Exact Model Counting of Query Expressions: Limitations of Propositional Methods

PAUL BEAME, University of Washington
JERRY LI, MIT
SUDEEPA ROY, Duke University
DAN SUCIU, University of Washington

We prove exponential lower bounds on the running time of the state-of-the-art exact model counting algorithms—algorithms for exactly computing the number of satisfying assignments, or the satisfying probability, of Boolean formulas. These algorithms can be seen, either directly or indirectly, as building *Decision-Decomposable Negation Normal Form (decision-DNNF)* representations of the input Boolean formulas. Decision-DNNFs are a special case of *d*-DNNFs where *d* stands for *deterministic*. We show that any knowledge compilation representations from a class (called DLDDs in this article) that contain decision-DNNFs can be converted into equivalent *Free Binary Decision Diagrams (FBDDs)*, also known as *Read-Once Branching Programs*, with only a quasi-polynomial increase in representation size. Leveraging known exponential lower bounds for FBDDs, we then obtain similar exponential lower bounds for decision-DNNFs, which imply exponential lower bounds for model-counting algorithms. We also separate the power of decision-DNNFs from *d*-DNNFs and a generalization of decision-DNNFs known as AND-FBDDs.

We then prove new lower bounds for FBDDs that yield exponential lower bounds on the running time of these exact model counters when applied to the problem of query evaluation in tuple-independent probabilistic databases—computing the probability of an answer to a query given independent probabilities of the individual tuples in a database instance. This approach to the query evaluation problem, in which one first obtains the lineage for the query and database instance as a Boolean formula and then performs weighted model counting on the lineage, is known as *grounded inference*. A second approach, known as *lifted inference* or *extensional query evaluation*, exploits the high-level structure of the query as a first-order formula. Although it has been widely believed that lifted inference is strictly more powerful than grounded inference on the lineage alone, no formal separation has previously been shown for query evaluation. In this article, we show such a formal separation for the first time. In particular, we exhibit a family of database queries for which polynomial-time extensional query evaluation techniques were previously known but for which query evaluation via grounded inference using the state-of-the-art exact model counters requires exponential time.

This is an extended and improved version of the papers titled "Lower Bounds for Exact Model Counting and Applications in Probabilistic Databases," which appeared in the proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI) 2013 [Beame et al. 2013], and "Model Counting of Query Expressions: Limitations of Propositional Methods," which appeared in the proceedings of the International Conference on Database Theory (ICDT) 2014 [Beame et al. 2014].
This research was partially supported by NSF Awards CCF-1217099, CCF-1524246, IIS-1115188, IIS-0911036, and IIS-0915054.
Authors' addresses: P. Beame and D. Suciu, University of Washington, Department of Computer Science and Engineering, Box 352350 Seattle, WA 98195-2350, USA; J. Li, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 32 Vassar Street Cambridge MA 02141, USA; S. Roy, Department of Computer Science, Duke University, Campus Box 90129, 308 Research Dr, Durham, NC 27708, USA. The bulk of the research was done at the University of Washington.

DOI: http://dx.doi.org/10.1145/2984632

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Languages—*Query languages*; F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes—*Relations among complexity measures*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation and Methods—*Representation languages*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Model counting, probabilistic databases, knowledge compilation, FBDD, read-once branching programs, DNNF, lower bounds

ACM Reference Format:
Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. 2017. Exact model counting of query expressions: Limitations of propositional methods. ACM Trans. Database Syst. 42, 1, Article 1 (February 2017), 46 pages. DOI: http://dx.doi.org/10.1145/2984632

1. INTRODUCTION

Model counting is the problem of computing the number, $\#\Phi$, of satisfying assignments of a Boolean formula Φ. In this article, we are concerned with the weighted version of model counting, which is the same as the probability computation problem on independent random variables. Although model counting is #P-hard in general (even for formulas where satisfiability is easy to check [Valiant 1979]), there have been major advances in practical algorithms that compute exact, weighted model counts. Exact model counting for propositional formulas are based on extensions of the Davis-Putnam-Logemann-Loveland (DPLL) family of algorithms [Davis and Putnam 1960; Davis et al. 1962] that were originally designed for satisfiability search. We focus on understanding the limitations of these exact model counting algorithms, both in general and, even more importantly, on their limitations in applications of probabilistic inference.

We are motivated by *tuple-independent probabilistic databases* [Suciu et al. 2011]. A probabilistic database is a relational database where each tuple t is annotated with a probability $p_t \in [0, 1]$. It defines a probability space over all subsets of the database, where an outcome, called a *possible world*, is obtained by randomly and independently including every tuple t with probability p_t. The query evaluation problem in a probabilistic database is the following: For a fixed Boolean query Q, given a probabilistic database D, compute the probability $\Pr[Q(D)]$ that the query is true in a randomly chosen possible world.

Query evaluation over probabilistic databases is a special case of weighted model counting. Indeed, one can associate to each query Q and probabilistic database D a Boolean formula Φ such that Q's probability equals that of Φ. The formula Φ is called the *lineage* of Q on D (reviewed in Definition 2.5). Intuitively, it states which tuples must be present in a possible world for Q to be true. For example, if $Q = \exists x(R(x) \wedge S(x))$ and the database instance has two relations $R = S = \{1, 2, \ldots, n\}$, then the lineage is $(R(1) \wedge S(1)) \vee (R(2) \wedge S(2)) \vee \cdots (R(n) \wedge S(n))$, in essence saying that at least one pair of tuples $R(k), S(k)$ must be present in the possible world for Q to be true. Therefore, an obvious way to compute $\Pr[Q(D)]$ is to first compute the lineage Φ and then perform model counting on Φ. We call this the *grounded inference* approach. In general, grounded inference may be inefficient, in part because Φ is a polynomially large propositional formula (measured in data complexity where Q is fixed and D is variable) that depends on many tuples in the database, while the first-order query Q is much smaller. As we demonstrate, this is only part of the story.

