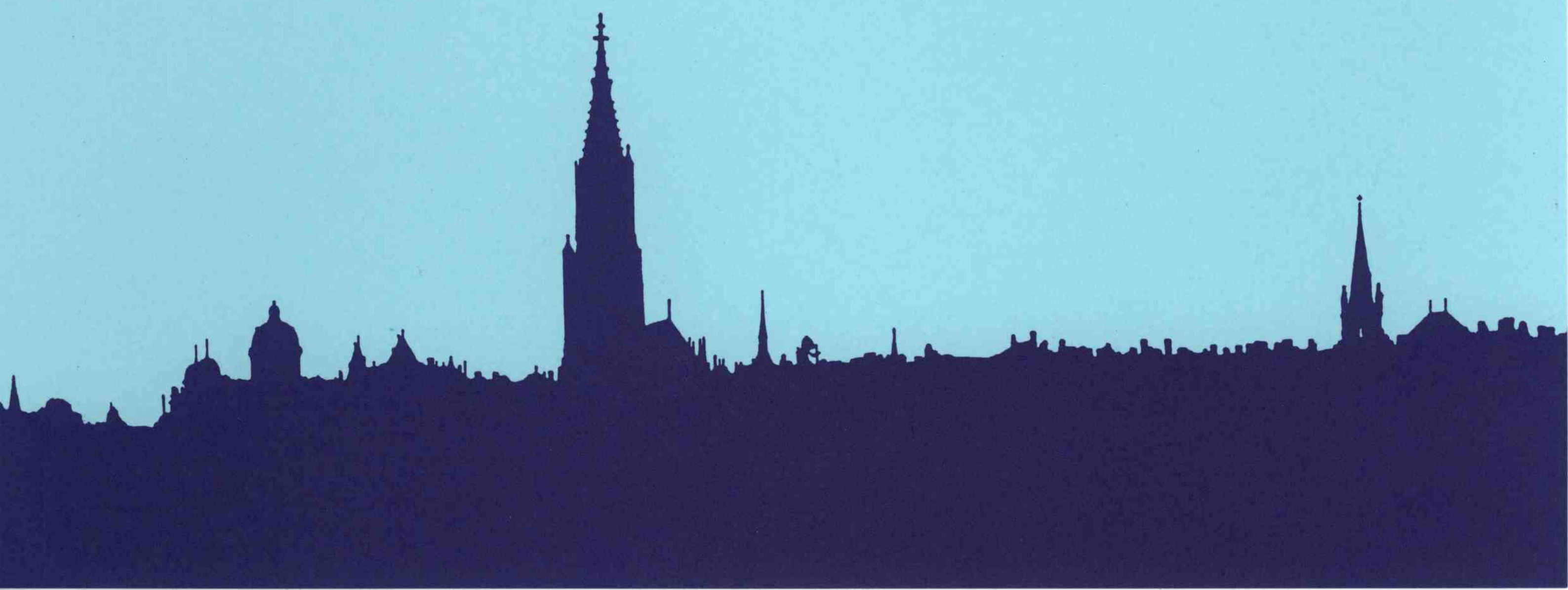
universität bern
institut für informatik
und angewandte mathematik

Logical Compilation of Bayesian Networks

Michael Wachter & Rolf Haenni

iam-06-006

August 2006



Address:
Institut für Informatik und angewandte Mathematik
Universität Bern
Neubrückstrasse 10
CH-3012 Bern

e-mail: office@iam.unibe.ch
Phone: 0041 (0)31 631 86 81
Fax: 0041 (0)31 631 32 62

Logical Compilation of Bayesian Networks

Michael Wachter & Rolf Haenni
University of Bern
Institute of Computer Science and Applied Mathematics
CH-3012 Bern, Switzerland
{wachter,haenni}@iam.unibe.ch

Abstract

This paper presents a new approach to inference in Bayesian networks with Boolean variables. The principal idea is to encode the network by logical sentences and to compile the resulting CNF into a deterministic DNNF. From there, all possible queries are answerable in linear time relative to its size. This makes it a potential solution for real-time applications of probabilistic inference with limited computational resources. The underlying idea is similar to Darwiche's differential approach to inference in Bayesian networks, but the core of the proposed CNF encoding is slightly different. This alternative encoding enables a more intuitive and elegant solution, which is apparently more efficient.

CR Categories and Subject Descriptors:

I.2.3 [Artificial Intelligence]: Deduction and Theorem Proving – uncertainty, "fuzzy" and probabilistic reasoning;

I.2.3 [Artificial Intelligence]: Deduction and Theorem Proving – inference engines;

I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods – Representation languages

General Terms:
Algorithms, Theory

1 Introduction

As Bayesian networks are more and more applied to complex real-world applications, the development of fast and flexible inference methods becomes increasingly important. In the last decades, researchers have developed various kinds of exact and approximate inference algorithms, each of them with corresponding advantages and disadvantages. Some methods are particularly designed for real-time inference with limited computational resources such as time or memory. See [13] for a comprehensive and compact survey.

A particular real-time inference method is the differential approach proposed in [9]. It suggests to view a Bayesian network as a *multi-linear function* (MLF), the so-called *network polynomial*, from which answers to probabilistic queries are retrieved by differentiating the polynomial. Relative to the given Bayesian network, the network polynomial is exponential in size, but it is possible to efficiently encode it by a CNF of linear size. As suggested in [8], this CNF is then compiled into a *decomposable negation normal form* (DNNF) with the additional properties of *smoothness* and *determinism* [7]. The resulting sd-DNNF is an intermediate step, from which an arithmetic circuit is extracted, whose size is not necessarily exponential relative to the original Bayesian network. This arithmetic circuit is guaranteed to compute the original network polynomial, and can therefore be used to obtain all necessary partial derivatives in time (and space) linear in its size. In its essence, the aim of the whole procedure is to generate a preferably optimal factoring of the network polynomial.

Such a logical approach is beneficial in many ways. The most important advantage is the ability to encode *context-specific independences*, i.e. local regularities in the given conditional probability tables [2]. This inherently includes appropriate solutions for the particular type of CPT obtained from noisy-OR and noisy-AND nodes or from purely logical relations. In comparison with classical inference methods such as join-tree propagation or message-passing, which do not directly exploit such local structures, there has been reports of tremendous improvements in both compile time and online inference [3, 4]. Another advantage is the ability to efficiently update numerical computations with minimal computational overhead. This is a key prerequisite for experimental sensitivity analyses.

1.1 Overview of Method

The starting point of our method is a logical representation ψ of the Bayesian network. For this, a proposition $\theta_{x|\mathbf{y}}$ is attributed to each CPT entry $P(x|\mathbf{y})$ of a network variable X. In this paper, our discussion is restricted to Boolean variables, i.e. we can attribute an additional proposition x to each network variable X and use it for the event $X{=}true$ and its negation for $X{=}false$. As a result, the logical representation ψ consists of two types of propositions, the ones linked to the CPT entries and the ones linked to the network variables. The corresponding sets of propositions are denoted by Θ and Δ, respectively.

