Theory of Computing Systems
https://doi.org/10.1007/s00224-019-09930-2

Connecting Knowledge Compilation Classes Width Parameters

Antoine Amarilli[1] · **Florent Capelli**[2,3] · **Mikaël Monet**[1,3] · **Pierre Senellart**[1,3,4]

Published online: 10 June 2019

Abstract

The field of *knowledge compilation* establishes the tractability of many tasks by studying how to *compile* them to Boolean circuit classes obeying some requirements such as structuredness, decomposability, and determinism. However, in other settings such as intensional query evaluation on databases, we obtain Boolean circuits that satisfy some width bounds, e.g., they have bounded treewidth or pathwidth. In this work, we give a systematic picture of many circuit classes considered in knowledge compilation and show how they can be systematically connected to width measures, through upper and lower bounds. Our upper bounds show that bounded-treewidth circuits can be constructively converted to d-SDNNFs, in time linear in the circuit size and singly exponential in the treewidth; and that bounded-pathwidth circuits can similarly be converted to uOBDDs. We show matching lower bounds on the compilation of monotone DNF or CNF formulas to structured targets, assuming a constant bound on the arity (size of clauses) and degree (number of occurrences of each variable): any d-SDNNF (resp., SDNNF) for such a DNF (resp., CNF) must be of exponential size in its treewidth, and the same holds for uOBDDs (resp., n-OBDDs) when considering pathwidth. Unlike most previous work, our bounds apply to *any* formula of this class, not just a well-chosen family. Hence, we show that pathwidth and treewidth respectively characterize the efficiency of compiling monotone DNFs to uOBDDs and d-SDNNFs with compilation being singly exponential in the corresponding width parameter. We also show that our lower bounds on CNFs extend to unstructured compilation targets, with an exponential lower bound in the treewidth (resp., pathwidth) when compiling monotone CNFs of constant arity and degree to DNNFs (resp., nFBDDs).

Keywords Knowledge compilation · Treewidth · Pathwidth · Circuit · Boolean function

This article is part of the Topical Collection on *Special Issue on Database Theory (2018)*

✉ Mikaël Monet
mikael.monet@imfd.cl

Extended author information available on the last page of the article.

1 Introduction

Knowledge compilation studies how problems can be solved by compiling them into classes of Boolean circuits or binary decision diagrams (BDDs) to which general-purpose algorithms can be applied. This field has introduced numerous such classes or *compilation targets*, defined by various restrictions on the circuits or BDDs, and studied which operations can be solved on them; e.g., the class of *d-DNNFs* requires that negation is only applied at the leaves, $\wedge$-gates are on disjoint variable subsets, and $\vee$-gates have mutually exclusive inputs. However, a different way to define restricted classes is to bound some graph-theoretic *width parameters*, e.g., treewidth, which measures how the data can be decomposed as a tree, or pathwidth, the special case of treewidth with path-shaped decompositions. Such restrictions have been used in particular in the field of database theory and probabilistic databases [49] in the so-called *intensional approach* where we compute a *lineage circuit* [34] that represents the output of a query or the possible worlds that make it true, and where these circuits can sometimes be shown to have bounded treewidth [2, 3, 33].

At first glance, classes such as *bounded-treewidth circuits* seem very different from usual knowledge compilation classes such as d-DNNF. Yet, for some tasks such as probability computation (computing the probability of the circuit under an independent distribution on the variables), both classes are known to be tractable: the problem can be solved in linear time on d-DNNFs by definition of the class [26], and for bounded-treewidth circuits we can use *message passing* [36] to solve probability computation in time linear in the circuit and exponential in the treewidth. This hints at the existence of a connection between traditional knowledge compilation classes and bounded-width classes.

This paper presents such a connection and shows that the width of circuits is intimately linked to many well-known knowledge compilation classes. Specifically, we show a link between the *treewidth* of Boolean circuits and the width of their representations in common circuit targets; and show a similar link between the *pathwidth* of Boolean circuits and the width of their representation in BDD targets. We demonstrate this link by showing *upper bound* results on compilation targets, to show that bounded-width circuits can be compiled to circuits or BDD targets in linear time and with singly exponential complexity in the width parameter. We also show corresponding *lower bound* results that establish that these compilation targets must be exponential in the width parameters, already for a restricted class of Boolean formulas. We now present our contributions and results in more detail.

The first contribution of this paper (in Section 3) is to give a systematic picture of the 12 knowledge compilation circuit classes that we investigate. We classify them along three independent axes:

– *Conjunction*: we distinguish between *BDD classes*, such as OBDDs (ordered binary decision diagrams [18]), where logical conjunction is only used to test the value of a variable and where computation follows a *path* in the structure; and *circuit classes* which allow decomposable conjunctions and where computation follows a *tree*.

- *Structuredness*: we distinguish between *structured classes*, where the circuit or BDD always decomposes the variables along the same order or v-tree [40], and *unstructured classes* where no such restriction is imposed except *decomposability* (each variable must be read at most once).
- *Determinism*: we distinguish between classes that feature *no disjunctions* beyond decision on a variable value (OBDDs, FBDDs, and dec-DNNFs), classes that feature *unambiguous* or *deterministic disjunctions* (uOBDDs, uFBDDs, and d-DNNFs), and classes that feature *arbitrary disjunctions* (nOBDDs, nFBDDs, and DNNFs).

This landscape is summarized in Fig. 1, and we review known translations and separation results that describe the relative expressive power of these features.

The second contribution of this paper (in Sections 4 and 5) is to show an *upper bound* on the compilation of bounded-treewidth classes to d-SDNNFs, and of bounded-pathwidth classes to OBDD variants. For *pathwidth*, existing work had already studied the compilation of bounded-pathwidth circuits to OBDDs [34, Corollary 2.13], which can be made constructive [4, Lemma 6.9]. Specifically, they show that a circuit of pathwidth $\leqslant k$ can be converted in polynomial time into an OBDD of width $\leqslant 2^{(k+2)2^{k+2}}$. Our first contribution is to show that, by using *unambiguous OBDDs* (*uOBDDs*), we can do the same but with linear time complexity, and with the size of the uOBDD as well as its *width* (in the classical knowledge compilation sense) being singly exponential in the pathwidth. Specifically:

Result 1 (**see Theorem 4.4**) *Given as input a Boolean circuit C of pathwidth k on n variables, we can compute in time $O(|C| \times f(k))$ a complete uOBDD equivalent to C of width $\leqslant f(k)$ and size $O(n \times f(k))$, where f is singly exponential.*

For *treewidth*, we show that bounded-treewidth circuits can be compiled to the class of *d-SDNNF circuits*:

Result 2 (**see Corollary 4.3**) *Given as input a Boolean circuit C of treewidth k on n variables, we can compute in time $O(|C| \times f(k))$ a complete d-SDNNF equivalent to C of width $\leqslant f(k)$ and size $O(n \times f(k))$, where f is singly exponential.*

The proof of Result 2, and its variant that shows Result 1, is quite challenging: we transform the input circuit bottom-up by considering all possible valuations of the gates in each bag of the tree decomposition, and keeping track of additional information to remember which guessed values have been substantiated by a corresponding input. Result 2 generalizes a recent theorem of Bova and Szeider in [16] which we improve in two ways. First, our result is constructive, whereas [16] only shows a bound on the size of the d-SDNNF, without bounding the complexity of effectively computing it. Second, our bound is singly exponential in k, whereas [16] is doubly exponential; this allows us to be competitive with message passing (also singly exponential in k), and we believe it can be useful for practical applications. We also explain how Result 2 implies the tractability of several tasks on bounded-treewidth

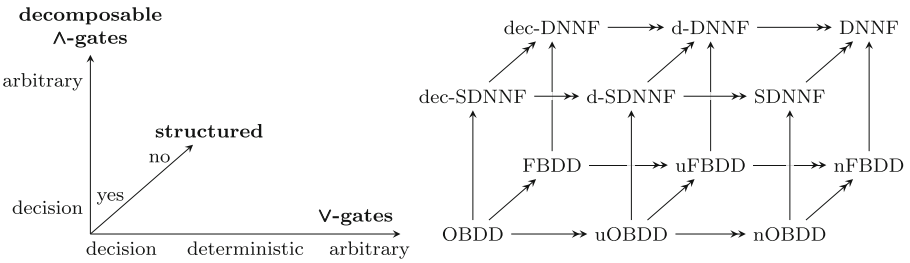

Fig. 1 Dimensions of the knowledge compilation classes consider in this paper (left), and diagram of the classes (right). Arrows indicate polynomial-time compilation; all classes are separated and no arrows are missing (except those implied by transitivity). Double arrows indicate that the classes are exponentially separated; when an arrow is not double, quasi-polynomial compilation in the reverse direction exists

circuits, e.g., probabilistic query evaluation, enumeration [1], quantification [22], MAP inference [31], etc.

The third contribution of this paper is to show *lower bounds* on how efficiently we can convert from width-based classes to the compilation targets that we study. Our bounds already apply to a weaker formalism of width-based circuits, namely, monotone formulas in CNF (conjunctive normal form) or DNF (disjunctive normal form). Our first two bounds (in Sections 6 and 7) are shown for *structured* compilation targets, i.e., OBDDs, where we follow a fixed order on variables, and SDNNFs, where we follow a fixed v-tree; and they apply to arbitrary monotone CNFs and DNFs. The first lower bound concerns pathwidth and OBDD representations: we show that, up to factors in the formula arity (maximal size of clauses) and degree (maximal number of variable occurrences), any OBDD for a monotone CNF or DNF must be of width exponential in the pathwidth $\mathrm{pw}(\varphi)$ of the formula φ. Formally:

Result 3 (**Corollary 7.5**) *For any monotone CNF φ (resp., monotone DNF φ) of constant arity and degree, the size of the smallest nOBDD (resp., uOBDD) computing φ is $2^{\Omega(\mathrm{pw}(\varphi))}$.*

This result generalizes several existing lower bounds in knowledge compilation that exponentially separate CNFs from OBDDs, such as [30] and [15, Theorem 19].

Our second lower bound shows the analogue of Result 3 for the treewidth $\mathrm{tw}(\varphi)$ of the formula φ and (d-)SDNNFs:

Result 4 (**Corollary 7.6**) *For any monotone CNF φ (resp., monotone DNF φ) of constant arity and degree, the size of the smallest SDNNF (resp., d-SDNNF) computing φ is $2^{\Omega(\mathrm{tw}(\varphi))}$.*

These two lower bounds contribute to a vast landscape of knowledge compilation results giving lower bounds on compiling specific Boolean functions to restricted circuits classes, e.g., [15, 30, 43] to OBDDs, [19] to dec-SDNNF, [8] to *sentential decision diagrams* (SDDs), [14, 41] to d-SDNNF, [14, 20, 21] to d-DNNFs and DNNFs. However, all those lower bounds (with the exception of some results in

[20, 21]) apply to well-chosen families of Boolean functions (usually CNF), whereas Results 3 and 4 apply to *any* monotone CNF and DNF. Together with Result 1, these generic lower bounds point to a strong relationship between width parameters and structure representations, on monotone CNFs and DNFs of constant arity and degree. Specifically, the smallest width of OBDD representations of any such formula φ is in $2^{\Theta(\mathrm{pw}(\varphi))}$, i.e., precisely singly exponential in the pathwidth; and an analogous bound applies to d-SDNNF size and treewidth of DNFs.

To prove these two lower bounds, we leverage known results from knowledge compilation and communication complexity [14] (in Section 6) of which we give a unified presentation. Specifically, we show that Boolean functions captured by uOBDDs (resp., nOBDDs) and d-SDNNF (resp., SDNNF) variants can be represented via a small cover (resp., disjoint cover) of so-called *rectangles*. We also show two Boolean functions (*set covering* and *set intersection*) which are known not to have any such covers. We then bootstrap the lower bounds on these two functions to a general lower bound in Section 7, by rephrasing pathwidth and treewidth to new notions of *pathsplitwidth* and *treesplitwidth*, which intuitively measure the performance of a variable ordering or v-tree. We then show that, for DNFs and CNFs with a high pathsplitwidth (resp., treesplitwidth), we can find the corresponding hard function "within" the CNF or DNF, and establish hardness.

Our last lower bound result is shown in Section 8, where we lift the assumption that the compilation targets are structured:

Result 5 (**Corollary 8.5**) *For any monotone CNF φ of constant arity and degree, the size of the smallest nFBDD computing φ is $2^{\Omega(\mathrm{pw}(\varphi))}$, and the size of the smallest DNNF computing φ is $2^{\Omega(\mathrm{tw}(\varphi))}$.*

This result generalizes Results 3 and 4 by lifting the structuredness assumption, but they only apply to CNFs (and not to DNFs). The proof of these results reuses the notions of pathsplitwidth and treesplitwidth, along with a more involved combinatorial argument on the size of rectangle covers.

The current article extends the conference article [5] in many ways:

- We added Section 3 which gives a systematic presentation of knowledge compilation classes and reviews known results that relate them.
- In Section 4, the upper bound result on uOBDDs (Result 1) was added, the results were rephrased in terms of width, and the size of the circuit has been improved[1] to be linear in the number of variables like in [16].
- The presentation of the lower bounds in Sections 6 and 7 was restructured to clarify the connection with communication complexity. The lower bounds on OBDDs (Result 3) was extended to uOBDDs and nOBDDs.
- The bounds on unstructured representations in Section 8 are new.
- We include full proofs for all results.

[1] This observation is due to Stefan Mengel and is adapted from the recent article [22].

2 Preliminaries

We give preliminaries on trees, hypergraphs, treewidth, and Boolean functions.

Graphs, Trees, and DAGs We use the standard notions of directed and undirected graphs, of paths in a graph, and of cycles. All graphs considered in the paper are finite.

A *tree* T is an undirected graph that has no cycles and that is connected (i.e., there exists exactly one path between any two different nodes). Its *size* $|T|$ is its number of edges. A tree T is *rooted* if it has a distinguished node r called the *root* of T. Given two adjacent nodes n_1, n_2 of a rooted tree T with root r, if n_1 lies on the (unique) path from r to n_2, we say that n_1 is the *parent* of n_2 and that n_2 is a *child* of n_1. A *leaf* of T is a node that has no child, and an *internal node* of T is a node that is not a leaf. Given a set U of nodes of T, we denote the set of leaves of U by $\mathsf{Leaves}(U)$. A node n' is a *descendant* of a node n in a rooted tree if $n \neq n'$ and n lies on the path from n' to the root. For $n \in T$, we denote by T_n the subtree of T rooted at n. A rooted tree is *binary* if all nodes have at most two children, and it is *full* if all internal nodes have exactly two children. A rooted full binary tree is called *right-linear* if the children of each internal node are ordered (we then talk of a *left* or *right child*), and if every right child is a leaf.

A *directed acyclic graph* (or *DAG*) D is a directed graph that has no cycles. A DAG D is *rooted* if it has a distinguished node r such that there is a path from r to every node in D. A *leaf* of D is a node that has no child.

Hypergraphs, Treewidth, Pathwidth A *hypergraph* $H = (V, E)$ consists of a finite set of *nodes* (or *vertices*) V and of a set E of *hyperedges* (or simply *edges*) which are non-empty subsets of V. We always assume that hypergraphs have at least one edge. For a node v of H, we write $E(v)$ for the set of edges of H that contain v. The *arity* of H, written $\mathrm{arity}(H)$, is the maximal size of an edge of H. The *degree* of H, written $\mathrm{degree}(H)$, is the maximal number of edges to which a vertex belongs, i.e., $\max_{v \in V} |E(v)|$.

A *tree decomposition* of a hypergraph $H = (V, E)$ is a rooted tree T, whose nodes b (called *bags*) are labeled by a subset $\lambda(b)$ of V, and which satisfies:

(i) for every hyperedge $e \in E$, there is a bag $b \in T$ with $e \subseteq \lambda(b)$;
(ii) for all $v \in V$, the set of bags $\{b \in T \mid v \in \lambda(b)\}$ is a connected subtree of T.

For brevity, we often identify a bag b with its domain $\lambda(b)$. The *width* of T is $\max_{b \in T} |\lambda(b)| - 1$. The *treewidth* of H, denoted $\mathrm{tw}(H)$, is the minimal width of a tree decomposition of H. Pathwidth (denoted $\mathrm{pw}(H)$) is defined similarly but with *path decompositions*, tree decompositions where all nodes have at most one child.

It is NP-hard to determine the treewidth of a hypergraph (V, E), but we can compute a tree decomposition in linear time when parametrizing by the treewidth. This can be done in time $O(|V| \times 2^{(32+\epsilon)k^3})$ with the classical result of [9], or, using a recent algorithm by Bodlaender et al. [10], in time $O(|V| \times 2^{ck})$:

Theorem 2.1 [10] *There exists a constant c such that, given a hypergraph $H = (V, E)$ and an integer $k \in \mathbb{N}$, we can check in time $O(|V| \times 2^{ck})$ whether $\mathrm{tw}(H) \leqslant k$, and if yes output a tree decomposition of H of width $\leqslant 5k + 4$.*

For simplicity, we will often assume that a tree decomposition is *v-friendly*, for a node $v \in V$, meaning that:

1. it is a full binary tree, i.e., each node has exactly zero or two children;
2. for every internal bag b with children b_l, b_r we have $b \subseteq b_l \cup b_r$;
3. for every leaf bag b we have $|b| \leqslant 1$;
4. the root bag of T only contains the node v.

Assuming a tree decomposition to be v-friendly for a fixed v can be done without loss of generality:

Lemma 2.2 *Given a tree decomposition T of a hypergraph H of width k and a node v of H, we can compute in time $O(k \times |T|)$ a v-friendly tree decomposition T' of H of width k.*

Proof We first create a bag b_{root} containing only the node v, and make this bag the root of T by connecting it to a bag of T that contains v (if there is no such bag then we connect b_{root} to an arbitrary bag of T). Then, we make the tree decomposition binary (but not necessarily full) by replacing each bag b with children $b_1, \ldots, b_n$ with $n > 2$ by a chain of bags with the same label as b to which we attach the children $b_1, \ldots, b_n$. This process is in time $O(|T|)$ and does not change the width.

We then ensure the second and third conditions, by applying a transformation to leaf bags and to internal bags. We first modify every leaf bag b containing more than one vertex by a chain of at most k internal bags with leaves where the vertices are added one after the other. Then, we modify every internal bag b that contains elements $v_1, \ldots, v_n$ not present in the union D of its children: we replace b by a chain of at most k internal bags $b'_1, \ldots, b'_n$ containing respectively $b, b \setminus \{v_n\}, b \setminus \{v_n, v_{n-1}\}, \ldots, D$, each bag having a child introducing the corresponding gate v_i. This is in time $O(k \times |T|)$, and again it does not change the width; further, the result of the process is a tree decomposition that satisfies the second, third and fourth conditions and is still a binary tree.

The only missing part is to ensure that the tree decomposition is full, which we can simply do in linear time by adding bags with an empty label as a second children for internal nodes that have only one child. This is obviously in linear time, does not change the width, and does not affect the other conditions, concluding the proof. □

Boolean Functions A (Boolean) *valuation* of a set V is a function $\nu : V \to \{0, 1\}$, which can also be seen as the set of elements of V mapped to 1. A *Boolean function* φ on variables V is a mapping $\varphi : 2^V \to \{0, 1\}$ that associates to each valuation ν of V a Boolean value $\varphi(\nu)$ in $\{0, 1\}$ called the *evaluation* of φ according to ν. We write $\#\varphi$ the number of satisfying valuations of φ. Given two Boolean functions φ_1, φ_2, we write $\varphi_1 \Rightarrow \varphi_2$ when every satisfying valuation of φ also satisfies φ_2. We write $\bot$ the Boolean function that maps every valuation to 0.

Let X, Y be two disjoint sets, ν a valuation on X and ν' a valuation on Y. We denote by $\nu \cup \nu'$ the valuation on $X \cup Y$ such that $(\nu \cup \nu')(a)$ is $\nu(a)$ if $a \in X$ and is $\nu'(a)$ if $a \in Y$. Let φ be a Boolean function on V, and ν be a valuation on a set $X \subseteq V$. We denote by $\nu(\varphi)$ the Boolean function on variables $V \setminus X$ such that, for any valuation ν' of $V \setminus X$, $\nu(\varphi)(\nu') = \varphi(\nu \cup \nu')$. When ν is a Boolean valuation on V and $X \subseteq V$, we denote by $\nu|_X$ the Boolean valuation on X defined by $\nu|_X(x) = \nu(x)$ for all x in X.

Two simple formalisms for representing Boolean functions are *Boolean circuits* and formulas in *conjunctive normal form* or *disjunctive normal form*. We will discuss more elaborate formalisms, namely binary decision diagrams and decomposable normal negation forms, in Section 3.

Boolean Circuits A *(Boolean) circuit* $C = (G, W, g_{\mathsf{output}}, \mu)$ is a DAG (G, W) whose vertices G are called *gates*, whose edges W are called *wires*, where $g_{\mathsf{output}} \in G$ is the *output gate*, and where each gate $g \in G$ has a *type* $\mu(g)$ among var (a *variable gate*), $\neg$, $\vee$, $\wedge$. The *inputs* of a gate $g \in G$ is the set $W(g)$ of gates $g' \in G$ such that $(g', g) \in W$; the *fan-in* of g is its number of inputs. We require $\neg$-gates to have fan-in 1 and var-gates to have fan-in 0. The *treewidth* of C is that of the hypergraph (G, W'), where W' is $\{\{g, g'\} \mid (g, g') \in W\}$. Its *size* $|C|$ is the number of wires. The set C_{var} of *variable gates* of C are those of type var. Given a valuation ν of C_{var}, we extend it to an *evaluation* of C by mapping each variable $g \in C_{\mathsf{var}}$ to $\nu(g)$, and evaluating the other gates according to their type. We recall the convention that $\wedge$-gates (resp., $\vee$-gates) with no input evaluate to 1 (resp., 0). The Boolean function on C_{var} *captured* (or *computed*, or *represented*) by the circuit is the one that maps ν to the evaluation of g_{output} under ν. Two circuits are *equivalent* if they capture the same function.