Several alternative approaches to grounded inference have been proposed, collectively called *lifted inference* in the statistical relational model literature [Jaeger and Van den Broeck 2012] and *extensional query evaluation* in probabilistic databases [Suciu et al. 2011]. They compute $\Pr[Q(D)]$ by exploiting the concise, first-order expression of Q, and either postpone, or avoid completely, computing the lineage. There is no consensus on a precise definition of lifted inference. Early definitions asked for the

entire inference to be performed at the first-order level, without any grounding at all, and therefore in time that is constant in the size of the domain of the database instance. This only makes sense when weights are attached to relations, so any two ground atoms of the same relation contribute the same weight (or probability) to a model, which represents a very restricted model, called symmetric probabilistic databases [Jaeger and Van den Broeck 2012; Beame et al. 2015]. We are interested only in asymmetric databases in this article, and most lifted inference techniques developed in the literature apply to this general case. Their common characteristic is that they exploit the high level structure of the first-order formula in order to guide the probabilistic inference. Van den Broeck [2011] defines *domain lifted inference* as any inference algorithm that runs in polynomial time in the size of the domain, thus allowing the algorithm to refer freely to the first-order formula or its grounding. For example, consider the query $\exists x \exists y(R(x) \wedge S(y))$. Grounded inference would first expand it to the Boolean formula $\Phi = \bigvee_{i,j}(R(i) \wedge S(j))$ and then compute its probability, which is a challenging task for today's DPLL-based model counters (see, e.g., Theorem 3.2), while lifted inference would first rewrite the first-order formula as $(\exists x R(x)) \wedge (\exists y S(y))$ and then compute $\Pr[Q(D)]$ as the product of two probabilities, $\Pr[\exists x R(x)]$ and $\Pr[\exists y S(y)]$, both computable in linear time in the size of the database.

Since lifted inference methods can still use grounding when needed, they are, in theory, at least as powerful as grounded inference. A natural question is whether they are strictly more powerful. While there have been examples in other contexts where provable separations have been shown [Sabharwal 2009], no formal separation has previously been shown in the context of query evaluation. We show such a formal separation for the first time. We describe a class of Unions of Conjunctive Queries (UCQ) Q (i.e., first-order formulas restricted to the connectors $\exists, \vee, \wedge$) whose probability can be computed in polynomial time, yet where any DPLL-based modern model counting algorithm takes provably exponential time. Thus, grounded inference is strictly weaker than lifted inference, at least on the class of queries considered here. Note that our result is a positive statement about lifted inference, and for that reason we shall avoid defining lifted inference formally in this article. Instead, we will only describe an algorithm that computes $\Pr[Q(D)]$. Since this algorithm runs in polynomial time, this algorithm is domain lifted according to Van den Broeck's formal definition [Van den Broeck 2011], and it can also be considered "lifted" according to the more informal criteria in the literature, of performing inference on the first-order expressions. However, our result is a negative statement about DPLL-based grounded algorithms, and this requires a careful definition. The cornerstone of our argument is a new simulation of DPLL-based algorithms to a simple representations of Boolean functions for which lower bounds have previously been shown for specific functions and for which we prove new lower bound needed in our context. This allows us to prove that any DPLL-based algorithm takes exponential time on the Boolean formula representing the lineage of any UCQ query in our class, while a simple lifted inference algorithm computes the same probability in polynomial time. This proves that current propositional model counting techniques are strictly weaker than their lifted counterparts on every query in our class.

Lower Bounds on State-of-the-Art Approaches to Exact Model Counting

Modern exact model counting algorithms use a variety of techniques; see Gomes et al. [2009] for a survey. Many are based on extensions of backtracking search using the *DPLL* family of algorithms [Davis and Putnam 1960; Davis et al. 1962] that were originally designed for satisfiability search. In the context of model counting (and related problems of exact Bayesian inference), extensions include caching the results of solved sub-problems [Majercik and Littman 1998], dynamically decomposing residual formulas into components (Relsat [Bayardo et al. 2000]) and caching their counts

[Bacchus et al. 2003], and applying dynamic component caching together with conflict-directed clause learning to further prune the search (Cachet [Sang et al. 2004] and sharpSAT [Thurley 2006]).

The other major approach, known as *knowledge compilation*, is to convert the input formula into a representation of the Boolean function that the formula defines and from which the model count can be computed efficiently in the size of the representation [Darwiche 2001a, 2001b; Huang and Darwiche 2007; Muise et al. 2012].

It is known that the trace of a DPLL-based algorithm is a type of knowledge representation called a Decision-Decomposable Negation Normal Form *(decision-DNNF)* [Huang and Darwiche 2005, 2007], which is a syntactic subclass of *d*-DNNF representations [Darwiche 2001b; Darwiche and Marquis 2002]. This has been noted both in the work on `c2d` based on component caching [Huang and Darwiche 2007] and in that on Dsharp based on sharpSAT [Muise et al. 2012]. While the details of various DPLL-based algorithms may differ, all the methods for exact model counting surveyed in Gomes et al. [2009] have a trace that is a decision-DNNF and thus can be converted to knowledge compilation algorithms that produce decision-DNNF representations, without any significant increase in their running time.[1]

In this article, we prove exponential lower bounds on the size of decision-DNNFs for natural classes of formulas. Therefore our results immediately imply exponential lower bounds for the running time of all DPLL-based modern exact model counting algorithms. These bounds are unconditional—they do not depend on any unproved complexity-theoretic assumptions.

Our bounds apply to very simple classes of Boolean formulas, which occur frequently both in uncertainty reasoning and in probabilistic inference. We also show that our lower bounds extend to the evaluation of the properties of a large class of database queries, which have been studied in the context of probabilistic databases.

We derive our exponential lower bounds by showing how to translate any decision-DNNF to an equivalent Free Binary Decision Diagrams *(FBDD)*, a less powerful representation for Boolean functions.[2] Our translation increases the size by at most a quasipolynomial amount: Every decision-DNNF of size N can be converted into an FBDD of size at most $N2^{\log^2 N}$ (Theorem 3.1). We can thus obtain many exponential lower bounds for exact model counting immediately by using well-established exponential lower bounds for FBDDs. This translation from decision-DNNFs to FBDDs is of independent interest: It is simple, and efficient, in the sense that it can be computed in time linear in the size of the output FBDD. In fact, it extends to a somewhat broader class of representations that we define and term *Decomposable Logic Decision Diagrams (DLDDs)*.

