

Representing Boolean Functions with Propositional Directed Acyclic Graphs

Michael Wachter¹

Abstract. Propositional directed acyclic graphs (PDAG) continue the line of research on knowledge compilation in the context of Negational Normal Forms (NNF) and Binary Decision Diagrams (BDD). This paper summarizes previous results and open problems concerning knowledge representation based on PDAG and its sub-languages. In addition, it gives a short introduction to some application areas.

1 Introduction

Boolean Functions (BF) are fundamental knowledge representation tools, but to work with BFs presupposes efficient ways to represent them. There are many existing approaches to represent BFs such as truth tables, Karnaugh maps, canonical sum-of-products, *binary decision diagrams* (BDD) and their derivatives [1, 4, 5], *negation normal forms* (NNF) and their derivatives [6, 9]. Some of them are known to be impractical, as they impose representations of size $\mathcal{O}(2^n)$ for *most* possible n -ary functions [10] whereas BDDs and NNFs are more sophisticated and provide polynomial representations at least for *many* functions.

In Darwiche’s terminology, the sets of all possible NNFs or all possible BDDs are *languages*, i.e. BDD, the set of BDDs, is a sub-language of NNF, the set of NNFs. Other NNF sub-languages are obtained from a number of different properties which may or may not hold. Some of these NNF sub-languages allow certain classes of queries to be answered and certain transformations to be preformed in polynomial time.

The following section provides a basic introduction to PDAG, the language of PDAGs, reviews present results, and mentions so far unsolved problems. Thereafter we consider possible application areas of two sub-languages of PDAG.

2 Propositional DAGs

A first observation is that a certain property, identified as *simple-negation* in [14], implicitly holds for NNF and all its sub-languages. A leaf node of a NNF with a negative literal attached to it is like the *negation* of a leaf node with the corresponding positive literal attached to it. Thus, the idea is to adjust NNF by extending it with negations, i.e. additional non-leaf nodes labeled with \neg (logical not), and by restricting the leaf nodes to propositional symbols only. Therefore, negations are not imposed to be “at the bottom” of a *propositional directed acyclic graph* (PDAG). A PDAG is a graphical representation of a Boolean function (resp. of a sentence in propositional logic).

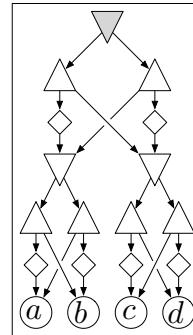


Figure 1. A (cd-)PDAG representing the odd parity function.

Formally, a PDAG is a rooted, directed acyclic graph in which each leaf node is represented by \circ and labeled with \top (true), \perp (false), or a propositional symbol, e.g. x . Each non-leaf node is represented by Δ (logical and), ∇ (logical or), or \diamond (logical not). The set of all possible PDAGs is called a *language* and denoted by PDAG. Figure 1 shows an example.

Leaves labeled with \top (\perp) represent the constant BF which always evaluates to 1 (0). A leaf labeled with the propositional symbol x 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 Δ -node is the one that evaluates to 1, iff the BFs of all its children evaluate to 1. Similarly, a ∇ -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 $\varphi \in \text{PDAG}$ is denoted by f_φ . Two PDAGs $\varphi, \psi \in \text{PDAG}$ are *equivalent*, iff $f_\varphi = f_\psi$. This is denoted by $\varphi \equiv \psi$. The number of edges of $\varphi \in \text{PDAG}$ is called its *size*.

According to the first results given in [14], the most important properties are *decomposability* and *determinism*. A decomposable and deterministic PDAG is called cd-PDAG. cd-PDAG is used to refer to the corresponding language, a sub-language of PDAG.

Other sub-languages are obtained from considering further properties: d-DNNF (*deterministic, decomposable NNF*), i.e. cd-PDAG satisfying *simple-negation*; OBDD (*ordered BDD*), i.e. d-DNNF satisfying *decision, read-once, and ordering*; and d-DNF (*disjoint DNF*), i.e. d-DNNF satisfying *flatness* and *simple-conjunction*. In the case of DNFs, determinism leads to disjoint terms², which is why deterministic DNFs are usually called disjoint DNFs. Note that c and d have

¹ University of Bern, Institute of Computer Science and Applied Mathematics, CH-3012 Bern, Switzerland, wachter@iam.unibe.ch

² Two terms t_1, t_2 are called *disjoint* iff $t_1 \wedge t_2 \equiv \perp$.

the same meaning, namely that decomposability is satisfied by each conjunction. Using c for decomposability instead of D makes it easier to distinguish it from d for determinism, since c is a property of conjunctions whereas d is a property of disjunctions. For a more comprehensive overview and a detailed discussion consider [9, 14]. These languages are analyzed according to their succinctness and the set of queries and transformations supported in polynomial time. The following subsections review some present results of [14].

