Compiling Knowledge into Decomposable Negation Normal Form

Adnan Darwiche
Cognitive Systems Laboratory
Department of Computer Science
University of California
Los Angeles, CA 90024
darwiche@cs.ucla.edu

Abstract

We propose a method for compiling propositional theories into a new tractable form that we refer to as decomposable negation normal form (DNNF). We show a number of results about our compilation approach. First, we show that every propositional theory can be compiled into DNNF and present an algorithm to this effect. Second, we show that if a 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. Third, we show that once a propositional theory is compiled into DNNF, a number of reasoning tasks, such as satisfiability and forgetting, can be performed in linear time. Finally, we 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 for their ability to answer queries.

1 Introduction

Compiling propositional theories has emerged as a new technique for enhancing the computational efficiency of automated reasoning systems. The basic idea here is to split the computational effort of such systems into two phases, off-line and on-line. In the off-line phase, a propositional theory is compiled into a tractable form which is then used in an on-line phase to answer multiple queries. The main value of such compilation is that most of the computational overhead is shifted into the off-line phase, which is amortized over all on-line queries.

One of the key approaches for compiling propositional theories has been proposed in [7]. Here, a propositional theory is compiled in an off-line phase into a Horn theory, which is used in an on-line phase to answer multiple queries. As it is not always possible to compile a propositional theory into a Horn theory, the propositional theory is generally compiled into two Horn theories, which approximate the original theory from below and above in terms of logical strength.

In this paper, we propose to compile propositional theories into a new form, which we call *decomposable negation normal form (DNNF).* This form is a generalization of disjunctive normal form (DNF) and a specialization of *negation normal form (NNF)* [I]. DNNF is tractable as the satisfiability of theories expressed in DNNF can be decided in linear time. In fact, a number of other interesting reasoning tasks, such as *forgetting* [6], can also be performed in linear time on theories expressed in DNNF.

We show a number of results about our compilation approach. First, contrary to compilations into Horn theories, we show that every propositional theory can be compiled into DNNF and present an algorithm to this effect. Second, we show that if a clausal form has a bounded treewidth, then it has a linear DNNF compilation which can be computed in linear time. Here, treewidth is a graph-theoretic parameter which measures the connectivity of a given clausal form. Even when the clausal form does not have a bounded treewidth, we show that its DNNF compilation is exponential only in its treewidth and linear in all other aspects. Finally, we present two techniques for approximating the DNNF compilation of a propositional theory in case 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 queries.

This paper is structured as follows. Section 2 introduces DNNF and its various properties. Section 3 discusses the compilation of propositional theories into DNNF. Section 4 discusses the two techniques for approximating a DNNF compilation and Section 5 focuses on the operation of forgetting. Finally, Section 6 closes with some concluding remarks. Proofs of theorems can be found in the long version of the paper [4].