In the database context, it is necessary to handle a broader class of representations than decision-DNNFs. Our preliminary work [Beame et al. 2013] gave a quasipolynomial translation from decision-DNNFs to FBDDs, but decision-DNNFs only capture the traces of DPLL-based search algorithms on Conjunctive Normal Form (CNF) expressions, not the Disjunctive Normal Form (DNF) expressions that arise as lineages of a database queries. Our generalization of this result to DLDDs captures the traces of any natural extension of DPLL search to Boolean formulas (in CNF, DNF, or any other form).

[1]Recent work has suggested sentential decision diagrams (SDDs) as an alternative format for knowledge compilation [Darwiche 2011]; the code for SDD-based model counting was made available shortly after this research was completed [SDD 2014]. This raises the question of whether our main result also extends to inference algorithms whose trace is an SDD. In subsequent work, Beame and Liew [2015] answered this affirmatively by showing that SDD representations are exponentially less succinct than decision-DNNF representations for simple query lineage formulas for which lifted inference is in polynomial time.

[2]FBDDs are also known as *Read-Once Branching Programs (ROBPs or 1BPs)*.

It is interesting to note that with formula caching, but without dynamic component caching, the trace extensions of DPLL-based searches yield FBDDs rather than decision-DNNFs [Huang and Darwiche 2005]. Hence, the difference between FBDDs and decision-DNNFs is precisely the ability of the latter to take advantage of decompositions into connected components of subformulas of the formula being represented. Our conversion shows that these connected component decompositions can only provide quasipolynomial improvements in efficiency.

Complexity of Compilation. Efficiency for knowledge compilation depends on both the size of the representation and the time required to construct it. The time required to construct a decision-DNNF from an input Boolean formula may greatly exceed the size of the representation. An extreme example is that of an unsatisfiable Boolean formula: The function evaluates to the constant 0 and hence the decision-DNNF is of size 1, but checking unsatisfiability may take exponential time. Indeed, DPLL with caching and conflict-directed clause learning is a special case of resolution theorem proving [Beame et al. 2004]. There are large numbers of unsatisfiable formulas for which exponential lower bounds are known for every resolution refutation (see, e.g., Ben-Sasson and Wigderson [2001]), and hence this compilation process must be exponential for such formulas.[3] The same issues can arise in ruling out parts of the space of assignments for satisfiable formulas. However, we do not know of any lower bounds for this excess compilation time that directly apply to the kinds of simple highly satisfiable instances that we discuss in this article. Our lower bound on the runtime of DPLL-based algorithms exploits only the size of the representation and not the time to construct it.

Separating Lifted and Grounded Inference for Probabilistic Databases

We use our translation from decision-DNNFs to FBDDs to show a formal separation between the lifted and grounded inference for queries on probabilistic databases. For that we need two results.

First, we give a lower bound on the representation for a family of queries, called h_k, $k \geq 1$. These queries are interesting because they have been shown to be the simplest UCQ queries for which the probability computation is #P-hard [Dalvi and Suciu 2012]. Each such query is in FO^2, first-order logic restricted to two logical variables: Van den Broeck et al. [2014] have shown that the probability of any query in FO^2 can be computed in polynomial time on symmetric probabilistic databases, but, in this article, we study asymmetric databases, and here all queries h_k are #P-hard. Each of these queries has a simple lineage, which is a 2-DNF of size $O(n^2)$. We prove *unconditional* exponential lower bounds of the form $2^{\Omega(\sqrt{n})}$ on the sizes of decision-DNNF representations of these lineages, which is the first non-trivial decision-DNNF lower bound for h_k (Theorem 3.6).

In particular, we prove that any FBDD for the Boolean formula representing the lineage of h_k requires at least $(2^n - 1)/n$ size for a domain of size n. Together, Theorems 3.1 and 3.6 imply our lower bound of $2^{\Omega(\sqrt{n})}$ on the sizes of decision-DNNF representations of h_k. (We note that a lower bound on the size of the FBDD for h_1 was known previously [Jha and Suciu 2013], but that bound, $2^{\Omega(\log^2 n)}$, is insufficient to yield any decision-DNNF lower bound using our translation.) This lower bound improves on the bounds on FBDD sizes for the queries h_k as well as the separation between lifted and grounded inference stated in a preliminary version of this work [Beame et al. 2014] in which proofs were omitted or only sketched.

[3]DPLL with formula caching, but not clause learning, can be simulated by even simpler *regular* resolution, though in general it is not quite as powerful as regular resolution [Beame et al. 2010].

This implies that any DPLL-based algorithm for computing the probability of the (lineage of) h_k runs in exponential time. This result, however, while new, comes at no surprise, since all queries h_k were known to be #P-hard. For our second result, we consider the fact that each UCQ h_k is the disjunction of $k+1$ conjunctive queries, $h_k = h_{k0} \vee h_{k1} \vee \ldots \vee h_{kk}$, and prove that *any* query Q defined as a Boolean combination $F(h_{k0}, \ldots, h_{kk})$ has the same unconditional lower bounds on its decision-DNNF or FBDD as h_k. For example, F may be the disjunction of these $k+1$ queries (in which case Q is just h_k), their conjunction, or any other combination: For any such Q the lower bound on the size of the FBDD, and hence on the decision-DNNF, continues to hold. The only restriction on F is that it has to depend on all $k+1$ queries; for example, F cannot be $F(h_{k0}, \ldots, h_{kk}) = h_{k0}$. This is necessary, otherwise the query Q is known to be inversion free and hence admits an ordered binary decision diagrams (OBDD)[4] of size linear in the size of the active domain [Jha and Suciu 2011]. In that case, a DPLL-based algorithm could conceivably compute Q's probability in polynomial time (assuming it does perfect caching and chooses the variable order optimally).

