) \mid \alpha &\equiv \bigvee_\beta ((\Delta(t_l) \mid \alpha) \mid \beta) \wedge ((\Delta(t_r) \mid \alpha) \mid \beta) \wedge \beta, \\ &\equiv \bigvee_\beta (\Delta(t_l) \mid \alpha \wedge \beta) \wedge (\Delta(t_r) \mid \alpha \wedge \beta) \wedge \beta,\end{aligned}$$

and $\Delta(t) \mid \alpha$ is a DNNF. Moreover, by the induction hypothesis, we have

$$\Delta(t) \mid \alpha \equiv \bigvee_\beta \mathsf{dnnf1}(t_l, \alpha \wedge \beta) \wedge \mathsf{dnnf1}(t_r, \alpha \wedge \beta) \wedge \beta,$$

which is exactly what is being returned by the algorithm.

We used two properties of conditioning in this proof. First, that $(\Delta_1 \vee \Delta_2) \mid \beta = (\Delta_1 \mid \beta) \vee (\Delta_2 \mid \beta)$. Second, that $(\Delta \mid \beta_1) \mid \beta_2 = \Delta \mid (\beta_1 \wedge \beta_2)$ when β_1 and β_2 share no atoms. $\Box$

Proof of Theorem 15

By Theorem 14, $\mathsf{dnnf1}(t,\alpha)$ returns $\Delta(t) \mid \alpha$ and $\mathsf{dnnf1}(t,\alpha')$ returns $\Delta(t) \mid \alpha'$. Therefore, if α and α' agree on $\mathsf{atoms}(t)$, then $\Delta(t) \mid \alpha$ and $\Delta(t) \mid \alpha'$ must be equivalent by the definition of conditioning and, hence, $\mathsf{dnnf1}(t,\alpha)$ must be equivalent to $\mathsf{dnnf1}(t,\alpha')$. $\Box$

LEMMA 5. *The cache associated with node t in* dnnf2 *has no more than* $2^{|\mathsf{context}(t)|}$ *entries.*

Proof. For any two calls $\mathsf{dnnf2}(t,\alpha)$ and $\mathsf{dnnf2}(t,\alpha')$, the instantiations α and α' can only disagree on the atoms in $\mathsf{atoms}^{\uparrow}(t)$. Since we take the subsets of these instantiations pertaining to $\mathsf{atoms}(t)$, these subsets will be stored in different cache entries only if they disagree on $\mathsf{atoms}(t) \cap \mathsf{atoms}^{\uparrow}(t) = \mathsf{context}(t)$. Therefore, we can have no more than $2^{|\mathsf{context}(t)|}$ entries in the cache. $\Box$

Proof of Theorem 16

First, note that for non-leaf node t,

$$\mathsf{cluster}(t) = \mathsf{cutset}(t) \cup \mathsf{context}(t),$$

which means that $\mid \mathsf{cutset}(t) \mid \leq w$ and $\mid \mathsf{context}(t) \mid \leq w$. In fact, one can show that $\mathsf{cutset}(t) \cap \mathsf{context}(t) = \emptyset$. Hence, $\mid \mathsf{cluster}(t) \mid = \mid \mathsf{cutset}(t) \mid + \mid \mathsf{context}(t) \mid$.

From the statement of the algorithm, it is clear that the space and time complexity are equal.

A function call $\mathsf{dnnf2}(t,.)$ will either return immediately after doing a constant amount of work, or will recurse. If the call recurses, the amount of work it does, excluding the work done by its recursive calls, is $O(w2^{|\mathsf{cutset}(t)|})$ where $O(2^{|\mathsf{cutset}(t)|})$ is the number of β instantiations it will consider and $O(w)$ is the size of each such instantiation.