DNFs and CNFs We also study other representations of Boolean functions, namely, Boolean formulas in *conjunctive normal form* (*CNFs*) and in *disjunctive normal form* (*DNFs*). A CNF (resp., DNF) φ on a set of variables V is a conjunction (resp., disjunction) of *clauses*, each of which is a disjunction (resp., conjunction) of *literals* on V, i.e., variables of V (a *positive* literal) or their negation (a *negative* literal).

A *monotone CNF* (resp., monotone DNF) is one where all literals are positive, in which case we often identify a clause to the set of variables that it contains. We always assume that monotone CNFs and monotone DNFs are *minimized*, i.e., no clause is a subset of another. This ensures that every monotone Boolean function has a unique representation as a monotone CNF (the disjunction of its prime implicants), and likewise for monotone DNF. In addition, when we consider a valuation ν of a subset X of the variables of a monotone CNF/DNF φ, we always take the Boolean function $\nu(\varphi)$ to be the equivalent minimized monotone CNF/DNF.

We assume that monotone CNFs and DNFs always contain at least one non-empty clause (in particular, they cannot represent constant functions). Monotone CNFs and DNFs φ are isomorphic to hypergraphs: the vertices are the variables of φ, and the hyperedges are the clauses of φ. We often identify φ with its hypergraph. In particular, the *pathwidth* $pw(\varphi)$ and *treewidth* $\mathrm{tw}(\varphi)$ of φ, and its *arity* and *degree*, are defined as that of its hypergraph.

Observe that there is a connection between the treewidth and pathwidth of a monotone CNF or DNF with the treewidth and pathwidth of the natural Boolean circuit computing it. Namely:

Observation 1 *For any monotone CNF or DNF formula φ, there is a circuit C_φ capturing φ such that* $\mathrm{tw}(C_\varphi) \leqslant \mathrm{tw}(\varphi) + 2$ *and* $\mathrm{pw}(C_\varphi) \leqslant \mathrm{pw}(\varphi) + 2$.

Proof Fix φ and define C_φ as follows if φ is a CNF: C_φ has one input gate i_x for each variable x of φ, one $\vee$-gate v_K for each clause K of φ whose inputs are the i_x for each variable x of K, and one $\wedge$-gate o which is the output gate of C_φ and whose inputs are all the v_K. If φ is a DNF we replace $\vee$-gates by $\wedge$-gates and vice-versa.

We claim that $\mathrm{tw}(C_\varphi) \leqslant \mathrm{tw}(\varphi) + 2$. Indeed, given a tree decomposition T of the hypergraph of φ of width $\mathrm{tw}(\varphi)$, we can construct a tree decomposition T' of C_φ as follows. First, we replace each bag b of T in T' by $b' = \{i_x \mid x \in b\} \cup \{o\}$. Second, for every clause K, we consider a bag $\beta(K)$ of T containing all variables of K (which must exist because T is a tree decomposition of φ): we modify T' to make β injective by creating sufficiently many copies of each bag in the image of β and connecting them to the original bag. Third, we add v_K to the domain of $\beta(K)$ for each clause K.

The connectedness of T' follows by the connectedness of T and by the fact that for each clause K, v_K appears only once in T' and o appears in every bag of T'. Moreover, every edge of C_φ is covered by some bag of T': indeed, both the edges of the form (o, v_K) and the edges of the form (v_K, i_x) are covered in the (only) bag of T' containing v_K. It is then immediate that T' has the prescribed width.

The proof for pathwidth is the same, because if T is a path then T' can also be constructed to be a path. □

Note that there is no obvious converse to this result, e.g., the family of single-clause CNFs $x_1 \vee \cdots \vee x_n$ has unbounded treewidth but can be captured by a circuit of treewidth 1. A finer connection can be made by considering the *incidence treewidth* [50] of CNFs and DNFs, but we do not investigate this in this work.

3 Knowledge Compilation Classes: BDDs and DNNFs

We now review some representation formalisms for Boolean functions that are used in knowledge compilation, based either on *binary decision diagrams* (also known as *branching programs* [52]) or on Boolean circuits in *decomposable negation normal form* [25]; in the rest of the paper we will study translations between bounded-width Boolean circuits and these classes. The classes that we consider have all been introduced in the literature (see, in particular, [29] for the main ones) but we sometimes give slightly different (but equivalent) definitions in order to see them in a common framework. An element C of a knowledge compilation class $\mathcal{C}$ is associated with its *size* $|C|$ (describing how compact it is) and with the Boolean function φ that it *captures*.

A summary of the classes considered is provided in Fig. 1. This figure also shows (with arrows) when a class can be compiled into another in polynomial-time (i.e., when one can transform an element C of class $\mathcal{C}$ capturing a Boolean function φ into C' of class $\mathcal{C}'$ capturing φ, in time polynomial in $|C|$). All classes shown in Fig. 1 are unconditionally *separated* and some (cf. double arrows) are *exponentially separated*. Specifically, we say that a class $\mathcal{C}$ is separated (resp., exponentially separated) from a class $\mathcal{C}'$ if there exists a family of Boolean functions (φ_k) captured by elements (C_k) of class $\mathcal{C}$ such that all families of class $\mathcal{C}'$ capturing (φ_k) have size $\Omega(|C_k|^\alpha)$ for all $\alpha \geq 0$ (resp., of size $\Omega(2^{|C_k|^\alpha})$ for some $\alpha > 0$).

We first consider general classes, then *structured* variants of these classes, and further introduce the notion of *width* of these structured classes. When introducing classes of interest, we recall or prove non-trivial polynomial-time compilation and separation results related to that class.

3.1 Unstructured Classes

We start by defining general, unstructured classes, i.e., those in the background of Fig. 1, namely (non-deterministic) free binary decision diagrams and circuits in decomposable negation normal form.

3.1.1 Free Binary Decision Diagrams

A *non-deterministic binary decision diagram* (or *nBDD*) on a set of variables $V = \{v_1, \ldots, v_n\}$ is a rooted DAG D with labels on edges and nodes, verifying:

(i) there are exactly two leaves (also called *sinks*), one being labeled by 0 (the *0-sink*), the other one by 1 (the *1-sink*);
(ii) internal nodes are labeled either by $\vee$ or by a variable of V;
(iii) each internal node that is labeled by a variable has two outgoing edges, labeled 0 and 1.

The *size* $|D|$ of D is its number of edges. Let ν be a valuation of V, and let π be a path in D from the root to a sink of D. We say that π is *compatible* with ν if for every node n of π that is labeled by a variable x of V, the path π goes through the outgoing edge of n labeled by $\nu(x)$. Observe that multiple paths might be compatible with ν, because no condition is imposed on nodes of the path that are labeled by $\vee$; in other words, the behaviour at $\vee$ nodes is non-deterministic. An nBDD D *captures* a Boolean function φ on V defined as follows: for every valuation ν of V, if there exists a path π from the root to the 1-sink that is compatible with ν, then $\varphi(\nu) = 1$, else $\varphi(\nu) = 0$. An nBDD is *unambiguous* when, for every valuation ν, there exists at most one path φ from the root to the 1-sink that is compatible with ν. A *BDD* is an nBDD that has no $\vee$-nodes.

The most general form of nBDDs that we will consider in this paper are *non-deterministic free binary decision diagrams* (*nFBDDs*): they are nBDDs such that for every path from the root to a leaf, no two nodes of that path are labeled by the same variable. In addition to the nFBDD class, we will also study the class *uFBDD* of unambiguous nFBDDs, and the class *FBDD* of nFBDDs having no $\vee$-nodes.

Proposition 3.1 *nFBDDs are exponentially separated from uFBDDs, and uFBDDs are exponentially separated from FBDDs.*

Proof The exponential separation between nFBDDs and uFBDDs is shown in [14]: Proposition 7 of [14] shows that there exists an nFBDD of size $O(n^2)$ for the Sauerhoff function [46] over n^2 variables, while Theorem 9 of [14], relying on [46, Theorem 4.10], shows that any representation of this function as a d-DNNF (a formalism that generalizes uFBDD, see our Proposition 3.4) necessarily has size $2^{\Omega(n)}$.

To separate uFBDDs from FBDDs, we rely on the proof of exponential separation of PBDDs and FBDDs in [12, Theorem 11] (see also [52, Theorem 10.4.7]). Consider the Boolean function φ_n on n^2 variables that tests whether, in an $n \times n$ Boolean matrix, either the number of 1's is odd and there is a row full of 1's, or the number of 1's is even and there is a column full of 1's. As shown in [12, 52], an FBDD for φ_n has necessarily size $2^{\Omega(n^{1/2})}$. On the other hand, it is easy to construct an FBDD of size $O\left(n^2\right)$ to test if the number of 1's is odd and there is a row full of 1's (enumerating variables in row order), and to construct an FBDD of size $O(n^2)$ to test if the number of 1's is even and there is a column full of 1's (enumerating variables in column order). An uFBDD for φ_n is then obtained by simply adding an $\vee$-gate joining these two FBDDs, since only one of these two functions can evaluate to 1 under a given valuation. □

3.1.2 Decomposable Negation Normal Forms

We say that a circuit C is in *negation normal form* (NNF) if the inputs of $\neg$-gates are always variable gates. For a gate g in a Boolean circuit C, we write $\mathsf{Vars}(g)$ for the set of variable gates that have a directed path to g in C. An $\wedge$-gate g of C is *decomposable* if it has at most two inputs and if, in case it has two input gates $g_1 \neq g_2$, then we have $\mathsf{Vars}(g_1) \cap \mathsf{Vars}(g_2) = \emptyset$. We call C *decomposable* if each $\wedge$-gate is. We write DNNF for an NNF that is decomposable. Some of our proofs will use the standard notion of a *trace* in an NNF:

Definition 3.2 Let C be an DNNF and g be a gate of C. A *trace of C starting at g* is a set Ξ of gates of C that is minimal by inclusion and where:

- We have $g \in \Xi$;
- If $g' \in \Xi$ and g' is an $\wedge$-gate, then all inputs of g' are in Ξ, i.e., $W(g') \subseteq \Xi$;
- If $g' \in \Xi$ and g' is an $\vee$-gate, then exactly one input of g' is in Ξ;
- If $g' \in \Xi$ and g' if a $\neg$-gate with input variable gate g'', then g'' is in Ξ.

Observe that a gate $g \in C$ is *satisfiable* (i.e., there exists a valuation ν such that g evaluates to 1 under ν) if and only if there exists a trace of C starting at g. Indeed, given such a trace, define the valuation ν that maps to 0 all the variables x such that a $\neg$-gate with input x is in Ξ, and to 1 all the other variables: this valuation clearly satisfies g, noting in particular that each variable occurs at most once in Ξ thanks to decomposability. Conversely, when g is satisfiable, it is clear that one can obtain

a trace starting at g whose literals (variable gates, and negations of the variables that are an input to a $\neg$-gate) evaluate to 1 under the witnessing valuation ν. This means that we can check in linear time whether a DNNF is *satisfiable*, i.e., if it has an accepting valuation, by computing bottom-up the set of gates at which a trace starts.

As we will later see, the tractability of satisfiability of DNNFs does not extend to some other tasks (e.g., model counting or probability computation). For these tasks, a useful additional requirement on circuits is *determinism*. An $\vee$-gate g of C is *deterministic* if there is no pair $g_1 \neq g_2$ of input gates of g and valuation ν of C_{var} such that g_1 and g_2 both evaluate to 1 under ν. A Boolean circuit is *deterministic* if each $\vee$-gate is. We write d-DNNF for an NNF that is both decomposable and deterministic. Model counting and probability computation can be done in linear time for d-DNNFs thanks to decomposability and determinism (in fact this does not even use the restriction of being an NNF).

Observe that, while decomposability is a syntactical restriction that can be checked in linear time, the determinism property is semantic, and it is co-NP-complete to check if a given $\vee$-gate of a circuit is deterministic: hardness comes from the fact that an arbitrary Boolean circuit C is unsatisfiable iff $\vee(C, 1)$ is deterministic. This motivates the notion of *decision* gates, which gives us a syntactic way to impose determinism. Formally, an $\vee$-gate is a *decision* gate if it is of the form $(x \wedge C_1) \vee (\neg x \wedge C_2)$, for some variable x and (generally non-disjoint) subcircuits C_1, C_2. A *dec-DNNF* is a DNNF where all $\vee$-gates are decision gates: it is in particular a d-DNNF.

Proposition 3.3 *DNNFs are exponentially separated from d-DNNFs, and d-DNNFs are exponentially separated from dec-DNNFs.*

Proof The exponential separation of DNNFs and d-DNNFs is in [14, Proposition 7 and Theorem 9], by a similar argument to the proof of our Proposition 3.1.

The exponential separation of d-DNNFs and dec-DNNFs is in [7, Corollary 3.5]. □

3.1.3 Connections between FBDDs and DNNFs

We have presented our unstructured classes of decision diagrams (namely FBDDs, uFBDDs, and nFBDDs), and of decomposable NNF circuits (dec-DNNF, d-DNNF, and DNNF). We now discuss the relationship between these various classes. We first observe that nFBDDs (and their subtypes) can be compiled to DNNF (and their subtypes):

Proposition 3.4 *nFBDDs (resp., uFBDDs, FBDDs) can be compiled to DNNFs (resp., d-DNNFs, dec-DNNFs) in linear time.*

Proof We first describe the linear-time compilation of an nFBDD to a DNNF that captures the same function: recursively rewrite every internal node n labeled with variable x by a circuit $(x \wedge D_0) \vee (\neg x \wedge D_1)$, where D_0 and D_1 are the (not necessarily

disjoint) rewritings of the nodes to which n respectively had a 0-edge and a 1-edge. We note that the new $\vee$-gate is a decision gate and the two $\wedge$-gates are decomposable. Furthermore:

- if D is unambiguous, all $\vee$-gates in the rewriting are deterministic, so we obtain a d-DNNF;
- if D is an FBDD, then $\vee$-gates are only introduced in the rewriting, so we obtain a dec-DNNF. □

The proof above implies that nFBDDs (resp., uFBDDs, FBDDs) are the restriction of DNNFs (resp., d-DNNFs, dec-DNNFs) to the case where $\wedge$-gates, in addition to being decomposable, are also all *decision* $\wedge$-gates, i.e., $\wedge$-gates appearing as children of a decision $\vee$-gate.

Unlike previous compilation results, Proposition 3.4 does not come with an exponential separation: we can compile in the other direction at a *quasi-polynomial* cost, i.e., in time $2^{O((\log n)^{\alpha})}$ for some fixed $\alpha > 0$:

Proposition 3.5 *DNNFs (resp., d-DNNFs, dec-DNNFs) can be compiled to nFBDDs (resp., uFBDDs, FBDDs) in quasi-polynomial time.*

Proof Quasi-polynomial compilation has been first shown for dec-DNNFs and FBDDs in in [6, Corollary 3.2]. This result was extended in [8, Section 5] to the compilation of DNNFs to nFBDDs. Finally, in [11, Proposition 1], it is shown that the same compilation yields a uFBDD when applied to a d-DNNF. □

We will see in Proposition 3.10 that these quasi-polynomial time compilations cannot be made polynomial-time, which will conclude the separation of all classes in the background of Fig. 1. We now move to structured classes, that are in the foreground of Fig. 1.

3.2 Structured Classes

The classes introduced so far are *unstructured*: there is no particular order or structure in the way variables appear within an nFBDD, or within a DNNF circuit. In this section, we introduce *structured* variants of these classes, which impose additional constraints on how variables are used. Such additional restrictions often help with the tractability of some operations: for example, given two FBDDs F_1, F_2 capturing Boolean functions φ_1, φ_2, it is NP-hard to decide if $\varphi_1 \wedge \varphi_2$ if satisfiable [37, Lemma 8.14]. By contrast, with the *ordered binary decision diagrams* [17] (OBDDs) that we now define, we can perform this task tractably: given two OBDDs O_1 and O_2 that are ordered in the same way, we can compute in polynomial time an OBDD representing $O_1 \wedge O_2$, for which we can then decide satisfiability. We first present OBDDs, and we then present *SDNNFs* which are the structured analogues of DNNF.

3.2.1 Ordered Binary Decision Diagrams

A *non-deterministic ordered binary decision diagram* (*nOBDD*) is an nFBDD O with a total order $\mathbf{v} = v_{i_1}, \ldots, v_{i_n}$ on the variables which *structures* O, i.e., for every path π from the root of O to a leaf, the sequence of variables which labels the internal nodes of π (ignoring $\vee$-nodes) is a subsequence of $\mathbf{v}$. We say that the nOBDD is *structured by* $\mathbf{v}$. We also define *uOBDDs* as the unambiguous nOBDDs, and *OBDDs* as the nOBDDs without any $\vee$-node.

Like in the unstructured case (Proposition 3.1), these classes are exponentially separated:

Proposition 3.6 *nOBDDs are exponentially separated from uOBDDs, and uOBDDs are exponentially separated from OBDDs.*

Proof The exponential separation between nOBDDs and uOBDDs will follow from our lower bounds on uOBDDs. Indeed, Corollary 7.5 shows a lower bound on the size of uOBDDs representing bounded-degree and bounded-arity monotone DNFs of high pathwidth. But there exists a family of DNFs $(\varphi_n)_{n\in\mathbb{N}}$ of bounded degree and arity whose treewidth (hence pathwidth) is linear in their size: for instance, DNFs built from *expander graphs* (see [32, Theorem 5 and Proposition 1]). Hence, for such a family $(\varphi_n)_{n\in\mathbb{N}}$ we have that any uOBDD for φ_n is of size $2^{\Omega(|\varphi_n|)}$. By contrast, it is easy to see that any DNF φ can be represented as an nOBDD in linear time. To do so, fix an arbitrary variable order $\mathbf{v}$ of the variables of φ. Any clause K of φ can clearly be represented as a small OBDD with order $\mathbf{v}$. Taking the disjunction of all these OBDDs then yields an nOBDD equivalent to φ of linear size.

For the separation between uOBDDs and OBDDs, consider the *Hidden Weighted Bit* function HWB_n on variables $V = \{x_1, \ldots, x_n\}$, defined for a valuation ν of V by:

$$\mathrm{HWB}_n(\nu) := \begin{cases} 0 & \text{if } \sum_{i=1}^{n} \nu(x_i) = 0; \\ \nu(x_k) & \text{if } \sum_{i=1}^{n} \nu(x_i) = k \neq 0. \end{cases}$$

Bryant [17] showed that OBDDs for HWB_n have size $2^{\Omega(n)}$. By contrast, it is not too difficult to construct uOBDDs of polynomial size for HWB_n. This was observed in [52, Theorem 10.2.1] for nOBDDs, with a note [52, Proof of Corollary 10.2.2] that the constructed nBDDs are unambiguous. See also [13, Theorem 3], which covers the case of *sentential decision diagrams* instead of uOBDDs. □

3.2.2 Structured DNNFs

For NNFs, as with BDDs, it is possible to introduce a notion of *structuredness*, that goes beyond that of decomposability.

Here, the analogue of a total order of variables (that structured an OBDD) is what is called a *v-tree*, which is a tree whose leaves correspond to variables. More formally, a *v-tree* [40] over a set V is a rooted full binary tree T whose leaves are in bijection with V. We always identify each leaf with the associated element of V. We will also use the notion of an *extended v-tree* T [22] over a set V, which is like a v-tree,

except that there is only an injection between V and $\mathsf{Leaves}(T)$. That is, some leaves can correspond to no element of V: we call those leaves *unlabeled* (and they can intuitively stand for constant gates in the circuit).

Definition 3.7 A *structured DNNF* (resp., *extended structured DNNF*), denoted *SDNNF* (resp., *extended SDNNF*), is a triple (D, T, ρ) consisting of:

- a DNNF D;
- a v-tree (resp., extended v-tree) T over D_{var};
- a mapping ρ labeling each $\wedge$-gate of g with a node of T that satisfies the following: for every $\wedge$-gate g of D with $1 \leqslant m \leqslant 2$ inputs g_1, g_m, the node $\rho(g)$ *structures* g, i.e., there exist m distinct children n_1, n_m of $\rho(g)$ such that $\mathsf{Vars}(g_i) \subseteq \mathsf{Leaves}(T_{n_i})$ for all $1 \leqslant i \leqslant m$.

We also define d-SDNNF and dec-SDNNF as structured d-DNNF and dec-DNNF, and define extended d-SDNNF and extended dec-SDNNF in the expected way.

