# Backdoor Decomposable Monotone Circuits and Propagation Complete Encodings

**Petr Kučera,**[1] **Petr Savický**[2]

[1] Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, Czech Republic

[2] Institute of Computer Science of the Czech Academy of Sciences, Czech Republic

kucerap@ktiml.mff.cuni.cz, savicky@cs.cas.cz

## Abstract

We describe a compilation language of backdoor decomposable monotone circuits (BDMCs) which generalizes several concepts appearing in the literature, e.g. DNNFs and backdoor trees. A $\mathcal{C}$-BDMC sentence is a monotone circuit which satisfies decomposability property (such as in DNNF) in which the inputs (or leaves) are associated with CNF encodings from a given base class $\mathcal{C}$. We consider the class of propagation complete (PC) encodings as a base class and we show that PC-BDMCs are polynomially equivalent to PC encodings. Additionally, we use this to determine the properties of PC-BDMCs and PC encodings with respect to the knowledge compilation map including the list of efficient operations on the languages.

## Introduction

In knowledge compilation (Darwiche and Marquis 2002; Marquis 2015), we are concerned with transforming a given propositional theory into a form which allows efficient query answering and manipulation. The form of the output representation is specified by a target compilation language. Lots of target languages were described in knowledge compilation map (Darwiche and Marquis 2002) which was later extended with other languages and their disjunctive closures (Fargier and Marquis 2008). We are in particular interested in languages based on conjunctive normal forms (CNF) which are used to encode various constraints into SAT. Since unit propagation is a basic procedure used in DPLL based SAT solvers including CDCL solvers, it has become a common practice to require that unit propagation maintains at least some level of local consistency in the constraints being encoded into a CNF formula.

Close connection between unit propagation in SAT solvers and maintaining generalized arc consistency (GAC) was investigated for example by Bacchus (2007); Bessiere et al. (2009). A stronger notion of propagation complete (PC) encodings was introduced by Bordeaux and Marques-Silva (2012) as a generalization of unit refutation complete (URC) encodings (del Val 1994). A formula $\varphi$ is propagation complete, if its consistency with a partial assignment can be checked by unit propagation and in case the formula is consistent with a partial assignment, unit propagation derives all implied literals. When encoding a constraint into a CNF we usually distinguish two kinds of variables, the main variables which directly correspond to the variables of a constraint, and auxiliary variables. A well known example is encoding of a circuit into a CNF formula using Tseitin's encoding (Tseitin 1983) where the main variables correspond to the inputs of the circuit and auxiliary variables correspond to the gates. Let us note that PC encodings treat all variables in the same way: the propagation properties of a PC encoding with respect to the auxiliary variables is the same as with respect to the main variables. By way of contrast, if the encoding only maintains GAC (see e.g. Bacchus 2007) or, equivalently, domain consistency, the propagation properties are required only for the main variables (GAC encoding in this paper). A systematic study of encodings of multi-valued decision diagrams (MDDs) with different propagation strength is presented by (Abío et al. 2016) including a construction of a polynomial size PC encoding for an arbitrary MDD.

Let us include a few remarks concerning a weaker notion of consistency checker (CC encoding in this paper) considered, for example, by (Bessiere et al. 2009; Abío et al. 2016). In the case of a CC encoding, unit propagation detects inconsistency with a partial assignment of the main variables. Following the framework of closures initiated by Fargier and Marquis (2008), Bordeaux et al. (2012) studied disjunctive closures of different types of CNF encodings and their placement into knowledge compilation map. In particular, they consider CC encodings denoted as $\exists$ URC-C and one of their results is that the language of disjunctions of CC encodings is polynomially equivalent to the language of CC encodings. We generalize this by proving that the language of disjunctions of PC encodings is polynomially equivalent to the language of PC encodings. Using a slightly simpler construction, we also prove that the language of disjunctions of URC encodings is polynomially equivalent to the language of URC encodings.

The above results are a consequence of a more general construction. We introduce a new target compilation language parameterized with a class $\mathcal{C}$ of encodings. For the special case when $\mathcal{C}$ is the class of PC encodings, we demonstrate a transformation of a sentence in this language into a single PC encoding.

A sentence in the proposed language is a monotone cir-

cuit whose inputs called leaves are defined by CNF encodings from a suitable base class $\mathcal{C}$. Additionally, the conjunctions in the circuit satisfy a decomposability property with respect to the input variables similar to the language of decomposable negation normal forms (DNNF, introduced by Darwiche 1999). We call this structure backdoor decomposable monotone circuit with respect to the base class $\mathcal{C}$ ($\mathcal{C}$-BDMC), because it is closely related to $\mathcal{C}$-backdoor trees introduced by Samer and Szeider (2008). By definition, the language of URC-BDMCs is a strict superset of the language of disjunctions of URC encodings studied in Bordeaux et al. (2012) as $\texttt{URC-C}[\vee, \exists]$.

BDMC generalizes also other concepts appearing in the literature. A DNNF can be understood as a special case of $\mathcal{C}$-BDMC for any class $\mathcal{C}$ containing the literals. If we consider circuits with only one node, we obtain that PC-BDMC sentences generalize PC formulas and URC-BDMC sentences generalize URC formulas. In the rest of the paper, we mostly consider PC-BDMCs, but the results can be transferred to URC-BDMCs as well. Note that monotone CNFs are propagation complete and thus they are special cases of PC-BDMCs. Combining this with the results of Bova et al. (2014) or Bova et al. (2016) and the fact that DNNFs are also special cases of PC-BDMCs, we obtain that the language of PC-BDMCs is strictly more succinct than the language of DNNFs in the sense of the knowledge compilation map (see Darwiche and Marquis 2002).

Generalizing the idea of PC-backdoor trees introduced by Samer and Szeider (2008) towards a language for representing boolean functions, we obtain a special case of PC-BDMCs in which the circuit part has the form of an outarborescence consisting of decision nodes whose leaves are associated with PC encodings without auxiliary variables. The difference from the original backdoor trees is that the formulas in the leaves are not necessarily restrictions of the same formula. We show that PC-BDMCs are strictly more succinct than the PC-backdoor trees generalized in this way.

The main result of our paper is that a PC-BDMC can be compiled into a PC encoding of size polynomial with respect to the total size of the input BDMC. As a consequence, we get that both PC-BDMCs and PC encodings share the same algorithmic properties while being equally succinct. Moreover, we argue that the properties of PC-BDMCs and PC encodings with respect to query answering and transformations described in the knowledge compilation map (Darwiche and Marquis 2002) are the same as in the case of DNNFs. At the same time, both PC-BDMCs and PC encodings are strictly more succinct than DNNFs which makes these languages good target compilation languages.

A compilation of a smooth DNNF into a URC or PC encoding of polynomial size was described by Kučera and Savický (2019b) building on previous consistency checking and domain consistency maintaining (or GAC) encodings described by Abío et al. (2016); Gange and Stuckey (2012); Jung et al. (2008). We generalize these results to a more general structure, where the leaves contain PC encodings instead of single literals. The omitted proofs can be found in the full version of the paper (Kučera and Savický 2019a).