On the other hand, we describe a simple lifted-inference algorithm that computes the probability of $Q = F(h_{k0}, \ldots, h_{kk})$ in polynomial time, provided the Boolean function F satisfies a certain property. The crux of the algorithm is to apply the inclusion/exclusion formula on the first-order expression Q: While the inclusion/exclusion formula is exponential in k, the exponent depends only on the query, not on the database, and thus the algorithm runs in polynomial time in the size of the active domain. This proves a $2^{\Omega(\sqrt{n})}$ versus $n^{O(1)}$ separation between propositional and lifted methods for queries of this type (Theorem 3.14).

We briefly discuss the practical implication of our result. In essence, it shows that there exist three types of UCQ queries: those for which DPLL-based algorithm may run in polynomial time (e.g., have a polynomial size OBDD); those for which DPLL-based algorithm provably run in exponential time, yet their probability can be computed in polynomial time; and those which are provably #P-hard. Specifically, we have proven that the second category is non-empty, and, moreover, we have described an infinite family of UCQ queries belonging to the second category. We illustrate such queries in Example 3.13 and, more concretely, in Example 3.15. Ideally, for practical purposes, we would like to have a decision procedure that, given a UCQ query Q, classifies Q into one of these three categories. However, currently only the third class (#P-hard queries) has been completely characterized in Dalvi and Suciu [2012]; for the first two categories, we only have sufficient criteria. For example, inversion-free queries have been proven to have linear-size OBDDs [Jha and Suciu 2011] and thus belong to the first category, while all UCQ queries in the class described in this article belong to the second category. A complete characterization of the first two classes remains open.

As we have discussed so far, our lower bounds on the running time of weighted model counting algorithm apply to decision-DNNF-based model counting algorithms. Their input is a CNF, and their component rule writes a residual formula as $\Phi = \Phi_1 \wedge \Phi_2$ where Φ_1, Φ_2 are sets of clauses with no common variables. On the other hand, queries in probabilistic databases have lineage expressions that are DNF formulas, where a more natural decomposition would be $\Phi = \Phi_1 \vee \Phi_2$, where Φ_1, Φ_2 are formulas with no common variables. A natural question is whether a simple extension of a model counting algorithm with this kind of decomposition could significantly improve their power. Our extension of the simulation of decision-DNNFs by FBDDs to include DLDDs extends our lower bounds to apply to algorithms that use more general decompositions, with any unary or binary operators, including independent AND, independent OR, and negation.

[4]An OBDD is a very simple form of FBDD where every path from the root to a leaf tests the Boolean variables in the same order.

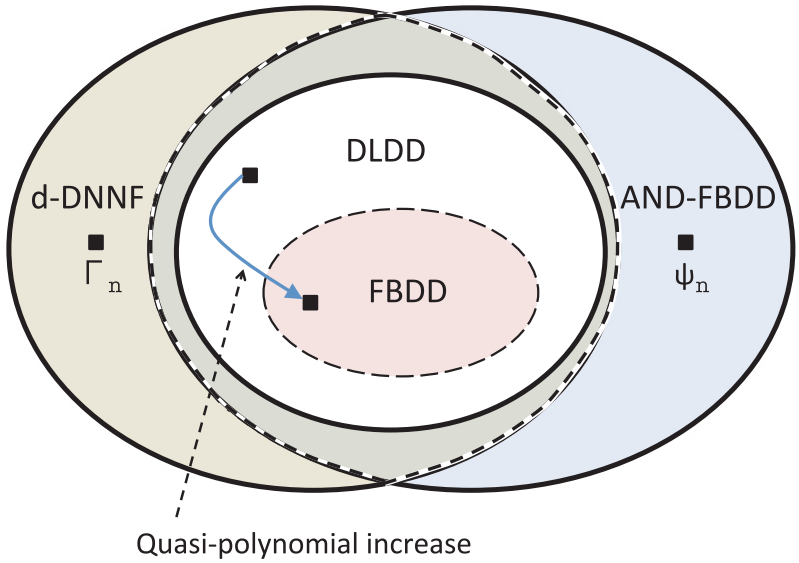


Fig. 1. A summary of our contributions on lower bounds of model counting algorithms (see Section 3.1). Here, one representation is contained in another if and only if the first can be (locally) translated into the second with at most a polynomial increase in size. Dashed borders indicate that regions are not known to be separated.

Separation of Representations of Boolean Functions

Two other consequences of our simulation of decision-DNNFs by FBDDs are provable exponential separations between the representational power of decision-DNNFs and that of either *d*-DNNFs or AND-FBDDs. There are functions, involving simple tests on the rows and columns of Boolean matrices, that require exponential size FBDDs but have linear size representations as AND-FBDDs and *d*-DNNFs, respectively (cf. Darwiche [2001b] and Theorems 10.3.8, 10.4.7. in Wegener [2000]; see Beame et al. [2013] for details); our simulation shows that these lower bounds carry over to decision-DNNFs, yielding the claimed separations. A comparison of these representations in terms of their succinctness is given in Figure 1.

Roadmap

We discuss some useful background concepts in Section 2. We describe our main results in Section 3 and then prove these results in Sections 4, 5, 6, and 7. Finally, we conclude with directions for related work in Section 8.

2. BACKGROUND

We review some knowledge compilation representations and concepts from probabilistic databases in this section.

2.1. Knowledge Compilation Representations

Though closely related, FBDDs and decision-DNNFs originate in completely different approaches for representing (or computing) Boolean functions. FBDDs are special kinds of *Binary Decision Diagrams (BDDs)* [Akers 1978], also known as *Branching Programs* [Masek 1976]. These represent a function using a directed acyclic graph with *decision* nodes, each of which queries a Boolean variable representing an input bit and has two out-edges, one labeled 0 and the other 1 (see Figure 2(a)). The graph has a single source node (root node) and has sink nodes labeled by output values of the function. Given an assignment of the Boolean variables, the value of the function is the label