As in the case of FBDDs and DNNFs, observe that an OBDD (resp., uOBDD, nOBDD) is a special type of dec-SDNNF (resp., d-SDNNF, SDNNF). Namely, the transformation described above Proposition 3.4, when applied to an OBDD (resp., uOBDD, nOBDD), yields a dec-SDNNF (resp., d-SDNNF, SDNNF) that is structured by a v-tree that is right-linear (recall the definition from Section 2). Hence, we have:

Proposition 3.8 *nOBDDs (resp., uOBDDs, OBDDs) can be compiled to SDNNFs (resp., d-SDNNFs, dec-SDNNFs) in linear time.*

Proof Given the variable order $\mathbf{v} = v_1 \dots v_n$ of an nOBDD, we construct our right-linear v-tree T as having a root r_1, internal nodes r_i with r_i being the left child of r_{i-1} for $2 \leqslant i \leqslant n-1$, leaf nodes $v_1 \dots v_{n-1}$ with v_i being the right child of r_i, and leaf node v_n being the right child of r_{n-1}. We then apply as-is the translation described in the proof of Proposition 3.4. □

As in the unstructured case (Proposition 3.4), there is no exponential separation result: indeed, analogously to Proposition 3.5 in the unstructured case, there exist quasi-polynomial compilations in the other direction:

Proposition 3.9 *SDNNFs (resp., d-SDNNFs, dec-SDNNFs) can be compiled to nOBDDs (resp., uOBDDs, OBDDs) in quasi-polynomial time.*

Proof Quasi-polynomial time compilation of a SDNNF into an nOBDD is proved in [11, Theorem 2], by adapting the compilation of [8] from DNNFs to nFBDDs. Furthermore, [11, Proposition 2] shows that the resulting nOBDD is unambiguous if the SDNNF is deterministic. But it is easy to see that the same compilation [11, Simulation 2] yields an OBDD if the input is a dec-SDNNF: indeed, in a dec-SDNNF there are no $\vee$-gates that are not decision gates, so no $\vee$-gates are produced in the output. □

3.3 Comparing Structured and Unstructured Classes

To obtain all remaining separations in Fig. 1, and justify that no arrows are missing, we need two last results in which we will compare structured and unstructured classes.

The first result describes the power of decomposable $\wedge$-gates as opposed to decision gates: it shows that the least powerful class that has arbitrary decomposable $\wedge$-gates (dec-SDNNF) cannot be compiled to the most powerful class with decision $\wedge$-gates (nFBDD) without a super-polynomial size increase.

Proposition 3.10 *There exists a family of functions (φ_n) that has $O(n^2)$ dec-SDNNF but no nFBDD of size smaller than $n^{\Omega(\log(n))}$.*

Proof In [44], Razgon constructs for every k a family of 2CNF $(\varphi_n^k)_{n\in\mathbb{N}}$ such that φ_n^k has n variables and treewidth k. He proves ([44, Theorem 1]) a $n^{\Omega(k)}$ lower bound on the size of any nFBDD computing φ_n^k (Razgon refers to nFBDD as NROBP in his paper). It is known from [25, Section 3] that one can compile any CNF with n variables and with treewidth k into a dec-SDNNF of size $2^{\Omega(k)}n$. Thus, φ_n^k can be computed by a dec-SDNNF of size $2^{\Omega(k)}n$.

Taking $k := \lfloor \log(n) \rfloor$ gives the desired separation: φ_n^k can be computed by a dec-SDNNF of size $O(n^2)$ but by no nFBDD of size smaller than $n^{\Omega(\log(n))}$. □

Proposition 3.10 implies that no DNNF class in the upper level of Fig. 1 can be polynomially compiled into any BDD class in the lower level of Fig. 1.

The second result describes the power of unstructured formalisms as opposed to structured ones: it shows that the least powerful unstructured class (FBDD) cannot be compiled to the most powerful structured class (SDNNF) in size less than exponential.

Proposition 3.11 *FBDDs are exponentially separated from SDNNFs: there exists a family of functions (φ_n) that has FBDDs of size $O(n)$ but no SDNNF of size smaller than $2^{\Omega(n)}$.*

Proof This separation was proved independently by Pipatsrisawat and Capelli in their PhD theses (see [42, Appendix D.2], and [20, Section 6.3]).

In his work, Pipatsrisawat considers the Boolean function *circular bit shift* $CBS(S, X, Y)$: it is defined on a tuple (S, X, Y) of variables with $N = s_1 \dots s_k$, $X = x_1 \dots x_{2^k}$, $Y = y_1 \dots y_{2^k}$ for some $k \in \mathbb{N}$, and it evaluates to 1 on valuation ν iff shifting the bits of $\nu(X)$ by S (as written in binary) positions yields $\nu(Y)$. Pipatsrisawat shows that the CBS function on n variables has an FBDD of size $O(n^2)$, but that any SDNNF for CBS has size $2^{\Omega(n)}$.

The proof of Capelli uses techniques close to the ones used in Section 7. □

Proposition 3.11 implies that no unstructured class (in the background of Fig. 1) can be polynomially compiled into any structured class (in the foreground of Fig. 1).

Looking back at Fig. 1, we see that, indeed, all classes are separated and no arrows are missing. The separation is exponential except when moving (on the vertical axis

in the figure) from BDD-like classes to NNF-like classes, in which case we know (cf. Propositions 3.5 and 3.9) that quasi-polynomial compilations exist in the other direction.

3.4 Completeness and Width

Two last notions that will be useful for our results are the notions of *completeness* and *width* for structured classes. Intuitively, completeness further restricts the structure of how variables are tested in the circuit or BDD: in addition to the structuredness requirement, we impose that no variables are "skipped". We will be able to assume completeness without loss of generality, it will be guaranteed by our construction, and it will be useful in our lower bound proofs.

On complete classes, we will additionally be able to define a notion of *width* that we will use to show finer lower bounds.

Complete OBDDs An nOBDD O on V is *complete* if every path from the root to a sink tests every variable of V. For $x \in V$, the $x - width$ of a complete nOBDD O is the number of nodes labeled with variable x. The *width* of O is $\max_{x \in V} x$-width of O.

It is immediate that partially evaluating a complete nOBDD does not increase its width:

Lemma 3.12 *Let O be a complete nOBDD (resp., uOBDD) on variables V, with order $\mathbf{v}$ and of width $\leqslant w$, and let φ be the Boolean function that O captures. Let $X \subseteq V$, and ν be a valuation of X. Then there exists a complete nOBDD (resp., uOBDD) O', on variables $V \setminus X$, of order $\mathbf{v}|_{V \setminus X}$ and width $\leqslant w$, that computes $\nu(\varphi)$.*

Complete SDNNFs The notion of completeness and width of OBDDs extends naturally to SDNNFs. Following [22], we say that a (d-)SDNNF (resp., extended (d-)SDNNF) (D, T, ρ) is *complete* if ρ labels *every* gate of D (not just $\wedge$-gates) with a node of T and the following conditions are satisfied:

1. The output gate of D is an $\vee$-gate;
2. For every variable gate g of D, we have $\rho(g) = g$;
3. For every $\neg$-gate g of D, letting g' be the variable gate that feeds g, we have $\rho(g) = g'$;
4. For every $\vee$-gate g of D, for any input g' of g, the gate g' is not an $\vee$-gate, and moreover we have $\rho(g') = \rho(g)$;
5. For every $\wedge$-gate g of D, for any input g' of g, the gate g' is an $\vee$-gate, and we have that $\rho(g')$ is a child of $\rho(g)$;
6. For every $\wedge$-gate g of D and any two inputs $g' \neq g''$ of g, we have $\rho(g') \neq \rho(g'')$;
7. For every $\wedge$-gate g of D such that $\rho(g)$ is an internal node of T, g has exactly two inputs.

For a node n of T, the $n - width$ of a complete (d-)SDNNF (resp., extended complete) (D, T, ρ) is the number of $\vee$-gates that are structured by n. The *width* of (D, T, ρ) is the maximal n-width for a node of T.

One of the advantages of complete (d-)SDNNFs of bounded width is that we can work with extended v-trees, and then compress their size in linear time, so that the v-tree becomes non-extended and the size of the circuit is linear in the number of variables. When doing so, the extended v-tree is modified in a way that we call a *reduction*:

Definition 3.13 Let T, T' be two extended v-trees over variables V. We say that T' is a *reduction of* T if, for every internal node n of T, there exists an internal node n' of T' such that $\mathsf{Leaves}(T') \cap \mathsf{Leaves}(T \setminus T_n) \subseteq \mathsf{Leaves}(T' \setminus T'_{n'})$ and $\mathsf{Leaves}(T') \cap \mathsf{Leaves}(T_n) \subseteq \mathsf{Leaves}(T'_{n'})$.

We can now show how to compress extended complete (d-)SDNNFs:

Lemma 3.14 ([22]) *Let (D, T, ρ) be an extended complete (d-)SDNNF of width w on n variables. We can compute in linear time a complete (d-)SDNNF (D', T', ρ') of width w such that T' is a reduction of T and such that $|D'|$ is in $O(n \times w^2)$.*

Proof We present a complete proof, inspired by the proof in [22, Lemma 4]. As a first prerequisite, we preprocess D in linear time so that the number of $\wedge$-gates structured by a same node n of T is in $O(w^2)$. This can be done, as in [22, Observation 3], by noticing that there can be at most w^2 inequivalent $\wedge$-gates that are structured by a node n. Indeed, this is clear if n is a leaf, as such an $\wedge$-gate cannot have an input (so there is at most one inequivalent $\wedge$-gate). If n is an internal node with children n_1 and n_2, then, by item (7) of the definition of being an extended complete SDNNF, every $\wedge$-gate structured by n has one input among the $\leqslant w$ $\vee$-gates structured by n_1, and one input among the $\leqslant w$ $\vee$-gates structured by n_2. Hence there are w^2 possible inequivalent $\wedge$-gates. We can then merge all the $\wedge$-gates that are equivalent, and obtain a complete (d-)SDNNF where for each node n of the v-tree, at most w^2 $\wedge$-gates are structured by n.

The second prerequisite is to eliminate the gates that are not connected to the output of D, and then to propagate the constants in the circuit (i.e., to evaluate it partially). In other words, eliminate all gates (and their wires) that are not connected to the output of D, and then repeat the following until convergence:

- For every constant 1-gate g (i.e., an $\wedge$-gate with no input) and wire $g \to g'$, if g' is an $\wedge$-gate then simply remove the wire $g \to g'$, and if g' is an $\vee$-gate, then replace g by a constant 1-gate; then remove g and all the wires connected to it.
- For every constant 0-gate g (i.e., an $\vee$-gate with no input) and wire $g \to g'$, if g' is an $\vee$-gate then simply remove the wire $g \to g'$, and if g' is an $\wedge$-gate, then replace g by a constant 0-gate; then remove g and all the wires connected to it.

This again can be done in linear time (by a DFS traversal of the circuit, for instance), and it does not change the properties of the circuit or the captured function. Further, it

ensures that $\vee$- and $\wedge$-gates of the resulting circuit always have at least one input, or that we get to one single constant gate (if the circuit captures a constant Boolean function): as this second case of constant functions is uninteresting, we assume that we are in the first case. We call the resulting circuit C. It is clear that C is still structured by T (by taking the restriction of ρ to the gates that have not been removed).

Having enforced these prerequisites on C, the idea is to eliminate unlabeled leaves l in he v-tree one by one by merging the parent and the sibling of l. Formally, whenever we can find in T an unlabeled leaf l with parent n and sibling n', we perform these two steps:

1. Remove from T the leaf l (and its parent edge) noticing that no gate of C was structured by l because we propagated the constants in the circuit in our second preprocessing step; then replace the parent n in T by its remaining child n' so that it is again binary and full.
2. We now need to modify C so that C is an extended complete (d-)SDNNF structured by the new v-tree. There is nothing to do in the case that n' was an unlabeled leaf, because then no gate of C was structured by n', or even by n (since we propagated constants). In the case where n' was a variable leaf or an internal node, then, for every $\vee$-gate g that was structured by n, we compute the set I_g of gates g' that were structured by n', that are not an $\vee$-gate, and such that there is a path from g' to g in C. Thanks to our first preprocessing step, the set I_g can be computed in time $O(w^2)$ as this bounds the number of $\wedge$-gates structured by n and by n'. Observe that gates in I_g can be either $\wedge$-gates that were structured by n' (in case n' was an internal node), or $\neg$-gates or variable gates (in case n' was a variable leaf). Now, remove from C all the $\vee$-gates that were structured by n', all the $\wedge$-gates that were structured by n, and all the edges connected to them. For each $\vee$-gate g that was structured by n, set its new inputs to be all the gates in I_g. One can check that the resulting circuit captures the same function (this uses the fact that we propagated constants), and that determinism cannot be broken in case the original circuit was a d-SDNNF. Moreover, C is now an extended complete (d-)SDNNF structured by the new v-tree T.

By iterating this process, we will end up with a v-tree T' that is not extended, and the resulting circuit will be an equivalent complete (d-)SDNNF of width $\leqslant w$ and size $O(n \times w^2)$. The total time is linear since we spend $O(w^2)$ time to eliminate each single unlabeled leaf. Moreover it is clear that the final v-tree obtained is a reduction of the original v-tree, as the property is preserved by each elimination. □

Like for OBDDs (Lemma 3.12), we will use the fact that partially evaluating a complete (d-)SDNNF cannot increase the width:

Lemma 3.15 *Let (D, T, ρ) be a complete (d-)SDNNF of width $\leqslant w$ over variables V, and let φ be the Boolean function that D captures. Let $X \subseteq V$, and ν be a valuation of X. Then there exists a complete (d-)SDNNF (D', T', ρ') of width $\leqslant w$ on variables $V \setminus X$ computing $\nu(\varphi)$ such that T' is a reduction of T.*

Proof We replace every leaf l of T that corresponds to a variable $x \in X$ by an unlabeled leaf, replace every variable gate x in D by a constant $\nu(x)$-gate, replace every $\neg$-gate with input variable x by a constant $(1 - \nu(x))$-gate, and then propagate constants as in the second prerequisite in the proof of Lemma 3.14. This yields an *extended* complete (d-)SDNNF computing $\nu(\varphi)$. We then conclude by applying Lemma 3.14. □

Making nOBDDs and SDNNFs Complete Imposing completeness on nOBDDs or SDNNFs is in fact not too restrictive, as we can assume that OBDDs and SDNNFs are complete up to multiplying the size by the number of variables:

Lemma 3.16 *For any nOBDD (resp., SDNNF) D on variables V, there exists an equivalent complete nOBDD (resp., SDNNF) of size at most $(|V| + 1) \times |D|$.*

Proof The result will follow from a more general completion result on unstructured classes given later in the paper (Lemma 8.4); it is straightforward to observe that applying the constructions of that lemma yield structured outputs when the input representations are themselves structured. □

4 Upper Bound

In this section we study how to compile Boolean circuits to d-SDNNFs (resp., uOBDDs), parameterized by the treewidth (resp., pathwidth) of the input circuits. We first present our results in Section 4.1, then show some examples of applications in Section 4.2, before providing full proofs in Section 5.

4.1 Results

To present our upper bounds, we first review the independent result that was recently shown by Bova and Szeider [16] on compiling bounded-treewidth circuits to d-SDNNFs:

Theorem 4.1 ([16, Theorem 3 and Equation (22)]) *Given a Boolean circuit C of treewidth $\leqslant k$, there exists an equivalent d-SDNNF of size $O(f(k) \times |C_{\mathsf{var}}|)$, where f is doubly exponential.*

Their result has two drawbacks: (i) it has a doubly exponential dependency on the width; and (ii) it is nonconstructive, because [16] gives no time bound on the computation, leaving open the question of effectively compiling bounded-treewidth circuits to d-SDNNFs. The nonconstructive aspect can easily be tackled by encoding in linear time the input circuit C into a relational instance of same treewidth, and then use [4, Theorem 6.11] to construct in linear time a d-SDNNF representation of the provenance on I of a fixed MSO formula describing how to evaluate Boolean circuits (see the conference version of this paper [5] for more details). This "naïve" approach computes a d-SDNNF in time $O(|C| \times f(k))$, but where f is a superexponential

function that does not address the first drawback. We show that we can get f to be *singly* exponential.

Treewidth Bound Our main upper bound result addresses both drawbacks and shows that we can compile in time linear in the circuit and singly exponential in the treewidth. Our proof is independent from [16]. Formally, we show:

Theorem 4.2 *There exists a function $f(k)$ that is in $O\left(2^{(4+\epsilon)k}\right)$ for any $\epsilon \geq 0$ such that the following holds. Given as input a Boolean circuit C and tree decomposition T of width $\leqslant k$ of C, we can compute a complete extended d-SDNNF equivalent to C of width $\leqslant 2^{2(k+1)}$ in time $O\left(|T| \times f(k)\right)$.*

This result assumes that the tree decomposition is provided as input; but we can instead use Theorem 2.1 to obtain it. We can also apply Lemma 3.14 to the resulting circuit to get a proper (non-extended) d-SDNNF and reduce its size so that it only depends on the number of variables of the input circuit C (i.e., $|C_{\mathsf{var}}|$ rather than $|C|$), which allows us to truly generalize Theorem 4.1. Putting all of this together, we get:

Corollary 4.3 *There exists a constant $c \in \mathbb{N}$ such that the following holds. Given as input a Boolean circuit C of treewidth $\leqslant k$, we can compute in time $O\left(|C| \times 2^{ck}\right)$ a complete d-SDNNF equivalent to C of width $O\left(2^{ck}\right)$ and size $O\left(|C_{\mathsf{var}}| \times 2^{ck}\right)$.*

However, Corollary 4.3 is mainly of theoretical interest, since the constant hidden in Theorem 2.1 is huge. In practice, one would first use a heuristic to compute a tree decomposition, and then use our construction of Theorem 4.2 on that decomposition. We will prove Theorem 4.2 in Section 5.

Pathwidth Bound A by-product of our construction is that, in the special case where we start with a path decomposition, it turns out that the d-SDNNF computed is in fact an uOBDD. The compilation of bounded-pathwidth Boolean circuits to OBDDs had already been studied in [4, 34]: Corollary 2.13 of [34] shows that a circuit of pathwidth $\leqslant k$ has an equivalent OBDD of width $\leqslant 2^{(k+2)2^{k+2}}$, and [4, Lemma 6.9] justifies that the transformation can be made in polynomial time. Our second upper bound result is that, by using uOBDDs instead of OBDDs, we can get a singly exponential dependency:

Theorem 4.4 *There exists a function $f(k)$ that is in $O(2^{(2+\epsilon)k})$ for any $\epsilon \geq 0$ such that the following holds. Given as input a Boolean circuit C and path decomposition P of width $\leqslant k$ of C, we can compute a complete uOBDD equivalent to C of width $\leqslant 2^{2(k+1)}$ in time $O\left(|P| \times f(k)\right)$.*

While we do not know if the doubly exponential dependence on k in [34] is tight for OBDDs, we will show in Section 7 that the singly exponential dependence for uOBDDs is indeed tight.

4.2 Applications

Theorem 4.2 implies several consequences for bounded-treewidth circuits. The first one deals with *probability computation*: we are given a *probability valuation* π mapping each variable $g \in C_{\mathsf{var}}$ to a probability that g is true (independently from other variables), and we wish to compute the probability $\pi(C)$ that C evaluates to true under π, assuming that arithmetic operations (sum and product) take unit time. More formally, we define the *probability* $\pi(\nu)$ of a valuation $\nu : C_{\mathsf{var}} \to \{0, 1\}$ as

$$\pi(\nu) := \left(\prod_{g \in C_{\mathsf{var}}, \nu(g)=1} \pi(g) \right) \left(\prod_{g \in C_{\mathsf{var}}, \nu(g)=0} (1 - \pi(g)) \right).$$

The *probability* $\pi(C)$ *of Boolean circuit* C *with probability assignment* π is then the total probability of the valuations that satisfy φ. Formally:

$$\pi(C) := \sum_{\nu \text{ satisfies } \varphi} \pi(\nu).$$

When $\pi(x) = 1/2$ for every variable, the probability computation problem simplifies to the *model counting problem*, i.e., counting the number of satisfying valuations, noted $\#C$. Indeed, in this case we have $\#C = 2^{|C_{\mathsf{var}}|} \times \pi(C)$. Hence, the probability computation problem is #P-hard for arbitrary circuits. However, it is tractable for deterministic decomposable circuits [26]. Thus, our result implies the following, where $|\pi|$ denotes the size of writing the probability valuation π:

Corollary 4.5 *Let $f(k)$ be the function from Theorem 4.2. Given a Boolean circuit C, a tree decomposition T of width $\leqslant k$ of C, and a probability valuation π of C_{var}, we can compute $\pi(C)$ in $O(|\pi| + |T| \times f(k))$.*

Proof Use Theorem 4.2 to compute an equivalent d-SDNNF (D, T', ρ); as C and D are equivalent, it is clear that $\pi(C) = \pi(D)$. Now, compute the probability $\pi(D)$ in linear time in D and $|\pi|$ by a simple bottom-up pass, using the fact that D is a d-DNNF [26]. □

This improves the bound obtained when applying message passing techniques [36] directly on the bounded-treewidth input circuit (as presented, e.g., in [3, Theorem D.2]). Indeed, message passing applies to *moralized* representations of the input: for each gate, the tree decomposition must contain a bag containing all inputs of this gate *simultaneously*, which is problematic for circuits of large fan-in. Indeed, if the original circuit has a tree decomposition of width k, rewriting it to make it moralized will result in a tree decomposition of width $3k^2$ (see [2, Lemmas 53 and 55]), and the bound of [3, Theorem D.2] then yields an overall complexity of

$O\left(|\pi| + |T| \times 2^{3k^2}\right)$ for message passing. Our Corollary 4.5 achieves a more favorable bound because Theorem 4.2 directly uses the associativity of $\wedge$ and $\vee$. We note that the connection between message-passing techniques and structured circuits has also been investigated by Darwiche, but his construction [27, Theorem 6] produces arithmetic circuits rather than d-DNNFs, and it also needs the input to be moralized.