2.1 Succinctness

A language L_1 is *more succinct* than another language L_2 , $L_1 \preceq L_2$, if any sentence $\alpha_2 \in L_2$ has an equivalent sentence $\alpha_1 \in L_1$ whose size is polynomial in the size of α_2 . A language L_1 is *strictly more succinct* than another language L_2 , $L_1 \prec L_2$, iff $L_1 \preceq L_2$ and $L_2 \not\preceq L_1$. Two languages L_1 and L_2 are *equally succinct*, iff $L_1 \preceq L_2$ and $L_2 \preceq L_1$. The above-mentioned languages have the following succinctness relationships:

$$\text{PDAG} \prec \text{cd-PDAG} \preceq \text{d-DNNF} \prec \begin{cases} \text{OBDD} \\ \text{d-DNNF} \end{cases}$$

It is still unknown whether cd-PDAG is strictly more succinct than d-DNNF or not. Since d-DNNF is a sub-language of cd-PDAG, d-DNNF will never be strictly more succinct than cd-PDAG. On the other hand, there are BFs for which the minimal size of a corresponding d-DNNF is at least a multiple (> 1) of the minimal size of a corresponding cd-PDAG. For example, the cd-PDAG $\neg(x_1 \wedge \dots \wedge x_n)$ has $n+1$ edges. Whereas, an equivalent minimal d-DNNF is $\neg x_1 \vee (x_1 \wedge (\neg x_2 \vee (x_2 \wedge \dots)))$, which has at least $4(n-1)$ edges.

2.2 Queries

A *query* is an operation that returns information about a PDAG representing a BF without changing it. The most important queries for BFs are: *consistency* (CO) or *satisfiability* (SAT), *validity* (VA), *clause entailment* (CE), *term implication* (IM), *sentential entailment* (SE), *equivalence* (EQ), *model counting* (CT), *model enumeration* (ME), *counter-model enumeration* (ME^c), *probabilistic equivalence* (PEQ), and *probability computation* (PR).

If a language supports a query in polynomial time with respect to the size of the PDAG(s) (in the case of model or counter-model enumeration, the reference size is both the size of the PDAG and size of the satisfying set or its complement), we simply say that it *supports* this query. Table 1 shows the supported queries of the most important languages, according to [9, 14].

	CO/CE/ME	VA/IM/ME ^c	CT/PR/PEQ	EQ	SE
PDAG	o	o	o	o	o
cd-PDAG	√	√	√	?	o
d-DNNF	√	√	√	?	o
OBDD	√	√	√	√	o
d-DNF	√	√	√	?	?

Table 1. Sub-languages of the PDAG language and their supported queries. The symbol \checkmark means “supports”, \circ means “does not support unless $P = NP$ ”, and $?$ means “unknown”.

It is easy to show that cd-PDAG supports CT. The method $val(\varphi)$ for counting models of a d-DNNF [7] can be adapted for \circ -, Δ -, and

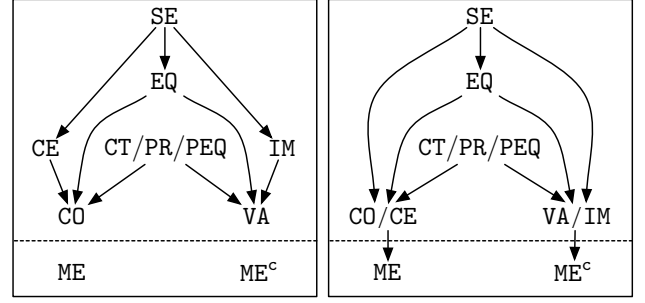


Figure 2. The correlations between supported queries: general case on the left, for languages supporting term conditioning on the right. $Q1/Q2$ means that $Q1$ is supported when $Q2$ is supported and vice versa. $Q1 \rightarrow Q2$ means that $Q2$ is supported when $Q1$ is supported, but it is also possible that $Q2$ is supported even if $Q1$ is not supported.