In order to use the logical representation ψ to compute the posterior probability $P(\mathbf{q}|\mathbf{e}) = P(\mathbf{q}\wedge\mathbf{e})/P(\mathbf{e})$ of a query event $\mathbf{q} = q_1 \wedge \cdots \wedge q_r$ given the evidence $\mathbf{e} = e_1 \wedge \cdots \wedge e_s$, it is sufficient to look at the simpler problem of computing prior probabilities $P(\mathbf{x})$ of arbitrary conjunctions $\mathbf{x} = x_1 \wedge \cdots \wedge x_r$ in order to obtain corresponding numerators $P(\mathbf{q} \wedge \mathbf{e})$ and denominators $P(\mathbf{e})$. Our solution for this consists of the following three steps:

1. Condition ψ on $\mathbf{x}$ to obtain $\psi|\mathbf{x}$.
2. Eliminate (forget) from $\psi|\mathbf{x}$ the propositions Δ. The resulting logical representation of $[\psi|\mathbf{x}]^{-\Delta}$ consists of propositions from Θ only.
3. Compute the probability of the event represented by $[\psi|\mathbf{x}]^{-\Delta}$ to obtain $P(\mathbf{x}) = P([\psi|\mathbf{x}]^{-\Delta})$. For this, we assume that the propositions $\theta_{x|\mathbf{y}} \in \Theta$ are probabilistically independent and that $P(\theta_{x|\mathbf{y}}) = P(x|\mathbf{y})$ are the respective marginal probabilities.

For the choice of an appropriate target compilation language for ψ, it is thus necessary to select a language that supports two transformations (conditioning and forgetting) and one query (probability computation) in polynomial time. At first sight, just by looking at the results given in [11] or [18], it seems that no such language exists. However, as we will see in this paper, we can exploit the fact that the propositional variables in Δ satisfy a certain property w.r.t. ψ. The particular form of forgetting such *deterministic* variables will be called *deterministic forgetting*, and we will see that it is (at least) supported by DNNFs and d-DNNFs. Among them, probability computations are only supported by d-DNNFs, and our search for an appropriate target compilation language for Bayesian networks thus leads to d-DNNFs, the only representation language that supports all necessary operations of the above procedure in polynomial time.

1.2 Contribution and Outline

The conclusion that d-DNNFs should be used as target compilation language for Bayesian networks confirms Darwiche's precursory work in [8], but it also shows that Darwiche's additional requirement of smoothness is dispensable. While the actual reasons for this conclusion and the exact role of smoothness remain rather nebulous in [8], a precise and conclusive explanation in terms of the (extended) knowledge compilation map is given in this paper.

Another contribution of this paper is the proposal of an alternative CNF encoding, which finally enables a more direct computational procedure in terms of a few basic operations of the knowledge compilation map. In our opinion, this is a significant simplification over Darwiche's original method of viewing posterior probabilities as partial derivatives of multi-linear functions, from which the rather cumbersome process of transforming the CNF encoding via a smooth d-DNNF to an arithmetic circuit (with all negative literals set to 1) results. In the light of this paper, some steps of this process appear as an unnecessary detour.

In a nutshell, we believe that the method of this paper is an important contribution to the area of compiling Bayesian networks, mainly as a significant advancement in terms of clarity and simplicity. First steps towards empirically testing the efficiency of the proposed method also report some considerable improvements (see Section 4), but this has not yet been verified on a broader scale.

The structure of this paper is as follows. Section 2 provides a short summary of possible representations of Boolean functions and the corresponding knowledge compilation map. We will also introduce the concepts of deterministic variables and deterministic forgetting, and extend the knowledge compilation map accordingly. The topic of Section 3 is the logical representation and evaluation of Bayesian networks. This part includes the main theorems of the paper. Section 4 displays the differences to the logical representation proposed by Darwiche. Section 5 concludes the paper.

2 Representing Boolean Functions

Consider a set V of r Boolean variables and a Boolean function (BF) $f : \mathbb{B}^r \rightarrow \mathbb{B}$ with $\mathbb{B} = \{0, 1\}$. Such a function f can also be viewed as the set of r-dimensional vectors $\mathbf{x} \in \mathbb{B}^r$ for which f evaluates to 1. This is the so-called *satisfying set* $S_f = \{\mathbf{x} \in \mathbb{B}^r : f(\mathbf{x}) = 1\}$ of f, for which an efficient representation has to be found [5].

2.1 Representation Languages

To start with the least restrictive view w.r.t. possible representation languages, consider the concept of a *propositional* DAG (or PDAG for short). According to [18], PDAGs are rooted, directed, acyclic graphs, in which each leaf node is represented by ○ and labeled with $\top$ (true), $\bot$ (false), or $x \in V$. Each non-leaf node is represented by △ (logical and), ▽ (logical or), or ⋄ (logical not). The set of all possible PDAGs of V is called *language* and denoted by $\texttt{PDAG}_V$ or simply `PDAG`. The example depicted in Fig. 1 represents the odd parity function with respect to $V = \{a, b, c, d\}$.

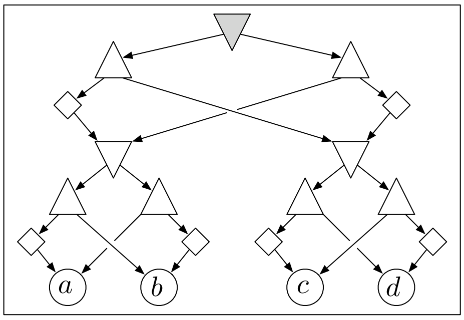


Figure 1: A PDAG representing the odd parity function with respect to $V = \{a, b, c, d\}$.

Leaves labeled with $\top$ ($\bot$) represent the constant BF which evaluates to 1 (0) for all $\mathbf{x} \in \mathbb{B}^r$. A leaf labeled with $x \in V$ is interpreted as the assignment $x = 1$, i.e. it represents the BF which evaluates to 1 iff $x = 1$. The BF represented by a $\triangle$-node is the one that evaluates to 1, iff the BFs of all its children evaluate to 1. Similarly, a $\triangledown$-node represents the BF that evaluates to 1, iff the BF of at least one child evaluates to 1. Finally, a $\diamond$-node represents the complementary BF of its child, i.e. the one that evaluates to 1, iff the BF of its child evaluates to 0. The BF of an arbitrary $\varphi \in \texttt{PDAG}$ will be denoted by f_φ and its satisfying set by S_φ. Two PDAGs $\varphi, \psi \in \texttt{PDAG}$ are *equivalent*, $\varphi \equiv \psi$, iff $f_\varphi = f_\psi$.

Our convention is to denote PDAGs by lower-case Greek letters such as φ, ψ, or the like. The set of variables included in $\varphi \in \texttt{PDAG}$ is denoted by $vars(\varphi) \subseteq V$. The number of edges of φ is called its *size* and denoted by $|\varphi|$. PDAGs may satisfy various properties [11, 18], but in the context of this paper, only three of them are relevant:

- *Decomposability*: the sets of variables of the children of each $\triangle$-node α in φ are pairwise disjoint (i.e. if $\beta_1, \ldots, \beta_l$ are the children of α, then $vars(\beta_i) \cap vars(\beta_j) = \emptyset$ for all $i \neq j$);
- *Determinism*: the children of each $\triangledown$-node α in φ are pairwise logically contradictory (i.e. if $\beta_1, \ldots, \beta_l$ are the children of α, then $\beta_i \wedge \beta_j \equiv \bot$ for all $i \neq j$);
- *Simple-negation:* the child of each $\diamond$-node in φ is a leaf.