## Definitions and Notation

We assume the reader is familiar with the basics of propositional logic, especially with the notion of entailment $\models$ and the notation related to formulas in conjunctive normal form (*CNF formula*). We use $\mathrm{lit}(\mathbf{x})$ to denote the set of literals ($x$, $\neg x$) over the set of variables $\mathbf{x}$. We treat a clause as a set of literals and a CNF formula as a set of clauses. A *partial assignment* $\alpha$ of values to variables in $\mathbf{z}$ is a subset of $\mathrm{lit}(\mathbf{z})$ that does not contain a complementary pair of literals. A full assignment $\mathbf{a} : \mathbf{x} \to \{0, 1\}$ is a special type of partial assignment and we use the representations by a function or by a set of literals interchangeably. We consider encodings of boolean functions defined as follows.

**Definition 1** (Encoding)**.** Let $f(\mathbf{x})$ be a boolean function on variables $\mathbf{x} = (x_1, \ldots, x_n)$. Let $\varphi(\mathbf{x}, \mathbf{y})$ be a CNF formula on $n + m$ variables where $\mathbf{y} = (y_1, \ldots, y_m)$. We call $\varphi$ a *CNF encoding* of $f$ if

$$f(\mathbf{x}) \equiv (\exists \mathbf{y})\, \varphi(\mathbf{x}, \mathbf{y})\,. \tag{1}$$

The variables in $\mathbf{x}$ and $\mathbf{y}$ are called *main variables* and *auxiliary variables*, respectively.

We use $\varphi \vdash_1 C$ to denote the fact that a clause $C$ can be derived by unit propagation interpreted as unit resolution from a CNF formula $\varphi$ (in particular, if $C \in \varphi$, then $\varphi \vdash_1 C$). The notion of a propagation complete CNF formula was introduced by Bordeaux and Marques-Silva (2012) as a generalization of a unit refutation complete CNF formula introduced by del Val (1994).

**Definition 2** (Propagation complete encoding)**.** Let $\varphi(\mathbf{x}, \mathbf{y})$ be a CNF encoding of a boolean function defined on a set of variables $\mathbf{x}$. We say that the encoding $\varphi$ is *propagation complete (PC)*, if for every partial assignment $\alpha \subseteq \mathrm{lit}(\mathbf{x} \cup \mathbf{y})$ and for each $l \in \mathrm{lit}(\mathbf{x} \cup \mathbf{y})$, such that

$$\varphi(\mathbf{x}, \mathbf{y}) \wedge \alpha \models l \tag{2}$$

we have

$$\varphi(\mathbf{x}, \mathbf{y}) \wedge \alpha \vdash_1 l \quad \text{or} \quad \varphi(\mathbf{x}, \mathbf{y}) \wedge \alpha \vdash_1 \bot\,. \tag{3}$$

## Backdoor Decomposable Monotone Circuits and the Main Result

In this section we introduce a language of backdoor decomposable monotone circuits (BDMC) and state the main result of the paper. BDMCs form a common generalization of decomposable negation normal forms (DNNF) introduced by Darwiche (1999) and $\mathcal{C}$-backdoor trees introduced by Samer and Szeider (2008) if the base class $\mathcal{C}$ contains the literals as formulas. It consists of sentences formed by a combination of a decomposable monotone circuit with CNF encodings from a suitable class $\mathcal{C}$ at the leaves. More precisely, let $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$ for $i = 1 \ldots, \ell$ be encodings from $\mathcal{C}$ with auxiliary variables $\mathbf{y}_i$ whose main variables $\mathbf{x}_i$ are subsets of a set of variables $\mathbf{x}$ and let us consider their combination by a monotone circuit $D$ with $\ell$ inputs. This is a DAG with nodes $V$, root $\rho \in V$, the set of edges $E$, and the set of leaves $L \subseteq V$ of size $\ell$. The inner nodes in $V$ are labeled with $\wedge$ or $\vee$ and represent connectives or gates. Each

edge $(v, u)$ in $D$ connects an inner node $v$ labeled $\wedge$ or $\vee$ with one of its inputs $u$. The edge is directed from $v$ to $u$, so the inputs of a node are its successors (or child nodes). We assume that there is a one to one correspondence between the leaves of $D$ and the formulas $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \ldots, \ell$ and we say that a leaf is labeled or associated with the corresponding formula. For a given index $i \in \{1, \ldots, \ell\}$, the leaf associated with $\varphi_i$ is denoted as $\mathrm{leaf}(i)$. Given a literal $l \in \mathrm{lit}(\mathbf{x})$, let us denote $\mathrm{range}(l)$ the set of indices of formulas in the leaves which contain variable $\mathrm{var}(l)$, i.e. $\mathrm{range}(l) = \{i \in \{1, \ldots, \ell\} \mid \mathrm{var}(l) \in \mathbf{x}_i\}$. Given two different formulas $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$ and $\varphi_j(\mathbf{x}_j, \mathbf{y}_j)$, we assume that $\mathbf{y}_i \cap \mathbf{y}_j = \emptyset$, i.e. the sets of auxiliary variables of encodings in different leaves are pairwise disjoint.

For a node $v \in V$, let us denote $\mathrm{var}(v)$ the set of main variables from $\mathbf{x}$ that appear in the leaves which can be reached from $v$ by a path. In particular, $\mathrm{var}(v) = \mathbf{x}_i$ for a leaf $v$ associated with $\varphi(\mathbf{x}_i, \mathbf{y}_i)$. We assume that $\mathrm{var}(\rho) = \mathbf{x}$, i.e. each variable $x \in \mathbf{x}$ is in some leaf.

Given a node $v \in V$, let $f_v(\mathbf{x}_i)$ be the function defined on the variables $\mathrm{var}(v)$ as follows. If $v$ is a leaf node associated with $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$, then $f_v(\mathbf{x}_i)$ is the function with encoding $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$. If $v$ is a $\wedge$-node or a $\vee$-node, $f_v$ is the conjunction or the disjunction, respectively, of the functions $f_u$ represented by the inputs $u$ of $v$.

**Definition 3** (Backdoor Decomposable Monotone Circuit)**.** Let $\mathcal{C}$ be a base class of CNF encodings containing every literal as a formula. A sentence in the language of *backdoor decomposable monotone circuits with respect to base class* $\mathcal{C}$ (*$\mathcal{C}$-BDMC*) is a directed acyclic graph as described above, where each leaf node is labeled with a CNF encoding from $\mathcal{C}$ and each internal node is labeled with $\wedge$ or $\vee$ and can have arbitrarily many successors. Moreover, the nodes labeled with $\wedge$ satisfy the *decomposability* property, which means that for every $\wedge$-node $v = v_1 \wedge \cdots \wedge v_k$, the sets of variables $\mathrm{var}(v_1), \ldots, \mathrm{var}(v_k)$ are pairwise disjoint. The function represented by the sentence is the function $f_\rho$ defined on the variables $\mathbf{x} = \mathrm{var}(\rho)$.