∇ -nodes. For a \diamond -node φ with child ψ , simply use

$$val(\varphi) = 2^{|\text{vars}(\varphi)|} - val(\psi),$$

which is polynomial in the size of φ since $|\text{vars}(\varphi)| \leq |\varphi|$. To show that cd-PDAG supports not only CT but also CO, CE, ME, VA, IM, ME^c, PR, and PEQ, we make use of two facts. Firstly, that the correlations between supported queries and the fact that the transformation *term conditioning* is supported by all sub-languages of PDAG. The correlations are shown in Figure 2. Consider [14] for the proofs of the correlations and of the fact that all sub-languages of PDAG support term conditioning.

2.3 Transformations

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. Let’s consider the following transformations: *term conditioning* (TC), *forgetting* (FO), *singleton forgetting* (SFO), *conjunction* (AND), *binary conjunction* (AND₂), *disjunction* (OR), *binary disjunction* (OR₂), and *negation* (NOT).

If a language supports a transformation in polynomial time with respect to the size of the PDAG(s), we simply say that it *supports* this transformation. Table 2 shows the supported transformations of the most important languages, according to [9, 14].

	TC	FO	SFO	AND	AND ₂	OR	OR ₂	NOT
PDAG	√	o	√	√	√	√	√	√
cd-PDAG	√	o	o	o	o	o	o	√
d-DNNF	√	o	o	o	o	o	o	?
OBDD	√	•	√	•	o	•	o	√
d-DNF	√	?	?	?	?	?	?	?

Table 2. Sub-languages of the PDAG language and their supported transformations. \checkmark means “supports”, \bullet means “does not support”, \circ means “does not support unless $P = NP$ ”, and $?$ means “unknown”.

cd-PDAG supports NOT by definition. The proofs of the remaining transformations for cd-PDAG are similar to the proofs for d-DNNF. Term conditioning φ on $\psi = \bigwedge_i \sigma_i$ is denoted by $\varphi|\psi$, where σ_i is a literal of the propositional symbol x_i . Note that term conditioning can be extended to a more general form of conditioning with respect to sub-PDAGs of φ instead of literals. This more general form will be called *conditioning* (CD).

Two new transformations are introduced in [12], namely *deterministic forgetting* (FO_d) and *deterministic singleton forgetting* (SFO_d). They are similar to FO and SFO but the involved variables have to be *deterministic*. For $\varphi \in \text{PDAG}$, the variable x is called *deterministic* w.r.t. φ , denoted by $x \parallel \varphi$, iff $\varphi|x \wedge \varphi|\neg x \equiv \perp$. More generally, a set of variables $\{x_1, \dots, x_n\}$ is called *deterministic* w.r.t. φ , denoted by $\{x_1, \dots, x_n\} \parallel \varphi$ or simply $x_1, \dots, x_n \parallel \varphi$, iff $\varphi|\mathbf{x} \wedge \varphi|\mathbf{x}' \equiv \perp$ for all instantiations $\mathbf{x} \neq \mathbf{x}'$ of the variables x_1, \dots, x_n . Interestingly, d-DNNF supports FO_d (and therefore SFO_d), although it does not support FO .

2.4 Future Work

Many questions regarding PDAG and its sub-languages are already answered, but some questions remain open or have not been considered yet. These open questions include several succinctness relationships concerning cd-PDAG, d-DNNF, d-DNF, DNNF (*decomposable* NNF), PI (*prime implicates*), IP (*prime implicants*), and MODS (*models*). The open problems concerning supported queries and transformations are indicated by ? in Table 1 and Table 2. In addition, it is unknown whether FBDD (*free* BDD) supports EQ or not. It is an open question whether cd-PDAG supports FO_d .

Another important goal is to build compilers for PDAGs. Starting with a representation φ_1 of a BF in language L_1 , a *compiler* generates an equivalent representation φ_2 in another language L_2 . Compilers are used to adjust set of satisfied properties, such that all required queries and transformations are supported, of course this may change the size of the representation.