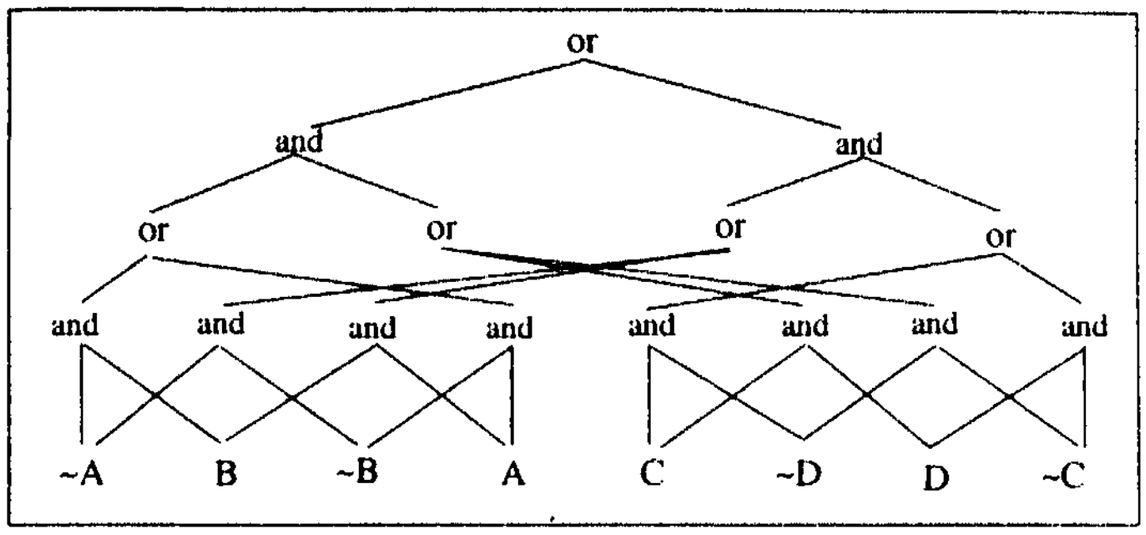

Figure 1: A propositional sentence in DNNF.

2 Decomposable NNF

A propositional sentence is in negation normal form (NNF) if it is constructed from literals using only the conjoin and disjoin operators [I]. Figure 1 shows a sentence in NNF depicted as a rooted, directed acyclic graph where the children of each node are shown below it in the graph. Each leaf node represents a literal and each non-leaf node represents a conjunction or a disjunction. We will also allow *true* and -^ *false* to appear as leaves in a DNNF to denote a conjunction with no conjuncts. Similarly, we will allow *false* and *-trne* as leaves to represent a disjunction with no disjuncts. The size of an NNF is measured by the number of edges in its graphical representation. Note that every disjunctive normal form (DNF) is an NNF, and that every conjunctive normal form (CNF) is an NNF. There are NNFs, however, that are neither DNFs nor CNFs.

Our concern here is mainly with a subclass of NNFs:

Definition 1 *A decomposable negation normal form (DNNF) is a negation normal form satisfying the decomposability property: for any conjunction /\,* a, *appearing in the form, no atoms are si tared by the conjuncts* a^.

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.[1] Therefore, all properties that we shall prove of DNNFs also hold for DNFs. A question that may arise then is why not compile propositional theories into DNFs? As it turns out, there are propositional theories that have linear DNNF representations, yet exponential DNF representations. 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 representation for *n — A.*

Propositional theories in DNNF are tractable:

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

1. Deciding the satisfiability of a DNNF can be done in linear time.
2. Forgetting about some atoms in a DNNF can be done in linear time [6].
3. Computing the minimum cardinality of models that satisfy a DNNF can be done in linear time, where cardinality is the number of atoms that are set to true (or false) by the model [3].

The last task has applications to model-based diagnosis and is outside the scope of this paper. Our focus here will be on the first two tasks, which we consider next.

By a clause (over distinct atoms *p1,...* ,p_n), we will mean a disjunction $l_1 \vee \ldots \vee l_n$, where l_i is either p_i or $\neg p_i$. By an instantiation (of distinct atoms $p_1, \ldots, p_n$), we will mean a conjunction $l_1 \wedge \ldots \wedge l_n$. We start with a linear test for deciding the satisfiability of NNFs.

Definition 2 *Let* SAT? *be a predicate over NNFs defined as follows.* SAT?(/) *is true where I is a literal.* $\text{SAT?}(\bigwedge_i \alpha_i)$ *is true iff each* $\text{SAT?}(\alpha_i)$ *is true,* $\text{SAT?}(\bigvee_i \alpha_i)$ *is true iff some* $\text{SAT?}(\alpha_i)$ *is true.*

It should be clear that the predicate SAT?(a) can be evaluated in time which is linear in the size of NNF a. The previous test is sound and complete for DNNFs:

Theorem 1 *DNNF α is satisfiable iff* $\text{SAT?}(\alpha)$ *is true.*

Now that we have a satisfiability test, we can also define an entailment test. Specifically, to test whether α entails clause β, we only need to test whether $\alpha \wedge \neg\beta$ is satisfiable. Note, however, that even though both *a* and $\neg\beta$ may be in DNNF, their conjunction $\alpha \wedge \neg\beta$ is not guaranteed to be in DNNF as α and $\neg\beta$ may share atoms. This can be easily dealt with, however, using the notion of conditioning:

Definition 3 *Let α be a propositional sentence and let γ be an instantiation. The conditioning of α on γ, written $\alpha \mid \gamma$, is the sentence which results from replacing each atom p in α with true if the positive literal p appears in γ and with false if the negative literal $\neg p$ appears in γ.*