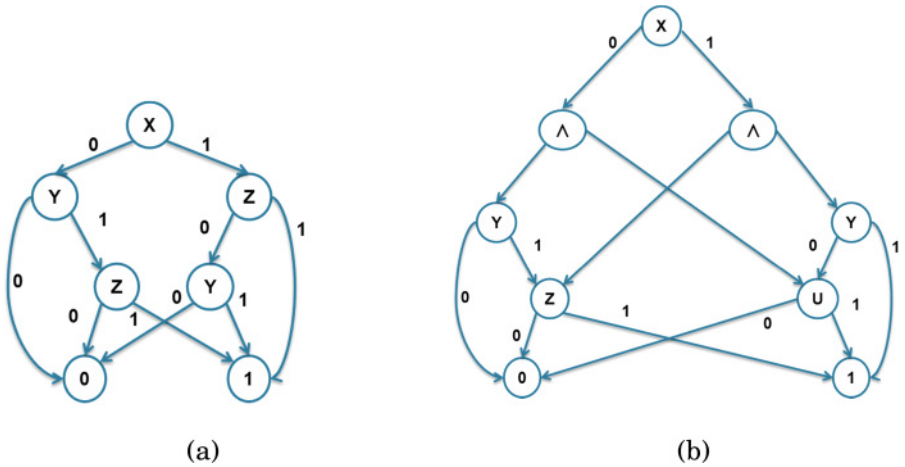


Fig. 2. (a) An FBDD representing the Boolean formula $(\neg X)YZ \vee XY \vee XZ$. (b) A decision-DNNF representing the Boolean formula $(\neg X)YZU \vee XYZ \vee XZU$.

of the unique sink node reached from the root by following that assignment: at each node, take the 0-edge if the node's variable is false, or the 1-edge if the variable is true. *FBDDs*, also known as *ROBPs*, have the property that each input variable is queried at most once on each source-sink path.[5] There are many variants and extensions of these decision-based representations; for an extensive discussion of their theory, see the monograph [Wegener 2000]. These include nondeterministic extensions of FBDDs called *OR-FBDDs*, as well as their corresponding co-nondeterministic extensions called *AND-FBDDs*, which have additional internal AND nodes through which any input can pass—the output value is 1 for an input iff *every* consistent source-sink path leads to a sink labeled 1.

Decision-DNNFs originate in the desire to find restricted forms of Boolean circuits that have better properties for knowledge representation. *Negation Normal Form (NNF)* circuits are those that have unbounded fan-in AND and OR nodes (gates) with all negations pushed to the input level using De Morgan's laws. Darwiche [2001a] introduced *DNNF*, which restricts NNF by requiring that the sub-circuits leading into each AND gate are defined on disjoint sets of variables. He also introduced *d*-DNNFs [Darwiche 2001a; Darwiche and Marquis 2002] that have the further restriction that DNNFs are *deterministic*, that is, the sub-circuits leading into each OR gate never simultaneously evaluate to 1; *d*-DNNFs have the advantage of probabilistic polynomial-time equivalence testing [Huang and Darwiche 2007]. Most subsequent papers have used these *d*-DNNFs. An easy way of ensuring determinism is to have a single variable x that evaluates to 1 on one branch and 0 on the other, so *d*-DNNFs can be produced by the subcircuit $(x \wedge A) \vee (\neg x \wedge B)$, which is equivalent to having decision nodes as above; moreover, the decomposability ensures that x does not appear in either A or B. The subclass of *d*-DNNFs in which all OR nodes are of this form is called decision-DNNFs [Huang and Darwiche 2005, 2007]. Virtually all algorithmic methods that use *d*-DNNFs, including those used in exact model counting and Bayesian inference, actually ensure determinism by using decision-DNNFs. Decision-DNNFs have the further

[5]The term *free* contrasts with *ordered* binary decision diagrams (OBDDs) [Bryant 1986], in which each root-leaf path must query the variables in the same order. For each variable order, reduced OBDDs are canonical representations for Boolean functions, making them extremely useful for a vast number of applications. Unfortunately, OBDDs are often also simply referred to as BDDs, which leads to confusion with the original general model.

advantage of being *syntactically* checkable; by comparison, given a general DNNF, it is NP-hard to check whether it satisfies the semantic restriction of being a *d*-DNNF.

It is immediate that one can get a completely equivalent representation to the above definition by using a decision node on x in place of each OR of ANDs involving x and in place of each leaf variable or its negation; the decomposability property ensures that no root-leaf path in the circuit queries the same variable more than once. Figure 2(b) illustrates such a representation of a decision-DNNF. Clearly, these form a special subclass of the AND-FBDDs discussed above, in which each AND node is required to have the decomposability property that the different branches below each AND node query disjoint sets of variables. Though formally there are insignificant syntactic differences between the definitions, we use the term decision-DNNFs to refer to these *decomposable* AND-FBDDs.

Definition 2.1. An *FBDD* is a rooted directed acyclic graph (DAG) $\mathcal{F}$ that computes m Boolean functions $\mathbf{\Phi} = (\Phi_1, \ldots, \Phi_m)$. $\mathcal{F}$ has two kinds of nodes: *decision nodes*, which are labeled by a Boolean variable X and have two outgoing edges labeled 0 and 1, and *sink nodes*, labeled with an element from $\{0,1\}^m$. $\mathcal{F}$ must satisfy the following *read-once* property: Every path from the root to some sink node may test a Boolean variable X at most once. For every node u in an FBDD $\mathcal{F}$, we denote $\mathcal{F}_u$ the sub-DAG of $\mathcal{F}$ rooted at u and denote $\mathbf{\Phi}_u$ the m Boolean functions defined inductively by the following rules[6]:

$$\begin{aligned} &\mathbf{\Phi}_u = (\neg X)\mathbf{\Phi}_{u_0} \vee X\mathbf{\Phi}_{u_1} && \text{if } u \text{ is a decision node labeled } X, \text{ with children } u_0, u_1 \\ &\mathbf{\Phi}_u = L && \text{if } u \text{ is a sink node labeled } L \in \{0,1\}^m. \end{aligned} \tag{1}$$

The FBDD $\mathcal{F}$ computes $\mathbf{\Phi} = \mathbf{\Phi}_r$, where r is the root. The *size* of the FBDD $\mathcal{F}$ is the number nodes in $\mathcal{F}$. Typically, $m = 1$, but we will also consider FBDDs $\mathcal{F}$ with $m > 1$ and call $\mathcal{F}$ a *multi-output FBDD*.