A de**c**omposable and **d**eterministic PDAG is called cd-PDAG, and `cd-PDAG` refers to the corresponding language, a sub-language of `PDAG`. The example shown in Fig. 1 is a cd-PDAG.

Darwiche's family of `NNF` (= `n-PDAG`) languages are sub-languages of `PDAG` satisfying simple-**n**egation, i.e. `DNNF` (= `cn-PDAG`) is the sub-language of `NNF` satisfying decomposability and `d-DNNF` (= `cdn-PDAG`) is the sub-language of `DNNF` satisfying determinism [11]. Other sub-languages are obtained from considering further properties, e.g. `OBDD` (ordered binary decision diagrams) is the sub-language of `d-DNNF` satisfying *decision*, *read-once*, and *ordering*, and `sd-DNNF` is the sub-language of `d-DNNF` satisfying *smoothness.*[1] The latter is used in [8] as target compilation language for Bayesian networks. For a more comprehensive overview and a detailed discussion we refer to [11, 18].

2.2 Succinctness, Queries, and Transformations

A language $\texttt{L}_1$ is *equally or more succinct* than another language $\texttt{L}_2$, $\texttt{L}_1 \preceq \texttt{L}_2$, if any sentence $\alpha_2 \in \texttt{L}_2$ has an equivalent sentence $\alpha_1 \in \texttt{L}_1$ whose size is polynomial in the size of α_2. A language

[1] Smoothness means that $vars(\beta_i) = vars(\beta_j)$ holds for each pair of children (β_i, β_j) of each $\triangledown$-node in φ.

$\texttt{L}_1$ is *strictly* more succinct than another language $\texttt{L}_2$, $\texttt{L}_1 \prec \texttt{L}_2$, iff $\texttt{L}_1 \preceq \texttt{L}_2$ and $\texttt{L}_2 \not\preceq \texttt{L}_1$. With respect to the above-mentioned languages, we have the following proven relationships [18]:

$$\texttt{PDAG} \prec \left\{ \begin{array}{cc} \texttt{DNNF} & \prec \\ \texttt{cd-PDAG} & \preceq \end{array} \right\} \texttt{d-DNNF} \prec \texttt{OBDD}.$$

It is still unknown whether `cd-PDAG` is strictly more succinct than `d-DNNF` or not.

Queries are operations that return information about a BF without changing its PDAG representation. The most important queries are *consistency* (`CO`) or *satisfiability* (SAT), *validity* (`VA`), *clause entailment* (`CE`), *term implication* (`IM`), *sentential entailment* (`SE`), *equivalence* (`EQ`), *model counting* (`CT`), *probabilistic equivalence* (`PEQ`), and *probability computation* (`PR`).

Finally, a *transformation* is an operation that returns a PDAG representing a modified BF. The new PDAG is supposed to satisfy the same properties as the language in use. The most important transformations are (*term*) *conditioning* (`TC`), *forgetting* (`FO`), *singleton forgetting* (`SFO`), *general/binary conjunction* ($\texttt{AND}/\texttt{AND}_2$), *general/binary disjunction* ($\texttt{OR}/\texttt{OR}_2$), and *negation* (`NOT`).

If a language supports a query or transformation in polynomial time with respect to the size of the involved PDAGs, we say that it *supports* this query or transformation. Table 1 shows the supported queries and transformations of the considered languages [11, 18].

	CO/CE	VA/IM	CT/PR/PEQ	EQ	SE	TC	FO	SFO	AND	$\texttt{AND}_2$	OR	$\texttt{OR}_2$	NOT
PDAG	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\surd$	$\circ$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
DNNF	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$	$\surd$	$\surd$	$\surd$	$\circ$	$\circ$	$\surd$	$\surd$	$\circ$
cd-PDAG	$\surd$	$\surd$	$\surd$	?	$\circ$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\surd$
d-DNNF	$\surd$	$\surd$	$\surd$	?	$\circ$	$\surd$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	?
OBDD	$\surd$	$\surd$	$\surd$	$\surd$	$\circ$	$\surd$	$\bullet$	$\surd$	$\bullet$	$\circ$	$\bullet$	$\circ$	$\surd$

Table 1: Sub-languages of the `PDAG` language and their supported queries and transformations. $\surd$ means "supports", $\bullet$ means "does not support", $\circ$ means "does not support unless $P = NP$", and ? means "unknown".

2.3 Deterministic Variables

It is interesting to see in Table 1 that forgetting is supported by `DNNF` but not by `d-DNNF` or `cd-PDAG`. This is a consequence of the fact that forgetting does not preserve determinism in general. Let us now have a look at the particular case of variables which preserve determinism while being forgotten.

Definition 1. For $\varphi \in \texttt{PDAG}$, the variable $x \in V$ is called *deterministic* w.r.t. φ, denoted by $x \,||\, \varphi$, iff $\varphi|x \wedge \varphi|\neg x \equiv \bot$.

The process of forgetting deterministic variables will be discussed in the next subsection. Before, let's have a look at some basic properties of deterministic variables.

Theorem 1. *$x \,||\, \varphi$ implies $x \,||\, \psi$ for all $\psi \models \varphi$.*

Theorem 2. *$x \notin vars(\varphi)$ implies $x \,||\, x \leftrightarrow \varphi$.*

The proofs of these theorems are included in the appendix. An immediate consequence is the following corollary, which will be useful to prove one of the main theorems of Section 3.

Corollary 1. $x \notin vars(\varphi)$ *implies* $x \,||\, (x \leftrightarrow \varphi) \wedge \psi$.

For the forgetting of more than one variable, it is useful to generalize the definition of a single deterministic variable to sets of deterministic variables.

Definition 2. For $\varphi \in$ `PDAG`, the set of variables $\{x_1, \ldots, x_n\} \subseteq V$ is called *deterministic* w.r.t φ, denoted by $\{x_1, \ldots, x_n\} \,||\, \varphi$ or simply $x_1, \ldots, x_n \,||\, \varphi$, iff $\varphi|\mathbf{x} \wedge \varphi|\mathbf{x}' \equiv \bot$ for all instantiations $\mathbf{x} \neq \mathbf{x}'$ of the variables $x_1, \ldots, x_n$.

Note that $x, y \,||\, \varphi$ implies $x \,||\, \varphi$ and $y \,||\, \varphi$, while the converse is not always true.

2.4 Deterministic Forgetting

Let $W \subseteq V$ be a subset of variables, $x \in V$ a single variable, and φ an arbitrary PDAG. Forgetting the variables W from φ generates a new PDAG φ^{-W}, in which the variables from W are no longer included, and such that its satisfying set $S_{\varphi^{-W}}$ is the projection of S_φ to the restricted set of variables $V \setminus W$. In the literature, forgetting was originally called *elimination of middle terms* [1], but it is also common to call it *projection*, *variable elimination*, or *marginalization* [16]. There is also a one-to-one analogy to the elimination of existential quantifiers in *quantified Boolean formulas* [12], as shown below.