The compilers CNF2cd-PDAG and DNF2cd-PDAG mentioned in [14] do not exploit all possibilities of omitting simple-negation, because they were originally designed for d-DNNF. Furthermore, we are interested in a more general compiler from PDAG to cd-PDAG. This includes the study of techniques to make a single Δ - or ∇ -node decomposable or deterministic respectively. For example, a decomposable cd-PDAG representing the conjunction of $\varphi, \psi \in \text{cd-PDAG}$ with only one common sub-PDAG ω can be obtained by CD, as shown in Figure 3.

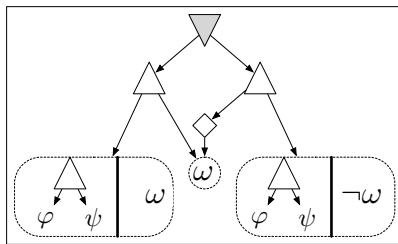


Figure 3. Obtaining decomposability with CD.

3 Application Areas

BFs are important in many areas of computer science and mathematics. In the following, we will consider three of them and show how PDAGs, more precisely cd-PDAGs, could be useful.

3.1 Reliability and Diagnostics of Systems

Reliability theory and *diagnostics* are two different but closely related fields. The first one concerns the overall *reliability* of the en-

tire system, which is usually reciprocally proportional to the system complexity. The second one is used if the system (or some of its components) is observed to be malfunctioning. The problem then is to find possible *diagnoses* explaining the cause of defect.

The classical areas of reliability are safety-critical technical and industrial systems such as airplanes or nuclear plants, but today similar techniques are also applied to investigate software systems, networks, or human-dependent administration or management systems. A related area of research is *risk analysis* [2]. Diagnosing malfunctioning systems is an important research topic in the area of Artificial Intelligence.

A system $\mathcal{S} = (C, f)$ consists of *components* $C = \{c_1, \dots, c_r\}$, $r \geq 1$, and a (*global*) *structure function* f . The structure function is a BF $f : \{0, 1\}^r \rightarrow \{0, 1\}$, which connects the components' states of operation with the state of operation of the entire system. To make a quantitative reliability or diagnostic analysis of a system $\mathcal{S} = (C, f)$, suppose that the components fail *independently* of each other, and let the probability that a component $c_i \in C$ is properly working be denoted by $p(c_i)$.

In [15], we have shown that representing the structure function by a cd-PDAG turns out to be an adequate computational technique for both reliability and diagnosis. Decomposability and determinism guarantee that the computation of probabilities is polynomial with respect to the size of the cd-PDAG.

For a given *modular* system description, which additionally consists of an *organizing tree* and corresponding *local* structure functions, it is possible to obtain a cd-PDAG for the system's global structure function by recursively replacing the nodes of the organizing tree by respective cd-PDAGs of the local structure functions. The probability of this global cd-PDAG corresponds then to the reliability of the system. This procedure is more difficult with OBDD, d-DNNF and d-DNF.

For finding the most likely diagnoses, we need to handle observations concerning the state (working or malfunctioning) of components, modules, or the system itself. The cd-PDAG from above corresponds to the situation without any observation, and the organizing tree and CD ensure that any set of observations can be handled in a tractable way.

Again, the advantage of cd-PDAG is its succinctness compared to common techniques such as OBDD or d-DNF.

3.2 Bayesian Networks

Bayesian networks (BN) are an important area for research and applications in Artificial Intelligence and beyond. A BN is a compact graphical model of a complex probability distribution over various variables [11]. 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. Figure 4 depicts a small BN with three Boolean variables X, Y , and Z . Within this paper they are also called *network variables*.

The goal of a BN is to compute posterior probabilities given some observations. In the following, we will shortly discuss two alternative computational techniques for BNs. While most conventional methods are purely numerical, these approaches will generate a logical representation of the BN. 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 [3, 8]. Another advantage is the ability to efficiently update numerical computations with minimal computational overhead. This is a key prerequisite for an experimental

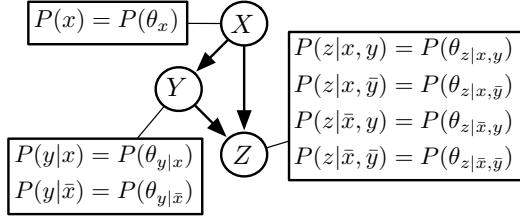


Figure 4. A small Bayesian network.