One can check that the probability of each of the m functions can be computed in time linear in the size of the FBDD using a simple dynamic program: $\Pr[\mathbf{\Phi}_u] = (1 - p(X))\Pr[\mathbf{\Phi}_{u_0}] + p(X)\Pr[\mathbf{\Phi}_{u_1}]$.

The multi-output FBDD definition for $m > 1$ is standard [Wegener 2000]. When computing m Boolean functions, it is also typical to consider m separate single-output FBDDs with overlapping node sets. We will find multi-output FBDDs as defined above to be natural intermediate objects in our constructions in the proof of Theorem 3.9.

For our purposes, it will also be useful to consider FBDDs with *no-op nodes*. A no-op node is not labeled by any variable and has a single child; the meaning is that we do not test any variable, but simply continue to its unique child. Every FBDD with no-op nodes can be transformed into an equivalent FBDD without no-op nodes by simply skipping over any no-op node.

Definition 2.2. An *AND-FBDD* [Wegener 2000] is an FBDD with an additional kind of node, called AND-nodes. Similarly to FBDDs, an AND-FBDD must satisfy the read-once condition. The function $\mathbf{\Phi_u}$ is defined as follows as in Equation (1) for decision nodes and sink nodes and is defined as $\mathbf{\Phi_u} = \mathbf{\Phi}_{\boldsymbol{u_1}} \wedge \cdots \wedge \mathbf{\Phi}_{\boldsymbol{u_u}}$ for an AND-node.

Any CNF formula Φ has an AND-FBDD of size linear in the CNF expression, which implies that, in general, computing probability of the function defined by an AND-FBDD in #P-hard. We assume, without loss of generality, that every AND-node in an AND-FBDD has exactly two children u_1, u_2, a property that can be enforced by at most doubling the number of edges in the DAG.

[6]We write conjunctions $\phi \wedge \psi$ as products $\phi\psi$. If $\mathbf{\Phi} = (\Phi_1, \ldots, \Phi_m)$, $\mathbf{\Phi'} = (\Phi'_1, \ldots, \Phi'_m)$, and X is a literal, then $X\mathbf{\Phi}$ denotes $(X\Phi_1, \ldots, X\Phi_m)$, while $\mathbf{\Phi} \vee \mathbf{\Phi'}$ denotes $(\Phi_1 \vee \Phi'_1, \ldots, \Phi_m \vee \Phi'_m)$.

Definition 2.3. A *decision-DNNF* [Huang and Darwiche 2005, 2007] is an AND-FBDD $\mathcal{D}$ satisfying the following property: For every AND-node u with children u_1, u_2, the sub-DAGS $\mathcal{D}_{u_1}$ and $\mathcal{D}_{u_2}$ do not mention any common Boolean variables.

The additional constraint on AND-nodes allows the probability of a decision-DNNF to be computed in linear time by noting that for every decomposable AND-node u, $\Pr[\mathbf{\Phi}_{\boldsymbol{u}}] = \Pr[\mathbf{\Phi}_{\boldsymbol{u_1}}] \cdot \Pr[\mathbf{\Phi}_{\boldsymbol{u_2}}]$.

Finally, we introduce here *Decomposable Decision Logic Diagrams* or DLDDs by further generalizing decision-DNNFs.

Definition 2.4. A *DLDD* is a rooted directed acyclic graph $\mathcal{D}$ that has four kinds of nodes: *decision nodes*, *sink nodes* labeled with 0 or 1, and, in addition to these nodes that they have in common with FBDDs, unary *NOT-nodes* labeled by $\neg$ and binary *decomposable operator nodes* (OP-nodes), each labeled by some two-input Boolean function. Similarly to FBDDs, $\mathcal{D}$ must satisfy the read-once condition, and, similarly to decision-DNNFs, for any binary decomposable OP-node u with two children u_1, u_2, the sub-DAGS $\mathcal{D}_{u_1}$ and $\mathcal{D}_{u_2}$ must not mention any common Boolean variables. The function $\mathbf{\Phi}_{\boldsymbol{u}}$ is defined as in Equation (1), where u is a decision node or a sink node; at a NOT node with child v, $\Phi_u = \neg\Phi_v$ and if u is an OP-node labeled by function g_u, then $\mathbf{\Phi}_{\boldsymbol{u}} = g_u(\mathbf{\Phi}_{\boldsymbol{u_1}}, \mathbf{\Phi}_{\boldsymbol{u_2}})$. A DLDD is called *positive* if it has no NOT nodes and all its binary OP-nodes are labeled by monotone Boolean functions, $\vee$ or $\wedge$.

A decision-DNNF is a special case of a positive DLDD where there are no NOT-nodes and every binary OP-node is an AND-node.

The probability of a DLDD can be computed in linear time, bottom up, by noting that for each decomposable operator node u with children u_1, u_2, if we denote $p = \Pr[\mathbf{\Phi}_{\boldsymbol{u_1}}]$ and $q = \Pr[\mathbf{\Phi}_{\boldsymbol{u_2}}]$, then:

$$\Pr[\mathbf{\Phi}_{\boldsymbol{u}}] = g(0,0)\cdot(1-p)\cdot(1-q) + g(0,1)\cdot(1-p)\cdot q + g(1,0)\cdot p\cdot(1-q) + g(1,1)\cdot p\cdot q.$$

Similarly, for a NOT-node u with a unique child u_1,

$$\Pr[\mathbf{\Phi}_{\boldsymbol{u}}] = (1-p).$$

2.2. Queries and Lineages

A UCQ is a first-order formula over a fixed relational vocabulary, consisting of only positive relational atoms and the connectives $\exists, \vee, \wedge$. A Boolean UCQ is a UCQ with no free variables. A probabilistic database D is a relational database where each tuple t is associated with a probability $p_t \in [0,1]$. The probabilistic database defines a probability space where the outcomes are all the subsets $W \subseteq D$, called *possible worlds*, and where the probability of an outcome W is $\prod_{t\in W} p_t \cdot \prod_{t\notin W}(1-p_t)$. The marginal probability of a query Q is $\Pr[Q(D)] = \sum_{W:W\models Q} \Pr[W]$.