Singleton forgetting is forgetting with $W = \{x\}$. A general and simple way to realize singleton forgetting is by constructing a PDAG of the form

$$\varphi^{-x} = \varphi|x \vee \varphi|\neg x.$$

Note that φ^{-x} is logically equivalent to the quantified Boolean formula $(\exists x)\varphi$. It is easy to see that singleton forgetting preserves the properties of simple-negation and decomposability (if present), while determinism is not preserved (the two children of the new $\triangledown$-node are not necessarily logically contradictory). This is the reason why singleton forgetting is only supported by `PDAG` and `DNNF`, but not by `cd-PDAG` or `d-DNNF` (see Table 1).

Forgetting multiple variable is usually realized as a sequence of singleton forgetting. In general, this may result in an exponential blow-up of the PDAG size, but the decomposability of `DNNF` allows to keep this blow-up under control. This is the reason why `DNNF` is the only language to support forgetting in general. For the details of a corresponding algorithm, we refer to [6].

Now let's turn our attention to the special case of forgetting deterministic variables. One way to look at it is to define two additional transformations called *deterministic forgetting* ($\mathtt{FO_d}$) and *deterministic singleton forgetting* ($\mathtt{SFO_d}$). They correspond to `FO` and `SFO`, respectively, but the involved variables have to be deterministic.

For $x \,||\, \varphi$, the two children of the new $\triangledown$-node of $\varphi|x \vee \varphi|\neg x$ are logically contradictory by definition. In other words, forgetting deterministic variables preserves determinism. This enables us to adopt Darwiche's `DNNF` forgetting algorithm from [6] one-to-one to the case of deterministic forgetting in the language `d-DNNF`. As a consequence, $\mathtt{SFO_d}$ and $\mathtt{FO_d}$ are both supported by `d-DNNF`, as stated in the following theorem.

Theorem 3.

a) `PDAG` *supports* $\mathtt{SFO_d}$*, but it does not support* $\mathtt{FO_d}$ *unless* $P = NP$.

b) `DNNF` *and* `d-DNNF` *support* $\mathtt{FO_d}$ *and* $\mathtt{SFO_d}$.

c) `cd-PDAG` *and* `OBDD` *support* $\mathtt{SFO_d}$.

The proof is included in the appendix. Whether `cd-PDAG` and `OBDD` support $\mathtt{FO_d}$ is an open question.

3 Compiling Bayesian Networks

The goal of this section is to show that the probability distribution induced by a Bayesian network can be represented by a CNF (1st subsection) and that the d-DNNF compilation of this CNF can be used to efficiently compute arbitrary posterior probabilities (2nd subsection). The proposed CNF representation is similar but not equivalent to the one proposed by Darwiche in [8] (see Section 4).

A *Bayesian network* (BN) is a compact graphical model of a complex probability distribution over a set of variables $N = \{X_1, \ldots, X_n\}$ [17]. It consists of two parts: a DAG representing the direct influences among the variables, and a set of conditional probability tables (CPT) quantifying the strengths of these influences. The whole BN represents the exponentially sized *joint probability distribution* over its variables in a compact manner by

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | parents(X_i)),$$

where $parents(X_i)$ denotes the parents of node X_i in the DAG. Fig. 2 depicts a small BN with three Boolean variables X, Y, and Z. In this paper, we will restrict our discussion to Boolean variables. A Boolean variable X allows us to write x for the event $X{=}true$ and $\bar{x}$ or $\neg x$ for the event $X{=}false$.

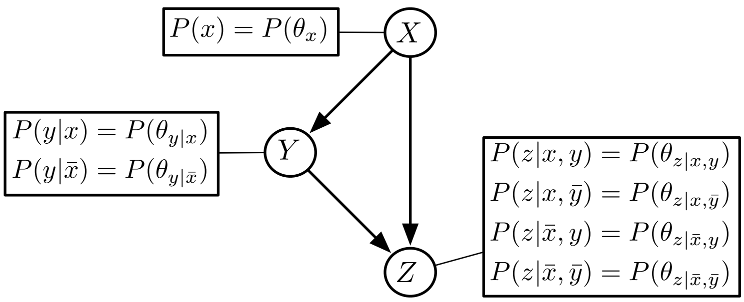


Figure 2: Example of a Bayesian network.

3.1 Logical Representation

Consider a variable $X \in N$ with $parents(X) = \{Y_1, \ldots, Y_n\}$ and the corresponding CPT. Since X has n parents, the CPT will have 2^n entries. For each CPT entry $P(x|\mathbf{y})$, a proposition $\theta_{x|\mathbf{y}}$ is introduced, where $\mathbf{y}$ is the corresponding instantiation of $parents(X)$. Assuming that the propositions $\theta_{x|\mathbf{y}}$ represent probabilistically independent events, we define their respective marginal probabilities by $P(\theta_{x|\mathbf{y}}) = P(x|\mathbf{y})$.

To see how this logical representation of the BN works, take a closer look at one particular instantiation $\mathbf{y}$ of $parents(X)$. The idea is that if $\mathbf{y}$ happens to be the true state of $parents(X)$, then $\theta_{x|\mathbf{y}}$ (resp. $\neg\theta_{x|\mathbf{y}}$) logically implies x (resp. $\neg x$). For $\mathbf{y} = (y_1, \ldots, y_n)$, this logical relationship between the propositions x, y_1 to y_n, and $\theta_{x|y_1,\ldots,y_n}$ is expressed by the first two implications in the following logical expression. By taking the conjunction of all such implications over all instantiations $\mathbf{y}$, we obtain a logical representation ψ_X of the node X with its relationship to its parents:

$$\psi_X = \bigwedge \left\{ \begin{array}{rcrcl} y_1 \wedge \cdots \wedge & y_n \wedge & \theta_{x|y_1,\dots,y_n} & \rightarrow & x \\ y_1 \wedge \cdots \wedge & y_n \wedge & \neg\theta_{x|y_1,\dots,y_n} & \rightarrow & \neg x \\ \vdots & \vdots & & & \vdots \\ \neg y_1 \wedge \cdots \wedge & \neg y_n \wedge & \theta_{x|\bar{y}_1,\dots,\bar{y}_n} & \rightarrow & x \\ \neg y_1 \wedge \cdots \wedge & \neg y_n \wedge & \neg\theta_{x|\bar{y}_1,\dots,\bar{y}_n} & \rightarrow & \neg x \end{array} \right\}.$$

A logical representation ψ_N of the whole BN is the conjunction

$$\psi_N = \bigwedge_{X \in N} \psi_X$$

over all network variables $X \in N$. This sentence includes two types of propositions, the ones linked to the CPT entries and the ones linked to the network variables. The respective sets of propositions are denoted by Θ and Δ, respectively.[2] Note that ψ_X and therewith ψ_N is a CNF, as each of its implications can be written as a clause. For the BN of Fig. 2, we get

$$\psi_N = \bigwedge \left(\left\{ \begin{array}{rcl} \theta_x & \rightarrow & x \\ \neg\theta_x & \rightarrow & \neg x \end{array} \right\} \cup \left\{ \begin{array}{rrcl} x \wedge & \theta_{y|x} & \rightarrow & y \\ \neg x \wedge & \theta_{y|\bar{x}} & \rightarrow & y \\ x \wedge & \neg\theta_{y|x} & \rightarrow & \neg y \\ \neg x \wedge & \neg\theta_{y|\bar{x}} & \rightarrow & \neg y \end{array} \right\} \cup \left\{ \begin{array}{rrrcl} x \wedge & y \wedge & \theta_{z|x,y} & \rightarrow & z \\ x \wedge & \neg y \wedge & \theta_{z|x,\bar{y}} & \rightarrow & z \\ \neg x \wedge & y \wedge & \theta_{z|\bar{x},y} & \rightarrow & z \\ \neg x \wedge & \neg y \wedge & \theta_{z|\bar{x},\bar{y}} & \rightarrow & z \\ x \wedge & y \wedge & \neg\theta_{z|x,y} & \rightarrow & \neg z \\ x \wedge & \neg y \wedge & \neg\theta_{z|x,\bar{y}} & \rightarrow & \neg z \\ \neg x \wedge & y \wedge & \neg\theta_{z|\bar{x},y} & \rightarrow & \neg z \\ \neg x \wedge & \neg y \wedge & \neg\theta_{z|\bar{x},\bar{y}} & \rightarrow & \neg z \end{array} \right\} \right).$$