If the number of times the algorithm will recurse at node t is r_t, then the total amount of work done by the algorithm is:

$$\sum_t r_t O(w2^{|\mathsf{cutset}(t)|}).$$

Each time a call $\mathsf{dnnf2}(t,.)$ recurses, an entry is added to the cache of t. Since the number of entries in this cache cannot exceed $2^{|\mathsf{context}(t)|}$ by Lemma 5, the number of times that call $\mathsf{dnnf2}(t,.)$ will recurse is then $r_t = O(2^{|\mathsf{context}(t)|})$. Hence, the total amount of work done by the algorithm is

$$\sum_t O(2^{|\mathsf{context}(t)|})O(w2^{|\mathsf{cutset}(t)|}) = \sum_t O(w2^{|\mathsf{cluster}(t)|}) = O(nw2^w).$$

Proof of Theorem 18

That $\mathsf{dnnf}_u(t,\alpha)$ is an NNF follows from the statement of the algorithm . We will show that $\mathsf{dnnf}_u(t,\alpha)$ is equivalent to $\mathsf{dnnf2}(t,\alpha)$ by induction on the structure of a decomposition tree.

If t is a leaf node, then the result is immediate. Suppose that t is a non-leaf node. Suppose further that $\mathsf{dnnf}_u(t_l,\alpha\wedge\beta) \equiv \mathsf{dnnf2}(t_l,\alpha\wedge\beta)$ and $\mathsf{dnnf}_u(t_r,\alpha\wedge\beta) \equiv$

$\mathsf{dnnf2}(t_r, \alpha \wedge \beta)$. We then have:

$$
\begin{aligned}
\mathsf{dnnf}_u(t,\alpha) &= \bigvee_\beta \mathsf{dnnf}_u(t_l, \alpha \wedge \beta) \wedge \mathsf{dnnf}_u(t_r, \alpha \wedge \beta) \wedge \beta \\
&\equiv \bigvee_\beta \mathsf{dnnf2}(t_l, \alpha \wedge \beta) \wedge \mathsf{dnnf2}(t_r, \alpha \wedge \beta) \wedge \beta \\
&\equiv \bigvee_\beta (\Delta(t_l) \mid \alpha \wedge \beta) \wedge (\Delta(t_r) \mid \alpha \wedge \beta) \wedge \beta \\
&\equiv \bigvee_\beta ((\Delta(t_l) \mid \alpha) \mid \beta) \wedge ((\Delta(t_r) \mid \alpha) \mid \beta) \wedge \beta \\
&\equiv \bigvee_\beta ((\Delta(t_l) \wedge \Delta(t_r)) \mid \alpha) \wedge \beta \\
&\equiv (\Delta(t_l) \wedge \Delta(t_r)) \mid \alpha \\
&\equiv \Delta(t) \mid \alpha \\
&\equiv \mathsf{dnnf2}(t,\alpha). \;\square
\end{aligned}
$$

Proof of Theorem 19

That $\mathsf{dnnf}_l(t,\alpha)$ is a DNNF follows since even though we are dropping some instantiations β, each such instantiation still mentions all atoms common between $\Delta(t_l) \mid \alpha$ and $\Delta(t_r) \mid \alpha$.

We will now show that $\mathsf{dnnf}_l(t,\alpha)$ entails $\mathsf{dnnf2}(t,\alpha)$ by induction on the structure of a decomposition tree. If t is a leaf, the result is immediate. If t is a non-leaf node, then suppose that $\mathsf{dnnf}_l(t_l, \alpha \wedge \beta) \models \mathsf{dnnf2}(t_l, \alpha \wedge \beta)$ and $\mathsf{dnnf}_l(t_r, \alpha \wedge \beta) \models \mathsf{dnnf2}(t_r, \alpha \wedge \beta)$. Let $\mathbf{X}$ be the atoms that appear in $\mathsf{atoms}(t_l) \cap \mathsf{atoms}(t_r) - \mathsf{atoms}(\alpha)$. We have

$$\mathsf{dnnf2}(t,\alpha) = \bigvee_\beta \mathsf{dnnf2}(t_l, \alpha \wedge \beta) \wedge \mathsf{dnnf2}(t_r, \alpha \wedge \beta) \wedge \beta,$$