A second consequence concerns the task of *enumerating* the accepting valuations of circuits, i.e., producing them one after the other, with small *delay* between each accepting valuation. The valuations are concisely represented as *assignments*, i.e., as a set of variables that are set to true, omitting those that are set to false. This task is of course NP-hard on arbitrary circuits (as it implies that we can check whether an accepting valuation exists), but was recently shown in [1] to be feasible on d-SDNNFs with linear-time preprocessing and delay linear in the Hamming weight of each produced assignment. Hence, we have:

Corollary 4.6 *Let $f(k)$ be the function from Theorem 4.2. Given a Boolean circuit C and a tree decomposition T of width $\leqslant k$ of C, we can enumerate the satisfying assignments of C with preprocessing in $O(|T| \times f(k))$ and delay linear in the size of each produced assignment.*

Proof Use Theorem 4.2 to compute an equivalent d-SDNNF (D, T', ρ), which has the same accepting valuations. We conclude using [1, Theorem 2.1]. □

This corollary refines some existing results about enumerating the satisfying valuations of some circuit classes with *polynomial* delay [29], and also relates to results on the enumeration of monomials of arithmetic circuits [48] or of solutions to constraint satisfaction problems (CSP) [24], again with polynomial delay. It also relates to recent incomparable results on constant-delay enumeration for classes of DNF formulae [23].

A third consequence concerns the tractability of *quantifying variables* in bounded-treewidth circuits. Let φ be a Boolean function on variables V, and let $X_1, \ldots X_n$ be disjoint subsets of V. A *quantifier prefix* of length n is a prefix of the form $\Pi := Q_1 X_1 \ldots Q_n X_n$, where each Q_i is either $\exists$ or $\forall$, with $Q_i \neq Q_{i+1}$. Let $\Pi(\varphi)$ be the Boolean function on variables $V \setminus (X_1 \cup \cdots \cup X_n)$, with the obvious semantics. Then [22] shows:

Theorem 4.7 ([22, Theorem 5]) *There is an algorithm that, given a complete SDNNF (D, T, ρ) on variables V of width k and $Z \subseteq V$, computes in time $2^{O(k)}|D|$ a complete d-SDNNF of width at most 2^k having a designated gate computing $\exists Z\, D$ and another designated gate computing $\neg\exists Z\, D$.*

By iterating the construction of Theorem 4.7 and using the identity $\forall X.D \equiv \neg\exists X \neg D$, one can easily get:

Corollary 4.8 *Let $\Pi := Q_1 X_1 \ldots Q_n X_n$ be a quantifier prefix of length n with $Q_n = \exists$. There is an algorithm that, given a complete d-SDNNF (D, T, ρ),*

computes in time $\exp^n(k) \times |D|$ *a complete structured d-SDNNF of width* $\leqslant \exp^n(k)$ *representing* $\Pi(D)$*, where* $\exp^n(k) = \underbrace{2^{\cdot^{\cdot^{2^{O(k)}}}}}_{n}$.

We can then combine Corollary 4.8 with our Theorem 4.2 to show:

Corollary 4.9 *Let C be a Boolean circuit of treewidth $\leqslant k$, and let Π be a quantifier prefix of length n that ends with $\exists$. We can compute in time $\exp^{n+1}(k) \times |C|$ a d-SDNNF of width at most $\exp^{n+1}(k)$ representing $\Pi(C)$.*

This generalizes the corresponding result of [22], which applies to bounded-treewidth CNFs instead of bounded-treewidth circuits. (However, we note that [22, Theorem 10] also applies to CNFs of bounded *incidence* treewidth, which can be smaller than the primal treewidth that we use in our article.) We refer to [22] for a discussion of the related work on model counting of quantified formulas.

Other applications of Theorem 4.2 include counting the number of satisfying valuations of the circuit (a special case of probability computation), MAP inference [31], or random sampling of possible worlds (which can easily be done on the d-SDNNF).

5 Proof of the Upper Bound

We first present in Section 5.1 the construction used for Theorem 4.2, then prove in Section 5.2 that this construction is correct and can be done within the prescribed time bound. We then explain how to specialize the construction to the case of bounded-pathwidth circuits and uOBBDs in Section 5.3.

5.1 Construction

Let C be the input circuit on n variables, and T the input tree decomposition of C of width $\leqslant k$. We start with prerequisites.

Prerequisites Let g_{output} be the output gate of C. Thanks to Lemma 2.2, we can assume that T is g_{output}-friendly. For every variable gate $x \in C_{\mathsf{var}}$, we choose a leaf bag b_x of T such that $\lambda(b_x) = \{x\}$. Such a leaf bag exists because T is friendly (specifically, thanks to bullet points 2 and 3). We say that b_x is *responsible for the variable gate* x. We can obviously choose such a b_x for every variable gate x in linear time in T.

To abstract away the type of gates and their values in the construction, we will talk of *strong* and *weak* values. Intuitively, a value is *strong* for a gate g if any input g' of g which carries this value determines the value of g; and *weak* otherwise. Formally:

Definition 5.1 Let g be a gate and $c \in \{0, 1\}$:

- If g is an $\wedge$-gate, we say that $c = 0$ is *strong* for g and $c = 1$ is *weak* for g;
- If g is an $\vee$-gate, we say that $c = 1$ is *strong* for g and $c = 0$ is *weak* for g;

- If g is a $\neg$-gate, $c = 0$ and $c = 1$ are both *strong* for g;
- For technical convenience, if g is a var-gate, $c = 0$ and $c = 1$ are both *weak* for g.

If we take any valuation $\nu : C_{\mathsf{var}} \to \{0, 1\}$ of the circuit $C = (G, W, g_{\mathsf{output}}, \mu)$, and extend it to an evaluation $\nu : G \to \{0, 1\}$, then ν will respect the semantics of gates. In particular, it will *respect strong values*: for any gate g of C, if g has an input g' for which $\nu(g')$ is a strong value, then $\nu(g)$ is determined by $\nu(g')$, specifically, it is $\nu(g')$ if g is an $\vee$- or an $\wedge$-gate, and $1 - \nu(g')$ if g is a $\neg$-gate. In our construction, we will need to guess how gates of the circuit are evaluated, focusing on a subset of the gates (as given by a bag of T); we will then call *almost-evaluation* an assignment of these gates that respects strong values. Formally:

Definition 5.2 Let U be a set of gates of C. We call $\nu : U \to \{0, 1\}$ a (C, U)-*almost-evaluation* if it *respects strong values*, i.e., for every gate $g \in U$, if there is an input g' of g in U such that $\nu(g')$ is a strong value for g, then $\nu(g)$ is determined from $\nu(g')$ in the sense above.

Respecting strong values is a necessary condition for such an assignment to be extensible to a valuation of the entire circuit. However, it is not sufficient: an almost-evaluation ν may map a gate g to a strong value even though g has no input that can justify this value. This is hard to avoid: when we focus on the set U, we do not know about other inputs of g. For now, let us call *unjustified* the gates of U that carry a strong value that is not justified by ν:

Definition 5.3 Let U be a set of gates of a circuit C and ν a (C, U)-almost-evaluation. We call $g \in U$ *unjustified* if $\nu(g)$ is a strong value for g, but, for every input g' of g in U, the value $\nu(g')$ is weak for g; otherwise, g is *justified*. The set of unjustified gates is written $\mathrm{Unj}(\nu)$.

Let us start to explain in a high-level manner how to construct the d-SDNNF D equivalent to the input circuit C (we will later describe the construction formally). We do so by traversing T bottom-up, and for each bag b of T we create gates $G_b^{\nu,S}$ in D, where ν is a (C, b)-almost-evaluation and S is a subset of $\mathrm{Unj}(\nu)$ which we call the *suspicious gates* of $G_b^{\nu,S}$. We will connect the gates of D created for each internal bag b with the gates created for its children in T, in a way that we will explain later. Intuitively, for a gate $G_b^{\nu,S}$ of D, the *suspicious gates* g in the set S are gates of b whose strong value is not justified by ν (i.e., $g \in \mathrm{Unj}(\nu)$), and is not justified either by any of the almost-evaluations at descendant bags of b to which $G_b^{\nu,S}$ is connected. We call *innocent* the other gates of b; hence, they are the gates that are justified in ν (in particular, those who carry weak values), and the gates that are unjustified in ν but have been justified by an almost-evaluation at a descendant bag b' of b. Crucially, in the latter case, the gate g' justifying the strong value in b' may no longer appear in b, making g unjustified for ν; this is why we remember the set S.

We still have to explain how we connect the gates $G_b^{\nu,S}$ of D to the gates $G_{b_l}^{\nu_l,S_l}$ and $G_{b_r}^{\nu_r,S_r}$ created for the children b_l and b_r of b in T. The first condition is that ν_l and ν_r must *mutually agree*, i.e., $\nu_l(g) = \nu_r(g)$ for all $g \in b_l \cap b_r$, and ν must then be the union of ν_l and ν_r, restricted to b. We impose a second condition to prohibit suspicious gates from escaping before they have been justified, which we formalize as *connectibility* of a pair (ν, S) at bag b to the parent bag of b.

Definition 5.4 Let b be a non-root bag, b' its parent bag, and ν a (C, b)-almost-evaluation. For any set $S \subseteq \mathrm{Unj}(\nu)$, we say that (ν, S) is *connectible* to b' if $S \subseteq b'$, i.e., the suspicious gates of ν must still appear in b'.

If a gate $G_b^{\nu,S}$ is such that (ν, S) is not connectible to the parent bag b', then this gate will not be used as input to any other gate, but we do not try to preemptively remove these useless gates in the construction (but note that this will be taken care of at the end, when we will apply Lemma 3.14). We are now ready to give the formal definition that will be used to explain how gates are connected:

Definition 5.5 Let b be an internal bag with children b_l and b_r, let ν_l and ν_r be respectively (C, b_l) and (C, b_r)-almost-evaluations that mutually agree, and $S_l \subseteq \mathrm{Unj}(\nu_l)$ and $S_r \subseteq \mathrm{Unj}(\nu_r)$ be sets of suspicious gates such that both (ν_l, S_l) and (ν_r, S_r) are connectible to b. The *result* of (ν_l, S_l) and (ν_r, S_r) is then defined as the pair (ν, S) where:

- ν is defined as the restriction of $\nu_l \cup \nu_r$ to b.
- $S \subseteq \mathrm{Unj}(\nu)$ is the new set of suspicious gates, defined as follows. A gate $g \in b$ is innocent (i.e., $g \in b \setminus S$) if it is justified for ν or if it is innocent for some child. Formally, $b \setminus S := (b \setminus \mathrm{Unj}(\nu)) \cup [b \cap [(b_l \setminus S_l) \cup (b_r \setminus S_r)]]$.

We point out that (ν, S) is not necessarily a (C, b)-almost-evaluation.

Construction We now use these definitions to present the construction formally. For every variable gate g of C, we create a corresponding variable gate $G^{g,1}$ of D, and we create $G^{g,0} := \neg(G^{g,1})$. For every internal bag b of T, for each (C, b)-almost-evaluation ν and set $S \subseteq \mathrm{Unj}(\nu)$ of suspicious gates of ν, we create one $\vee$-gate $G_b^{\nu,S}$. For every leaf bag b of T, we create one $\vee$-gate $G_b^{\nu,S}$ for every (C, b)-almost-evaluation ν, where we set $S := \mathrm{Unj}(\nu)$; intuitively, in a leaf bag, an unjustified gate is always suspicious (it cannot have been justified at a descendant bag).

Now, for each internal bag b of T with children b_l, b_r, for each pair of gates $G_{b_l}^{\nu_l,S_l}$ and $G_{b_r}^{\nu_r,S_r}$ that are both connectible to b and where ν_l and ν_r mutually agree, letting (ν, S) be the result of (ν_l, S_l) and (ν_r, S_r), if (ν, S) is a (C, b)-almost-evaluation then we create a gate $G_b^{\nu_l,S_l,\nu_r,S_r} = \wedge\left(G_{b_l}^{\nu_l,S_l}, G_{b_r}^{\nu_r,S_r}\right)$ and make it an input of $G_b^{\nu,S}$. We now explain where the variables gates are connected. For every leaf bag b that is responsible for a variable gate x (i.e., b is b_x), for $\nu \in \{\{x \mapsto 1\}, \{x \mapsto 0\}\}$, we set the gate $G^{x,\nu(x)}$ to be the (only) input of the gate $G_b^{\nu,S}$. Last, for every leaf bag b that is not responsible for a variable gate, for every valuation ν of b, we create a constant

1-gate (i.e., an $\wedge$-gate with no inputs), and we make it the (only) input of $G_b^{\nu,S}$. The output gate of D is the gate $G_{b_{\text{root}}}^{\nu,\emptyset}$ where b_{root} is the root of T and ν maps g_{output} to 1 (remember that b_{root} contains only g_{output}).

We now construct the extended v-tree T' together with the mapping ρ. T' has the same skeleton than T (in particular, it is a full binary tree). For every node b of T, let us denote by b' the corresponding node of T'. For every leaf bag b of T, b' is either x if b is responsible for the variable x, or unlabeled otherwise. For every bag b of T and every gate g of the form $G_b^{\nu,S}$ or $G_b^{\nu_l,S_l,\nu_r,S_r}$, we take $\rho(g) = b'$. For every leaf bag b of T that is not responsible for a variable, for any gate g of the form $G_b^{\nu,S}$ (there can be either one (if b is empty) or two of them), letting g' be the (only) input of g (i.e., g' is a constant 1-gate), we set $\rho(g') = b'$. For every leaf bag b of T that is responsible for a variable x, we set $\rho(G^{x,1}) = \rho(G^{x,0}) = b'$.

5.2 Proof of Correctness

We now prove that (D, T', ρ) is indeed an extended complete d-SDNNF equivalent to the initial circuit C, that its width is $\leqslant 2^{2(k+1)}$, and that it can be constructed in time $O\left(|T| \times 2^{(4+\epsilon)k}\right)$ for any $\epsilon > 0$.

5.2.1 (D, T', ρ) is an Extended Complete SDNNF of the Right Width

Negations only apply to the input gates, so D is an NNF. It is easy to check that (D, T', ρ) satisfies the conditions of being a complete extended SDNNF. Now, for every leaf l of T', there at most two $\vee$-gates of D that are structured by l (remember that leaf bags of the friendly tree decomposition T contain one or zero gates of C). For every internal node n of T' corresponding to a bag b of T, the $\vee$-gates that are structured by n are of the form $G_b^{\nu,S}$, so they are at most $2^{2(k+1)}$, which shows our claim about the width of (D, T', ρ).

5.2.2 D is Equivalent to C

We now show that D is equivalent to the original circuit C. Recall the definition of a trace of a DNNF from Definition 3.2. Our first step is to prove that traces have exactly one almost-evaluation corresponding to each descendant bag, and that these almost-evaluations mutually agree.

Lemma 5.6 *Let $G_b^{\nu,S}$ a gate in D and Ξ be a trace of D starting at $G_b^{\nu,S}$. Then for any bag $b' \leqslant b$ (meaning that b' is b or a descendant of b), Ξ contains exactly one gate of the form $G_{b'}^{\nu',S'}$. Moreover, over all $b' \leqslant b$, all the almost-evaluations of the gates $G_{b'}^{\nu',S'}$ that are in Ξ mutually agree.*

Proof The fact that Ξ contains exactly one gate $G_{b'}^{\nu',S'}$ for any bag $b' \leqslant b$ is obvious by construction of D, as $\vee$-gates are assumed to have exactly one input in Ξ. For the second claim, suppose by contradiction that not all the almost-evaluations of the gates

$G_{b'}^{\nu',S'}$ that are in Ξ mutually agree. We would then have $G_{b_1}^{\nu_1,S_1}$ and $G_{b_2}^{\nu_2,S_2}$ in Ξ and $g \in b_1 \cap b_2$ such that $\nu_1(g) \neq \nu_2(g)$. But because T is a tree decomposition, g appears in all the bags on the path from b_1 and b_2, and by construction the almost-evaluations of the gates $G_{b'}^{\nu',S'}$ on this path that are in Ξ mutually agree, a contradiction. □

Therefore, Lemma 5.6 allows us to define the union of the almost-evaluations in such a trace:

Definition 5.7 Let G_b^ν a gate in D and Ξ be a trace of D starting at G_b^ν. Then $\gamma(\Xi) := \bigcup_{G_{b'}^{\nu',S'} \in \Xi} \nu'$ (the union of the almost-evaluations in Ξ, which is a valuation from $\bigcup_{G_{b'}^{\nu',S'} \in \Xi} b'$ to $\{0, 1\}$) is properly defined.

We now need to prove a few lemmas about the behavior of gates that are innocent (i.e., not suspicious).

Lemma 5.8 *Let $G_b^{\nu,S}$ a gate in D and Ξ be a trace of D starting at $G_b^{\nu,S}$. Let $g \in b$ be a gate that is innocent ($g \notin S$). Then the following holds:*

- *If $\nu(g)$ is a weak value of g, then for every input g' of g that is in the domain of $\gamma(\Xi)$ (i.e., g' appears in a bag $b' \leqslant b$), we have that $\gamma(\Xi)$ maps g' to a weak value of g;*
- *If $\nu(g)$ is a strong value of g, then there exists an input g' of g that is in the domain of $\gamma(\Xi)$ such that $\gamma(\Xi)(g')$ is $\nu(g)$ if g is an $\wedge$- or $\vee$-gate, and $\gamma(\Xi)(g')$ is $1 - \nu(g)$ if g is a $\neg$-gate.*

Proof We prove the claim by bottom-up induction on $b \in T$. One can easily check that the claim is true when b is a leaf bag, remembering that in this case we must have $S = \mathrm{Unj}(\nu)$ by construction (that is, all the gates that are unjustified are suspicious). For the induction case, let b_l, b_r be the children of b. Suppose first that $\nu(g)$ is a weak value of g, and suppose for a contradiction that there is an input g' of g in the domain of $\gamma(\Xi)$ such that $\gamma(\Xi)(g')$ is a strong value of g. By the occurrence and connectedness properties of tree decompositions, there exists a bag $b' \leqslant b$ in which both g and g' occur. Consider the gate $G_{b'}^{\nu',S'}$ that is in Ξ: by Lemma 5.6, this gate exists and is unique. By definition of $\gamma(\Xi)$ we have $\nu'(g') = \gamma(\Xi)(g')$. Because ν' is a (C, b')-almost-evaluation that maps g' to a strong value of g, we must have that $\nu'(g)$ is also a strong value of g, thus contradicting our hypothesis that $\nu(g) = \gamma(\Xi)(g) = \nu'(g)$ is a weak value for g.

Suppose now that $\nu(g)$ is a strong value of g. We only treat the case when g is an $\vee$- or an $\wedge$-gate, as the case of a $\neg$-gate is similar. We distinguish two sub-cases:

- g is justified. Then, by definition of ν being a (C, b)-almost-evaluation, there must exist an input g' of g that is also in b such that $\nu(g')$ is a strong value of g, which proves the claim.
- g is unjustified. But since g is innocent ($g \notin S$), by construction (precisely, by the second item of Definition 5.5) g must then be innocent for a child of b. The claim then holds by induction hypothesis. □

Lemma 5.8 allows us to show that for a gate g that is not a variable gate, letting b be the topmost bag in which g appears (hence, each input of g must occur in some bag $b' \leqslant b$), if g is innocent then for any trace Ξ starting at a gate for bag b, $\gamma(\Xi)$ respects the semantics of g. Formally, recalling that $W(g)$ denotes the inputs of g:

Lemma 5.9 *Let $G_b^{\nu,S}$ a gate in D and Ξ be a trace of D starting at $G_b^{\nu,S}$. Let $g \in b$ be a gate that is not a variable gate and such that b is the topmost bag in which g appears (hence $W(g) \subseteq domain(\gamma(\Xi))$). If g is innocent ($g \notin S$) then $\gamma(\Xi)$ respects the semantics of g, that is, $\gamma(\Xi)(g) = \bigodot \gamma(\Xi)(W(g))$ where $\bigodot$ is the type of g.*

Proof Clearly implied by Lemma 5.8. □

We need one last lemma about the behavior of suspicious gates, which intuitively tells us that if we have already seen all the input gates of a gate g and g is still suspicious, then g can never escape:

Lemma 5.10 *Let $G_b^{\nu,S}$ a gate in D and Ξ be a trace of D starting at $G_b^{\nu,S}$. Let g be a gate such that the topmost bag b' in which g appears is $\leqslant b$, and consider the unique gate of the form $G_{b'}^{\nu',S'}$ that is in Ξ. If $g \in S'$ then $b' = b$ (hence $G_b^{\nu,S} = G_{b'}^{\nu',S'}$ by uniqueness).*

Proof Let $g \in S'$. Suppose by contradiction that $b' \neq b$. Let p be the parent of b' (which exists because $b' < b$). It is clear that by construction (ν', S') is connectible to p (recall Definition 5.4), hence g must be in p, contradicting the fact that b' should have been the topmost bag in which g occurs. Hence $b' = b$. □

We now have all the results that we need to show that $D \implies C$, i.e. that, for every valuation χ of the variables of C, if $\chi(D) = 1$ then $\chi(C) = 1$ (we see χ both as a valuation of the variables of C, and as a valuation of the (corresponding) variables of D). We prove a stronger result. Given a valuation χ of the variable gates of D and a gate g of D, we say that a *trace of D starting at g according to χ* is a trace of D starting at g such that χ satisfies every gate in Ξ. We show:

Lemma 5.11 *Let χ be a valuation of the variable gates, $G_{\mathrm{root}(T)}^{\nu,\emptyset} \in D$ a gate that evaluates to 1 under χ, and Ξ a trace of D starting at $G_{\mathrm{root}(T)}^{\nu,\emptyset}$ according to χ. Then $\gamma(\Xi)$ corresponds to the evaluation χ of C.*

Proof We prove by induction on C (as its graph is a DAG) that for all $g \in C$, $\gamma(\Xi)(g) = \chi(g)$. When g is a variable gate, consider the leaf bag b_g that is responsible of g, and consider the gate $G_{b_g}^{\nu',S'}$ that is in Ξ: this gate exists and is unique according to Lemma 5.6. This gate evaluates to 1 under χ (because it is in the trace), which is only possible if $G^{g,\nu'(g)}$ evaluates to 1 under χ, hence by construction we must have $\nu'(g) = \chi(g)$ and then $\gamma(\Xi)(g) = \chi(g)$. When g is a constant gate (an $\vee$- or $\wedge$-gate with no inputs), consider the topmost bag b' in which g appears, and consider the unique $G_{b'}^{\nu',S'}$ that is in Ξ. According to Lemma 5.9, we have that $\gamma(\Xi)(g) = \chi(g)$. Now suppose that g is an internal gate of type $\bigodot$, and consider the

topmost bag b' in which g appears. Consider again the unique $G_{b'}^{\nu',S'}$ that is in Ξ. By induction hypothesis, we know that $\gamma(\Xi)(g') = \chi(g')$ for every input g' of g. We now distinguish two cases:

- $b' = \mathrm{root}(T)$. Therefore by Lemma 5.9 we know that $\gamma(\Xi)$ respects the semantics of g, which means that $\gamma(\Xi)(g) = \bigodot \gamma(\Xi)(W(g)) = \bigodot \chi(W(g)) = \chi(g)$ (where the second equality comes from the induction hypothesis and the third equality is just the definition of the evaluation χ of C), which proves the claim.
- $b' < \mathrm{root}(T)$. But then by Lemma 5.10 we must have $g \notin S'$ (because otherwise we should have $b' = \mathrm{root}(T)$ and then $S' = \emptyset$, a contradiction), that is g is innocent for $G_{b'}^{\nu',S'}$. Therefore, again by Lemma 5.9, it must be the case that $\gamma(\Xi)$ respects the semantics of g, and we can again show that $\gamma(\Xi)(g) = \chi(g)$, concluding the proof. □

This indeed implies that $D \implies C$: let χ be a valuation of the variable gates and suppose $\chi(D) = 1$. Then by definition of the output of D, it means that the gate $G_{\mathrm{root}(T)}^{\nu,\emptyset}$ such that $\nu(g_{\mathsf{output}}) = 1$ evaluates to 1 under χ. But then, considering a trace Ξ of D starting at $G_{\mathrm{root}(T)}^{\nu,\emptyset}$ according to χ, we have that $\chi(g_{\mathsf{output}}) = \gamma(\Xi)(g_{\mathsf{output}}) = \nu(g_{\mathsf{output}}) = 1$.