We will omit prefix $\mathcal{C}$ and write simply BDMC in case the choice of a particular class of formulas $\mathcal{C}$ is not essential. The language of DNNFs is the class of those $\mathcal{C}$-BDMCs, whose leaves are the literals on the input variables. Since a decision node can be represented as a disjunction of two conjunctions in which one of the conjuncts is a literal, we can also conclude that $\mathcal{C}$-backdoor trees (Samer and Szeider 2008) form a subclass of $\mathcal{C}$-BDMCs.

For the construction of the encoding, we consider the case when $\mathcal{C}$ is equal to the class of PC encodings. This class admits a polynomial time satisfiability test. However, Babka et al. (2013) proved that the corresponding membership test whether a given formula is PC is co-NP-complete. For this reason, when the complexity of algorithms searching for a BDMC for a given function is in consideration, a different suitable class of encodings with a polynomial time membership test can be used, such as prime 2-CNF (which are PC) or (renamable) Horn formulas (which are URC).

The function represented by a BDMC is described above by a recursion. In order to relate this function to the encodings constructed later, we describe the function using the following notion. A *minimal satisfying subtree* $T$ of $D$ is a rooted subtree of $D$ (also called out-arborescence) which has the following properties:

- The root $\rho$ of $D$ is also the root of $T$.
- For each $\wedge$-node $v$ in $T$, all edges $(v, u)$ in $D$ are in $T$.
- For each $\vee$-node $v$ in $T$, exactly one of the edges $(v, u)$ in $D$ is also in $T$.

If $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$ and $\varphi_j(\mathbf{x}_j, \mathbf{y}_j)$ are formulas associated with two different leaves of $T$, then by decomposability of $D$ we have that $\mathbf{x}_i \cap \mathbf{x}_j = \emptyset$. We can observe that if $\mathcal{T}$ denotes the set of all minimal satisfying subtrees of $D$, then we have

$$f(\mathbf{x}) \equiv \bigvee_{T \in \mathcal{T}} \bigwedge_{\mathrm{leaf}(i) \in V(T)} (\exists \mathbf{y}_i) \varphi_i(\mathbf{x}_i, \mathbf{y}_i)\,. \qquad (4)$$

A smooth BDMC is defined similarly to a smooth DNNF.

**Definition 4** (Smooth BDMC)**.** We say that a $\mathcal{C}$-BDMC $D$ is *smooth* if for every $\vee$-node $v = v_1 \vee \cdots \vee v_k$ we have $\mathrm{var}(v) = \mathrm{var}(v_1) = \cdots = \mathrm{var}(v_k)$.

If $D$ is a smooth BDMC representing a function $f(\mathbf{x})$ and $T$ is a minimal satisfying subtree of $D$, then for every $x \in \mathbf{x}$ there is a leaf of $T$ which is associated with a formula $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$ such that $x \in \mathbf{x}_i$.

The definition of a smooth BDMC restricts only the occurrences of the main variables. The auxiliary variables of the encodings in the leaves are local to the leaves and we assume that the sets of auxiliary variables in the encodings associated with two differrent leaves are disjoint. Darwiche (2001) showed that a DNNF can be transformed into a smooth DNNF with a polynomial increase of size. The same approach can be used to make an arbitrary $\mathcal{C}$-BDMC smooth. This is one of the places, where we use the assumption that every formula consisting of a single literal belongs to $\mathcal{C}$.

Let us state the main result of this paper proven later using Theorem 6 below.

**Theorem 1.** *Let $D$ be a smooth PC-BDMC representing a function $f(\mathbf{x})$. Then we can construct in polynomial time a PC encoding of $f(\mathbf{x})$.*

Following Fargier and Marquis (2008), two languages $\mathbf{L}_1$ and $\mathbf{L}_2$ are called *polynomially equivalent* if any sentence in $\mathbf{L}_1$ can be translated in polynomial time into an equivalent sentence in $\mathbf{L}_2$ and vice versa. As a corollary of Theorem 1 we get the following.

**Corollary 2.** *Languages of PC encodings and PC-BDMCs are polynomially equivalent.*

## Relations to Other Target Compilation Languages

Let us first recall the notion of succinctness introduced by Gogic et al. (1995) and used later extensively by Darwiche and Marquis (2002).

**Definition 5** (Succinctness)**.** Let $\mathbf{L}_1$ and $\mathbf{L}_2$ be two representation languages. We say that $\mathbf{L}_1$ is *at least as succinct as*

$\mathbf{L}_2$, iff there exists a polynomial $p$ such that for every sentence $\varphi \in \mathbf{L}_2$, there exists an equivalent sentence $\psi \in \mathbf{L}_1$ where $|\psi| \leq p(|\varphi|)$. We say that $\mathbf{L}_1$ is *strictly more succinct* than $\mathbf{L}_2$ if $\mathbf{L}_1$ is at least as succinct as $\mathbf{L}_2$ but $\mathbf{L}_2$ is not at least as succinct as $\mathbf{L}_1$.

Corollary 2 implies that the languages of PC encodings and PC-BDMCs are equally succinct. Bova et al. (2014, 2016) show examples of monotone CNF formulas which have only exponentially bigger DNNFs. Given the fact that every monotone CNF formula is PC, we get that PC encodings and PC-BDMCs are strictly more succinct than DNNFs.

Let us relate PC-BDMCs and PC encodings to backdoor trees introduced by Samer and Szeider (2008) when we consider PC formulas as a base class. Backdoor trees were introduced in the context of parameterized SAT solving as an auxiliary data structure that allows to make SAT solving of a given CNF formula easy. Given a base class $\mathcal{C}$, a $\mathcal{C}$ backdoor tree $T$ for a CNF formula $\varphi$ is defined as a decision tree on some of the variables in $\varphi$ which satisfies the following property: If $\alpha$ is a partial assignment specified by a path from the root of $T$ to a leaf, then $\varphi(\alpha)$ (i.e., formula $\varphi$ after we apply partial assignment $\alpha$) belongs to class $\mathcal{C}$. In particular, all the formulas in the leaves are restrictions of the same original formula. For proving a lower bound on the size of a PC backdoor tree, we remove this assumption, so we only require that the formula in every leaf represents the restriction of the original function according to the assignments on the path from the root to the leaf. We will call this structure *generalized $\mathcal{C}$ backdoor tree* and it is precisely the subclass of $\mathcal{C}$-BDMCs that satisfy that the only gates allowed in the circuit part are decision gates (disjunctions of two conjunctions), the directed graph in the circuit part is an out-arborescence, and leaves are associated with $\mathcal{C}$ encodings without auxiliary variables.

Let us point out another generalization of backdoor trees within the framework of BDMCs obtained by including decomposable conjunctions. This is a model that appears as an intermediate state in several CNF to DNNF compilers, whose final output is a Decision DNNF (see e.g., Lagniez and Marquis 2017). From the compilation perspective, it thus makes sense to consider a variant of BDMCs which only allows conjunctions and decision nodes as inner nodes. One of the variants of the lower bound below separates generalized PC backdoor trees with decomposable conjunctions from generalized PC backdoor trees.

In this section, we present a family of boolean functions which have PC-BDMCs of polynomial size, but any generalized PC backdoor tree has exponential size. For this purpose, we measure the sizes of generalized PC backdoor trees and PC-BDMCs in the same way, namely, we sum the sizes of the formulas associated with the leaves with the number of the edges in the circuit part.