where β ranges over all instantiations of atoms $\mathbf{X}$ and

$$\mathsf{dnnf}_l(t,\alpha) = \bigvee_\beta \mathsf{dnnf}_l(t_l, \alpha \wedge \beta) \wedge \mathsf{dnnf}_l(t_r, \alpha \wedge \beta) \wedge \beta,$$

where β ranges over some instantiations of atoms $\mathbf{X}$. Therefore, all we are doing in dnnf_l is dropping some disjuncts from $\mathsf{dnnf2}$ and possibly strengthening some of the remaining disjuncts (by the induction hypothesis). This will only possibly strengthen the whole sentence. Therefore, $\mathsf{dnnf}_l(t,\alpha) \models \mathsf{dnnf2}(t,\alpha)$. $\square$

Proof of Theorem 20

We need to prove three things about the call $\mathsf{fbdd2dnnf}(m)$:

(1) it returns a DNNF;

(2) the DNNF it returns represents the same Boolean function represented by m;

(3) its time and space complexity are $O(n)$.

The time complexity follows since the call $\mathsf{fbdd2dnnf}(m)$ does a depth-first traversal on the DAG rooted at node m. It recurses on each node only once, performing a constant amount of work and adding a constant number of nodes and edges to the constructed DNNF. Hence, the time and space complexity of the algorithm is $O(V+E)$, where V is the number of nodes in the DAG rooted at m and E is its number of edges. Since each node in the DAG has at most two outgoing edges, the time and space complexity of the algorithm is $O(n)$

The proof of the first two properties follow by induction on the structure of FBDD rooted at m. If m is a leaf node, the above results follow immediately. Suppose that m is a non-leaf node with $\mathsf{var}(m) = X_i$.

(1) Suppose that $\mathsf{fbdd2dnnf}(\mathsf{low}(m))$ and $\mathsf{fbdd2dnnf}(\mathsf{high}(m))$ return DNNFs. Then $\mathsf{fbdd2dnnf}(m)$ is clearly an NNF since negations are only introduced next to variables. Now, suppose that $\mathsf{fbdd2dnnf}(m)$ is not decomposable. Then either $\mathsf{fbdd2dnnf}(\mathsf{low}(m))$ or $\mathsf{fbdd2dnnf}(\mathsf{high}(m))$ must contain an occurance of the variable X_i. But this implies that variables X_i is tested more than once on some computation path, which is a contradiction. Therefore, $\mathsf{fbdd2dnnf}(m)$ is a DNNF.

(2) Let $f(m)$ denote the Boolean function represented by the FBDD rooted at node m. Then $f(m) = f(\mathsf{low}(m))\overline{X_i} + f(\mathsf{high}(m))X_i$ by the definition of an FBDD. Suppose that $\mathsf{fbdd2dnnf}(\mathsf{low}(m))$ is a DNNF representation of $f(\mathsf{low}(m))$ and $\mathsf{fbdd2dnnf}(\mathsf{high}(m))$ is a DNNF representation of $f(\mathsf{high}(m))$. The answer returned by $\mathsf{fbdd2dnnf}(m)$,

$$(\mathsf{fbdd2dnnf}(\mathsf{low}(m)) \wedge \neg X_i) \vee (\mathsf{fbdd2dnnf}(\mathsf{high}(m)) \wedge X_i),$$

must therefore be a representation of the Boolean function $f(m)$. □

Proof of Theorem 22

Let β be an $\mathbf{A}$-instantiation. We want to show that $\Delta \cup \{\alpha\} \not\models \neg\beta$ precisely when $\mathsf{CSD}(\Delta, \mathbf{A}, \mathbf{O}) \wedge \alpha \not\models \neg\beta$. It suffices to show that $\Delta \cup \{\alpha\} \models \neg\beta$ precisely when $\mathsf{CSD}(\Delta, \mathbf{A}, \mathbf{O}) \wedge \alpha \models \neg\beta$. It also suffices to show that $\Delta \models \neg\alpha \vee \neg\beta$ precisely when $\mathsf{CSD}(\Delta, \mathbf{A}, \mathbf{O}) \models \neg\alpha \vee \neg\beta$. Since $\neg\alpha \vee \neg\beta$ is an $(\mathbf{A} \cup \mathbf{O})$-sentence, the result follow directly from Part 2 of Definition 20. □