The first block corresponds to ψ_X, the second block to ψ_Y, and the third block to ψ_Z. The two sets of propositional variables are $\Delta = \{x, y, z\}$ and $\Theta = \{\theta_x, \theta_{y|x}, \theta_{y|\bar{x}}, \theta_{z|x,y}, \theta_{z|x,\bar{y}}, \theta_{z|\bar{x},y}, \theta_{z|\bar{x},\bar{y}}\}$.

3.2 Computing Posterior Probabilities

The goal of a BN is the computation of the posterior probability $P(\mathbf{q}|\mathbf{e}) = P(\mathbf{q} \wedge \mathbf{e})/P(\mathbf{e})$ of a query event $\mathbf{q} = q_1 \wedge \cdots \wedge q_r$ given the observed evidence $\mathbf{e} = e_1 \wedge \cdots \wedge e_s$. As mentioned in Section 1, it is sufficient to look at the simpler problem of computing prior probabilities $P(\mathbf{x})$ of arbitrary conjunctions of literals $\mathbf{x}$. The following theorem states that the essential step to solve this problem is to forget the propositions Δ from ψ_N (or any equivalent form of it) conditioned on $\mathbf{x}$.

Theorem 4. $P(\mathbf{x}) = P\left([\psi_N|\mathbf{x}]^{-\Delta}\right)$.

[2] The representation of a Bayesian network by a logical sentence ψ_N over two sets of variables Θ and Δ, together with the given marginal probabilities for the variables in Θ and the corresponding independence assumptions, puts this approach in the broader context of *probabilistic argumentation* [14, 15]. This is a theory of formal reasoning which aims at unifying the classical fields of logical and probabilistic reasoning. The principal idea is to evaluate the credibility of a hypothesis by non-additive *probabilities of provability* (or *degrees of support*). This is a natural extension of the classical concepts of probability (in probability theory) and provability (in logic) [14]. The non-additivity of this measure is an important characteristic to distinguish properly between uncertainty and ignorance, but the particularity of the model in this paper always causes the resulting probabilities of provability to degenerate into ordinary (additive) probabilities. The embedding into the theory of probabilistic argumentation has no practical significance for the method and goals of this paper, but it allows inference in Bayesian network to be seen from a totally new perspective. We expect this perspective to be useful as a starting point to study inference in Bayesian networks with missing parameters.

This guarantees that the computed values are correct. To ensure that this computation requires only polynomial time, we need to compile ψ_N into an appropriate language, one that simultaneously supports `TC`, `FO`, and `PR`. The following theorem allows us to replace `FO`, not supported by `d-DNNF`, by $\texttt{FO}_\texttt{d}$, supported by `d-DNNF`.

Theorem 5. $\Delta \,||\, \psi_N$.

As a consequence of this simple theorem, we arrive at the main message of this paper, namely that `d-DNNF` is the most suitable target compilation language for Bayesian networks, since it supports `TC`, $\texttt{FO}_\texttt{d}$, and `PR`, and thus allows to compute posterior probabilities in polynomial time. For the compilation of the CNF ψ_N into a d-DNNF, we refer to the state-of-the-art CNF to d-DNNF compilers [7, 10]. Another option is to use any CNF to OBDD compiler, and to regard the result as a d-DNNF.

4 Comparison with Darwiche's Approach

Darwiche proposed a similar logical compilation for Bayesian networks [8, 3]. His approach focusses on the encoding the *multi-linear function* (or *network polynomial*) of a Bayesian network, rather than the Bayesian network itself. The resulting CNF is then compiled into a smooth d-DNNF, which defines a corresponding arithmetic circuit. The inference is then performed in time linear in the size of this circuit. Before the two methods are compared, Darwiche's encoding will be recapitulated shortly. We will use `DA` to refer to Darwiche's approach and `WH` to refer to the approach introduced in this paper.

For each network variable $X \in N$ with values $\{x_1, \ldots, x_k\}$ and $\mathit{parents}(X) = \{Y_1, \ldots, Y_n\}$, `DA` defines an *indicator variable* λ_{x_i} for each value x_i, $1 \leq i \leq k$. In addition, a *parameter variable* $\theta_{x_i|\mathbf{y}}$ is generated for each CPT entry $P(x_i|\mathbf{y})$ of X. Let $\mathbf{Y}$ denote the set of all instantiations $\mathbf{y} = (y_1, \ldots, y_n)$ of $\mathit{parents}(X)$. The CNF representation for X and its CPT consists of three distinct sets of clauses [3]:

$$\text{Indicator clauses: } \{\lambda_{x_1} \vee \cdots \vee \lambda_{x_k}\} \cup \{\neg\lambda_{x_i} \vee \neg\lambda_{x_j} : 1 \leq i < j \leq k\},$$

$$\text{IP clauses: } \{\lambda_{y_1} \wedge \cdots \wedge \lambda_{y_n} \wedge \lambda_{x_i} \rightarrow \theta_{x_i|\mathbf{y}} : \mathbf{y} \in \mathbf{Y},\, 1 \leq i \leq k\},$$

$$\text{PI clauses: } \begin{array}{l} \{\theta_{x_i|\mathbf{y}} \rightarrow \lambda_{x_i} : \mathbf{y} \in \mathbf{Y},\, 1 \leq i \leq k\} \cup \\ \{\theta_{x_i|\mathbf{y}} \rightarrow \lambda_{y_j} : \mathbf{y} \in \mathbf{Y},\, 1 \leq i \leq k,\, 1 \leq j \leq n\}. \end{array}$$

The CNF representation ψ_N of the entire Bayesian network is the conjunction of all indicator, IP, and PI clauses. This CNF is then compiled into a smooth d-DNNF, which finally leads to an arithmetic circuit to perform the inference. For more details on this we refer to [8, 3].

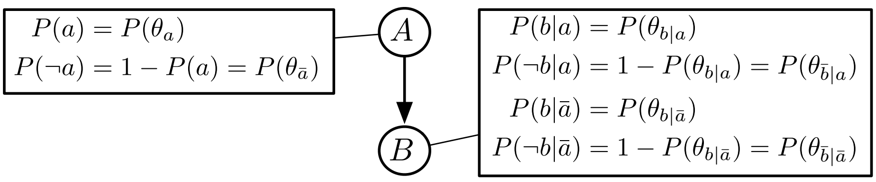


Figure 3: A very small Bayesian network.