Definition 2.5. Fix a finite domain Dom. Associate to each ground tuple t over the domain Dom a unique Boolean variable X_t. Given a Boolean UCQ Q, the *lineage* of Q on Dom, denoted by Φ_Q^{Dom}, is inductively defined on the structure of Q as follows:

$$\begin{aligned}
\Phi_Q^{\mathrm{Dom}} &= X_t \qquad \text{if } Q \text{ is the ground tuple } t\\
\Phi_{Q_1\wedge Q_2}^{\mathrm{Dom}} &= \Phi^{\mathrm{Dom}} D_{Q_1} \wedge \Phi_{Q_2}^{\mathrm{Dom}}\\
\Phi_{Q_1\vee Q_2}^{\mathrm{Dom}} &= \Phi^{\mathrm{Dom}} D_{Q_1} \vee \Phi_{Q_2}^{\mathrm{Dom}}\\
\Phi_{\exists x.Q}^{\mathrm{Dom}} &= \bigvee_{a\in\mathrm{Dom}} \Phi_{Q[a/x]}^{\mathrm{Dom}}.
\end{aligned}$$

Patient

name	disease	
Ann	asthma	$X_1,\ 0.2$
Bob	asthma	$X_2,\ 1.0$
Carl	flue	$X_3,\ 0.9$

Friend

name1	name2	
Ann	Joe	$Z_{11},\ 0.3$
Ann	Tom	$Z_{12},\ 0.4$
Bob	Tom	$Z_{22},\ 0.3$
Carl	Tom	$Z_{32},\ 0.7$

Smoker

name	
Joe	$Y_1,\ 0.8$
Tom	$Y_2,\ 0.6$

Fig. 3. An example of a probabilistic database.

Here $Q[a/x]$ denotes the query obtained by replacing a variable x in query Q by a constant a; that is, the variable x is *grounded* to the constant a. Given a relational database instance D, the lineage of Q on D, denoted by Φ_Q^D, is defined as $\Phi_Q^D = \Phi_Q^{\text{Dom}}$, where Dom is the active domain of the database D.

The probability of a Boolean formula Φ over variables X_t is defined as the probability of Φ being true when each variable X_t is set to true independently, with probability p_t. It follows that $\Pr[Q(D)] = \Pr[\Phi_Q^D]$.

Example 2.6. Consider a vocabulary with three relation symbols: `Patient(name, diseases)`, `Friend(name1, name2)`, `Smoker(name)`. Figure 3 shows a tuple-independent probabilistic database D on these three relations; there are 2^9 possible worlds. To each tuple we associate a Boolean variable, and a probability, denoted X_1, p_1, X_2, p_2, and so on. Consider the Boolean query:

$$\text{Query } Q_s := \exists x\, \exists y\, \texttt{Patient}(x, \texttt{'asthma'}) \wedge \texttt{Friend}(x, y).$$

The query is `true` on a possible world W iff W contains an asthma patient who has some friend. Its lineage is as follows:

$$\Phi_D^Q = X_1 Z_{11} \vee X_1 Z_{12} \vee X_2 Z_{22}.$$

This formula is also `true` precisely in those possible worlds where some asthma patient has some friend; thus, it should be clear that $\Pr[Q(D)] = \Pr[\Phi_D^Q]$. The grounded inference method will first compute the lineage and then compute $\Pr[\Phi_s^D]$ using the probabilities of the variables as given in Figure 3 not considering the structure of the query Q_s in the evaluation process.

Lifted inference methods postpone grounding. For example, *safe query evaluation* [Suciu et al. 2011] computes $\Pr[Q(D)]$ by first grounding only the variable x:

$$\begin{aligned}\Pr[Q] &= \Pr[Q[\texttt{Ann}/x] \vee Q[\texttt{Bob}/x] \vee Q[\texttt{Carl}/x]] \\ &= 1 - (1 - \Pr[Q[\texttt{Ann}/x]])(1 - \Pr[Q[\texttt{Bob}/x]])(1 - \Pr[Q[\texttt{Carl}/x]]).\end{aligned}$$

Each of the three residual queries occurring above is also computed using lifted inference. We illustrate only with the first query:

$$\begin{aligned}\Pr[Q[\texttt{Ann}/x]] &= \Pr[\exists y(\texttt{Patient(Ann, 'Asthma')} \wedge \texttt{Friend(Ann}, y))] \\ &= \Pr[\texttt{Patient(Ann, 'Asthma')} \wedge \exists y(\texttt{Friend(Ann}, y))] \\ &= \Pr[\texttt{Patient(Ann, 'Asthma')}] \cdot \Pr[\exists y(\texttt{Friend(Ann}, y))] \\ &= 0.2 \cdot \Pr[\exists y(\texttt{Friend(Ann}, y))].\end{aligned}$$

Finally, the last expression is $\Pr[\exists y(\texttt{Friend(Ann}, y))] = 1 - (1 - 0.3) \cdot (1 - 0.4)$.

3. MAIN RESULTS

Here we formally state our main results, discuss their implications, and defer the proofs to the following sections.

3.1. Exponential Lower Bounds on the Size of Knowledge Representations

Our goal is to prove lower bounds on the running time of algorithms that are used in modern model counters. For that, we use the fact that the trace of any DPLL-based algorithm with caching and components is a decision-DNNF, and therefore any lower bound on the size of decision-DNNFs represents a lower bound on the running time of such algorithms. In this section, we give several such lower bounds on decision-DNNFs.

In fact, for a general statement of our results, we prove lower bounds for the more general DLDDs instead of decision-DNNFs. The important difference between DLDDs and decision-DNNFs for model counting is that DLDDs allow negation at arbitrary levels of the representation. Allowing negation is necessary since decision-DNNFs are not closed under complement (complementing a decision-DNNF can result in an exponential size increase), and one should not say that a formula is hard for counting if its complement is easy to count.[7] Handling negation at the output of the representation turns out to be sufficient to handle it at all levels; small representations that extend decision-DNNFs by allowing negation anywhere would still yield efficient model counting and could be substantially more efficient than those that allow negation only at the output. DLDDs extend decision-DNNFs by including these internal negations as well as additional binary decompositions other than $\wedge$; we allow these additional decompositions to state Theorem 3.1, and our lower bounds, in as strong a way as possible.