To show the converse ($C \implies D$), one can simply observe the following phenomenon:

Lemma 5.12 *Let χ be a valuation of the variable gates. Then for every bag $b \in T$, the gate $G_b^{\chi_{|b},S}$ evaluates to 1 under χ, where S is the set of gates $g \in \mathrm{Unj}(\nu)$ such that for every input g' of g that appears in some bag $b' \leqslant b$, then $\chi(g')$ is a weak value of g.*

Proof Easily proved by bottom-up induction. □

Now suppose $\chi(C) = 1$. By Lemma 5.12 we have that $G_{\mathrm{root}(T)}^{\chi_{|\mathrm{root}(T)},\emptyset}$ evaluates to 1 under χ, and because $\chi(g_{\mathsf{output}}) = 1$ we have that $\chi(D) = 1$. This shows that $C \implies D$. Hence, we have proved that D is equivalent to C.

5.2.3 *D* is Deterministic

We now prove that D is deterministic, i.e., that every $\vee$-gate in D is deterministic. Recall that the only $\vee$-gates in D are the gates of the form $G_b^{\nu,S}$. We will in fact prove that for every valuation χ and every $\vee$-gate of D, there exists at most one trace of D starting at that gate according to χ, which clearly implies that all the $\vee$-gates are deterministic.

We start by proving the following lemma:

Lemma 5.13 *Let $G_b^{\nu,S}$ be a gate in D and Ξ be a trace of D starting at $G_b^{\nu,S}$. Let $g \in b$. Then the following is true:*

- *if g is innocent ($g \notin S$) and $\nu(g)$ is a strong value of g, then there exists an input g' of g in the domain of $\gamma(\Xi)$ such that $\gamma(\Xi)(g')$ is a strong value for g.*
- *if $g \in S$, then for every input g' of g that is in the domain of $\gamma(\Xi)$, we have that $\gamma(\Xi)(g')$ is a weak value for g.*

Proof We prove the two claims independently:

- Let $g \in b$ such that $g \notin S$ and $\nu(g)$ is a strong value for g. Then the claim directly follows from the second item of Lemma 5.8.
- We prove the second claim via a bottom-up induction on T. When b is a leaf then it is trivially true because g has no input g' in b because $|b| \leqslant 1$ because T is friendly. For the induction case, let $G_{b_l}^{\nu_l, S_l}$ and $G_{b_r}^{\nu_r, S_r}$ be the (unique) gates in Ξ corresponding to the children b_l, b_r of b. By hypothesis we have $g \in S$. By definition of a gate being suspicious, we know that $\nu(g)$ is a strong value for g. To reach a contradiction, assume that there is an input g' of g in the domain of $\gamma(\Xi)$ such that $\gamma(\Xi)(g')$ is a strong value for g. Clearly this g' is not in b, because g is unjustified by ν (because $S \subseteq \mathrm{Unj}(\nu)$). Either g' occurs in a bag $b'_l \leqslant b_l$, or it occurs in a bag $b'_r \leqslant b_r$. The two cases are symmetric, so we assume that we are in the former. As $g \in b$ and $g' \in b'_l$, by the properties of tree decompositions and because $g' \notin b$, we must have $g \in b_l$. Hence, by the contrapositive of the induction hypothesis on b_l applied to g, we deduce that $g \notin S_l$. But then by the second item of Definition 5.5, g should be innocent for $G_b^{\nu, S}$, that is $g \notin S$, which is a contradiction. □

We are ready to prove that traces starting at $\vee$-gates are unique (according to a valuation of the variable gates). Let us first introduce some useful notations: Let U, U' be sets of gates, ν, ν' be valuations having the same domain. We write $(\nu, U) = (\nu', U')$ to mean $\nu = \nu'$ and $U = U'$, and for g in the domain of ν we write $(\nu, U)(g) = (\nu', U')(g)$ to mean that $\nu(g) = \nu'(g)$ and that we have $g \in U$ iff $g \in U'$. We show the following:

Lemma 5.14 *Let χ be a valuation of the variable gates, $G_b^{\nu, S}$ be a gate of D. Then there exists at most one trace of D starting at $G_b^{\nu, S}$ according to χ.*

Proof Fix the valuation χ. We will prove the claim by bottom-up induction on T. The case when b is a leaf is trivial because gates of the form $G_b^{\nu, S}$, for b a leaf of T, have either:

- exactly one input g' that is a constant 1-gate;
- or, exactly one input that is a variable gate $G^{x,1}$, if b is responsible of x and $\nu(g) = 1$;
- or, exactly one input $G^{x,0}$ that is the $\neg$-gate $\neg(G^{x,1})$, if b is responsible of x and $\nu(g) = 0$.

In all cases, there can be at most one trace. For the inductive case, let b be an internal bag with children b_l and b_r. By induction hypothesis for every $G_{b_l}^{\nu_l, S_l}$ (resp., $G_{b_r}^{\nu_r, S_r}$), there exists at most one trace of D starting at $G_{b_l}^{\nu_l, S_l}$ (resp., $G_{b_r}^{\nu_r, S_r}$) according

to χ. Hence, if by contradiction there are at least two traces of D starting at $G_b^{\nu,S}$ according to χ, it can only be because $G_b^{\nu,S}$ is not deterministic, i.e., because for at least two different inputs of $G_b^{\nu,S}$, there is a trace that starts at this input according to χ, say $G_b^{\nu_l,S_l,\nu_r,S_r}$ and $G_b^{\nu'_l,S'_l,\nu'_r,S'_r}$ with $(\nu_l, S_l) \neq (\nu'_l, S'_l)$ or $(\nu_r, S_r) \neq (\nu'_r, S'_r)$. We can suppose that it is $(\nu_l, S_l) \neq (\nu'_l, S'_l)$, since the other case is symmetric. Hence there exists $g_0 \in b_l$ such that $(\nu_l, S_l)(g_0) \neq (\nu'_l, S'_l)(g_0)$. Let Ξ_l be the trace of D starting at $G_{b_l}^{\nu_l,S_l}$ and Ξ'_l be the trace of D starting at $G_{b_l}^{\nu'_l,S'_l}$ (according to χ). We observe the following simple fact about Ξ_l and Ξ'_l:

(*) for any g, if $\gamma(\Xi_l)(g) \neq \gamma(\Xi'_l)(g)$ then $g \notin b$.

Indeed otherwise we should have $\nu(g) = \nu_l(g) = \gamma(\Xi_l)(g)$ and $\nu(g) = \nu'_l(g) = \gamma(\Xi'_l)(g)$, which is impossible.

Now we will define an operator θ that takes as input a gate g such that $(\gamma(\Xi_l), S_l)(g) \neq (\gamma(\Xi'_l), S'_l)(g)$, and outputs another gate $\theta(g)$ which is an input of g and such that again $(\gamma(\Xi_l), S_l)(\theta(g)) \neq (\gamma(\Xi'_l), S'_l)(\theta(g))$. This will lead to a contradiction because for any $n \in \mathbb{N}$, starting with g_0 and applying θ n times consecutively we would obtain a path of n mutually distinct gates (because C is acyclic), but C has a finite number of gates.

Let us now define θ: let g such that $(\gamma(\Xi_l), S_l)(g) \neq (\gamma(\Xi'_l), S'_l)(g)$. We distinguish two cases:

- We have $(\gamma(\Xi_l), S_l)(g) \neq (\gamma(\Xi'_l), S'_l)(g)$ because $\gamma(\Xi_l)(g) \neq \gamma(\Xi'_l)(g)$. Then by (*), we know for sure that $g \notin b$. Therefore the topmost bag b' in which g occurs is $\leqslant b_l$. Let $G_{b'}^{\nu',S'}$ be the gate in Ξ_l and $G_{b'}^{\nu'',S''}$ the gate in Ξ'_l (they exist and are unique by Lemma 5.6). We again distinguish two subcases:
  - g is a variable gate. But then it is clear that, by considering the bag b'' that is responsible for g, we have $b'' \leqslant b'$, and then that $\gamma(\Xi_l)(g) = \chi(g) = \gamma(\Xi'_l)(g)$, a contradiction.
  - g is not a variable gate. Observe that by Lemma 5.10 we must have $g \notin S'$ and $g \notin S''$, because otherwise we should have $b' = b$, which is not true. But then, by Lemma 5.9 we know that both $\gamma(\Xi_l)$ and $\gamma(\Xi'_l)$ respect the semantics of g. But we have $\gamma(\Xi_l)(g) \neq \gamma(\Xi'_l)(g)$, so there must exist an input g' of g such that $\gamma(\Xi_l)(g') \neq \gamma(\Xi'_l)(g')$. We can thus take $\theta(g)$ to be g'.
- We have $(\gamma(\Xi_l), S_l)(g) \neq (\gamma(\Xi'_l), S'_l)(g)$ because (without loss of generality) $g \notin S_l$ and $g \in S'_l$. Observe that this implies that $g \in b_l$, and that $\nu'_l(g)$ is a strong value for g. We can assume that $\nu_l(g) = \nu'_l(g)$, as otherwise we would have $\gamma(\Xi_l)(g) \neq \gamma(\Xi'_l)(g)$, which is a case already covered by the last item. Hence $\nu_l(g)$ is also a strong value for g, but we have $g \notin S_l$, so by the first item of Lemma 5.13 we know that there exists an input g' of g that occurs in some bag $\leqslant b_l$ and such that $\gamma(\Xi_l)(g')$ is a strong value for g. We show that $\gamma(\Xi'_l)(g')$ must in contrast be a weak value for g, so that we can take $\theta(g)$ to be g' and conclude the proof. Indeed suppose by way of contradiction that $\gamma(\Xi'_l)(g')$ is a

strong value for g. By the contrapositive of the second item of Lemma 5.13, we get that $g \notin S'_l$, which contradicts our assumption.

Hence we have constructed θ, which shows a contradiction, which means that in fact we must have $G_b^{\nu_l, S_l, \nu_r, S_r} = G_b^{\nu'_l, S'_l, \nu'_r, S'_r}$, so that $G_b^{\nu, S}$ is deterministic, which proves that there is at most one trace of D starting at $G_b^{\nu, S}$ according to χ, which was our goal. □

This concludes the proof that D is deterministic, and thus that D is a d-SDNNF equivalent to C.

5.2.4 Analysis of the Running Time

We last check that the construction can be performed in time $O(|T| \times f(k))$, for some function $f(k)$ that is in $O\left(2^{(4+\epsilon)k}\right)$ for any $\epsilon > 0$:

- From the initial tree decomposition T of C, in time $O(k|T|)$ we computed the g_{output}-friendly tree decomposition T_{friendly} of size $O(k|T|)$;
- In linear time in T_{friendly}, for every variable $x \in C_{\mathsf{var}}$ we selected a leaf bag b_x of T such that $\lambda(b_x) = \{x\}$;
- We can clearly compute the v-tree and the mapping in linear time in T_{friendly};
- For each bag b of T_{friendly} we have $2^{2|b|} \leqslant 2^{2k+2}$ different pairs of a valuation ν of b and of a subset S of b, and checking if ν is a (C, b)-almost-evaluation and if S is a subset of the unjustified gates of ν can be done in polynomial time in $|b| \leqslant k+1$ (we access the inputs and the type of each gate in constant time from C), hence we pay $O\left(|T_{\text{friendly}}| \times p(k) \times 2^{2k}\right)$ to create the gates of the form $G_b^{\nu, S}$, for some polynomial p;
- We constructed and connected in time $O(|T_{\text{friendly}}| \times p'(k) \times 2^{4k})$ the gates of the form $G_b^{\nu_l, S_l, \nu_r, S_r}$ (the polynomial is for testing if the result of (ν_l, S_l) and of (ν_r, S_r) is an almost-evaluation);
- In time $O(|T_{\text{friendly}}|)$ we connected the gates of the form $G_b^{\nu, S}$ for b a leaf to their inputs.

Hence, the total time is indeed in $O(|T| \times f(k))$, for some function $f(k)$ that is in $O(2^{(4+\epsilon)k})$ for any $\epsilon > 0$.

5.3 uOBDDs for Bounded-Pathwidth Circuits

We now argue that our construction for Theorem 4.2 can be specialized in the case of bounded-pathwidth circuits to compute uOBDDs, i.e., we prove Theorem 4.4. If the input circuit captures a constant Boolean function then there is no difficulty, so we assume that this is not the case.

Let C be the Boolean circuit with output gate g_{output}, and P be a path decomposition of C of width $\leqslant k$. It is clear that, by adapting Lemma 2.2, we can compute in linear time from P a g_{output}-friendly *tree* decomposition T that is further right-linear. We then use the same construction as the one we use in Theorem 4.2 with T. This

gives us an extended complete d-SDNNF (D, T', ρ) of width $\leqslant 2^{2(k+1)}$ equivalent to C, where T' is an extended v-tree that is right-linear. It is easy to verify that we can compute this d-SDNNF in time $O(|T| \times f(k))$ for some function f that is in $O(2^{(2+\epsilon)k})$ for any $\epsilon > 0$ (as opposed to $O(2^{(4+\epsilon)k})$ in the original construction). This is thanks to the fact that T is right-linear, and because the leaf bags of a friendly tree decomposition can contain at most one gate of C. We then use Lemma 3.14 to compress the extended complete d-SDNNF into a complete d-SDNNF (D', T'', ρ') of width $\leqslant 2^{2(k+1)}$, again equivalent to C. By inspection of the proof of this Lemma, it is clear that T'' is a (non extended) v-tree that is right-linear. Again by inspection of the proof of Lemma 3.14, we can see that we can rewrite every $\wedge$-gate g of D' to be of the form $g' \wedge x$ or $g' \wedge \neg x$, for some variable x and gate g': this is thanks to the fact that Lemma 3.14 starts by propagating constants, so that the right input of g can only be equivalent to x or to $\neg x$ (there is an $\vee$-gate in between, which we remove).

We now justify that we can transform D' into a complete uOBDD. First, create the 0-sink $\bot$ and the 1-sink $\top$. Then, we traverse the internal nodes n of T' top-down and inductively define an uOBDD $O(g)$ for every $\vee$-gate g of D' structured by n. The intuition is that $O(g)$ captures the subcircuit rooted at g. Remember that C captures a non-constant Boolean function, which implies that there are no $\wedge$- or $\vee$-gates without input left in D'. We proceed as follows, letting l be the right child of n, corresponding to variable x, and n' being the left child of n:

– For every $\vee$-gate g of D' structured by n we do the following. First, compute A^0_g, the set of gates g' such that there exists an $\wedge$-gate of the form $g' \wedge \neg x$ that is an input of g, and A^1_g, the set of gates g' such that there exists an $\wedge$-gate of the form $g' \wedge x$ that is an input of g. Then define $O^0(g)$ to be $\bigvee_{g' \in A^0_g} O(g')$; and similarly $O^1(g) := \bigvee_{g' \in A^1_g} O(g')$. Finally define $O(g)$ to consist of a node labeled by x, with outgoing 0-edge to $O^0(g)$ and outgoing 1-edge to $O^1(g)$.

There is one special case in this construction: we might not have defined $O(g)$ when g is a variable gate (resp., a $\neg$-gate) that is structured by the leftmost leaf of T. In that case, we define $O(g)$ to consist of a node labeled by the corresponding variable, with outgoing 1-edge to $\top$ (resp., $\bot$), and outgoing 0-edge to $\bot$ (resp., $\top$). It is then clear that, by considering the output gate G_{output} of D', we have that $O(G_{\mathsf{output}})$ is a complete uOBDD of width $\leqslant 2^{2(k+1)}$ that captures the same function as D'.

6 Lower Bounds for Structured Classes: SCOV_n and SINT_n

In this section, we start our presentation of our lower bound results. Our upper bound in Section 4 applied to arbitrary Boolean circuits; however, our lower bounds in this section and the next one will already apply to much weaker formalisms for Boolean functions, namely, monotone DNFs and monotone CNFs.

We first review some existing lower bounds on the compilation of monotone CNFs and DNFs into OBDDs and d-SDNNFs. Bova and Slivovsky have constructed a family of CNFs of bounded degree whose OBDD representations are exponential

[15, Theorem 19], following an earlier result of this type by Razgon [43, Corollary 1]. The result of Bova and Slivovsky is as follows:

Theorem 6.1 ([15, Theorem 19]) *There is a class of monotone CNF formulas of bounded degree and arity such that every formula φ in this class has OBDD size at least $2^{\Omega(|\varphi|)}$.*

By a similar approach, Bova, Capelli, Mengel, and Slivovsky show an exponential lower bound on the size of d-SDNNF representing a given family of DNFs [14, Theorem 14]. However, these bounds apply to well-chosen families of Boolean functions. We adapt some of these techniques to show a more general result. First, our lower bounds will apply to *any* monotone DNF or monotone CNF, not to one specific family. Second, our lower bounds apply to more expressive classes of binary decision diagrams than OBDDs, namely, uOBDDs and nOBDDs (recall their definitions from Section 3). Third, we obtain finer lower bounds on SDNNFs thanks to our new notion of width.

In essence, our result is shown by observing that the families of functions used in [14, 15] occur "within" any bounded-degree, bounded-arity monotone CNF or DNF. Here is the formal definition of these two families:

Definition 6.2 Let $n \in \mathbb{N}$ and consider two disjoint tuples $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$. The *set covering CNF* $\mathrm{SCOV}_n(X, Y)$ is the monotone CNF:

$$\mathrm{SCOV}_n(X, Y) = \bigwedge_{i=1}^{n} x_i \vee y_i.$$

Similarly, the *set intersection DNF* $\mathrm{SINT}_n(X, Y)$ is the monotone DNF:

$$\mathrm{SINT}_n(X, Y) = \bigvee_{i=1}^{n} x_i \wedge y_i.$$

As the order chosen on X and Y does not matter, we will often abuse notation and consider them as sets rather than tuples.

In this section, we prove lower bounds on the size of representations of the functions SCOV_n and SINT_n. We will then show in Section 7 how to extend these bounds to arbitrary monotone CNFs/DNFs.

Our lower bounds on the representations of $\mathrm{SCOV}_n(X, Y)$ and $\mathrm{SINT}_n(X, Y)$ will only apply to some specific variable orderings. Indeed, observe that both $\mathrm{SCOV}_n(X, Y)$ and $\mathrm{SINT}_n(X, Y)$ can easily be represented by complete OBDDs of size $O(n)$ with the variable ordering $\mathbf{v} := x_1 y_1 \ldots x_n y_n$. The idea of our bounds is that "inconvenient" variable orderings (or "inconvenient" v-trees) can force OBDD (or SDNNF) representations of SCOV and SINT to be of exponential size. We formalize our notion of inconvenient variable orderings and v-trees as follows:

Definition 6.3 Let V be a set of variables, X and Y be two disjoint subsets of V. We say that a total order $\mathbf{v} = v_1, \ldots, v_{|V|}$ of V *cuts* (X, Y) if there exists $1 \leqslant i \leqslant n$ such

that $X \subseteq \mathbf{v}_{<i}$ and $Y \subseteq \mathbf{v}_{\geq i}$. Similarly, we say that a v-tree T over V *cuts* (X, Y) if there exists a node n of T such that $X \subseteq \mathsf{Leaves}(T_n)$ and $Y \subseteq \mathsf{Leaves}(T \setminus T_n)$.

In the rest of this section, we show the following two theorems. The first theorem applies to OBDDs, and the second generalizes it to SDNNFs.