For a given $n$, let us define formula $\psi'_n$ on $2n$ variables $y_1, \ldots, y_n, z_1, \ldots, z_n$ as follows.

$$\psi'_n = (\neg z_1 \vee \cdots \vee \neg z_n) \wedge \bigwedge_{i=1}^{n} (\neg y_i \vee z_i)$$

It can be checked that $\psi'_n$ has $2^n$ implicates of form $C_I = \bigvee_{i \in I} \neg z_i \vee \bigvee_{i \notin I} \neg y_i$ for every set of indices $I \subseteq \{1, \ldots, n\}$ in addition to $n$ prime implicates $\neg y_i \vee z_i$, $i = 1, \ldots, n$. Clause $C_I$ is an implicate of $\psi'_n$, because it can be produced by resolving clause $\neg z_1 \vee ... \vee \neg z_n$ with $\neg y_i \vee z_i$ for $i \notin I$. Kučera and Savický (2020, Section 4.1) argued that formula

$$\psi_n = \bigwedge_{C \in \psi'_n} (x \vee C) \tag{5}$$

where $x$ is a new variable has only one prime PC representation which is the list of all its $2^n + n$ prime implicates. We use this property to show the main result of this section.

**Theorem 3.** *PC-BDMCs (and thus also PC encodings) are strictly more succinct than generalized PC backdoor trees.*

*Proof.* For a given $n$, let us consider three sets of variables: $\mathbf{x} = \{x_{i,j} \mid i, j \in \{1, \ldots, n\}\}$, $\mathbf{y} = \{y_{i,j,k} \mid i, j \in \{1, \ldots, n\}, k \in \{1, \ldots, n-1\}\}$, and $\mathbf{z} = \{z_{i,j} \mid i, j \in \{1, \ldots, n\}\}$. For every $i = 1, \ldots, n$, we introduce formulas

$$\begin{aligned} \gamma_i &= (\neg x_{i,1} \vee \cdots \vee \neg x_{i,n}) \\ \delta_i &= (\neg z_{i,1} \vee \cdots \vee \neg z_{i,n}) \\ &\quad \wedge \bigwedge_{s=1}^{n} (\neg y_{i,s,1} \vee \cdots \vee \neg y_{i,s,n-1} \vee z_{i,s}) \end{aligned}$$

Let $f(\mathbf{x}, \mathbf{y}, \mathbf{z})$ be the function represented by the disjunction of formulas $\Gamma = \bigwedge_{i=1}^{n} \gamma_i$ and $\Delta = \bigwedge_{j=1}^{n} \delta_j$. Since $\Gamma$ is a conjunction of clauses on pairwise disjoint sets of variables, it is immediate that $\Gamma$ is PC. Formulas $\delta_i$, $i = 1, \ldots, n$ are PC because each can be constructed by subsequently taking conjunction of PC formulas which share a single variable. This leads to a PC formula as shown by Bordeaux and Marques-Silva (2012, proof of Proposition 5). It follows that $f$ has a small PC-BDMC in form of the disjunction of $\Gamma$ and $\Delta$. Let $T$ be any generalized PC backdoor tree for $f$ which has leaves associated with PC formulas of size less than $2^n$. We claim that every leaf in $T$ has depth at least $n$ and thus the number of leaves of $T$ is at least $2^n$.

Let us consider any partial assignment $\alpha \subseteq \mathrm{lit}(\mathbf{x} \cup \mathbf{y} \cup \mathbf{z})$ of size $n-1$. Since all prime implicates of $\gamma_i$ and of $\delta_j$ for all indices $i$, $j$ have length at least $n$, we get that formulas $\gamma_i(\alpha)$ and $\delta_j(\alpha)$ are satisfiable. Moreover, there is a pair of indices $p$, $q$, such that $\alpha$ does not assign a value to any variable in $\gamma_p$ and $\delta_q$. Partial assignment $\alpha$ can be extended to a partial assignment $\alpha'$ such that $f(\alpha') \equiv \gamma_p \vee \delta_q$. This is possible, since $\gamma_i(\alpha)$, $\delta_j(\alpha)$ are pairwise independent and satisfiable. Let us further extend $\alpha'$ to satisfy all variables $x_{p,1}, \ldots, x_{p,n-1}$ and variables $y_{q,s,t}$ for $s = 1, \ldots, n$ and $t = 1, \ldots, n-2$. In this way, we obtain a restriction of $f$ represented by the formula

$$(\neg x_{p,n} \vee \neg z_{q,1} \vee \cdots \vee \neg z_{q,n}) \wedge \bigwedge_{s=1}^{n} (\neg x_{p,n} \vee \neg y_{q,s,n-1} \vee z_{q,s})$$

which has the same structure as $\psi_n$ defined by (5). This is a contradiction with the choice of $\alpha$, since any PC formula equivalent to (5) has size at least $2^n$. It follows that every leaf in $T$ has depth at least $n$. □

The separation presented in Theorem 3 is based on a function defined by a disjunction of suitably chosen formulas which is not allowed in generalized backdoor trees. A similar separation could be achieved with a function $g$ defined by a conjunction, namely, by the formula $\bigwedge_{i=1}^{n}(\gamma_i \vee \delta'_i)$ where $\delta'_i = (\neg z_{i,1} \vee \cdots \vee \neg z_{i,n}) \wedge \bigwedge_{j=1}^{n}(\neg y_{i,1} \vee z_{i,j})$.

## Queries and Transformations

In this section we shall look at PC encodings and PC-BDMCs as target compilation languages. We shall demonstrate that both these languages have the same properties as DNNFs when it comes to answering queries and transformations described by Darwiche and Marquis (2002).

Let us first look at queries. Darwiche and Marquis (2002) consider **CO** (consistency), **VA** (validity), **CE** (clausal entailment), **EQ** (equivalence), **SE** (sentential entailment), **IM** (implicant), **CT** (model counting), and **ME** (model enumeration). Out of these, **CO**, **CE** and **ME** can be done in polynomial time on a DNNF, the remaining ones cannot be performed in polynomial time unless P is equal to NP. Since DNNFs form a special case of PC-BDMCs, we have that also for PC encodings and PC-BDMCs, answering queries **VA**, **EQ**, **SE**, **IM**, and **CT** is hard. Consistency checking **CO** and clausal entailment **CE** can be performed on a PC encoding by unit propagation. PC encodings also satisfy **ME**, because the models of a PC encoding can be enumerated with polynomial delay by a simple backtrack procedure considering the fact that PC encodings are closed under the application of a partial assignment. In particular, to enumerate the models of a PC encoding $\varphi$, first check if it is satisfiable and if so, pick an unassigned variable of $\varphi$ and recursively enumerate the models of $\varphi(x)$ and $\varphi(\neg x)$ which originate from $\varphi$ by satisfying literals $x$ and $\neg x$ respectively. By Corollary 2 we have that PC-BDMCs and PC encodings are polynomially equivalent and thus they have the same properties with respect to query answering.