For example, conditioning the DNNF $(\neg A \wedge \neg B) \vee (B \wedge C)$ on instantiation $B \wedge D$ gives $(\neg A \wedge \neg true) \vee (true \wedge C)$ and conditioning it on $\neg B \wedge D$ gives $(\neg A \wedge \neg false) \vee (false \wedge C)$ Conditioning allows us to eliminate reference to atoms while preserving satisfiability:

Theorem 2 *For DNNF a and instantiation γ, $\alpha \mid \gamma$ is in DNNF, and $\alpha \mid \gamma$ is satisfiable i$\alpha \wedge \gamma$ s satisfiable.*

Therefore, to test whether DNNF α entails clause β„ we only need to test whether *a* | –ZB is satisfiable, which is guaranteed to be in DNNF. We can now define a linear entailment test for DNNFs. Actually, we will (more generally) define it for NNFs:

Definition 4 *For NNF α and clause β, define $\alpha \vdash \beta$ to be true when* $\text{SAT?}(\alpha \mid \neg\beta)$ *is false, where $\neg\beta$ is the instantiation negating clause β.*

This linear test is both sound and complete for DNNFs:

Theorem 3 *For DNNF a and clause β, $\alpha \vdash \beta$ iff $\alpha \models \beta$*

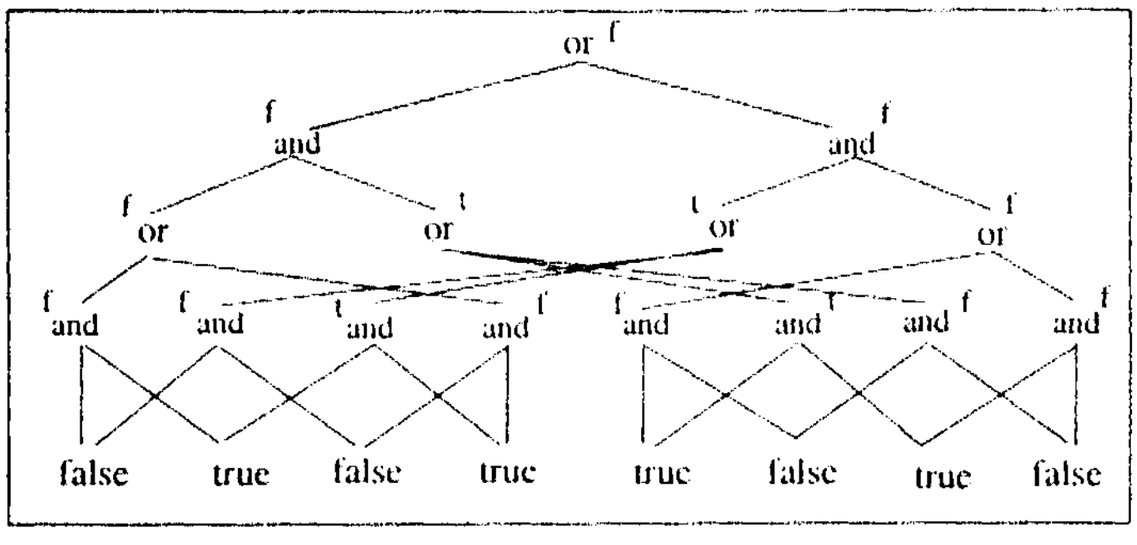

Figure 2: A prepositional sentence in DNNF.

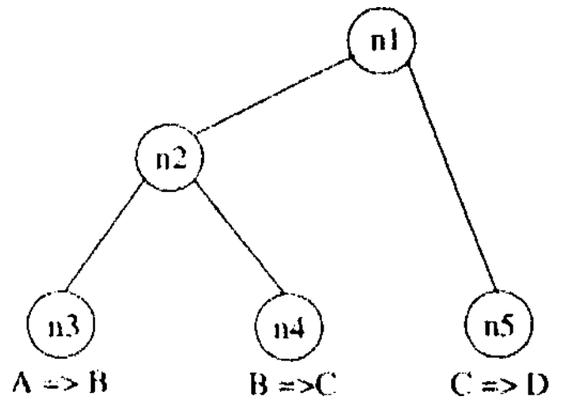

Figure 3: A decomposition tree.