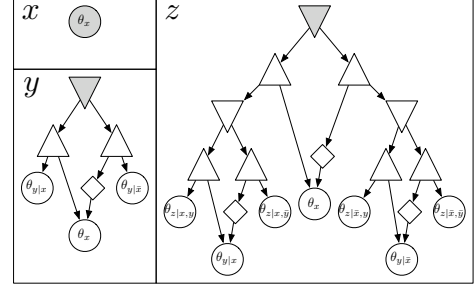


Figure 5. The three cd-PDAGs obtained from the small Bayesian network (Figure 4).

sensitivity analysis.

The discussion is restricted to Boolean variables, i.e. an additional proposition x is attributed to each network variable X and use it for the event $X=true$ and its negation (denoted $\neg x$ or \bar{x}) for $X=false$. Both methods attribute a proposition $\theta_{x|y}$ to each CPT entry $P(x|y)$ of a network variable X . The first approach is based on the previous subsection, using cd-PDAGs, whereas the second one is based on [12], using d-DNNFs.

3.2.1 First Method

Particular cases of BNs are modular systems mentioned in the previous section: the DAGs are the organizing trees, where the edges are directed from the bottom to the top, and the entries of the CPTs are the values of the local structure functions. Thus, the idea is to generalize the procedure of the previous section to handle general BNs. The following part describes some of the basic ideas.

Let N denote the set of network variables of a BN, and let $X \in N$ be a network variable with $parents(X) = \{Y_1, \dots, Y_s\}$ and $r = 2^s - 1$ CPT entries. Each CPT entry corresponds to one of the following conditional probability:

$$\begin{aligned} P(x|\bar{y}_1, \dots, \bar{y}_s) &= P(\theta_{x|y_0}) \\ P(x|\bar{y}_1, \dots, y_s) &= P(\theta_{x|y_1}) \\ &\dots \\ P(x|y_1, \dots, y_s) &= P(\theta_{x|y_r}) \end{aligned}$$

The idea is to obtain a cd-PDAG representing X . CPT2cd-PDAG constructs the corresponding cd-PDAG. The input parameters are the \circ -nodes θ_i labeled with $\theta_{x|y_i}$, $0 \leq i < r$ and the cd-PDAGs $\varphi_1, \dots, \varphi_s$ representing the parents of X . If the j -th bit of the binary coded representation of i is 1, then $\sigma_i(\varphi_j)$ denotes φ_j , otherwise it denotes the negation of φ_j . Figure 5 shows the result of applying CPT2cd-PDAG to each CPT of the BN given in Figure 4. These cd-PDAGs can be used to compute the prior probability of the corresponding node of the BN.

Algorithm 1: CPT2cd-PDAG($\{\theta_0, \dots, \theta_r\}, \{\varphi_1, \dots, \varphi_s\}$)

```

1  $\Psi \leftarrow \emptyset$ ;
2 for  $i \in \{0, \dots, r\}$  do
3    $\psi \leftarrow$  cd-PDAG equivalent to  $\Delta$ -node with children
    $\sigma_i(\varphi_1), \dots, \sigma_i(\varphi_s)$ ;
4    $\Psi \leftarrow \Psi \cup \{\Delta\text{-node with children } \psi \text{ and } \theta_i\}$ ;
5 end
6 return  $\nabla$ -node with children  $\Psi$ ;

```

The same cd-PDAGs will also allow us to handle observations and compute posterior probabilities thereof. For this, construct the cd-

PDAG representing the conjunction of the cd-PDAGs of the observations and the variable under examination. Unfortunately, this construction may not be performable in polynomial time, i.e. the method does not qualify for reasoning in polynomial time.

3.2.2 Second Method

The starting point of this method is a complete logical representation ψ of the entire Bayesian network. ψ consists of 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.

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 $\mathbf{q} = q_1 \wedge \dots \wedge q_r$ given the evidence $\mathbf{e} = e_1 \wedge \dots \wedge e_s$, it is sufficient to look at the simpler problem of computing prior probabilities $P(\mathbf{x})$ of arbitrary conjunctions $\mathbf{x} = x_1 \wedge \dots \wedge x_r$ in order to obtain corresponding numerators $P(\mathbf{q} \wedge \mathbf{e})$ and denominators $P(\mathbf{e})$. This is done in 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 $[\psi|\mathbf{x}]^{-\Delta}$ 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 $\theta_{x|y} \in \Theta$ are probabilistically independent

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. [12] shows that the propositions in Δ are deterministic w.r.t. ψ and that d-DNNF is the appropriate target compilation language, since FO can be replaced by FO_q which is supported by d-DNNF. For more details consider [12].