To reveal the difference between the two approaches, consider the BN of Fig. 3 and the resulting satisfying sets of the respective encodings. `DA` comes out with ten variables and the following four models:

λ_a	$\lambda_{\bar{a}}$	λ_b	$\lambda_{\bar{b}}$	θ_a	$\theta_{\bar{a}}$	$\theta_{b\|a}$	$\theta_{\bar{b}\|a}$	$\theta_{b\|\bar{a}}$	$\theta_{\bar{b}\|\bar{a}}$	MLF term
1	0	1	0	1	0	1	0	0	0	$\lambda_a \lambda_b \theta_a \theta_{b\|a}$
1	0	0	1	1	0	0	1	0	0	$\lambda_a \lambda_{\bar{b}} \theta_a \theta_{\bar{b}\|a}$
0	1	1	0	0	1	0	0	1	0	$\lambda_{\bar{a}} \lambda_b \theta_{\bar{a}} \theta_{b\|\bar{a}}$
0	1	0	1	0	1	0	0	0	1	$\lambda_{\bar{a}} \lambda_{\bar{b}} \theta_{\bar{a}} \theta_{\bar{b}\|\bar{a}}$

.

Note that each model in the satisfying set, by replacing each 1 with the corresponding variable and each 0 with 1, represents exactly one term of the multi-linear function $f = \lambda_a \lambda_b \theta_a \theta_{b|a} + \lambda_a \lambda_{\bar{b}} \theta_a \theta_{\bar{b}|a} + \lambda_{\bar{a}} \lambda_b \theta_{\bar{a}} \theta_{b|\bar{a}} + \lambda_{\bar{a}} \lambda_{\bar{b}} \theta_{\bar{a}} \theta_{\bar{b}|\bar{a}}$. The connection between a Bayesian network and its MLF is extensively discussed in [9].

In contrast to `DA`, `WH` represents the BN of Fig. 3 by a CNF over five variables and with the following eight models:

a	b	θ_a	$\theta_{b\|a}$	$\theta_{b\|\bar{a}}$		λ_a	$\lambda_{\bar{a}}$	λ_b	$\lambda_{\bar{b}}$	θ_a	$\theta_{\bar{a}}$	$\theta_{b\|a}$	$\theta_{\bar{b}\|a}$	$\theta_{b\|\bar{a}}$	$\theta_{\bar{b}\|\bar{a}}$
1	1	1	1	0		1	0	1	0	1	0	1	0	0	1
1	1	1	1	1		1	0	1	0	1	0	1	0	1	0
1	0	1	0	0		1	0	0	1	1	0	0	1	0	1
1	0	1	0	1	$\Leftrightarrow$	1	0	0	1	1	0	0	1	1	0
0	1	0	0	1		0	1	1	0	0	1	0	1	1	0
0	1	0	1	1		0	1	1	0	0	1	1	0	1	0
0	0	0	0	0		0	1	0	1	0	1	0	1	0	1
0	0	0	1	0		0	1	0	1	0	1	1	0	0	1

.

The table on the right hand side is obtained from the one on the left hand side by substituting $\lambda_a \Leftrightarrow a$, $\lambda_{\bar{a}} \Leftrightarrow \neg a$, $\lambda_b \Leftrightarrow b$, $\lambda_{\bar{b}} \Leftrightarrow \neg b$, $\theta_{\bar{a}} \Leftrightarrow \neg\theta_a$, $\theta_{\bar{b}|a} \Leftrightarrow \neg\theta_{b|a}$, and $\theta_{\bar{b}|\bar{a}} \Leftrightarrow \neg\theta_{b|\bar{a}}$. This shows that that the encodings in `DA` and `WH` are in fact different. The key difference comes from the PI clauses in `DA`, which are not included in `WH`. On the other hand, there is an analogy between the clauses in `WH` and the IP clause in `DA`:

$$\begin{array}{rlcrl} & y_1 \wedge \cdots \wedge y_n \wedge \theta_{x|\mathbf{y}} \rightarrow x & \qquad & & y_1 \wedge \cdots \wedge y_n \wedge \neg\theta_{x|\mathbf{y}} \rightarrow \neg x \\ \equiv & y_1 \wedge \cdots \wedge y_n \wedge \neg x \rightarrow \neg\theta_{x|\mathbf{y}} & & \equiv & y_1 \wedge \cdots \wedge y_n \wedge x \rightarrow \theta_{x|\mathbf{y}} \\ \Leftrightarrow & \lambda_{y_1} \wedge \cdots \wedge \lambda_{y_n} \wedge \lambda_{\bar{x}} \rightarrow \theta_{\bar{x}|\mathbf{y}}\ , & & \Leftrightarrow & \lambda_{y_1} \wedge \cdots \wedge \lambda_{y_n} \wedge \lambda_x \rightarrow \theta_{x|\mathbf{y}}\ . \end{array}$$

Another important difference is the fact that `WH` does not impose the d-DNNF to be smooth, and `WH` gets along with less variables. All this is reflected in the sizes of the CNFs and, more importantly, in the sizes of the resulting d-DNNFs.[3] Table 2 lists the number of variables and clauses of the CNFs and the number of nodes and edges of the resulting d-DNNFs for the BNs of Fig. 2 and Fig. 3.

[3] One could argue that these differences result from the generality of Darwiche's multivariate approach, and claim that they are likely to vanish in the Boolean case by substituting all appearances of λ_{x_1} and λ_{x_2} by x and $\neg x$, respectively, and all appearances of $\theta_{x_1|\mathbf{y}}$ and $\theta_{x_2|\mathbf{y}}$ by $\theta_{x|\mathbf{y}}$ and $\neg\theta_{x|\mathbf{y}}$, respectively, but this makes Darwiche's encoding logically contradictory.

	Example of Fig. 2				Example of Fig. 3			
	#variables	#clauses	#nodes	#edges	#variables	#clauses	#nodes	#edges
`WH`	10	14	47	54	5	6	21	22
`DA`	20	54	67	102	10	20	31	38

Table 2: Number of variables, clauses, nodes, and edges for `WH` and `DA`.

5 Conclusion

The approach proposed in this paper defines a new logical inference method for Bayesian networks with Boolean variables. We expect its contribution to be theoretically and practically significant. On the theoretical side, based on an extended knowledge compilation map, the paper provides a precise explanation of why d-DNNFs are apparently the most suitable logical representations for Bayesian networks. This is mainly a consequence of the fact that some of the involved variables are deterministic. The paper also demonstrates how to reduce the problem of logical inference in Bayesian networks to three basic logical operations. Compared to Darwiche's differential approach, this view fits much better into the picture of the knowledge compilation perspective, as the reduction to these essential elements no longer requires us to talk about network polynomials, multi-linear functions, partial derivatives, arithmetic circuits, or smoothness. In this sense, we also see our paper as an attempt to clarify the theoretical mechanisms and connections behind this kind of inference algorithms and as a good example to demonstrate the usefulness of the knowledge compilation map.

On the practical side, the paper provides precise step-by-step instructions to implement a new encoding and inference method for Bayesian networks in terms of a few simple operations for d-DNNFs. Compared to Darwiche's differential approach, this will lead to more transparent implementations. The efficiency of such an implementation has not yet been empirically verified on a broader scale, but significant improvements are already observable in very small examples. Finally, with respect to possible applications other than Bayesian networks, other situations with deterministic variables may be detected, for which forgetting becomes tractable in the case of d-DNNFs.