Consider the DNNF ά in Figure 1, the clause $\beta = \neg A \vee \neg B \vee \neg C \vee \neg D$, and suppose we want to test whether $\alpha \models \beta$. Theorem 3 suggests that we condition *ά* on $\neg\beta$, to yield $\alpha \mid \neg\beta$, and then test whether SAT?$(\alpha \mid \neg\beta)$ is false. Figure 2 depicts the conditioning of a on $\neg\beta$ and the result of applying the SAT? test. Since SAT?$(\alpha \mid \neg\beta)$ is false, we conclude $\alpha \vdash \beta$ and also $\alpha \models \beta$.

Before we close this section, we present three important results on DNNF entailment. First, that the entailment test $\vdash$ is sound with respect, to sentences in NNF:

Theorem 4 *For NNF* α, $\alpha \vdash \beta$ *only if* $\alpha \models \beta$.

That is, even though α 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 5 *NNF ά is decomposable, except on atoms X iff for any conjunction* $\bigwedge_i \beta_i$ *that appears in* α, *only atoms in X are shared by the conjunct* β_i.

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

Theorem 5 *Let* α *be an NNF -which, is decomposable except on X. Let* β *be a clause which mentions all atoms in X. Then* $\alpha \vdash \beta$ *iff* $\alpha \models \beta$

Consider the NNF $\alpha = (\neg A \vee B) \wedge (\neg B \vee C)$ and the queries $A \supset B$, $B \supset A$ and $A \supset C$. Since α 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 be. Finally:

Theorem 6 *For NNF* α, *which is decomposable except on X, and for clause* β, $\alpha \models \beta$ *iff* $\alpha \vdash \gamma_i \vee \beta$ *for each clause* γ_i *over atoms appearing in X but not in B.*

Therefore, if the size of *X* is bounded by a constant, then $\alpha \models \beta$ can be decided in linear time for any query B, even though *ά* itself is not decomposable.

3 Compiling Knowledge into DNNF

We established two main results in the previous section. First, we identified the class of DNNF theories. Second, we showed that satisfiability and entailment can be decided in linear time with respect to DNNF theories. 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.

The following theorem is the key to proving that even propositional theory can be converted into DNNF.

Theorem 7 *Let* Δ_1 *and* Δ_2 *be two propositional sentences in DNNF. Let* Δ *be the sentence* $\bigvee_\beta (\Delta_1 \mid \beta) \wedge (\Delta_2 \mid \beta) \wedge \beta$, *where* β *is an instantiation of all atoms shared by* Δ_1 *and* Δ_2. *Then* Δ *is in DNNF and* Δ *is equivalent to* $\Delta_1 \wedge \Delta_2$

Here is a recursive algorithm DNNF1(Δ), based on the above theorem, which converts any clausal form Δ intc an equivalent theory in DNNF:

1. If Δ contains a single clause *ά,* DNNFI$(\Delta) \leftarrow \alpha$.
2. Otherwise, DNNF1$(\Delta) \leftarrow \bigvee_\beta$ DNNF1$(\Delta_1 \mid \beta) \wedge$ 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 o the atoms shared by Δ_1 arrd Δ_2.

This algorithm converts any theory in clausal form intf an equivalent theory in DNNF, but at the expense o: increasing the theory size. The increase in size comet mainly from the case analysis performed on the atoms shared by the sub-theories Δ_1 and Δ_2. Consider the theory $\Delta = (A \supset B) \wedge (B \supset C)$ and let $\Delta_1 = A \supset B$ and $\Delta_2 = B \supset C$. Then DNNF1(Δ) = (DNNF1$(\Delta_1 \mid B) \wedge$ DNNF1$(\Delta_2 \mid B) \wedge B) \vee$ (DNNF1$(\Delta_1 \mid \neg B) \wedge$ 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 very 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 6 *A decomposition tree T for clausal form* Δ *is a full binary tree whose leaves correspond to the clauses in* Δ. *If N is the leaf node corresponding to clause A in* Δ, *then* $\Delta(N) \stackrel{def}{=} \{\alpha\}$.