Proof of Theorem 23

Follows immediately from the definitions of CSD, projection and Theorem 9. □

Proof of Theorem 24

Falls as a special case of Theorem 25. □

Proof of Theorem 25

We first show that $\mathsf{Project}(\Gamma \wedge \alpha, \mathbf{A}) \equiv \mathsf{Project}(\Gamma \mid \alpha, \mathbf{A})$. By Theorem 2, we have that $\Gamma \wedge \alpha$ is equivalent to $(\Gamma \mid \alpha) \wedge \alpha$, which is a DNNF. By definition of $\mathsf{Project}$ and since α is a conjunction of $\overline{\mathbf{A}}$-literals, we have that

$$\begin{aligned}
\mathsf{Project}(\Gamma \wedge \alpha, \mathbf{A}) &\equiv \mathsf{Project}((\Gamma \mid \alpha) \wedge \alpha, \mathbf{A}) \\
&\equiv \mathsf{Project}(\Gamma \mid \alpha, \mathbf{A}) \wedge \mathsf{Project}(\alpha, \mathbf{A}) \\
&\equiv \mathsf{Project}(\Gamma \mid \alpha, \mathbf{A}).
\end{aligned}$$

Now, the smallest number of possible faults is the minimum cardinality of diagnoses in the set $\mathsf{Diagnoses}(\Delta \cup \{\alpha\})$, which are the $\mathbf{A}$-instantiations consistent with $\Delta \cup \{\alpha\}$. By definition of projection, such $\mathbf{A}$-instantiations are in one-to-one correspondence with the models of $\mathsf{Project}(\Delta \wedge \alpha, \mathbf{A})$, which is equivalent to $\mathsf{Project}(\Delta \mid \alpha, \mathbf{A})$ as shown above. Therefore, the smallest number of possible faults is $\mathsf{MCard}(\mathsf{Project}(\Delta \mid \alpha, \mathbf{A}))$ by definition of MCard.

Since $\mathsf{Models}(\mathsf{Project}(\Gamma \mid \alpha, \mathbf{A}))$ correspond to the diagnoses in $\mathsf{Diagnoses}(\Delta \cup \{\alpha\})$, and by definition of $\mathsf{Minimize}$, $\mathsf{Models}(\mathsf{Minimize}(\mathsf{Project}(\Gamma \mid \alpha, \mathbf{A})))$ must correspond to the subset of these diagnoses having minimum cardinality. $\square$

REFERENCES

ARNBORG, S. 1985. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT 25*, 2–33.

ARNBORG, S., CORNEIL, D., AND PROSKUROWSKI, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Meth. 8*, 277–284.

ARNBORG, S. AND PROSKUROWSKI, A. 1985. Characterization and recognition of partial k-trees. *Congr. Numer. 47*, 69–75.

BARWISE, J. Ed. 1977. *Handbook of Mathematical Logic*. North-Holland, Amsterdam.

BEGLEZ, F. AND FUJIWARA, H. 1985. A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN. In *Proceedings of the IEEE symposium on Circuits and Systems* (1985). http://www.cbl.ncsu.edu/www/CBL_Docs/iscas85.html.

BERN, J., GERGOV, J., MEINEL, C., AND SLOBODOVA, A. 1994. Boolean manipulation with free bdds: first experimental results. In *EDAC'94, Paris, France* (1994), pp. 200–207.

BODLAENDER, H. L. 1993. A tourist guide through treewidth. *ACTA CYBERNETICA 11*, 1-2, 1–22.

BODLAENDER, H. L. 1996. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing 25*, 6, 1305–1317.

BOLLING, B. AND WEGENER, I. 1998. A very simple function that requires exponential size read-once branching programs. *Information Processing Letters 66*, 2, 53–57.

BOUFKHAD, Y., GREGOIRE, E., MARQUIS, P., MAZURE, B., AND SAIS, L. 1997. Tractable cover compilations. In *Proc. International Joint Conference on Artifical Intelligence (IJCAI)* (1997), pp. 122–127.