3.3 Formal Verification

A third application area is *formal verification* which is used to formally prove that two distinct representations of a BF are equivalent. This problem arises in various situations, such as verification of circuits, software (expressed as source code), and cryptographic protocols. As an example, consider the problem of checking if a manually modified circuit design functionally corresponds to the original one. Another typical example is the question of whether a gate-level implementation meets its functional specification.

The probabilistic equivalence test of cd-PDAG is an alternative to the missing exact equivalence test [13]. For an adequate choice of

parameters, the failure probabilities of this test converges quickly towards 0. It seems to be an interesting alternative to the techniques used in hardware design, which are mostly based on the language OBDD. According to [13], cd-PDAG is the most suitable language for probabilistic equivalence testing. The advantage of using cd-PDAG instead of OBDD is its succinctness.

4 Conclusion

A new graph-based language for representing Boolean functions, called PDAG, is obtained by removing the implicit simple-negation property of NNF. cd-PDAG is a promising sub-language of PDAG satisfying decomposability and determinism. The “good” succinctness, the “large” set of queries, supported in polynomial time, and the “simple” negation turn cd-PDAG into an interesting alternative to existing languages such as OBDD, d-DNF, or d-DNNF for representing Boolean functions. The considered application areas show the potential of cd-PDAG, but it is not limited to these areas. Furthermore, FQ_d seems to be an interesting transformation and should be very useful for knowledge compilation and inference.

Acknowledgment

Research supported by the Swiss National Science Foundation, Project No. PP002102652/1, and *The Leverhulme Trust*.

REFERENCES

- [1] S. B. Akers, ‘Binary decision diagrams’, *IEEE Transactions on Computers*, **27**(6), 509–516, (1978).
- [2] J. Andrews and B. Moss, *Reliability and Risk Assessment (2nd Edition)*, American Society of Mechanical Engineers, 2002.
- [3] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller, ‘Context-specific independence in Bayesian networks’, in *UAI’96, 12th Conference on Uncertainty in Artificial Intelligence*, eds., E. Horvitz and F. Jensen, pp. 115–123, Portland, USA, (1996).
- [4] R. E. Bryant, ‘Graph-based algorithms for Boolean function manipulation’, *IEEE Transactions on Computers*, **35**(8), 677–691, (1986).
- [5] R. E. Bryant, ‘Symbolic Boolean manipulation with ordered binary decision diagrams’, *ACM Computing Surveys*, **24**(3), 293–318, (1992).
- [6] A. Darwiche, ‘Decomposable negation normal form’, *Journal of ACM*, **48**(4), 608–647, (2001).
- [7] A. Darwiche, ‘On the tractability of counting theory models and its application to belief revision and truth maintenance’, *Journal of Applied Non-Classical Logics*, **11**(1-2), 11–34, (2001).
- [8] A. Darwiche, ‘A differential approach to inference in Bayesian networks’, *Journal of the ACM*, **50**(3), 280–305, (2003).
- [9] A. Darwiche and P. Marquis, ‘A knowledge compilation map’, *Journal of Artificial Intelligence Research*, **17**, 229–264, (2002).
- [10] F. J. Hill and G. R. Peterson, *Introduction to Switching Theory and Logical Design*, John Wiley and Sons, New York, USA, 1974.
- [11] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Mateo, USA, 1988.
- [12] M. Wachter and R. Haenni. Logical compilation of bayesian networks. (Draft), 2006.
- [13] M. Wachter and R. Haenni, ‘Probabilistic equivalence checking with propositional DAGs’, Technical Report iam-06-001, University of Bern, (2006).
- [14] M. Wachter and R. Haenni, ‘Propositional DAGs: a new graph-based language for representing Boolean functions’, in *KR’06, 10th International Conference on Principles of Knowledge Representation and Reasoning*, Lake District, U.K., (2006).
- [15] M. Wachter, R. Haenni, and J. Jonczy, ‘Reliability and diagnostics of modular systems: a new probabilistic approach’, in *DX’06, 17th International Workshop on Principles of Diagnosis*, Peñaranda de Duero, Spain, (2006).