Darwiche and Marquis (2002) consider the following transformations on compilation languages: **CD** (conditioning), **FO** (forgetting), **SFO** (singleton forgetting), $\wedge$**C** (conjunction), $\wedge$**BC** (bounded conjunction), $\vee$**C** (disjunction), $\vee$**BC** (bounded disjunction), and $\neg$**C** (negation). Unless P is equal to NP, DNNFs do not allow polytime $\wedge$**C**, $\wedge$**BC**, and $\neg$**C** and since DNNFs are a special case of PC-BDMCs, this is also the case for PC-BDMCs. It is well-known that these operations are not efficient also for PC encodings under the same assumption. The rest of transformations can be done in polynomial time on DNNFs. Let us look at these transformations on PC-BDMCs and PC encodings. **CD** can be done in polynomial time on PC encodings since partial assignment preserves propagation completeness. Both **FO** and **SFO** are trivial on a PC encoding, we just move the variables to be forgotten from the set of main variables to the set of auxiliary variables. Note that PC encodings also allow to forget a single auxiliary variable by means of Davis Putnam resolution. Both $\vee$**C** and $\vee$**BC** can be done on PC-BDMCs in the same way as on DNNFs, just connect the roots of the input PC-BDMCs with a disjunction gate. This transformation is not so trivial on PC encodings. By Theorem 1 we have that a PC-BDMC can be translated into a PC encoding in polynomial time and thus a disjunction of PC encodings can be transformed back into a PC encoding.

## Extended Implicational Dual Rail Encoding

We use the well-known dual rail encoding of partial assignments (see e.g., Bonet et al. 2018; Bryant et al. 1987; Ignatiev, Morgado, and Marques-Silva 2017; Manquinho et al. 1997; Morgado et al. 2019) to simulate unit propagation in a general CNF formula in the same way as Bessiere et al. (2009); Bordeaux et al. (2012); Kučera and Savický (2020). We use the form of the encoding with a special variable representing the fact that contradiction was not derived and extend it with clauses which make the encoding propagation complete if the input CNF formula is PC.

Let us introduce for every $l \in \mathrm{lit}(\mathbf{x})$ a *meta-variable* $[\![l]\!]$. In addition, we use special meta-variable $[\![\top]\!]$ intended to represent the value of a formula in a way suitable for propagating into the circuit part of BDMC. For this purpose, we implement deriving a contradiction as deriving the negative literal $\neg[\![\top]\!]$. The set of the meta-variables corresponding to a vector of variables $\mathbf{x}$ will be denoted

$$\mathrm{meta}(\mathbf{x}) = \{[\![l]\!] \mid l \in \mathrm{lit}(\mathbf{x}) \cup \{\top\}\}\,.$$

For notational convenience, we extend this notation also to sets of literals that are meant as a conjunction, especially to partial assignments. If $\alpha \subseteq \mathrm{lit}(\mathbf{x})$ is a set of literals, then $[\![\alpha]\!] = \{[\![l]\!] \mid l \in \alpha\}$ denotes the set of meta-variables associated with the literals in $\alpha$. If $[\![\alpha]\!]$ is used in a formula such as $\psi \wedge [\![\alpha]\!]$, we identify this set of literals with the conjunction of them, similarly to the interpretation of $\alpha$ in $\varphi \wedge \alpha$.

**Definition 6** (Extended implicational dual rail encoding)**.** Let $\varphi(\mathbf{x})$ be an arbitrary CNF formula. The *extended implicational dual rail encoding* of $\varphi$ is a formula on meta-variables $\mathbf{z} = \mathrm{meta}(\mathbf{x})$ denoted $\mathrm{DR}^+(\varphi, \mathbf{z})$ and defined as follows. If $\varphi$ contains the empty clause, then $\mathrm{DR}^+(\varphi, \mathbf{z}) = \neg[\![\top]\!]$. Otherwise, we set

$$\begin{aligned}\mathrm{DR}^+(\varphi, \mathbf{z}) = \bigwedge_{C \in \varphi} \bigwedge_{l \in C} \left( \bigwedge_{e \in C \setminus \{l\}} [\![\neg e]\!] \to [\![l]\!] \right) \\ \wedge \bigwedge_{x \in \mathbf{x}} ([\![x]\!] \wedge [\![\neg x]\!] \to \neg[\![\top]\!]) \\ \wedge \bigwedge_{l \in \mathrm{lit}(\mathbf{x})} ([\![\top]\!] \vee [\![l]\!]) \wedge \bigwedge_{x \in \mathbf{x}} ([\![x]\!] \vee [\![\neg x]\!])\ .\end{aligned} \tag{6}$$

A subset of extended implicational dual rail encoding is used in the first part of the proof of Theorem 1 by Bessiere et al. (2009) for a similar purpose as in this paper. A similar encoding is used also by Bordeaux et al. (2012) as a part of a larger formula. The proof of the following lemma is omitted, since it is a straightforward extension of the properties of the encodings used in the two papers cited above.

**Lemma 4.** *Let $\varphi(\mathbf{x})$ be a CNF formula without the empty clause, let $\alpha \subseteq \mathrm{lit}(\mathbf{x})$, and assume $\mathbf{z} = \mathrm{meta}(\mathbf{x})$. Then*

$$\varphi \wedge \alpha \vdash_1 \bot \iff \mathrm{DR}^+(\varphi, \mathbf{z}) \wedge [\![\alpha]\!] \vdash_1 \neg[\![\top]\!]$$

*and if* $\varphi \wedge \alpha \not\vdash_1 \bot$*, then for every* $l \in \mathrm{lit}(\mathbf{x})$ *we have*

$$\varphi \wedge \alpha \vdash_1 l \iff \mathrm{DR}^+(\varphi, \mathbf{z}) \wedge [\![\alpha]\!] \vdash_1 [\![l]\!] .$$

The clauses of form $[\![\top]\!] \vee [\![l]\!]$ imply that once $\neg[\![\top]\!]$ is derived by unit propagation, which rules out the possibility that $\varphi$ is consistent with a given partial assignment, all meta-variables $[\![l]\!]$, $l \in \mathrm{lit}(\mathbf{x})$ are derived as well. The clauses $[\![x]\!] \vee [\![\neg x]\!]$ guarantee that every satisfying assignment can be extended to a satisfying assignment of a formula obtained by adding the clauses $l \to [\![l]\!]$, $l \in \mathrm{lit}(\mathbf{x})$. We use the following property of $\mathrm{DR}^+(\varphi, \mathrm{meta}(\mathbf{x}))$ (see the full version (Kučera and Savický 2019a) for the proof).