Algorithm DNNF1

/* N is a tree node ; α is an instantiation */
DNNF1(N, α)
if N is a leaf node & $\Delta(N) = \{\phi\}$, then $\gamma \leftarrow \phi \mid \alpha$
else $\gamma \leftarrow \bigvee_\beta$ DNNF1$(N_l, \alpha \wedge \beta) \wedge$ DNNF1$(N_r, \alpha \wedge \beta) \wedge \beta$
where β ranges over all instantiations
of $atoms(N_l) \cap atoms(N_r) - atoms(\alpha)$
return γ

Figure 4: Compiling a theory into DNNF.

Algorithm DNNF2

DNNF2(N, α)
$\psi \leftarrow project(\alpha, atoms(N))$
if CACHE$_N(\psi) \neq$ NIL, return CACHE$_N(\psi)$
if N is a leaf node & $\Delta(N) = \{\phi\}$, then $\gamma \leftarrow \phi \mid \alpha$
else $\gamma \leftarrow \bigvee_\beta$ DNNF2$(N_l, \alpha \wedge \beta) \wedge$ DNNF2$(N_r, \alpha \wedge \beta) \wedge \beta$
where β ranges over all instantiations
of $atoms(N_l) \cap atoms(N_r)$ -- $atoms(\alpha)$
CACHE$_N(\psi) \leftarrow \gamma$
return γ

Figure 5: Compiling a theory into DNNF.

Figure 3 depicts a decomposition tree for the theory A 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 *N:* N_1 and N_r denote the loft and right children of *N,* respectively, and $\Delta(N) \stackrel{def}{=} \Delta(N_l) \cup \Delta(N_r)$. Second, *atoms(N)* is defined as the set of atoms appearing in clauses $\Delta(A^r)$. For example, in Figure 3, $\Delta(n_2)$ contains the clauses $A \supset B$ and $B \supset C$ and *atoms(112)* contains the atoms *A,B* and C.

Given a. decomposition tree for theory Δ, Figure 1 depicts an algorithm which compiles Δ into DNNF.

Theorem 8 DNNFI(N, α) *return* $\Delta_{\wedge}(A^r) \mid \alpha$ *in DNNF.*

Therefore, to convert a theory Δ into DNNF, we first construct a decomposition tree *T* for Δ and call DNNFI (N *true)* with N being the root of tree *T.* The following is an important observation about DNNFI:

Theorem 9 *If* α *and* α' *agree on atoms(N), then* DNNFI(N, α) *is equivalent to* DNNFI(N, α')

Therefore, we can improve on DNNFI by associating a cache with each node *N* to store[1] the result of DNNFI(N, α) indexed by the projection of instantiation α on *atoms(N),* denoted *project* $(\alpha, atoms(N))$. When another recursive call DNNFI(N, α') is made, we first check the cache of node N to see whether we have an entry for *project*$(\alpha', atoms(N))$. If we do. we return it. Otherwise, we continue with the recursion. This improvement leads to the refined algorithm in Figure 5.

We now address the complexity of DNNF2.

Definition 7 *Let N be a 'node in a decomposition tree T. The cluster of node N is defined as follows:*

- *If N is a leaf node, then its cluster is atoms(N).*
- *If N is an internal node, then its cluster is the set of atoms that appear either*
  - *— above and. below node N in the tree; or*
  - *— in the left and rigid subtrees of node N.*

The width of a decomposition tree is the size of its maximal cluster minus one.

In Figure 3, wo have *cluster* $[n_1)$ — *\C}, cluster* (n_2) — {B,C}, *clustering - {A,B}, cluster*$\{n_A)$ = *{B,C}* and *cluster*(n_5) = *{C,D}.* Therefore, the decomposition tree has width 1.

Theorem 10 *Let T be the decomposition tree used in Figure 5. The time and space complexity of the algorithm in Figure 5 is* $0\{nw2^w)$, *where n is the number of leaf nodes in tree T and w is its width.*[2]

Therefore, the complexity of compiling a propositional theory into DNNF depends crucially on the quality (width) of decomposition tree used. The question now is how to construct good decomposition trees (ones with small width)? As it turns out, there, is a device in the literature on graph-based reasoning, known as a jointree, which can be easily converted into a decomposition tree. A jointree also has a width and good jointrees are those with small width. We can easily convert, a jointree into a decomposition tree while maintaining its width. Therefore, any good method for constructing jointrees is also a good method for constructing decomposition trees. A jointree, however, is constructed for an undirected graph while a decomposition tree is constructed for a propositional theory. The following definition makes the connection.