Theorem 6.4 *Let* $\mathbf{v}$ *be a total order that cuts* (X, Y)*. Then the width of any complete nOBDD (resp., complete uOBDD) structured by* $\mathbf{v}$ *that computes* $\mathrm{SCOV}_n(X, Y)$ *(resp.,* $\mathrm{SINT}_n(X, Y)$*) is* $\geq 2^n - 1$.

Theorem 6.5 *Let* T *be a v-tree that cuts* (X, Y)*. Then the width of any complete SDNNF (resp., complete d-SDNNF) structured by* T *that computes* $\mathrm{SCOV}_n(X, Y)$ *(resp.,* $\mathrm{SINT}_n(X, Y)$*) is* $\geq 2^n - 1$.

These results are proved in two steps, presented in the next two sections. First, we show that Boolean functions computed by SDNNF or nOBDD (resp., d-SDNNF or uOBDD) of width w can be decomposed as a disjunction (resp., exclusive disjunction) of at most w very simple Boolean functions known as *rectangles*. We then appeal to known results about these functions that show that SCOV_n (resp., SINT_n) cannot be decomposed as a disjunction (resp., exclusive disjunction) of less than $2^n - 1$ rectangles, which implies the desired lower bound.

6.1 Rectangle Covers for Compilation Targets

Towards our desired bounds on the size of compilation targets, we start by formalizing the notion of decomposing Boolean functions as a *rectangle cover*.

Definition 6.6 Let V be a set of variables and (X, Y) be a partition of V. A (X, Y)-*rectangle* is a Boolean function $R : 2^V \to \{0, 1\}$ such that there exists $R_X : 2^X \to \{0, 1\}$ and $R_Y : 2^Y \to \{0, 1\}$ such that $R = R_X \wedge R_Y$. In other words, for any valuation ν of V, we have $R(\nu) = 1$ iff $R_X(\nu_{|X}) = 1$ and $R_Y(\nu_{|Y}) = 1$.

For any Boolean function $f : 2^V \to \{0, 1\}$, a (X, Y)-*rectangle cover* of f is a set S of (X, Y)-rectangles such that $f = \bigvee_{R \in S} R$. The *size* $|S|$ of S is the number of rectangles. We say that S is *disjoint* if for every $R, R' \in S$, we have $R \wedge R' = \bot$.

Connections between rectangle covers and compilation target sizes have already been successfully used to prove lower bounds, see [8, 14, 51]. We adapt these results to relate the size of rectangle covers to the width of compilation targets in our context. We give proofs for these results that are essentially self-contained: their main difference with existing proofs is that our proofs apply to our notion of width whereas existing results generally apply to size.

We start by relating the width of OBDDs with the size of rectangle covers:

Theorem 6.7 *Let* O *be a complete nOBDD on variables* V *structured by the total order* $\mathbf{v} = v_1, \ldots, v_{|V|}$*. Let* $1 \leqslant i \leqslant n$*, let* $\mathbf{v}_{<\mathbf{i}} = \{v_j \mid j < i\}$ *and let* $\mathbf{v}_{\geq \mathbf{i}} = \{v_j \mid$

$j \geq i$}. There exists a $(\mathbf{v}_{<\mathbf{i}}, \mathbf{v}_{\geq\mathbf{i}})$-rectangle cover S of O whose size is at most the v_i-width of O. Moreover, if O is an uOBDD, then S is disjoint.

Proof Let v be a node of O that tests variable v_i, and let R_v be the set of valuations of $2^{\mathbf{v}}$ that are accepted in O by a path going through v. We claim that R_v is a $(\mathbf{v}_{<\mathbf{i}}, \mathbf{v}_{\geq\mathbf{i}})$-rectangle. Indeed, let $R_{\mathbf{v}_{<\mathbf{i}}}$ be the set of valuations of $\mathbf{v}_{<\mathbf{i}}$ compatible with some path in O from the root to v, and let $R_{\mathbf{v}_{\geq\mathbf{i}}}$ be the set of valuations of $\mathbf{v}_{\geq\mathbf{i}}$ compatible with some path in O from v to the 1-sink. Any valuation of R_v can clearly be written as the union of one valuation from $R_{\mathbf{v}_{<\mathbf{i}}}$ and of one valuation from $R_{\mathbf{v}_{\geq\mathbf{i}}}$. Conversely, given any pair $\nu_{\mathbf{v}_{<\mathbf{i}}}$ and $\nu_{\mathbf{v}_{\geq\mathbf{i}}}$ of valuations of these two sets, we can combine any two witnessing paths for these valuations to obtain a path in O that witnesses that $\nu_{\mathbf{v}_{<\mathbf{i}}} \cup \nu_{\mathbf{v}_{\geq\mathbf{i}}}$ is in R_v. Hence, it is indeed the case that $R_v = R_{\mathbf{v}_{<\mathbf{i}}} \wedge R_{\mathbf{v}_{\geq\mathbf{i}}}$.

Consider now the set O_i of gates of O that test v_i. Since O is complete, every accepting path of O contains a gate of O_i. It follows that $\bigcup_{v \in O_i} R_v$ is a rectangle cover of O, and its size is at most $|O_i|$, i.e., the v_i-width of O.

Now, if O is an uOBDD, then let ν be a satisfying valuation of O. Since O is unambiguous, there exists a unique path π in O compatible with ν. Moreover, since O is complete, π contains exactly one node v that is labeled by v_i, so that ν is in R_v, and not in any other $R_{v'}$ for $v' \neq v$. Hence, $\bigcup_{v \in O_i} R_v$ is a disjoint rectangle cover of O. □

We now generalize this result from OBDDs to SDNNFs. To do so, we use the connections of [14, Theorem 13] and [41, Theorem 3] between rectangle covers and SDNNF size, and adapt them to our notion of width:

Theorem 6.8 ([14, Theorem 13] and [41, Theorem 3]) *Let (D, T, ρ) be a complete SDNNF on variables V. Let $n \in T$. There is a $(\mathsf{Leaves}(T_n), \mathsf{Leaves}(T \setminus T_n))$-rectangle cover S of D whose size is at most the n-width of (D, T, ρ). Moreover, if D is a d-SDNNF, then S is disjoint.*

Proof Recall the notion of trace (Definition 3.2). Given an $\vee$-gate g of D structured by n, we define R_g to be the set of valuations of 2^V that are accepted in D by a trace going through g. Then R_g defines a $(\mathsf{Leaves}(T_n), \mathsf{Leaves}(T \setminus T_n))$-rectangle. Intuitively, any trace going through g defines a trace on the variables $\mathsf{Leaves}(T_n)$ that starts at gate g, and one "partial" trace on $\mathsf{Leaves}(T \setminus T_n)$ where g is also used as a leaf; conversely, any pair of such traces can be combined to a complete trace in D that goes through g. The precise argument is given in [14, Theorem 1], where traces are called *certificates*.

Consider now the set D_n of $\vee$-gates structured by n. Since D is complete, every satisfying valuation of D has a corresponding trace containing a gate in D_n; this uses the facts that in complete d-SDNNFs, no input of an $\wedge$-gate is an $\wedge$-gate, and an $\wedge$-gate structured by an internal node of the v-tree has exactly two children. It follows that $S = \bigcup_{g \in D_n} R_g$ is a $(\mathsf{Leaves}(T_n), \mathsf{Leaves}(T \setminus T_n))$-rectangle cover of D of size $|D_n| \leqslant w$.

Now, if D is a d-SDNNF, it is not hard to see that every satisfying assignment has exactly one accepting trace: otherwise, considering any topmost gate where the two

traces differ, we see that this gate must be a $\vee$-gate where the two traces witness a violation of determinism. Moreover, if ν is a satisfying valuation of R_g for some $g \in D_n$ then its unique trace contains g and cannot contain another $g' \in D_n$: otherwise this would imply that one $\wedge$-gate is not decomposable, or that there is an $\vee$-gate having an $\vee$-gate as input which would be a violation of completeness. Thus ν is not in $R_{g'}$. In other words, S is disjoint. □

6.2 Rectangle covers of SCOV and SINT

The second step of the proof of Theorems 6.4 and 6.5 is to observe that $\mathrm{SCOV}_n(X, Y)$ (resp., $\mathrm{SINT}_n(X, Y)$) does not have small (X, Y)-rectangle covers (resp., disjoint covers). This is a folklore result in communication complexity: see, e.g., [47, Section 3] for the bound on $\mathrm{SCOV}_n(X, Y)$. For completeness, we state and prove the result here:

Theorem 6.9 *Let S be a (X, Y)-rectangle cover (resp., disjoint (X, Y)-rectangle cover) of $\mathrm{SCOV}_n(X, Y)$ (resp., of $\mathrm{SINT}_n(X, Y)$). Then $|S| \geq 2^n - 1$.*

Proof We first prove our claim for $\mathrm{SCOV}_n(X, Y)$ as in [47]. For any valuation ν of X, we denote by $\overline{\nu}$ the valuation of Y defined by $\nu(y_i) := \neg\nu(x_i)$ for all $1 \leqslant i \leqslant n$. Consider the set $\mathcal{F} := \{\nu \cup \overline{\nu} \mid \nu : X \to \{0, 1\}\}$: we have $|\mathcal{F}| = 2^n$, it is clear that $\mathrm{SCOV}_n(X, Y)$ evaluates to true on each valuation of $\mathcal{F}$, and we will now show that $\mathcal{F}$ is a *fooling set* in the terminology of [47]. Specifically, consider the (X, Y)-rectangle cover S and let us show that every rectangle contains at most one valuation of $\mathcal{F}$, which implies the bound. Assume by contradiction that some rectangle $R_X \wedge R_Y$ contains two valuations $\nu \cup \overline{\nu}$ and $\nu' \cup \overline{\nu'}$ of $\mathcal{F}$ for $\nu \neq \nu'$, so that $R_X(\nu) = R_X(\nu') = R_Y(\overline{\nu}) = R_Y(\overline{\nu'}) = 1$. This implies that the rectangle $R_X \wedge R_Y$ must also contain $\nu \cup \overline{\nu'}$ and $\nu' \cup \overline{\nu}$. However, as $\nu \neq \nu'$, there is $1 \leqslant i \leqslant n$ where $\nu(x_i) \neq \nu'(x_i)$. The first case is that we have $\nu(x_i) = 1$ but $\nu'(x_i) = 0$, in which case $\overline{\nu}(y_i) = 0$ and $\overline{\nu'}(y_i) = 1$, but then $\mathrm{SCOV}_n(X, Y)$ evaluates to 0 on $\nu' \cup \overline{\nu}$, a contradiction. The second case is that we have $\nu(x_i) = 0$ but $\nu'(x_i) = 1$ and we conclude symmetrically using $\nu \cup \overline{\nu'}$. Thus we have shown that any rectangle of S can contain at most one valuation from $\mathcal{F}$, so that $|S| \geq |\mathcal{F}| \geq 2^n$, in particular $|S| \geq 2^n - 1$.

The proof of our claim for $\mathrm{SINT}_n(X, Y)$ can be found in [14]. More precisely, it is exactly Theorem 15 of [14], together with the second-last sentence in the proof of Proposition 14 of [14]. □

We are now ready to prove Theorems 6.4 and 6.5:

Proof (of Theorem 6.4) Let O be a complete nOBDD (resp., complete uOBDD) computing $\mathrm{SCOV}_n(X, Y)$ (resp., $\mathrm{SINT}_n(X, Y)$), where O is structured by $\mathbf{v} = v_1, \ldots, v_{|V|}$ that cuts (X, Y). Let $1 \leqslant i \leqslant n$ witnessing that $\mathbf{v}$ cuts (X, Y), and let w_i be the v_i-width of O. By Theorem 6.7, $\mathrm{SCOV}_n(X, Y)$ (resp., $\mathrm{SINT}_n(X, Y)$) has a (X, Y)-rectangle cover (resp., disjoint rectangle cover) of size $\leqslant w_i$. Hence, by Theorem 6.9, we have $w_i \geq 2^n - 1$. But then this implies that the width of O is also $\geq 2^n - 1$. □

The proof of Theorem 6.5 is similar, using Theorem 6.8 instead of Theorem 6.7.

7 Lower Bounds for Structured Classes: General Case

In this section we extend the lower bounds of the previous section from the specific functions SCOV_n and SINT_n, to obtain lower bounds that apply to any family of CNFs and DNFs. Specifically, we will show:

Theorem 7.1 *Let φ be a monotone CNF (resp., monotone DNF) of pathwidth k, arity a and degree d. Then the width of any complete nOBDD (resp., any complete uOBDD) computing φ is $\geq 2^{\frac{k}{a^3d^2}} - 1$.*

Theorem 7.2 *Let φ be a monotone CNF (resp., monotone DNF) of treewidth k, arity a and degree d. Then the width of any complete SDNNF (resp., any complete d-SDNNF) computing φ is $\geq 2^{\frac{k}{3a^3d^2}} - 1$.*

Together with our upper bounds, this implies the following when we have a constant bound on arity and degree:

Corollary 7.3 *For any monotone CNF φ (resp., monotone DNF φ) of constant arity and degree, the width of the smallest complete nOBDD (resp., uOBDD) computing φ is $2^{\Theta(\mathrm{pw}(\varphi))}$.*

Corollary 7.4 *For any monotone CNF φ (resp., monotone DNF φ) of constant arity and degree, the width of the smallest complete SDNNF (resp., d-SDNNF) computing φ is $2^{\Theta(\mathrm{tw}(\varphi))}$.*

The completeness assumption can be lifted using the completion results (Lemma 3.16), to show a lower bound on representations that are not necessarily complete. However, if we do this, we no longer have a definition of width, so the lower bound is on the *size* of the representation (and thus is no longer tight).

Corollary 7.5 *For any monotone CNF φ (resp., monotone DNF φ) of constant arity and degree, the size of the smallest nOBDD (resp., uOBDD) computing φ is $2^{\Omega(\mathrm{pw}(\varphi))}$.*

Corollary 7.6 *For any monotone CNF φ (resp., monotone DNF φ) of constant arity and degree, the size of the smallest SDNNF (resp., d-SDNNF) computing φ is $2^{\Omega(\mathrm{tw}(\varphi))}$.*

In the case of OBDDs, it is easy to generalize width to non-complete OBDDs such that we can lift the completeness assumption in Corollary 7.3: see Theorem 15 of [5]. However, this result in [5] is only shown for OBDDs (not nOBDDs or uOBDDs), and it is open whether it extends to these larger classes. As for SDNNFs, we leave

to future work the task of generalizing width to non-complete circuits and showing comparable bounds.

To prove Theorem 7.1 and 7.2, we will explain how we can find SCOV_n (resp., SINT_n) in any monotone CNF (resp., DNF) of high pathwidth/treewidth. To this end, we first present a general notion of a set of clauses being *split* by two variable subsets. We will then show Theorem 7.1, and last show Theorem 7.2.

7.1 From Split Sets of Clauses to SCOV_n and SINT_n

To prove Theorem 7.1 and 7.2, we will need to find a subset of the clauses of the formula which is *split* by two subsets of variables. In this subsection, we introduce the corresponding notions. We first define the notions of *split clauses*, which we introduce in terms of hypergraphs because the definition is the same for DNF and CNF.

Definition 7.7 Let $H = (V, E)$ be a hypergraph, and let X', Y' be two disjoint subsets of V. We say that a set $E' \subseteq E$ of hyperedges is *split* by (X', Y') if every $e \in E'$ intersects X' and Y' nontrivially, i.e., $e \cap X' \neq \emptyset$ and $e \cap Y' \neq \emptyset$.

If we can find a set K' of clauses of a monotone CNF (resp., monotone DNF) that are split by some pair (X', Y') of disjoint variable subsets, then we can use it to find a partial valuation that yields $\mathrm{SCOV}_n(X, Y)$ (resp., $\mathrm{SINT}_n(X, Y)$) for some $X \subseteq X'$ and $Y \subseteq Y'$, where the number n of extracted clauses depends on the number of clauses in K' and on the arity and degree. Formally:

Proposition 7.8 *Let φ be a monotone CNF (resp., monotone DNF) with variable set V, arity a and degree d. Assume that there are two disjoint subsets X', Y' and a set K' of clauses of φ such that K' is split by (X', Y'). Let $n := \left\lfloor \frac{|K'|}{a^2 \times d^2} \right\rfloor$. Then we can find $X \subseteq X'$ and $Y \subseteq Y'$ such that $|X| = |Y| = n$, and a valuation ν of $V \setminus (X \cup Y)$ such that $\nu(\varphi) = \mathrm{SCOV}_n(X, Y)$ (resp., $\nu(\varphi) = \mathrm{SINT}_n(X, Y)$).*

We prove Proposition 7.8 in the rest of this subsection. Intuitively, the idea is to use the clauses of K' to achieve $\mathrm{SCOV}_n(X, Y)$ or $\mathrm{SINT}_n(X, Y)$, and assign the other variables to eliminate them from the clauses and eliminate the other clauses. Our ability to do this will rely on the monotonicity of φ, but it will also require a careful choice of a subset of clauses of K' that are "independent" in some sense: they should be pairwise disjoint and any pair of them should never intersect a common clause. We formalize this as an independent set in an *exclusion graph* constructed from the hypergraph of φ:

Definition 7.9 The *exclusion graph* of a hypergraph $H = (V, E)$ is the graph G_H whose vertices are the edges E of H, and where two edges $e \neq e'$ are adjacent if (1.) e and e' both intersect some edge $e'' \in E$, or if (2.) e and e' intersect each other: note that case (2.) is in fact covered by case (1.) by taking $e'' := e'$. Equivalently, e and e' are adjacent in G_H iff they are at distance $\leqslant 4$ in the so-called incidence graph of

H. Formally, we can define $G_H = (E, \{\{e, e'\} \in E^2 \mid e \neq e' \wedge \exists e'' \in E, (e \cap e'') \neq \emptyset \wedge (e' \cap e'') \neq \emptyset\}$.

Remember that an *independent set* of a graph $G = (V, E)$ is a subset S of V such that no two elements of S are adjacent in G. We will use the following easy lemma on independent sets:

Lemma 7.10 *Let $G = (V, E)$ be a graph and let $V' \subseteq V$. Then G has an independent set $S \subseteq V'$ of size at least $\left\lfloor \frac{|V'|}{\text{degree}(G)+1} \right\rfloor$.*

Proof We construct the independent S set with the following trivial algorithm: start with $S := \emptyset$ and, while V' is non-empty, pick an arbitrary vertex v in V', add it to S, and remove v and all its neighbors from G and from V'. It is clear that this algorithm terminates and adds the prescribed number of vertices to S, so all that remains is to show that S is an independent set at the end of the algorithm. This is initially true for $S = \emptyset$; let us show that it is preserved throughout the algorithm. Assume by way of contradiction that, at a stage of the algorithm, we add a vertex v to S and that it stops being an independent set. This means that S contains a neighbor v' of v which must have been added earlier; but when we added v' to S we have removed all its neighbors from G, so we have removed v and we cannot add it later, a contradiction. Hence, the algorithm is correct and the claim is shown. □

To use this lemma, let us bound the degree of G_H using the degree and arity of H:

Lemma 7.11 *Let H be a hypergraph. Then we have* $\text{degree}(G_H) \leqslant (\text{arity}(H) \times \text{degree}(H))^2 - 1$.

Proof Any edge e of H contains $\leqslant \text{arity}(H)$ vertices, each of which occurs in $\leqslant \text{degree}(H) - 1$ edges that are different from e, so any edge e of H intersects at most $n := \text{arity}(H) \times (\text{degree}(H) - 1)$ edges different from e. Hence, the degree of G_H is at most $n + n^2$ (counting the edges that intersect e or those at distance 2 from e). Now, we have $n + n^2 = n(n + 1)$, and as $\text{degree}(H) \geq 1$ and $\text{arity}(H) \geq 1$ (because we assume that hypergraphs contain at least one non-empty edge), the degree of G_H is $< \text{arity}(H) \times \text{degree}(H) \times (1 + \text{arity}(H) \times (\text{degree}(H) - 1))$, i.e., it is indeed $< (\text{arity}(H) \times \text{degree}(H))^2$, which concludes. □

We are now ready to show Proposition 7.8.

Proof (of Proposition 7.8) Let φ be the monotone formula with variable set V, arity a, and degree d, fix the sets X' and Y' and the set K' of split clauses, and let $n := \left\lfloor \frac{|K'|}{a^2 \times d^2} \right\rfloor$. Let G_φ be the exclusion graph of φ (seen as a hypergraph of clauses). By Lemma 7.11, the graph G_φ has degree $\leqslant a^2 \times d^2 - 1$, so by Lemma 7.10 it has an independent set $K'' \subseteq K'$ with $\left|K''\right| \geq n$. Let us pick any subset K of K'' that has cardinality exactly n: K is still an independent set of G_φ, and K is still split by (X', Y').

Let us now define X by choosing one element of X' in each clause of K, and define Y accordingly. We have $X \subseteq X'$, $Y \subseteq Y'$, so that X' and Y' are disjoint; and the cardinality of X and Y is exactly n, because the clauses of K are pairwise disjoint as none of them are adjacent in G_φ.

Let us now define the valuation ν of $V \setminus (X \cup Y)$. We first let $Z := \bigcup K$ be the set of variables occurring in the clauses of K: we have $X \cup Y \subseteq Z$. Let ν_1 be the partial valuation that assigns all variables of $V \setminus Z$ to 1 if φ is a CNF and to 0 if φ is a DNF, and consider the formula $\nu_1(\varphi)$: it consists precisely of the clauses of φ that only use variables of Z (in particular those of K), because the other clauses evaluate to true (if φ is a CNF) or false (if φ is a DNF) and so are simplified away.