Future work will focus on implementing this new approach, testing its efficiency, and extending it to Bayesian networks with multinomial variables and/or missing parameters.

Acknowledgment

Research supported by the *Swiss National Science Foundation*, Project No. PP002-102652/1, and *The Leverhulme Trust*.

Appendix: Proofs

Proof of Theorem 1. $\psi \models \varphi$ implies $\psi|x \models \varphi|x$ and $\psi|\neg x \models \varphi|\neg x$, and $x \,||\, \varphi$ is equivalent to $\varphi|x \wedge \varphi|\neg x \equiv \bot$. This implies $\psi|x \wedge \psi|\neg x \equiv \bot$, from which $x \,||\, \psi$ follows. ∎

Proof of Theorem 2. Holds since $(x \leftrightarrow \varphi)|x \wedge (x \leftrightarrow \varphi)|\neg x \equiv \varphi \wedge \neg\varphi \equiv \bot$. ∎

Proof of Corollary 1. Follows from

$$((x \leftrightarrow \varphi) \wedge \psi)|x \equiv \varphi \wedge \psi|x \text{ and } ((x \leftrightarrow \varphi) \wedge \psi)|\neg x \equiv \neg\varphi \wedge \psi|\neg x$$
$$\Rightarrow ((x \leftrightarrow \varphi) \wedge \psi)|x \wedge ((x \leftrightarrow \varphi) \wedge \psi)|\neg x \equiv \bot. \qquad \blacksquare$$

Lemma 1. For $\varphi \vee \psi \in$ d-DNNF with $x \,||\, \varphi \vee \psi$, we have

(a) $x \,||\, \varphi$,

(b) $x \,||\, \psi$,

(c) $\varphi^{-\{x\}} \wedge \psi^{-\{x\}} \equiv \bot$, i.e. $\varphi^{-\{x\}} \vee \psi^{-\{x\}} \in$ d-DNNF.

Proof. The proof of (a) and (b)goes as follows:

$$\begin{aligned} x \,||\, \varphi \vee \psi &\Leftrightarrow (\varphi \vee \psi)|x \wedge (\varphi \vee \psi)|\neg x \equiv \bot \\ &\Leftrightarrow (\varphi|x \vee \psi|x) \wedge (\varphi|\neg x \vee \psi|\neg x) \equiv \bot \\ &\Leftrightarrow (\varphi|x \wedge \varphi|\neg x) \vee (\varphi|x \wedge \psi|\neg x) \vee (\psi|x \wedge \varphi|\neg x) \vee (\psi|x \wedge \psi|\neg x) \equiv \bot \\ &\Rightarrow \begin{cases} \varphi|x \wedge \varphi|\neg x \equiv \bot,\ \psi|x \wedge \psi|\neg x \equiv \bot \Rightarrow x \,||\, \varphi,\ x \,||\, \psi & \text{(a), (b)} \checkmark \\ \varphi|x \wedge \psi|\neg x \equiv \bot,\ \psi|x \wedge \varphi|\neg x \equiv \bot & \text{(I)} \end{cases} \end{aligned}$$

Note that $\varphi \wedge \psi \equiv \bot$ implies $(\varphi \wedge \psi)|x \equiv \bot$ and $(\varphi \wedge \psi)|\neg x \equiv \bot$ (II). Finally, from (I) and (II) follows (c):

$$\begin{aligned} \varphi^{-\{x\}} \wedge \psi^{-\{x\}} \equiv \bot &\Leftrightarrow (\varphi|x \vee \varphi|\neg x) \wedge (\psi|x \vee \psi|\neg x) \equiv \bot \\ &\Leftrightarrow (\varphi|x \wedge \psi|x) \vee (\varphi|x \wedge \psi|\neg x) \vee (\varphi|\neg x \wedge \psi|x) \vee (\varphi|\neg x \wedge \psi|\neg x) \equiv \bot \\ &\Leftrightarrow (\varphi \wedge \psi)|x \vee (\varphi|x \wedge \psi|\neg x) \vee (\varphi|\neg x \wedge \psi|x) \vee (\varphi \wedge \psi)|\neg x \equiv \bot. \qquad \blacksquare \end{aligned}$$

Lemma 2. For $\varphi \wedge \psi \in$ d-DNNF with $x \,||\, \varphi \wedge \psi$, $x \in vars(\varphi)$, and $\psi \not\equiv \bot$, we have

(a) $vars\left(\varphi^{-\{x\}}\right) \cap vars(\psi) = \emptyset$,

(b) $x \,||\, \varphi$.

Proof. (a) follows from $vars\left(\varphi^{-\{x\}}\right) \subseteq vars(\varphi) \setminus \{x\}$. The proof of (b) goes as follows:

$$\begin{aligned} x \,||\, \varphi \wedge \psi &\Leftrightarrow (\varphi \wedge \psi)|x \wedge (\varphi \wedge \psi)|\neg x \equiv \bot \\ &\Leftrightarrow (\varphi|x \wedge \psi|x) \wedge (\varphi|\neg x \wedge \psi|\neg x) \equiv \bot \\ &\Leftrightarrow (\varphi|x \wedge \varphi|\neg x) \wedge (\psi|x \wedge \psi|\neg x) \equiv \bot \\ &\Leftrightarrow (\varphi|x \wedge \varphi|\neg x) \wedge \psi \equiv \bot \\ &\Rightarrow \varphi|x \wedge \varphi|\neg x \equiv \bot \Leftrightarrow x \,||\, \varphi. \qquad \blacksquare \end{aligned}$$

Proof of Theorem 3. We have:

a) PDAG supports $\texttt{SFO}_\texttt{d}$ since it supports SFO. If PDAG supports $\texttt{FO}_\texttt{d}$ it would also support FO. Since FO is not supported unless $P = NP$, $\texttt{FO}_\texttt{d}$ is not supported unless $P = NP$.

b) DNNF supports $\texttt{FO}_\texttt{d}$ and $\texttt{SFO}_\texttt{d}$ since it supports FO and SFO. According to Lemma 1 and Lemma 2, both determinism and decomposability are preserved by deterministic forgetting. Therefore, d-DNNF can use the algorithm of forgetting within DNNF as presented in [6].

c) OBDD supports $\texttt{SFO}_\texttt{d}$ since it supports SFO. cd-PDAG supports $\texttt{SFO}_\texttt{d}$ since forgetting a deterministic variable x of φ can be done by $\varphi|x \vee \varphi|\neg x$. ■

In the following $\sim$ denotes the compatibility relationship among variable instantiations. Hence, $\mathbf{x}{\sim}\mathbf{y}$ means that the instantiations $\mathbf{x}$ and $\mathbf{y}$ are compatible, i.e. they agree on every common variable. Furthermore, we use

$$lit(\theta_{x|\mathbf{y}}) = \begin{cases} \theta_{x|\mathbf{y}} & \text{if} \quad x{\sim}\mathbf{y}, \\ \neg\theta_{x|\mathbf{y}} & \text{if } \neg x{\sim}\mathbf{y}. \end{cases}$$

Lemma 3. For an instantiation $\mathbf{y}$ of all variables N of the BN, let $\mathbf{y}_x$ be the instantiation $\mathbf{y}$ restricted to the parents of X. This implies

$$\psi_N|\mathbf{y} \equiv \bigwedge_{x \in \Delta} lit(\theta_{x|\mathbf{y}_x}).$$