Definition 8 *[5] Let* Δ *be a propositional theory in clausal form. The interaction graph for* Δ *is the undirected graph G constructed as follows. The nodes of G arc the atoms of* Δ. *There is an edge between two atoms in G iff the atoms appear in the same clause of* Δ

Theorem 11 *Let* Δ *be a propositional theory in clausal form and let G be its interaction graph. Let J be a jointree for G with, width. w There is a decomposition tree for* Δ *which has width,* $\leq w$ *and which, can be constructed from J in time linear in the size of J.*

The width of the best jointree for a graph *G* is known as the *treewidth* of *G.* If the treewidth of a graph is bounded by a constant *w,* then one can construct an optimal jointree in linear time [2|. A major implication of this result is that if a clausal form Δ has an interaction graph with a bounded treewidth, then (a) computing an optimal decomposition tree (jointree) for that theory, (b) compiling the theory based on the computed decomposition tree, and (c) answering queries based on

[2]Note that if *T* is a decomposition tree for a clausal form Δ, then *n* is also the number of clauses in Δ

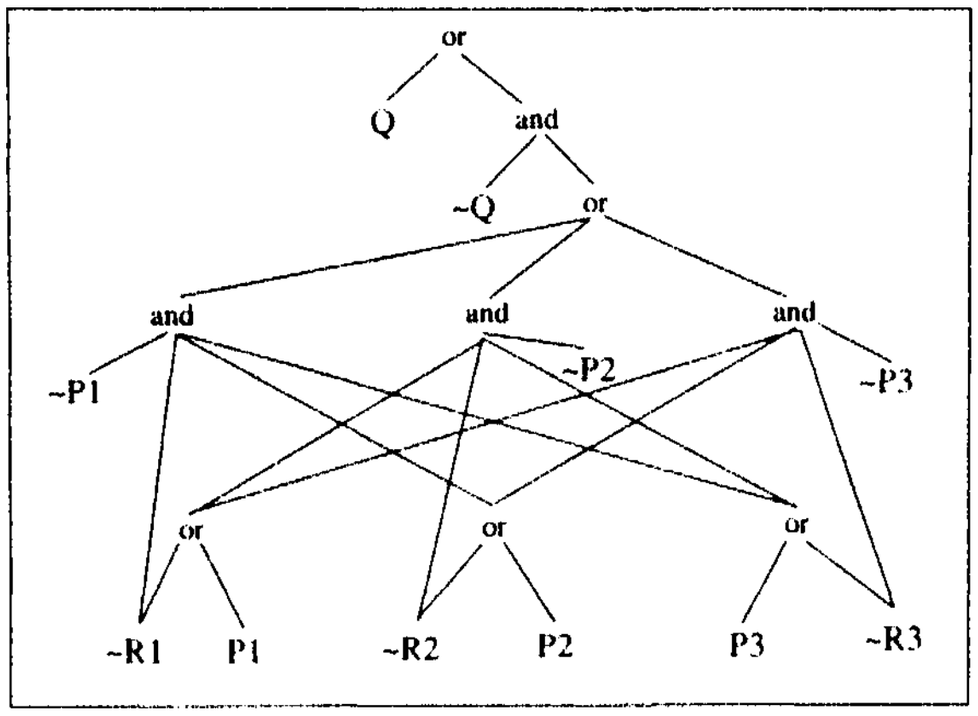

Figure 6: A prepositional sentence in DNNF.

the resulting compilation, can all be done in linear time. This is our central result on the complexity of compiling theories into DNNF.[3]

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 *0(n²)* (shown in Figure 6 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* [7], This shows that there are theories with exponential Horn approximations, yet polynomial DNNF representations.