Let us now observe that in fact $\nu_1(\varphi)$ precisely consists of the clauses of K. Indeed, the clauses of K are clearly in $\nu_1(\varphi)$, and for the converse let us assume by contradiction that φ contains a clause e'' that only uses variables of Z but is not a clause of K. We know that e'' cannot be the empty clause because we have disallowed it, so it contains a variable of Z, which means that it intersects a clause e of K. Now, by hypothesis e'' is not a clause of K, and as φ is minimized we know that e'' cannot be a subset of e, which means that it must contain some variable of Z which is not in e, hence it must intersect some other clause $e' \neq e$ of K. Hence, e'' intersects both e and e', which is impossible as K is an independent set of G_φ but e'' witnesses that e and e' are adjacent in G_φ. We conclude that $\nu_1(\varphi)$ precisely consists of the clauses of K.

Now, let us consider the partial valuation ν_2 of $\nu_1(\varphi)$ that assigns all variables of $Z \setminus (X \cup Y)$ to 0 (if φ is a DNF) or to 1 (if φ is a CNF). Let $\nu := \nu_1 \cup \nu_2$. It is clear that the clauses of $\nu(\varphi)$ are the intersection of the clauses of $\nu_1(\varphi)$ with $X \cup Y$, i.e., the intersection of the clauses of K with $X \cup Y$. Now, the definition of K ensures that each clause contains exactly one variable of X and one variable of Y, with each variable occurring in exactly one clause. Thus, it is the case that $\nu(\varphi) = \mathrm{SCOV}_n(X, Y)$ or $\nu(\varphi) = \mathrm{SINT}_n(X, Y)$, which concludes the proof. □

7.2 Proof of Theorem 7.1: Pathwidth and OBDDs

In this section, we explain how we can obtain SCOV_n (resp., SINT_n) by applying a well-chosen partial valuation to any monotone CNF (resp., monotone DNF). The key result is:

Proposition 7.12 *Let φ be a monotone CNF (resp., monotone DNF) of pathwidth $\geq k$ on variables V. Then, for any variable ordering $\mathbf{v}$ of V, there exist disjoint subsets X, Y of V such that $\mathbf{v}$ cuts (X, Y) and a valuation ν of $V \setminus (X \cup Y)$ such that $\nu(\varphi) = \mathrm{SCOV}_l(X, Y)$ (resp., $\mathrm{SINT}_l(X, Y)$) for some $l \geq \frac{k}{a^3 d^2}$.*

Thanks to this result, we can extend Theorem 6.4 to arbitrary monotone CNFs/DNFs, which is what we need to prove Theorem 7.1:

Proof (of Theorem 7.1) Let O be a complete nOBDD (resp., complete uOBDD) computing φ, and let $\mathbf{v}$ be its order on the variables V. By Proposition 7.12, there exist disjoint subsets X, Y of V such that $\mathbf{v}$ cuts (X, Y) and a valuation ν of $V \setminus (X \cup Y)$ such that $\nu(\varphi) = \mathrm{SCOV}_l(X, Y)$ (resp., $\mathrm{SINT}_l(X, Y)$) for $l \geq \frac{k}{a^3 d^2}$. By applying

Lemma 3.12 to O, we know that there is a complete nOBDD (resp., complete uOBDD) O' on variables $X \cup Y$ with order $\mathbf{v}' = \mathbf{v}|_{X \cup Y}$ computing $\mathrm{SCOV}_l(X, Y)$ (resp., $\mathrm{SINT}_l(X, Y)$), whose width is no greater than that of O. Now, it is clear that $\mathbf{v}'$ still cuts (X, Y), so that by Theorem 6.4 the width of O', and hence that of O, is $\geq 2^{\frac{k}{a^3 d^2}} - 1$. □

Hence, in the rest of this subsection, we prove Proposition 7.12.

Pathsplitwidth The first step of the proof of Proposition 7.12 is to rephrase the bound on pathwidth, arity, and degree, in terms of a bound on the performance of variable orderings. Intuitively, a good variable ordering is one which does not *split* too many clauses. Formally:

Definition 7.13 Let $H = (V, E)$ be a hypergraph, and $\mathbf{v} = v_1, \ldots, v_{|V|}$ be an ordering on the variables of V. For $1 \leqslant i \leqslant |V|$, we let $\mathrm{Split}_i(\mathbf{v}, H)$ be the set of hyperedges e of H that contain both a variable at or before v_i, and a variable strictly after v_i, i.e., $\mathrm{Split}_i(\mathbf{v}, H) := \{e \in E \mid \exists l \in \{1, \ldots, i\} \text{ and } \exists r \in \{i+1, \ldots, |V|\} \text{ such that } \{v_l, v_r\} \subseteq e\}$. Note that $\mathrm{Split}_{|V|}(\mathbf{v}, H)$ is always empty.

The *pathsplitwidth* of $\mathbf{v}$ relative to H is the maximum size of the split, formally, $\mathrm{psw}(\mathbf{v}, H) := \max_{1 \leqslant i \leqslant |V|} |\mathrm{Split}_i(\mathbf{v}, H)|$. The *pathsplitwidth* $\mathrm{psw}(H)$ of H is then the minimum of $\mathrm{psw}(\mathbf{v}, H)$ over all variable orderings $\mathbf{v}$ of V.

In other words, $\mathrm{psw}(H)$ is the smallest integer $n \in \mathbb{N}$ such that, for any variable ordering $\mathbf{v}$ of the nodes of H, there is a moment at which n hyperedges of H are split by $(\mathbf{v}_{\leqslant \mathbf{i}}, \mathbf{v}_{>\mathbf{i}})$, in the sense of Definition 7.7. We note that the pathsplitwidth of H is exactly the *linear branch-width* [39] of the dual hypergraph of H, but we introduced pathsplitwidth because it fits our proofs better. This being said, the definition of pathsplitwidth is also reminiscent of pathwidth, and we can indeed connect the two (up to a factor of the arity):

Lemma 7.14 *For any hypergraph H, we have* $\mathrm{pw}(H) \leqslant \mathrm{arity}(H) \times \mathrm{psw}(H)$.

Proof Let $H = (V, E)$ be a hypergraph, and let $\mathbf{v}$ be an enumeration of the nodes of H witnessing that H has pathsplitwidth $\mathrm{psw}(H)$. We will construct a path decomposition of H of width $\leqslant \mathrm{arity}(H) \times \mathrm{psw}(H)$. Consider the path $P = b_1, \cdots, b_{|V|}$ and the labeling function λ where $\lambda(b_i) := \{v_i\} \cup \bigcup \mathrm{Split}_i(\mathbf{v}, H)$ for $1 \leqslant i \leqslant |V|$. Let us show that (P, λ) is a path decomposition of H: once this is established, it is clear that its width will be $\leqslant \mathrm{arity}(H) \times \mathrm{psw}(H)$.

First, we verify the occurrence condition. Let $e \in E$. If e is a singleton $\{v_i\}$ then e is included in b_i. Now, if $|e| \geq 2$, then let v_i be the first element of e enumerated by $\mathbf{v}$. We have $e \in \mathrm{Split}_i(\mathbf{v}, H)$, and therefore e is included in b_i.

Second, we verify the connectedness condition. Let v be a vertex of H, then by definition $v \in b_i$ iff $v = v_i$ or there exists $e \in \mathrm{Split}_i(\mathbf{v}, H)$ with $v \in e$. We must show that the set T_v of the bags that contain v forms a connected subpath in P. To show

this, first observe that for every $e \in E$, letting $\text{Split}(e) = \{v_i \mid 1 \leqslant i < |V| \wedge e \in \text{Split}_i(\mathbf{v}, H)\}$, then $\text{Split}(e)$ is clearly a connected segment of $\mathbf{v}$. Second, note that for every e with $v \in e$, then either $v \in \text{Split}(e)$ or v and the connected subpath $\text{Split}(e)$ are adjacent (in the case where v is the last vertex of e in the enumeration). Now, by definition T_v is the union of the $b_{v'}$ for $v' \in \text{Split}(e)$ with $v \in e$ and of b_i, so it is a union of connected subpaths which all contain b_i or are adjacent to it: this establishes that T_v is a connected subpath, which shows in turn that (T, λ) is a path decomposition, concluding the proof. □

For completeness with the preceding result, we note that the following also holds, although we do not use it (the proof is in the extended version of [5]):

Lemma 7.15 *For any hypergraph H, it is the case that* $\text{psw}(H) \leqslant \text{degree}(H) \times (\text{pw}(H) + 1)$.

We are finally ready to prove Proposition 7.12:

Proof (of Proposition 7.12) Let φ be the monotone CNF (resp., monotone DNF) on variables V having pathwidth $\geq k$, a be its arity, d its degree, and let $\mathbf{v}$ be a variable ordering of V. Remember that we identify φ with its associated hypergraph. By Lemma 7.14, the pathsplitwidth k' of φ is $\geq \frac{k}{a}$. By definition of pathsplitwidth, there exists $1 \leqslant i \leqslant |V|$ such that $\text{Split}_i(\mathbf{v}, \varphi) \geq k'$. Let $X' := \mathbf{v}_{\leqslant \mathbf{i}}$ and $Y' := \mathbf{v}_{>\mathbf{i}}$, and let $K' := \text{Split}_i(\mathbf{v}, \varphi)$. Then by definition, K' is split by (X', Y'). Hence by Proposition 7.8, letting $n := \left\lfloor \frac{k'}{a^2 \times d^2} \right\rfloor \geq \left\lfloor \frac{k}{a^3 \times d^2} \right\rfloor$, we can find $X \subseteq X'$ and $Y \subseteq Y'$ of size n and a valuation ν of $V \setminus (X \cup Y)$ such that $\nu(\varphi) = \text{SCOV}_n(X, Y)$ (resp., $\nu(\varphi) = \text{SINT}_n(X, Y)$), which is what we wanted. □

7.3 Proof of Theorem 7.2: Treewidth and SDNNFs

In this section we show our general lower bound relating the treewidth of CNFs/DNFs to the width of equivalent (d)-SDNNFs. We proceed similarly to the previous section, and start by showing the analogue of Proposition 7.12 for treewidth and v-trees:

Proposition 7.16 *Let φ be a monotone CNF (resp., monotone DNF) of treewidth $\geq k$ on variables V. Then, for any v-tree T of V, there exist disjoint subsets X, Y of V such that T cuts (X, Y) and a valuation ν of $V \setminus (X \cup Y)$ such that $\nu(\varphi) = \text{SCOV}_l(X, Y)$ (resp., $\text{SINT}_l(X, Y)$) for some $l \geq \frac{k}{3a^3d^2}$.*

This allows us to extend Theorem 6.5 to arbitrary monotone CNFs/DNFs and to prove Theorem 7.2:

Proof (of Theorem 7.2) Let (D, T, ρ) be a complete SDNNF (resp., complete d-SDNNF) on V computing φ. By Proposition 7.16, there exist disjoint subsets X, Y of V such that T cuts (X, Y) and a valuation ν of $V \setminus (X \cup Y)$ such that $\nu(\varphi) = \text{SCOV}_l(X, Y)$ (resp., $\text{SINT}_l(X, Y)$) for $l \geq \frac{k}{3a^3d^2}$. By Lemma 3.15, there exists a

complete SDNNF (resp., complete d-SDNNF) (D', T', ρ') on variables $X \cup Y$ computing $\mathrm{SINT}_l(X, Y)$ whose width is no greater than that of (D, T, ρ), and such that T' is a reduction of T. Now, it is clear that T' still cuts (X, Y) (by definition of T' being a reduction of T), so that by Theorem 6.5 the width of (D', T', ρ'), and hence that of (D, T, ρ), is $\geq 2^{\frac{k}{3a^3d^2}} - 1$. □

Hence, in the rest of this subsection, we prove Proposition 7.16.

Treesplitwidth Informally, treesplitwidth is to v-trees what pathsplitwidth is to variable orders: it bounds the "best performance" of any v-tree.

Definition 7.17 Let $H = (V, E)$ be a hypergraph, and T be a v-tree over V. For any node n of T, we define $\mathrm{Split}_n(T, H)$ as the set of hyperedges e of H that contain both a variable in T_n and one outside T_n (recall that T_n denotes the subtree of T rooted at n). Formally $\mathrm{Split}_n(T, H)$ is defined as the following set of hyperedges:

$$\{e \in E \mid \exists v_\mathrm{i} \in \mathsf{Leaves}(T_n) \text{ and } \exists v_\mathrm{o} \in \mathsf{Leaves}(T \setminus T_n) \text{ such that } \{v_\mathrm{i}, v_\mathrm{o}\} \subseteq e\}$$

The *treesplitwidth* of T relative to H is $\mathrm{tsw}(T, H) := \max_{n \in T} |\mathrm{Split}_n(T, H)|$. The *treesplitwidth* $\mathrm{tsw}(H)$ of H is then the minimum of $\mathrm{tsw}(T, H)$ over all v-trees T of V.

We note that the treesplitwidth of H is exactly the *branch-width* [45] of the dual hypergraph of H, but treesplitwidth is more convenient for our proofs. As with pathsplitwidth and pathwidth (Lemma 7.14), we can bound the treewidth of a hypergraph by its treesplitwidth:

Lemma 7.18 *For any hypergraph H, we have* $\mathrm{tw}(H) \leqslant 3 \times \mathrm{arity}(H) \times \mathrm{tsw}(H)$.

Proof Let $H = (V, E)$ be a hypergraph, and T a v-tree over V witnessing that H has treesplitwidth $\mathrm{tsw}(H)$. We will construct a tree decomposition T' of H of width $\leqslant 3 \times \mathrm{arity}(H) \times \mathrm{tsw}(H)$. The skeleton of T' is the same as that of T. Now, for each node $n \in T$, we call b_n the corresponding bag of T', and we define the labeling $\lambda(b_n)$ of b_n.

If n is an internal node of T with children n_l, n_r (recall that v-trees are assumed to be binary), then we define $\lambda(b_n) := \bigcup \mathrm{Split}_n(T, H) \cup \bigcup \mathrm{Split}_{n_l}(T, H) \cup \bigcup \mathrm{Split}_{n_r}(T, H)$, and if n is a variable $v \in V$ (i.e., n is a leaf of T) then $\lambda(b_n) := \{v\}$. It is clear that the width of P is $\leqslant \max(3 \times \mathrm{arity}(H) \times \mathrm{tsw}(H), 1) - 1 \leqslant 3 \times \mathrm{arity}(H) \times \mathrm{tsw}(H)$.

The occurrence condition is verified: let e be an edge of H. If e is a singleton edge $\{v\}$ then it is included in b_v. If $|e| \geq 2$ then there must exists a node $n \in T$ such that $e \in \mathrm{Split}_n(T, H)$. If n is an internal node of T then $e \subseteq \bigcup \mathrm{Split}_n(T, H) \subseteq b_n$, and if n is a leaf node of T then it must have a parent p (since e is split), and $e \subseteq \bigcup \mathrm{Split}_n(T, H) \subseteq b_p$.

Connectedness is proved in the same way as in the proof of Lemma 7.14: for a given vertex $v \in V$, the nodes of T where each edge e containing v is split is a connected subtree of T without its root node: more precisely, they are all the ancestors of a leaf in e strictly lower than their the least common ancestor. Adding the missing root to each such subtree and unioning them all will result in the subtree of all ancestors of a vertex adjacent to v (included v itself) up to their least common ancestor a. Consequently, the set of nodes of T' containing v is a connected subtree of T', rooted in b_a. □

We are now ready to prove Proposition 7.16, similarly to the way we proved Proposition 7.12:

Proof (of Proposition 7.16) Let φ be the monotone CNF (resp., monotone DNF) on variables V having treewidth $\geq k$, a be its arity, d its degree, and let T be a v-tree of V. By Lemma 7.18, the treesplitwidth k' of φ is $\geq \frac{k}{3a}$. By definition of treesplitwidth, there exists $n \in T$ such that $\left|\mathrm{Split}_n(T, \varphi)\right| \geq k'$. Let X' be the variables in T_n, let Y' be the variables outside T_n, and let $K' := \mathrm{Split}_n(T, \varphi)$. Then by definition, K' is split by (X', Y'). Hence by Proposition 7.8, letting $n := \left\lfloor \frac{k'}{a^2 \times d^2} \right\rfloor \geq \left\lfloor \frac{k}{3a^3 \times d^2} \right\rfloor$, we can find $X \subseteq X'$ and $Y \subseteq Y'$ of size n and a valuation ν of $V \setminus (X \cup Y)$ such that $\nu(\varphi) = \mathrm{SCOV}_n(X, Y)$ (resp., $\nu(\varphi) = \mathrm{SINT}_n(X, Y)$), which is what we wanted. □

8 Lower Bounds for Unstructured Classes

Theorem 7.2 gives an exponential lower bound on the size of *structured* DNNFs computing monotone CNF formulas of treewidth k. Intuitively, the high treewidth of the CNF makes it possible to find a large instance of SCOV for some set of variables that are cut by the v-tree, and this implies a lower bound on the size of the SDNNF. However, this argument crucially depends on the fact that the whole circuit is structured by the same v-tree.

It turns out that we can nevertheless extend our exponential lower bound on monotone CNF of treewidth k; but this requires a completely different proof technique as we cannot isolate a single bad partition of variables anymore. As in the rest of our work, the same argument applies to decision diagrams with pathwidth.

As in the previous sections, our lower bounds in this section will apply to so-called *complete* circuits and decision diagrams. However, the definitions of completeness in Section 3.4 were only given for structured classes. We now give these missing definitions:

Definition 8.1 We say that an nFBDD is *complete* if, for any root-to-sink path π, all variables are tested along π, i.e., occur as the label of a node of π. For DNNFs, recalling the definition of a *trace* (see Definition 3.2), we say that a DNNF is *complete* if, for any trace Ξ starting at the output gate, all variable gates are in Ξ.

Observe that a complete nOBDD is indeed complete when seen as an nFBDD, and a complete SDNNF is also complete when seen as an DNNF.

Our main results in this section are the following analogues of Corollaries 7.5 and 7.6, where the structuredness assumption is lifted:

Theorem 8.2 *For any monotone CNF φ of constant arity and degree, the size of the smallest complete nFBDD computing φ is $2^{\Omega(\mathrm{pw}(\varphi))}$.*

Theorem 8.3 *For any monotone CNF φ of constant arity and degree, the size of the smallest complete DNNF computing φ is $2^{\Omega(\mathrm{tw}(\varphi))}$.*

Like in the previous section, we can in fact lift the completeness assumption because one can make any nFBDD or any DNNF complete by only increasing its size with a factor linear in the number of variables:

Lemma 8.4 *For any nFBDD (resp. DNNF) D on variables set V, there exists an equivalent complete nFBDD (resp. DNNF) of size at most $(|V| + 1) \times |D|$.*

The proof of Lemma 8.4 for nFBDD can be straightforwardly adapted from [52, Lemma 6.2.2] where it is shown for FBDD. As for Lemma 8.4 for DNNF, it can be shown similarly to the way that DNNF are made *smooth* in [26] (after Definition 4).

Combining Lemma 8.4 with Theorems 8.2 and 8.3 allows us to remove the completeness assumption as we did for Corollary 7.5 and 7.6 (since $(|V| + 1) \times |D| \leqslant |D|^2$, and because $\sqrt{2^{\Omega(k)}}$ is also $2^{\Omega(k)}$):

Corollary 8.5 *For any monotone CNF φ of constant arity and degree, the size of the smallest nFBDD computing φ is $2^{\Omega(\mathrm{pw}(\varphi))}$. Likewise, the size of the smallest DNNF computing φ is $2^{\Omega(\mathrm{tw}(\varphi))}$.*

We note that, in contrast with our results in Sections 6 and 7, the above results *only* apply to CNFs. By contrast, for DNFs, there is no hope of showing a lower bound on DNNFs, because any DNF is in particular a DNNF. A natural analogue would be a lower bound on *d-DNNFs* representations of DNFs, but this is a long-standing open problem in knowledge compilation [14, 29].

Corollary 8.5 generalizes a lower bound of Razgon [44] where he constructs a family $(F_n)_{n\in\mathbb{N}}$ of monotone 2CNFs such that for every n, F_n has n variables and treewidth k. He proves that every nFBDD computing F_n has size at least $n^{\Omega(k)}$. One can actually observe that F_n has pathwidth $\Omega(k \log(n))$, making Razgon's lower bound a consequence of Corollary 8.5. This observation is actually crucial in Razgon's reasoning and this is exactly this fact that makes his proof works. Our lower bound is more general though as it works for *any* monotone CNF and can be lifted to treewidth and DNNF. The proof technique is however roughly the same and rely on a similar technical result, Lemma 8.6 here, which corresponds to [44, Theorem 4]. We give a simpler and more generic presentation of the proof.

The proofs of Theorems 8.2 and 8.3 follow the same structure. We will first present the proof of Theorem 8.2 and then explain how the argument adapts from nFBDDs to DNNFs.

Both results rely on a bound on the number of satisfying valuations that a rectangle can cover when its underlying partition splits (remember Definition 7.7) a large number of clauses. The result can be understood as a variant of Proposition 7.8, coupled with a generalization of the notion of *fooling sets* in the proof of Theorem 6.9.

Lemma 8.6 *Let φ be a monotone CNF with variable set V, arity a and degree d. Let X, Y be a partition of V and R be an (X, Y)-rectangle such that $R \Rightarrow \varphi$. Assume that there exists a set K' of clauses of φ such that K' is split by (X, Y). Let $n := \left\lfloor \frac{|K'|}{a^2 \times d^2} \right\rfloor$. Then we can bound the number of satisfying valuations of R as follows: $\#R \leqslant (1 + \alpha_{d,a})^{-n} \times \#\varphi$ where $\alpha_{d,a} = 2^{-a^2 d} > 0$.*

Proof We start by extracting a subset $K \subseteq K'$ of size n from K' such that K is an independent set of G_φ, the exclusion graph of φ, as in the beginning of the proof of Proposition 7.8.