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

BRYANT, R. E. 1991. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE transactions on Computers 40*, 205–213.

BRYANT, R. E. 1992. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys 24*, 3, 293–318.

CADOLI, M. AND DONINI, F. M. 1997. A survey on knowledge compilation. *AI Communications 10*, 137–150.

DARWICHE, A. 1992. *A Symbolic Generalization of Probability Theory*. Ph. D. thesis, Stanford University.

DARWICHE, A. 1998a. Compiling devices: A structure-based approach. In *KR98* (1998), pp. 156–166.

DARWICHE, A. 1998b. Model–based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research 8*, 165–222.

DARWICHE, A. 1999. Compiling knowledge into decomposable negation normal form. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)* (1999), pp. 284–289.

DARWICHE, A. 2000b. On the tractable counting of theory models and its application to belief revision and truth maintenance. Technical Report D-113, Cognitive Systems Laboratory, Computer Science Department, UCLA, Los Angeles, Ca 90095. To appear in Journal of Applied Non-Classical Logics.

Darwiche, A. February, 2001. Recursive conditioning. *Artificial Intelligence 126*, 1-2, 5–41.

Darwiche, A. Summer, 2000a. Model-based diagnosis under real-world constraints. *AI Magazine 21*, 2, 57–73.

Darwiche, A. and Ginsberg, M. L. 1992. A symbolic generalization of probability theory. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)* (1992), pp. 622–627.

de Kleer, J. 1990. Compiling devices and processes. Presented at 4th International Workshop on Qualitative Physics, Lugano, Switzerland.

de Kleer, J., Mackworth, A. K., and Reiter, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence 56*, 2-3, 197–222.

de Kleer, J. and Williams, B. C. 1987. Diagnosing multiple faults. *Artificial Intelligence Journal 32*, 97–130.

Dechter, R. 1992. Constraint networks. In S. Shapiro Ed., *Encyclopedia of Artificial Intelligence*, pp. 276–285. John Wiley and Sons.

Dechter, R. and Rish, I. 1994. Directional resolution: The davis-putnam procedure, revisited. In *KR94* (1994), pp. 134–145.

del Val, A. 1994. Tractable databases: How to make propositional unit resolution complete through compilation. In *KR94* (1994), pp. 551–561.

del Val, A. 2000a. Personal communication.

del Val, A. 2000b. On some tractable classes in deduction and abduction. *Artificial Intelligence 116*, 297–313.

El Fattah, Y. and Dechter, R. 1996. An evaluation of structural paramters for probabilistic reasoning: Results on benchmark circuits. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)* (1996), pp. 244–251.

Forbus, K. D. and de Kleer, J. 1993. *Building Problem Solvers*. MIT Press.

Gergov, J. and Meinel, C. 1994. Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Transactions on Computers 43*, 10, 1197–1209.

Hamscher, W., Console, L., and de Kleer, J. 1992. *Readings in Model-Based Diagnosis*. Morgan Kaufmann Publishers, Inc., San Mateo, California.

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

Lin, F. and Reiter, R. 1994. Forget it! In *Working notes: AAAI Fall Symposium on Relevance* (1994).

Marquis, P. 1995. Knowledge compilation using theory prime implicates. In *Proc. International Joint Conference on Artifical Intelligence (IJCAI)* (1995), pp. 837–843.

Meinel, C. and Slobodova, A. 1998. Sample method for minimization of obdds. In *SOFSEM'98, Jasna (Slowakia)* (1998), pp. 419–428.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence 32*, 57–95.

Robertson, N. and Seymour, P. D. 1986. Graph minors II: Algorithmic aspects of treewidth. *J. Algorithms 7*, 309–322.

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

Sieling, D. and Wegener, I. 1995. Graph driven BDDs — a new data structure for Boolean functions. *Theoretical Computer Science 141*, 283–310.

Williams, B. C. and Nayak, P. P. 1997. A reactive planner for a model-based executive. In *Proc. International Joint Conference on Artifical Intelligence (IJCAI)* (1997).