4 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 5. It should be clear that the reason for possible intractability in this algorithm is the size of $\mathcal{S}(N,\alpha) \stackrel{def}{=} atoms$(N/) $\cap$ *atoms* (N_r) - *atoms(a),* which contains some of the atoms shared by sub-theories $\Delta(N_l)$ and $\Delta(N_r)$. Specifically, the algorithm will consider a number of instantiations β which is exponential in the size of $\mathcal{S}(N,\alpha)$. 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 $\mathcal{S}(N,\alpha)$ by performing a case analysis on only a subset of $\mathcal{S}(N,\alpha)$. That is, we consider all instantiations β of a subset of $\mathcal{S}(N,\alpha)$. 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 some instantiations β of $\mathcal{S}(N,\alpha)$. This leads to a variation on algorithm DNNF2 which we call DNNF/.

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 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 12 $\text{DNNF}_u(N,\alpha)$ *is in NNF and is equivalent to* $\text{DNNF2}(N,\alpha)$

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.

Corollary 1 $\text{DNNF}_u(N,\alpha) \vdash \beta$ *only if* $\text{DNNF2}(N,\alpha) \vdash \beta$.

Here is the guarantee about the second approximation:

Theorem 13 $\text{DNNF}_l(N,\alpha)$ is *in DNNF and* $\text{DNNF}_l(N,\alpha) \models \text{DNNF2}(N,\alpha)$

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/ is complete, although not necessarily sound, with respect to entailment.

Corollary 2 $\text{DNNF}_l(N,\alpha) \not\vdash \beta$ *only if* $\text{DNNF2}(N,\alpha) \not\vdash \beta$

Therefore, if the size of a DNNF compilation *T* is too large, we can replace it with two approximations T_1 and Γ_u. Given a query β, we first test whether $\Gamma_l \vdash \beta$ and $\Gamma_u \vdash \beta$. We have three possibilities: If $\not\vdash \beta$, 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 [7]. In their 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.

5 Compiling Out Atoms

Given a theory Δ, we may only be interested in queries β which do not mention atoms A'. In this case, it makes sense to compile Δ into a theory which does not mention atoms *X* either, yet is equivalent to Δ with respect to

[3]The satisfiability of this class of theories can also be decided in linear time using directional resolution [5].

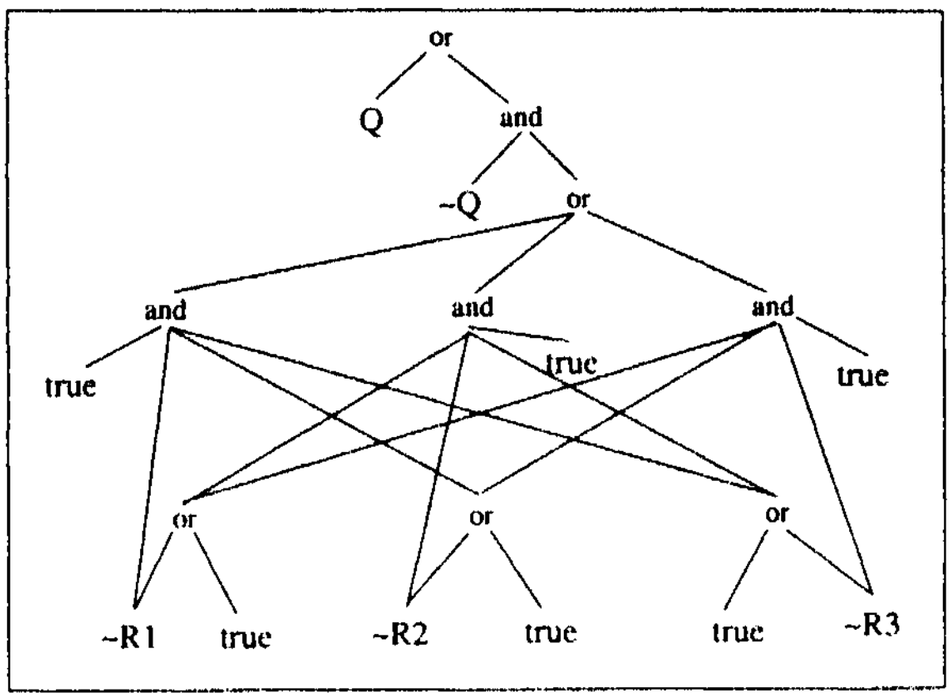

Figure 7: Forgetting atoms in a DNNF.

queries β. Such theory is the result of forgetting about atoms X in Δ [6]. As it turns out, once a theory is converted into DNNF, forgetting takes linear time:

Definition 9 *For DNNF α and atoms X, define* FORGET(α, X) *as the result of replacing each literal in á with true iff that literal refers to an atom in X.*

The following theorem shows that the above linear operation does correspond to forgetting as defined in [6]:

Theorem 14 *Let α be a sentence in DNNF and let β be a clause that does not mention atoms X. Then $\alpha \models \beta$ iff* FORGET$(\alpha, X) \models \beta$.