**Lemma 5.** *If* $\varphi(\mathbf{x})$ *is a PC formula, then* $\mathrm{DR}^+(\varphi, \mathrm{meta}(\mathbf{x}))$ *is a PC formula.*

We use extended implicational dual rail encodings of the formulas in all the leaves as part of the encoding of a BDMC. In order to make the propagation in them independent, the extended implicational dual rail encoding in $\mathrm{leaf}(i)$ uses the following set of meta-variables with indices in place of $\mathbf{z}$

$$\mathrm{meta}_i(\mathbf{x}) = \{[\![l]\!]_i \mid l \in \mathrm{lit}(\mathbf{x}) \cup \{\top\}\} .$$

This is the reason for including the set of variables $\mathbf{z}$ as part of the notation $\mathrm{DR}^+(\varphi, \mathbf{z})$. Encodings $\mathrm{DR}^+(\varphi_i, \mathbf{z}_i)$, $i = 1, \ldots, \ell$ are connected to the remaining parts by identifying the variable representing leaf node $\mathrm{leaf}(i)$ with the variable $[\![\top]\!]_i$ and the consistency with the input variables is guaranteed by adding the clauses $l \to [\![l]\!]_i$ for all $l \in \mathrm{lit}(\mathbf{x}_i)$ (see also Bordeaux et al. 2012). Note that these clauses imply $[\![x]\!]_i \vee [\![\neg x]\!]_i$ for each $x \in \mathbf{x}_i$. The implied clauses are explicitly present in $\mathrm{DR}^+(\varphi_i, \mathbf{z}_i)$ also for $x \in \mathbf{y}_i$.

## PC Encoding of a PC-BDMC

In this section we present the encoding promised in the introduction. To this end, let us fix a PC-BDMC $D$ which represents a function $f(\mathbf{x})$ on variables $\mathbf{x} = (x_1, \ldots, x_n)$. Let $V$ denote the set of nodes in $D$ and let $\rho$ denote the root of $D$. Let us assume that $D$ has $\ell$ leaves which are labeled with PC encodings $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \ldots, \ell$.

### Separators

The construction of the PC encoding relies on the notion of separators introduced by Kučera and Savický (2019b) and briefly described below. Construction of separators can require to include a polynomial number of new nodes into $D$. For every $i = 1, \ldots, n$, let us denote $H_i$ the set of nodes $v$ of $D$ for which $x_i \in \mathrm{var}(v)$. Let $D_i$ be the subgraph of $D$ induced by the vertices in $H_i$. We say that a set of nodes $S \subseteq H_i$ is a *separator* in $D_i$, if every path in $D_i$ from the root to a leaf contains precisely one node from $S$. We say that $D$ can be *covered by separators*, if for each $i = 1, \ldots, n$ there is a collection of separators $\mathcal{S}_i$ in $D_i$, such that the union of $S \in \mathcal{S}_i$ is $H_i$. Not every BDMC can be covered by separators, but every BDMC can be modified in polynomial time into an equivalent one which can be covered by separators (for more details, see Kučera and Savický 2019b).

For the purpose of the construction, let us assume that $D$ is a smooth BDMC covered by separators and let us denote $\mathcal{S}_i$ a set of separators which covers $D_i$. Let us further denote $\mathcal{S} = \bigcup_{i=1}^{n} \mathcal{S}_i$ the set of all separators considered for covering $D$. It follows by smoothness of $D$ that if $T$ is a minimal satisfying subtree of $D$, then for every $i = 1, \ldots, n$, the intersection of $T$ and $D_i$ is a path from the root $\rho$ to a leaf $v$ in $D_i$, i.e. a leaf $v$ satisfying $x_i \in \mathrm{var}(v)$. It follows that $T$ must intersect each separator $S \in \mathcal{S}$ in exactly one node.

### Encoding

We use the extended implicational dual rail encodings $\mathrm{DR}^+(\varphi_i, \mathbf{z}_i)$ of the formulas $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$ associated with the leaves of $D$, where $\mathbf{z}_i = \mathrm{meta}_i(\mathbf{x}_i \cup \mathbf{y}_i)$ is the set of meta-variables specific to the formula $\varphi_i$. The (disjoint) union of these sets of variables is denoted $\mathbf{z} = \bigcup_{i=1}^{\ell} \mathbf{z}_i$.

We associate a variable $v$ with every node $v \in V$. For a leaf $v \in V$ labeled with $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$ for some $i \in \{1, \ldots, \ell\}$, the variable $v$ is identified with $[\![\top]\!]_i$. This identification is understood as follows. The variable actually used in the encoding is $[\![\top]\!]_i$. The variable $v$ corresponding to the leaf is used for simplicity if we refer to its role within the monotone circuit part of $D$ while $[\![\top]\!]_i$ is used when referring to the role of the leaf within the extended implicational dual-rail encoding of $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$ which is a part of the encoding.

The set of variables associated with the inner nodes of $D$ (i.e. the $\wedge$-nodes and $\vee$-nodes) will be denoted $\mathbf{v}$. As explained above, a leaf is represented by a variable $[\![\top]\!]_i$ which belongs to $\mathbf{z}$. Altogether, the encoding described in this section uses three kinds of variables: $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{v}$.

Consider the list of clauses in Table 1. Clauses N1–N3 are the same as introduced by Abío et al. (2016) and (Kučera and Savický 2019b). Clauses N6 are the same as introduced by Kučera and Savický (2019b) and a special case of them was also used by Abío et al. (2016). Clauses N4 used in the cited constructions are not needed for encodings of BDMC. We use exactly-one constraints in group N6, similarly to Kučera and Savický (2019b), if we would use the at-most-one constraints instead, we would obtain a URC encoding provided the encodings associated with leaves are URC. Let us point out that group N6 contains a redundant unit clause $\rho$, since one of the separators in $\mathcal{S}$ is $\{\rho\}$.

Let us denote $\mathcal{P}(\mathbf{x}, \mathbf{z}, \mathbf{v})$ the encoding consisting of the clauses in Table 1. The encoding is clearly constructible in polynomial time and in particular, it has a polynomial size as well. Theorem 1 follows from the following proposition.

**Theorem 6.** *Let* $D$ *be a smooth PC-BDMC covered by separators and representing a function* $f(\mathbf{x})$*. Then* $\mathcal{P}(\mathbf{x}, \mathbf{z}, \mathbf{v})$ *is a PC encoding of* $f(\mathbf{x})$*.*

*Proof sketch.* We only provide a sketch of the proof, see (Kučera and Savický 2019a) for the full proof. As the first step we show that $\mathcal{P}$ is a CNF encoding of $f$. Assume that $\mathbf{a}$ is a model of $f$, i.e. $f(\mathbf{a}) = 1$. It follows that there is a minimal satisfying subtree $T$ of $D$. We form assignments $\mathbf{b}$ and $\mathbf{c}$ of values to variables in $\mathbf{z}$ and $\mathbf{v}$ respectively as follows. For every node $v \in \mathbf{v}$ set $\mathbf{c}(v) = 1$ if and only if $v$ is in $T$. A variable $v$ representing $\mathrm{leaf}(i)$ is identified with $[\![\top]\!]_i$ in $\mathbf{z}$. Consistently with the above, set $\mathbf{b}(v) = \mathbf{b}([\![\top]\!]_i) = 1$ if and only if $\mathrm{leaf}(i)$ is in $T$. For every $i$ such that $\mathbf{b}([\![\top]\!]_i) = 0$ we set $\mathbf{b}([\![l]\!]_i) = 1$ for every literal $l \in \mathrm{lit}(\mathbf{x}_i \cup \mathbf{y}_i)$. If

| group | clause | condition |
|---|---|---|
| N1 | $v \to v_1 \vee \cdots \vee v_k$ | $v = v_1 \vee \cdots \vee v_k$ |
| N2 | $v \to v_i$ | $v = v_1 \wedge \cdots \wedge v_k, i = 1, \ldots, k$ |
| N3 | $v \to p_1 \vee \cdots \vee p_k$ | $v \in V$ has incoming edges from $p_1, \ldots, p_k$ |
| N6 | $\mathrm{eo}(S)$ | $S \in \mathcal{S}$ |
| R | $\rho$ | $\rho$ is the root |
| E1 | $\mathrm{DR}^+(\varphi_i, \mathbf{z}_i)$ | $i = 1, \ldots, \ell$, $\mathbf{z}_i = \mathrm{meta}_i(\mathbf{x}_i \cup \mathbf{y}_i)$ |
| E2 | $l \to [\![l]\!]_i$ | $l \in \mathrm{lit}(\mathbf{x}_i), i = 1, \ldots, \ell$ |
| E3 | $\left(\bigwedge_{i \in \mathrm{range}(l)} [\![l]\!]_i\right) \to l$ | $l \in \mathrm{lit}(\mathbf{x})$ |