Proof. Performing the conjunction of two sequent line in the logical representation ψ_X of a varialbe X leads to

$$\psi_X \equiv \bigwedge \left\{ \begin{array}{ccc} y_1 \wedge \cdots \wedge & y_n \rightarrow (\theta_{x|y_1,\ldots,y_n} & \leftrightarrow x) \\ \vdots & \vdots & \vdots \\ \neg y_1 \wedge \cdots \wedge & \neg y_n \rightarrow (\theta_{x|\bar{y}_1,\ldots,\bar{y}_n} & \leftrightarrow x) \end{array} \right\},$$

i.e. $\psi_X|\mathbf{y} \equiv lit(\theta_{x|\mathbf{y}_x})$. Finally, $\psi_N = \bigwedge_{X \in N} \psi_X$ implies $\psi_N|\mathbf{y} \equiv \bigwedge_{x \in \Delta} lit(\theta_{x|\mathbf{y}_x})$. ■

Lemma 4. If $\mathbf{x}$ is an instantiation of some variables of the BN, then

$$[\psi_N|\mathbf{x}]^{-\Delta} \equiv \bigvee_{\mathbf{y}\sim\mathbf{x}} \psi_N|\mathbf{y},$$

where $\mathbf{y}$ is an instantiation of all variables of the BN, and $\mathbf{y}_x$ is the projection of $\mathbf{y}$ to $parents(X)$.

Proof. Let $\mathbf{Y}$ be the set of all instantiations $\mathbf{y}$. This implies

$$[\psi_N|\mathbf{x}]^{-\Delta} \equiv \bigvee_{\mathbf{y} \in \mathbf{Y}} [\psi_N|\mathbf{x}]\,|\mathbf{y} \equiv \bigvee_{\mathbf{y}\sim\mathbf{x}} \psi_N|\mathbf{y}.$$ ■

Proof of Theorem 4. Follows from Lemma 3 and Lemma 4. ■

Lemma 5. For $\varphi, \psi \in$ PDAG and $x \notin vars(\psi)$, we have $(\varphi \wedge \psi)^{-\{x\}} \equiv \varphi^{-\{x\}} \wedge \psi$.

Proof. Holds since $\psi \equiv \psi|x \equiv \psi|\neg x$. ■

Lemma 6. For all $x \in \Delta$, x is a deterministic variable of ψ_X.

Proof. By transforming ψ_X one can show that

$$\psi_X \equiv \left[\bigvee \left\{ \begin{array}{ccc} y_1 \wedge \cdots \wedge & y_n \wedge \theta_{x|y_1,\ldots,y_n} \\ \vdots & \vdots & \vdots \\ \neg y_1 \wedge \cdots \wedge & \neg y_n \wedge \theta_{x|\bar{y}_1,\ldots,\bar{y}_n} \end{array} \right\} \right] \leftrightarrow x.$$

Thus, x is a deterministic variable of ψ_X according to Theorem 1. ■

Lemma 7. For $\varphi \in \texttt{PDAG}$ with $x \,||\, \varphi$ and $y \,||\, \varphi^{-\{x\}}$, we get $x, y \,||\, \varphi$, $y \,||\, \varphi$, and $x \,||\, \varphi^{-\{y\}}$.

Proof. This lemma follows from

$$x \,||\, \varphi \Rightarrow \begin{cases} (\varphi| \;\; x, \;\; y) \wedge (\varphi|\neg x, \;\; y) \equiv \bot, \\ (\varphi| \;\; x, \neg y) \wedge (\varphi|\neg x, \neg y) \equiv \bot, \end{cases}$$

and

$$y \,||\, \varphi^{-\{x\}} \Rightarrow \begin{cases} (\varphi| \;\; x, y) \wedge (\varphi| \;\; x, \neg y) \equiv \bot, \\ (\varphi| \;\; x, y) \wedge (\varphi|\neg x, \neg y) \equiv \bot, \\ (\varphi|\neg x, y) \wedge (\varphi| \;\; x, \neg y) \equiv \bot, \\ (\varphi|\neg x, y) \wedge (\varphi|\neg x, \neg y) \equiv \bot. \end{cases}$$

■

Proof of Theorem 5. Starting with a leaf X of the BN, Lemma 6 and Theorem 1 ensure that x is a deterministic w.r.t. ψ_N. Since X has no children, all ψ_Y, $Y \neq X$, remain unchanged according to Lemma 5. This is repeated for the remaining variables of the BN, but X will no longer count as a child. According to Lemma 7, the order of the nodes is only important for the simplicity of the proof. ■

References

[1] G. Boole. *The Laws of Thought.* Walton and Maberley, London, 1854.

[2] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In E. Horvitz and F. Jensen, editors, *UAI'96, 12th Conference on Uncertainty in Artificial Inteligence*, pages 115–123, Portland, USA, 1996.

[3] M. Chavira and A. Darwiche. Compiling Bayesian networks with local structure. In *IJCAI'05, 19th International Joint Conference on Artificial Intelligence*, Edinburgh, U.K., 2005.

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

[5] P. Clote and E. Kranakis. *Boolean Functions and Computation Models.* Springer, 1998.

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

[7] A. Darwiche. A compiler for deterministic, decomposable negation normal form. In *AAAI'02, 18th National Conference on Artificial Intelligence*, pages 627–634, Edmonton, Canada, 2002.

[8] A. Darwiche. A logical approach to factoring belief networks. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M. A. Williams, editors, *KR'02, 8th International Conference on Principles and Knowledge Representation and Reasoning*, pages 409–420, Toulouse, France, 2002.

[9] A. Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

[10] A. Darwiche. New advances in compiling CNF to decomposable negational normal form. In *ECAI'04, 16th European Conference on Artificial Intelligence*, Valencia, Spain, 2004.

[11] A. Darwiche and P. Marquis. A knowlege compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[12] S. Davis and M. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7(3):201–215, 1960.

[13] H. Guo and W. H. Hsu. A survey of algorithms for real-time Bayesian network inference. In A. Darwiche and N. Friedman, editors, *AAAI/KDD/UAI'02, Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, Edmonton, Canada, 2002.

[14] R. Haenni. Towards a unifying theory of logical and probabilistic reasoning. In F. B. Cozman, R. Nau, and T. Seidenfeld, editors, *ISIPTA'05, 4th International Symposium on Imprecise Probabilities and Their Applications*, pages 193–202, Pittsburgh, USA, 2005.

[15] R. Haenni, J. Kohlas, and N. Lehmann. Probabilistic argumentation systems. In D. M. Gabbay and P. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5: Algorithms for Uncertainty and Defeasible Reasoning, pages 221–288. Kluwer Academic Publishers, Dordrecht, Netherlands, 2000.

[16] J. Kohlas. *Information Algebras: Generic Stuctures for Inference*. Springer, London, 2003.

[17] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Mateo, USA, 1988.

[18] M. Wachter and R. Haenni. Propositional DAGs: a new graph-based language for representing Boolean functions. In P. Doherty, J. Mylopoulos, and C. Welty, editors, *KR'06, 10th International Conference on Principles of Knowledge Representation and Reasoning*, pages 277–285, Lake District, U.K., 2006.