Consider the DNNF in Figure 6, which is equivalent to the theory $\Delta = \{P_1 \wedge P_2 \wedge P_3 \supset Q, R_1 \wedge \neg P_1 \supset Q, R_2 \wedge \neg P_2 \supset Q, R_3 \wedge \neg P_3 \supset Q\}$. Forgetting about atoms P_1, P_2, P_3 in this theory gives the DNNF in Figure 7, which can be easily simplified to $R_1 \wedge R_2 \wedge R_3 \supset Q$.

Forgetting has three major applications. First, reducing the size of a DNNF compilation by forgetting about atoms that would never appear in queries. Second, computing DNNF representations of Boolean functions. Consider the circuit in Figure 8 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 forget atoms *E, F* and G, 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.[4] A final application of forgetting is in computing the implications of a given theory on a particular set of atoms X (by forgetting about all atoms other than A"). This has proven to be very useful in model based diagnosis [3].

6 Conclusion

We have proposed an approach for compiling prepositional theories into a tractable form, which we refer to as decomposable negation normal form (DNNF). We have

[4]In [4], we compare DNNFs with Binary Decision Diagrams (BDDs) as a representation of Boolean functions.

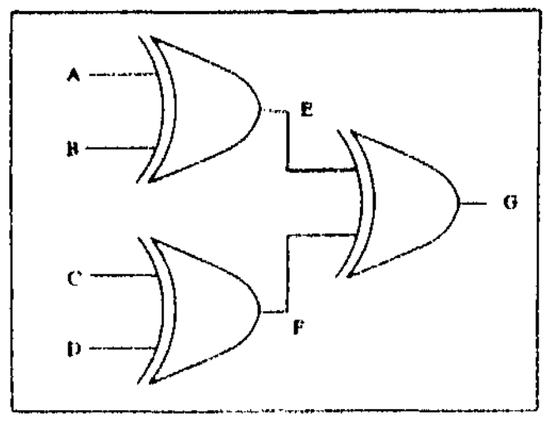

Figure 8: An odd-parity circuit.

shown that once a theory is compiled into that form, a number of reasoning tasks including entailment, satisfiability and forgetting can be accomplished in linear time. We have shown that every propositional theory can be compiled into DNNF and presented an algorithm to this effect. We then presented a key result according to which the time and space complexity of our compilation technique is linear given that the propositional theory has a clausal form with bounded treewidth. Finally, we presented two techniques for approximating DNNF compilations. One of the techniques generates sound compilations, the other generates complete compilations. Together, the two approximations bound the original theory from below and above in terms of their ability to answer queries. There are at least three key distinctions between our compilation approach and the one proposed in [7]. First, every theory has a DNNF representation, while not every theory lias a Horn representation. Second, there are theories with exponential Horn approximations, yet polynomial DNNF representations. Third, we have characterized a class of theories which is guaranteed to have linear DNNF compilations. We are not aware of a similar guarantee on the Horn approximations of this class of theories.

References

[I] Jon Barwise, editor. *Handbook of Mathematical Logic.* North-Holland, Amsterdam, 1977.

[2] Hans. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing,* 25(6):1305-1317, 1996.

[3] Adrian Darwiche. Compiling devices: A structure-based approach. In *KR98,* pages 156 166, 1998.

[4] Adnan Darwiche. Compiling knowledge into decomposable negation normal form. Technical Report R-262, Cognitive Systems Laboratory, UCLA, 1999.

[5] Rina Dechter and Irina Rish. Directional resolution: The davis-putnam procedure, revisited. In *KR94,* pages 134-145, 1994.

[6] Fangzhen Lin and Ray Reiter. Forget it! In *Working notes: AAAI Fall Symposium on Relevance,* 1994.

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