Table 1: List of clauses of the encoding $\mathcal{P}(\mathbf{x}, \mathbf{z}, \mathbf{v})$. $\mathcal{S} = \bigcup_{i=1}^{n} \mathcal{S}_i$ denotes a fixed collection of separators which covers $D$ and $\mathrm{eo}(S)$ denotes a suitable PC encoding of the exactly-one constraint on the variables in $S$. By construction, we have $v = [\![\top]\!]_i$ for a leaf node $v$ associated with formula $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$.

$\mathbf{b}([\![\top]\!]_i) = 1$, then $\varphi_i(\mathbf{x}_i, \mathbf{y}_i)$ is associated with a leaf of $T$ and thus it has a model consistent with $\mathbf{a}$ and we set the values $\mathbf{b}([\![l]\!]_i)$ according to this model for every $l \in \mathrm{lit}(\mathbf{x}_i, \mathbf{y}_i)$. One can check that $\mathcal{P}(\mathbf{a}, \mathbf{b}, \mathbf{c})$ is satisfied.

For the other direction assume that we have assignments $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ of values to variables in $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{v}$ respectively and which together satisfy $\mathcal{P}$. Let us denote $D'$ the subgraph of $D$ which is induced by the inner nodes $v$ for which $\mathbf{c}(v) = 1$ and leaves with indices $i$ for which $\mathbf{b}([\![\top]\!]_i) = 1$. We can show that $D'$ contains a minimal satisfying subtree $T$ of $D$ consistent with $\mathbf{a}$ and thus $\mathbf{a}$ is a model of $f$.

It remains to show that $\mathcal{P}$ is PC. Let $\psi_p$ be the formula formed by clauses of groups N1–N3, N6, and $\rho$. Formula $\psi_p$ is defined over variables which correspond to all nodes $V$ of $D$, i.e. $\mathbf{v}$ and the leaves. The formula denoted $\psi_p$ by Kučera and Savický (2019b) differs from the formula denoted in the same way here in the interpretation of the leaves. However, we can show that $\psi_p$ is PC in the same way as Kučera and Savický (2019b). Bordeaux and Marques-Silva (2012) showed that a formula which is composed as a conjunction of two PC formulas sharing a single variable is PC. By inductive use of this argument together with Lemma 5 we obtain that for each $i = 1, \ldots, \ell$ the formula $\psi_i = \psi_{i-1} \wedge \mathrm{DR}^+(\varphi_i, \mathbf{z}_i)$ is PC. It follows that formula $\mathcal{P}'(\mathbf{z}, \mathbf{v}) = \psi_\ell$ formed by clauses of groups N1–N3, N6, E1, and $\rho$ is PC. Clauses N6 ensure that any model of $\mathcal{P}'$ intersects every separator at exactly one node. The models of $\mathcal{P}'$ are thus precisely the encodings of minimal satisfying subtrees of $D$. Using smoothness, this implies that any model of $\mathcal{P}'$ can be extended to a model of $\mathcal{P}$ in which the values of the main variables are uniquely determined by clauses E2 and E3. This can be used to finally show that $\mathcal{P}$ is PC. □

Let us close the section with an asymptotic estimate of the size of encoding $\mathcal{P}$. We will assume that the exactly one constraints (clauses N6) are represented with a linear size encoding described by Kučera and Savický (2019b). The encoding then contains $O(ns+m)$ variables and $O(ns+e+r)$ clauses where $n$ denotes the number of input variables, $s$ the number of nodes of $D$, $e$ the number of edges of $D$, $m$ the total number of variables in the formulas associated with the leaves, and $r$ the total length of these formulas (the sum of the lengths of all the clauses).

## Conclusion and Further Research

We have introduced the language of $\mathcal{C}$-BDMCs which is a common generalization of DNNFs (introduced by Darwiche 1999) and $\mathcal{C}$-backdoor trees (introduced by Samer and Szeider 2008). Moreover, URC-BDMCs contain the disjunctive closure of URC encodings URC-C$[\vee, \exists]$ (Bordeaux et al. 2012) as a subset.

We have shown that PC-BDMCs are polynomially equivalent to PC encodings (introduced by Bordeaux and Marques-Silva 2012). In particular, the language of disjunctions of PC encodings is polynomially equivalent to PC encodings. By a slight modification of our construction, we get a similar result for URC encodings which is a generalization of the results of Bordeaux et al. (2012). We have demonstrated that PC-BDMCs and PC encodings have the same properties as DNNFs with respect to query answering and transformations considered in the knowledge compilation map (Darwiche and Marquis 2002). Using the results of Bova et al. (2014, 2016), PC encodings and PC-BDMCs are strictly more succinct than DNNFs and we have shown that they are also strictly more succinct than generalized PC-backdoor trees we have introduced when discussing the relation of PC-BDMCs to other target compilation languages.

Although PC encodings and PC-BDMCs are polynomially equivalent, a compilation from a CNF into a PC-BDMC can be easier than to a PC encoding. In particular, we might consider modifications of the techniques of compiling a CNF into a DNNF (see e.g., Darwiche 2004; Lagniez and Marquis 2017; Muise et al. 2012) such that the process of splitting the formula into pieces stops at encodings from a class $\mathcal{C}$, so earlier than at the literals. Note that some of the compilers into a DNNF (e.g., D4 introduced by Lagniez and Marquis 2017) actually compile into a Decision DNNF which is a DNNF where only decision nodes and decomposable $\wedge$-gates are allowed. It is natural to consider $\mathcal{C}$-BDMCs restricted in the same way. Such Decision $\mathcal{C}$-BDMCs form a language more restricted than general $\mathcal{C}$-BDMCs, but more general than generalized $\mathcal{C}$-backdoor trees. Further research is needed to determine whether one can construct a reasonably efficient compiler of a CNF formula into a $\mathcal{C}$-BDMC which uses for example prime 2-CNFs (which are PC) or renamable Horn formulas (which are URC) as the base class.