Let C be a clause of K. We denote by $C \cap X$ the (disjunctive) clause formed of the variables in X that occur in C. We claim that one of the following holds:

- every satisfying valuation ν of R satisfies $C \cap X$; or
- every satisfying valuation ν of R satisfies $C \cap Y$.

Indeed, assume toward a contradiction that there exist two satisfying valuations ν_1, ν_2 of R such that ν_1 does not satisfy $C \cap X$ and ν_2 does not satisfy $C \cap Y$. Consider now the valuation $\nu := \nu_{1|X} \cup \nu_{2|Y}$. As R is a rectangle, it is easy to see that ν should satisfy R, hence φ. However, by definition ν does not satisfy C, hence it does not satisfy φ, a contradiction. We point out here that this claim would not hold if we had a DNF formula instead of CNF.

Thus, for every clause $C \in K$, all satisfying valuations of R satisfy either $C \cap X$ or $C \cap Y$. Let K_X and K_Y be a partition of K such that all clauses in K_X (resp., in K_Y) are such that all satisfying valuations of R satisfy $C \cap X$ (resp., $C \cap Y$): if there is any clause such that both conditions hold, assign it to K_X or to K_Y arbitrarily.

This definition ensures that every satisfying valuation ν of R satisfies $C \cap X$ for each clause $C \in K_X$ and that ν satisfies $C \cap Y$ for each clause $C \in K_Y$; what is more, as $R \Rightarrow \varphi$, the valuation ν must also satisfy all clauses in φ. This means that we have $R \Rightarrow \psi(K_X, K_Y)$ where ψ is defined as follows (up to minimization, i.e., removing clauses that are supersets of other clauses):

$$\psi(K_X, K_Y) := \varphi \cup \{C \cap X \mid C \in K_X\} \cup \{C \cap Y \mid C \in K_Y\}.$$

We will now show that $\#\psi(K_X, K_Y) \leqslant (1 + \alpha_{d,a})^{-n} \times \#\varphi$. This is enough to conclude the proof since $R \Rightarrow \psi(K_X, K_Y)$, that is, $\#R \leqslant \#\psi(K_X, K_Y)$.

The proof is by induction on the size of $K_X \cup K_Y$. More precisely, we will show that given $C \in K_X$, it holds:

$$\#\psi(K_X, K_Y) \leqslant (1 + \alpha_{d,a})^{-1} \times \#\psi(K_X \setminus \{C\}, K_Y). \tag{*}$$

The same is true for $C \in K_Y$, but as the argument is symmetric, we only explain it for K_X. This will be enough to show that $\#\psi(K_X, K_Y) \leqslant (1 + \alpha_{d,a})^{-n} \times \#\varphi$, since

iteratively applying (*) to K_X and K_Y until $K_X \cup K_Y$ is empty, and recalling that $K_X \cup K_Y = K$, we get:

$$\#\psi(K_X, K_Y) \leqslant (1+\alpha_{d,a})^{-(|K_X \cup K_Y|)} \times \#\psi(\emptyset, \emptyset) = (1+\alpha_{d,a})^{-n} \times \#\varphi.$$

Hence, let us fix $C \in K_X$ and show (*). To do so, we will consider the satisfying valuations of $\psi(K_X \setminus \{C\}, K_Y)$ and distinguish those that happen to satisfy $C \cap X$ and those who do not. (Note that these valuations are not necessarily satisfying valuations of R.) Specifically, let us call S the set of satisfying valuations of $\psi(K_X \setminus \{C\}, K_Y)$ which do not satisfy $C \cap X$. Observe that all the other satisfying valuations of $\psi(K_X \setminus \{C\}, K_Y)$ also satisfy $\psi(K_X, K_Y)$. Hence, we have:

$$\#\psi(K_X \setminus \{C\}, K_Y) = |S| + \#\psi(K_X, K_Y). \tag{**}$$

This implies that to show (*), in combination with (**) it suffices to show:

$$\#\psi(K_X, K_Y) \leqslant \frac{|S|}{\alpha_{d,a}}. \tag{***}$$

To show (***), we define a function f from the satisfying valuations of $\psi(K_X, K_Y)$ to S, and show that each valuation in S has at most $1/\alpha_{d,a}$ preimages by f. To define f, let us map every satisfying valuation ν of $\psi(K_X, K_Y)$ to $\nu' = f(\nu)$ defined as follows:

- set $\nu'(x) := 0$ for every $x \in C \cap X$,
- set $\nu'(z) := 1$ for every $z \in V \setminus (C \cap X)$ that appears together in a clause with a variable of $C \cap X$;
- set $\nu'(z') := \nu(z')$ for every other z'. □

We first show that f is indeed a function that maps to S, in other words:

Claim For any satisfying valuation ν of $\psi(K_X, K_Y)$, letting $\nu' := f(\nu)$, we have $\nu' \in S$.

Proof First, by definition, ν' does not satisfy $C \cap X$ since $\nu'(x) = 0$ for every $x \in C \cap X$. Now we have to show that ν' also satisfies $\psi(K_X \setminus \{C\}, K_Y)$. First observe that ν' satisfies C since C has at least one variable y in Y and by definition (precisely, by the second item), $\nu'(y) = 1$.

Now let C' be a clause of φ. Then either C' was satisfied by ν thanks to a variable z such that $\nu'(z) = \nu(z) = 1$. In this case, C' is still satisfied by ν'. Otherwise, it may be that there is an $x \in C \cap X$ such that $x \in C'$ and $\nu(x) = 1$. In this case, it is not guaranteed anymore that C' is satisfied by ν'. However, since φ is minimized, there must exist $z \in C'$ such that $z \notin C$. By definition, z appears in the same clause C' as x, meaning that $\nu'(z) = 1$ (again by the second item) and thus C' is satisfied by ν'.

Finally let $C' \in (K_X \setminus \{C\})$ (the case $C' \in K_Y$ is similar). We have to check that $C' \cap X$ is satisfied by ν' as this clause appears in $\psi(K_X \setminus \{C\}, K_Y)$. Since C' is also in $\psi(K_X, K_Y)$, ν satisfies C', thus, there exists a variable y of C' such that $\nu(y) = 1$. Now, we claim that $\nu'(y) = 1$ too. Indeed, by definition of $K = K_X \cup K_Y$, all clauses of K form an independent set in G_φ. Thus, C' does not share any variable with C. In

other words, $\nu'(y) = \nu(y) = 1$ (by the third item), that is, ν' satisfies C'. Thus, we have established the claim.

Now it remains to count the maximal number of preimages of a valuation $\nu' \in S$ by f. To define ν' from ν we only changed the valuation of variables that occur in C or in a clause with a non-empty intersection with C. There are at most a^2d such variables: at most a variables in C, and for each of these variables, we have at most $d-1$ other clauses of size a incident to it, that is, we change the value of at most $a + a(d-1)(a-1) \leqslant da^2$ variables. Thus, given a valuation ν' of S, there are at most $2^{a^2d} = 1/\alpha_{d,a}$ satisfying valuations of $\psi(K_X \setminus \{C\}, K_Y)$ whose image by f is ν'. In other words, we have shown (***).

Combining this inequality with (**), we get:

$$\#\psi(K_X, K_Y) \leqslant (1+\alpha_{d,a})^{-1} \times \#\psi(K_X \setminus \{C\}, K_Y).$$

This concludes the proof of Lemma 8.6. □

Now that we have proved our technical lemma, we are ready to prove our main result on nFBDDs, Theorem 8.2:

Proof (of Theorem 8.2) Let V be the variables of φ, let $k := \mathrm{pw}(\varphi)$, let a be the arity of φ, and let d be the degree of φ. Let D be a complete nFBDD computing φ.

Let ν be a satisfying valuation of φ. By definition of D, there exists a root-to-sink path π in D compatible with ν. As we assumed D to be complete, this path induces a total order on V. By Lemma 7.14, since φ is of pathwidth k, it is also of pathsplitwidth at least k/a. In other words, we know that there exists a set of clauses K' of size at least k/a and a gate $g(\nu)$ in π such that every clause of K' has at least one variable tested before $g(\nu)$ in π and one variable tested after $g(\nu)$ in π. In other words, letting X (resp., Y) be the variables of V tested before $g(\nu)$ (resp., after $g(\nu)$), we have that K' is split by (X, Y).

We denote by $R_{g(\nu)}$ the set of satisfying valuations of φ having a compatible path going through $g(\nu)$. Observe now that, similarly to how we proved Theorem 6.7, $R_{g(\nu)}$ is an (X, Y)-rectangle such that $R_{g(\nu)} \Rightarrow \varphi$: this uses in particular the fact that, although all valuations of $R_{g(\nu)}$ may not test the variables in the same order, we know that all variables of X are tested before $g(\nu)$, and all variables of Y are tested after $g(\nu)$, thanks to the fact that each variable can be tested only once along root-to-sink paths in an nFBDD. Now, by Lemma 8.6, the number of valuations in $R_{g(\nu)}$ is $\leqslant (1+\alpha_{d,a})^{-n} \times \#\varphi$ where $n := \frac{k}{a^3d^2}$.

Now, observe that given $\nu \models \varphi$, we can always find such a gate $g(\nu)$ inducing a rectangle $R_{g(\nu)}$ such that $\nu \models R_{g(\nu)}$ and such that the previous inequality holds, i.e., $\left|R_{g(\nu)}\right|$ is small wrt $\#\varphi$. Let $S = \{g(\nu) \mid \nu \models \varphi\}$. We thus have: $\varphi = \bigvee_{g \in S} R_g$. Thus:

$$\begin{aligned}
\#\varphi &\leqslant \sum_{g \in S} \#R_g && (1)\\
&\leqslant |S|\,(1+\alpha_{d,a})^{-n} \times \#\varphi && (2)\\
&\leqslant |D|\,(1+\alpha_{d,a})^{-n} \times \#\varphi \text{ since } S \subseteq D. && (3)
\end{aligned}$$

Simplifying by $\#\varphi$ gives $|D| \geq (1+\alpha_{d,a})^n = 2^{\Omega(\mathrm{pw}(\varphi))}$ since d and a are constants. □

We have shown our result on nFBDDs, Theorem 8.2. We now explain how we can adapt the proof to DNNFs and show Theorem 8.3:

Proof (of Theorem 8.3) Let V be the variables of φ, let $k := \mathrm{tw}(\varphi)$, let a and d be the arity and degree of φ, and let D be a complete DNNF computing φ.

For any satisfying valuation ν of φ, we consider a trace Ξ of D starting at the output gate that witnesses that ν satisfies D. As D is complete, all variables of V occur in Ξ. By Lemma 7.18, we know that φ has treesplitwidth $\geq \frac{k}{3a}$. Hence, we can define a gate $g(\nu)$ of Ξ such that every clause of K' has at least one variable in a leaf which a descendant of $g(\nu)$ in Ξ (we call X the set of such variables) and one variable in a leaf which is not a descendant of $g(\nu)$ in Ξ (we call Y the set of such variables), i.e., K' is split by (X, Y).

We denote again by $R_{g(\nu)}$ the set of satisfying valuations of φ having a trace where $g(\nu)$ occurs. Similarly to how we proved Theorem 6.8, it is again the case that $R_{g(\nu)}$ is an (X, Y)-rectangle such that $R_{g(\nu)} \Rightarrow \varphi$: this again uses the fact that, even though the different traces using $g(\nu)$ may have a very different structure, decomposability ensures that the variables of X must occur as descendants of $g(\nu)$ in Ξ and the variables of Y must occur as non-descendants of $g(\nu)$ in Ξ. Now, Lemma 8.6 ensures that the number of valuations in $R_{g(\nu)}$ is $\leqslant (1 + \alpha_{d,a})^{-n} \times \#\varphi$ where $n := \frac{k}{3a^3d^2}$.

We conclude exactly as in Theorem 8.2 by considering the gates $g(\nu)$ for all satisfying valuations ν and writing the corresponding rectangle cover. This establishes the desired bound. □

9 Conclusion

We have shown tight connections between structured circuit classes and width measures on circuits. We constructively rewrite bounded-treewidth (resp., bounded-pathwidth) circuits to d-SDNNFs (resp., uOBDDs) in time linear in the circuit and singly exponential in the treewidth, and show matching lower bounds for arbitrary monotone CNFs or DNFs under degree and arity assumptions, also for CNFs in the unstructured case. Our upper bound results imply the tractability of several tasks (probability computation, enumeration, quantification, etc.) on bounded-treewidth and bounded-pathwidth circuits, whereas our lower bounds show that pathwidth and treewidth *characterize* compilability to these classes. Our lower bounds also have consequences for probabilistic query evaluation as described in the conference version [4].

Our work also raises a number of open questions. We leave to future work a more thorough study of the relationships between the knowledge compilation classes that we investigated, or their relationship to other classes such as *sentential decision diagrams* (SDD) [28] which we did not consider. Indeed, one interesting question is whether our upper bound result Theorem 4.2 could be modified to construct SDDs, or whether this is impossible and SDDs and d-SDNNFs can thus be separated. A related question would be to characterize the bounds on the compilation of bounded-pathwidth circuits to OBDDs (not uOBDDs): this can be done with doubly exponential complexity in the pathwidth by the results of [34, Corollary 2.13] but

it is unclear whether this is tight. Another intriguing question is whether we could improve our main lower bound of Theorem 4.2 by compiling to d-DNNFs that are not necessarily structured; could we then consider a less restrictive parameter than the treewidth of the original circuit?

In terms of our lower bounds, the main questions would be to investigate more general languages, e.g., where the arity or degree are not bounded, or where the functions are not monotone. There is also the question of proposing convincing width definitions for non-complete circuit formalisms, so as to remove all completeness assumptions from our results. Last, there is of course the tantalizing question of showing a lower bound for *unstructured* representations of DNF formulae, i.e., an analogue for DNF and d-DNNF of the results of Section 8, that would match the results shown in Section 7 for the structured case. This relates to the open problem in probabilistic databases of whether safe queries have tractable lineages [35, 38].

Acknowledgments We acknowledge Chandra Chekuri for his helpful comments at https://cstheory.stackexchange.com/a/38943/, as well as Stefan Mengel for pointing us to a notion of width for d-SDNNFs and suggesting a strengthening of our complexity upper bound in Theorem 4.2.

References

1. Amarilli, A., Bourhis, P., Jachiet, L., Mengel, S.: A circuit-based approach to efficient enumeration. In: ICALP (2017)
2. Amarilli, A., Bourhis, P., Monet, M., Senellart, P.: Combined tractability of query evaluation via tree automata and cycluits. In: ICDT (2017)
3. Amarilli, A., Bourhis, P., Senellart, P.: Provenance circuits for trees and treelike instances. In: ICALP (2015)
4. Amarilli, A., Bourhis, P., Senellart, P.: Tractable lineages on treelike instances: limits and extensions. In: PODS (2016)
5. Amarilli, A., Monet, M., Senellart, P.: Connecting width and structure in knowledge compilation. In: ICDT (2018)
6. Beame, P., Li, J., Roy, S., Suciu, D.: Lower bounds for exact model counting and applications in probabilistic databases. In: UAI (2013)
7. Beame, P., Li, J., Roy, S., Suciu, D.: Exact model counting of query expressions: Limitations of propositional methods. ACM Trans. Database Syst. (TODS) **42**(1), 1 (2017)
8. Beame, P., Liew, V.: New limits for knowledge compilation and applications to exact model counting. In: UAI (2015)
9. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **5**(6), 1305?-1317 (1996)
10. Bodlaender, H.L., Drange, P.G., Dregi, M.S., Fomin, F.V., Lokshtanov, D., Pilipczuk, M.: A c^k n 5-approximation algorithm for treewidth. SIAM J. Comput. **45**(2), 317?-378 (2016)
11. Bollig, B., Buttkus, M.: On the relative succinctness of sentential decision diagrams. arXiv:1802.04544 (2018)
12. Bollig, B., Wegener, I.: Complexity theoretical results on partitioned (nondeterministic) binary decision diagrams. In: MFCS (1997)
13. Bova, S.: SDDs are exponentially more succinct than OBDDs. In: AAAI (2016)
14. Bova, S., Capelli, F., Mengel, S., Slivovsky, F.: Knowledge compilation meets communication complexity. In: IJCAI (2016)
15. Bova, S., Slivovsky, F.: On compiling structured CNFs to OBDDs. In: International Computer Science Symposium in Russia, pp. 80–93. Springer (2015)
16. Bova, S., Szeider, S.: Circuit treewidth, sentential decision, and query compilation. In: PODS (2017)

17. Bryant, R.E.: On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. IEEE Trans. Comput. **40**(2), 205–213 (1991)
18. Bryant, R.E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput. Surv. **24**(3), 293–318 (1992)
19. Calí, A., Capelli, F., Razgon, I.: Non-FPT lower bounds for structural restrictions of decision DNNF. arXiv:1708.07767v1 (2017)
20. Capelli, F.: Structural restrictions of CNF-formulas: Applications to model counting and knowledge compilation. Ph.D. thesis, Université Paris-Diderot (2016)
21. Capelli, F.: Understanding the complexity of #SAT using knowledge compilation. In: LICS (2017)
22. Capelli, F., Mengel, S.: Tractable QBF by knowledge compilation. In: STACS (2019)
23. Capelli, F., Strozecki, Y.: Enumerating models of DNF faster: Breaking the dependency on the formula size. arXiv:1810.04006 (2018)
24. Creignou, N., Olive, F., Schmidt, J.: Enumerating all solutions of a Boolean CSP by non-decreasing weight. In: SAT (2011)
25. Darwiche, A.: Decomposable negation normal form. JACM **48**(4), 608–647 (2001)
26. Darwiche, A.: On the tractable counting of theory models and its application to truth maintenance and belief revision. J. Appl. Non-Classical Logics **11**(1-2), 11–34 (2001)
27. Darwiche, A.: A differential approach to inference in Bayesian networks. JACM **50**(3), 280–305 (2003)
28. Darwiche, A.: SDD: A new canonical representation of propositional knowledge bases. In: IJCAI (2011)
29. Darwiche, A., Marquis, P.: A knowledge compilation map. JAIR **17**, 229–264 (2002)
30. Devadas, S.: Comparing two-level and ordered binary decision diagram representations of logic functions. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **12**(5), 722–723 (1993)
31. Fierens, D., den Broeck, G.V., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., Raedt, L.D.: Inference and learning in probabilistic logic programs using weighted Boolean formulas. TPLP **15**(3), 358–401 (2015)
32. Grohe, M., Marx, D.: On tree width, bramble size, and expansion. J. Combinatorial Theory Series B **99**(1), 218–228 (2009)
33. Jha, A.K., Olteanu, D., Suciu, D.: Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In: EDBT (2010)
34. Jha, A.K., Suciu, D.: On the tractability of query compilation and bounded treewidth. In: ICDT (2012)
35. Jha, A.K., Suciu, D.: Knowledge compilation meets database theory: Compiling queries to decision diagrams. TCS **52**(3), 403-?440 (2013)
36. Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. J. Royal Statistical Society Series B (1988)
37. Meinel, C., Theobald, T.: Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications. Springer Science & Business Media (2012)
38. Monet, M., Olteanu, D.: Towards deterministic decomposable circuits for safe queries. In: AMW (2018)
39. Nordstrand, J.A.: Exploring graph parameters similar to tree-width and path-width. University of Bergen, Master's thesis (2017)
40. Pipatsrisawat, K., Darwiche, A.: New compilation languages based on structured decomposability. In: AAAI (2008)
41. Pipatsrisawat, K., Darwiche, A.: A lower bound on the size of decomposable negation normal form. In: AAAI (2010)
42. Pipatsrisawat, T.: Reasoning with propositional knowledge: Frameworks for Boolean satisfiability and knowledge compilation. Ph.D. thesis, University of California (2010)
43. Razgon, I.: On OBDDs for CNFs of bounded treewidth. In: KR (2014)
44. Razgon, I.: No small nondeterministic read-once branching programs for CNFs of bounded treewidth. In: IPEC (2014)
45. Robertson, N., Seymour, P.: Graph minors. X. Obstructions to tree-decomposition. Journal of Combinatorial Theory Series B **52**(2), 153–190 (1991)
46. Sauerhoff, M.: Approximation of boolean functions by combinatorial rectangles. Theor. Comput. Sci. **301**(1–3), 45–78 (2003)
47. Sherstov, A.A.: Communication complexity theory: Thirty-five years of set disjointness. In: MFCS (2014)

48. Strozecki, Y.: Enumeration complexity and matroid decomposition, p. 7. Ph.D. Thesis, Paris (2010)
49. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic databases. Morgan & Claypool (2011)
50. Szeider, S.: On fixed-parameter tractable parameterizations of SAT. In: SAT (2004)
51. Wegener, I.: The complexity of Boolean functions. Wiley, New York (1991)
52. Wegener, I.: Branching programs and binary decision diagrams: Theory and applications. SIAM (2000)

Affiliations

Antoine Amarilli[1] · Florent Capelli[2,3] · Mikaël Monet[1,3] · Pierre Senellart[1,3,4]

Antoine Amarilli
antoine.amarilli@telecom-paristech.fr

Florent Capelli
florent.capelli@univ-lille.fr

Pierre Senellart
pierre.senellart@ens.fr

1 LTCI, Télécom ParisTech, Université Paris-Saclay, Paris, France
2 CRIStAL, Université de Lille, CNRS, Lille, France
3 Inria, Lille, Paris, France
4 DI ENS, ENS, CNRS, PSL University, Paris, France