## Acknowledgements

Petr Kučera acknowledges the support by Czech-French Mobility program (Czech Ministry of Education, grant No. 7AMB17FR027). Both authors acknowledge the support by Grant Agency of the Czech Republic (grant No. GA19–19463S). Both authors would like to thank Pierre Marquis, Jean-Marie Lagniez, Gilles Audemard, and Stefan Mengel (from CRIL, U. Artois, Lens, France) for discussion on the compilation of CNF formulas into more general structures than DNNF.

## References

Abío, I.; Gange, G.; Mayer-Eichberger, V.; and Stuckey, P. J. 2016. On CNF Encodings of Decision Diagrams. In Quimper, C.-G., ed., *Integration of AI and OR Techniques in Constraint Programming*, 1–17. Cham: Springer International Publishing. ISBN 978-3-319-33954-2.

Babka, M.; Balyo, T.; Čepek, O.; Gurský, Š.; Kučera, P.; and Vlček, V. 2013. Complexity issues related to propagation completeness. *Artificial Intelligence* 203(0): 19 – 34. ISSN 0004-3702. doi:http://dx.doi.org/10.1016/j.artint.2013.07.006.

Bacchus, F. 2007. GAC Via Unit Propagation. In Bessière, C., ed., *Principles and Practice of Constraint Programming – CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, 133–147. Springer Berlin Heidelberg. ISBN 978-3-540-74969-1. doi:10.1007/978-3-540-74970-7_12.

Bessiere, C.; Katsirelos, G.; Narodytska, N.; and Walsh, T. 2009. Circuit Complexity and Decompositions of Global Constraints. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, 412–418.

Bonet, M. L.; Buss, S.; Ignatiev, A.; Marques-Silva, J.; and Morgado, A. 2018. MaxSAT resolution with the dual rail encoding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Bordeaux, L.; Janota, M.; Marques-Silva, J.; and Marquis, P. 2012. On Unit-refutation Complete Formulae with Existentially Quantified Variables. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, 75–84. AAAI Press. ISBN 978-1-57735-560-1.

Bordeaux, L.; and Marques-Silva, J. 2012. Knowledge Compilation with Empowerment. In Bieliková, M.; Friedrich, G.; Gottlob, G.; Katzenbeisser, S.; and Turán, G., eds., *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, 612–624. Springer Berlin / Heidelberg. ISBN 978-3-642-27659-0.

Bova, S.; Capelli, F.; Mengel, S.; and Slivovsky, F. 2014. A Strongly Exponential Separation of DNNFs from CNF Formulas. *arXiv preprint arXiv: 1411.1995* .

Bova, S.; Capelli, F.; Mengel, S.; and Slivovsky, F. 2016. Knowledge Compilation Meets Communication Complexity. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, 1008–1014. AAAI Press. ISBN 978-1-57735-770-4.

Bryant, R. E.; Beatty, D.; Brace, K.; Cho, K.; and Sheffler, T. 1987. COSMOS: A Compiled Simulator for MOS Circuits. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, DAC '87, 9–16. New York, NY, USA: Association for Computing Machinery. ISBN 0818607815. doi:10.1145/37888.37890.

Darwiche, A. 1999. Compiling Knowledge into Decomposable Negation Normal Form. In *Proceedings of the 16th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'99, 284–289. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Darwiche, A. 2001. On the Tractable Counting of Theory Models and its Application to Truth Maintenance and Belief Revision. *Journal of Applied Non-Classical Logics* 11(1-2): 11–34. doi:10.3166/jancl.11.11-34.

Darwiche, A. 2004. New Advances in Compiling CNF to Decomposable Negation Normal Form. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, 318–322. Amsterdam, The Netherlands: IOS Press. ISBN 978-1-58603-452-8.

Darwiche, A.; and Marquis, P. 2002. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research* 17: 229–264.

del Val, A. 1994. Tractable Databases: How to Make Propositional Unit Resolution Complete through Compilation. In *Knowledge Representation and Reasoning*, 551–561.

Fargier, H.; and Marquis, P. 2008. Extending the Knowledge Compilation Map: Closure Principles. In *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, 50–54. Amsterdam, The Netherlands, The Netherlands: IOS Press. ISBN 978-1-58603-891-5.

Gange, G.; and Stuckey, P. J. 2012. Explaining Propagators for s-DNNF Circuits. In Beldiceanu, N.; Jussien, N.; and Pinson, É., eds., *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, 195–210. Springer Berlin Heidelberg. ISBN 978-3-642-29828-8.

Gogic, G.; Kautz, H.; Papadimitriou, C.; and Selman, B. 1995. The Comparative Linguistics of Knowledge Representation. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, 862–869. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1-55860-363-8, 978-1-558-60363-9.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2017. On Tackling the Limits of Resolution in SAT Solving. In Gaspers, S.; and Walsh, T., eds., *Theory and Applications of Satisfiability Testing – SAT 2017*, 164–183. Cham: Springer International Publishing. ISBN 978-3-319-66263-3.

Jung, J. C.; Barahona, P.; Katsirelos, G.; and Walsh, T. 2008. Two Encodings of DNNF Theories. In *ECAI'08 Workshop on Inference methods based on Graphical Structures of Knowledge*.

Kučera, P.; and Savický, P. 2019a. Backdoor Decomposable Monotone Circuits and their Propagation Complete Encodings. *arXiv preprint arXiv:1811.09435* .

Kučera, P.; and Savický, P. 2019b. Propagation complete encodings of smooth DNNF theories. *arXiv preprint arXiv:1909.06673* .

Kučera, P.; and Savický, P. 2020. Bounds on the size of PC and URC formulas. *Journal of Artificial Intelligence Research* 69: 1395–1420. doi:10.1613/jair.1.12006.

Lagniez, J.-M.; and Marquis, P. 2017. An Improved Decision-DNNF Compiler. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 667–673. doi:10.24963/ijcai.2017/93.

Manquinho, V. M.; Flores, P. F.; Silva, J. P. M.; and Oliveira, A. L. 1997. Prime implicant computation using satisfiability algorithms. In *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence*, 232–239. ISSN 1082-3409. doi:10.1109/TAI.1997.632261.

Marquis, P. 2015. Compile! In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, 4112–4118. AAAI Press. ISBN 0262511290.

Morgado, A.; Ignatiev, A.; Bonet, M. L.; Marques-Silva, J.; and Buss, S. 2019. DRMaxSAT with MaxHS: First Contact. In *International Conference on Theory and Applications of Satisfiability Testing*, 239–249. Springer.

Muise, C.; McIlraith, S. A.; Beck, J. C.; and Hsu, E. I. 2012. Dsharp: Fast d-DNNF Compilation with sharpSAT. In Kosseim, L.; and Inkpen, D., eds., *Advances in Artificial Intelligence: 25th Canadian Conference on Artificial Intelligence, Canadian AI 2012, Toronto, ON, Canada, May 28-30, 2012. Proceedings*, 356–361. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-30353-1. doi:10.1007/978-3-642-30353-1_36.

Samer, M.; and Szeider, S. 2008. Backdoor Trees. In Fox, D.; and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 363–368. AAAI Press. ISBN 978-1-57735-368-3.

Tseitin, G. S. 1983. On the complexity of derivation in propositional calculus. In *Automation of Reasoning*, 466–483. Springer.