Theorem 3.1 allows us to reduce the problem of producing lower bounds for DLDDs to finding lower bounds for FBDDs.

THEOREM 3.1. *Let $\mathcal{D}$ be a DLDD with N nodes. Then there exists an equivalent FBDD $\mathcal{F}$ computing the same formula as $\mathcal{D}$, with at most $2N2^{\log^2 N}$ nodes and at most $N2^{\log^2 N}$ nodes if $\mathcal{D}$ is positive (e.g., when $\mathcal{D}$ is a decision-DNNF). Furthermore, $\mathcal{F}$ can be computed in linear time in its size.*

We prove this result in Section 4 where we describe the details of the conversion algorithm. Recently, Razgon [2016] has identified formulas that have decision-DNNFs of size $O(n^2)$ but require FBDD size (and even nondeterministic FBDD size) $\Omega(n^{\log_2 n})$, which shows that this conversion is asymptotically optimal.

In the remainder of this sub-section, we give some simple applications of the theorem. We begin with the following explicit 2-DNF formula introduced by Bollig and Wegener [1998]. For any set $E \subseteq [n] \times [n]$, define $\Psi_E = \bigvee_{(i,j)\in E} X_i Y_j$, where $X_1, \ldots, X_n, Y_1, \ldots, Y_n$ are Boolean variables. Let $n = p^2$ where p is a prime number; then each number $0 \le i < n$ can be uniquely written as $i = a + bp$, where $0 \le a, b < p$. Define $E_n = \{(i+1, j+1) \mid i = a + bp, j = c + dp, c \equiv (a + bd) \mod p\}$; thus, $|E_n| = p^3 = n^{3/2}$. Then:

THEOREM 3.2. *[Bollig and Wegener 1998, Theorem 3.1] Any FBDD for Ψ_{E_n} has $2^{\Omega(\sqrt{n})}$ nodes.*

The formula Ψ_{E_n} is not natural. However, we consider the formula $H_0 = \bigvee_{1\le i,j\le n} R(i)S(i,j)T(j)$, which is the lineage of a simple database query on a database instance (see Section 3.2). It is easy to see that Theorem 3.2 implies that any FBDD for H_0 has size $2^{\Omega(\sqrt{n})}$, because it can be converted into an FBDD for Ψ_{E_n} by setting $S(i,j) = 1$ or $S(i,j) = 0$ depending on whether (i,j) is in E_n.

COROLLARY 3.3. *Any decision-DNNF or DLDD for either Ψ_{E_n} or H_0 has $2^{\Omega(n^{1/4})}$ nodes.*

[7]This is particularly problematic since model counting algorithms are typically designed for CNF expressions, but the natural lineages of database query expressions are in DNF form (cf. Section 3.2); the lower bounds in Beame et al. [2013] did not handle this distinction.

PROOF. Denote N the size of a DLDD for the formula. By Theorem 3.1, we obtain an FBDD $\mathcal{F}$ of size $2^{\log^2 N+\log N+1}$, which must be $2^{\Omega(\sqrt{n})}$; thus $\log^2 N = \Omega(\sqrt{n})$, hence $\log N = \Omega(n^{1/4})$, and $N = 2^{\Omega(n^{1/4})}$. □

In particular, any DPLL-based algorithm whose trace can be seen as a DLDD will take exponential time on the formulas Ψ_{E_n} and H_0. We will later improve the lower bound on the size of FBDDs for H_0 to $2^{\Omega(n)}$, which implies a $2^{\Omega(\sqrt{n})}$ lower bound on the size of its DLDDs.

Separating decision-DNNFs from AND-FBDDs. We show that decision-DNNFs are strictly weaker than AND-FBDDs. In other words, the extra condition in decision-DNNFs requiring AND-nodes to be decomposable is significant. Define $\Psi'_{E_n} = \bigwedge_{(i,j)\in E_n}(X_i \vee Y_j)$, the CNF expression that is the dual of Ψ_{E_n}. Since Ψ'_{E_n} is a CNF formula, it admits an AND-FBDD with at most $O(n^{3/2})$ nodes (since $|E_n| = n^{3/2}$). On the other hand, Theorem 3.2 also implies that any FBDD for Ψ'_{E_n} has $2^{\Omega(\sqrt{n})}$ nodes, and therefore any decision-DNNF for this formula must have $2^{\Omega(n^{1/4})}$ nodes by the same argument as in Corollary 3.3. We have shown:

COROLLARY 3.4. *Decision-DNNFs are exponentially less concise than AND-FBDDs.*

Separating decision-DNNFs from d-DNNFs. Define Γ_n on the matrix of variables X_{ij} for $i, j \in [n]$ by $\Gamma_n(X) = f_n(X) \vee g_n(X)$ where f_n is 1 if and only if the parity of all the variables is even (meaning: $\bigoplus_{i,j} X_{ij} = 0$) and the matrix has an all-1 row and g_n is 1 if and only if the parity of all the variables is odd ($\bigoplus_{i,j} X_{ij} = 1$) and the matrix has an all-1 column. Theorem 10.4.7. in Wegener [2000] shows that any FBDD for Γ_n has $2^{\Omega(n)}$ nodes (therefore, every decision-DNNF requires $2^{\Omega(n^{1/2})}$ nodes). Γ_n can also be computed by an $O(n^2)$ size d-DNNF, because both f_n and g_n can be computed by $O(n^2)$ size OBDDs, and $f_n \wedge g_n \equiv \texttt{false}$. Hence:

COROLLARY 3.5. *decision-DNNFs are exponentially less concise than d-DNNFs.*

3.2. Lower Bounds for Simple Database Queries and Their Lineages

We now introduce some elementary queries that work as building blocks for the class of queries considered in these results [Dalvi and Suciu 2012; Jha and Suciu 2013]:

Let $[n]$ denote the set $\{1, \ldots, n\}$. Fix $k > 0$ as a parameter for the size of queries and $n > 0$ as the size of the domain of tuples in a database. We will use h with subscripts to denote Boolean first-order formulas (which represent the class of Boolean *UCQ*) and H with the same subscripts to denote the *lineages* of these formulas on a database with relations $R, T, S, S_1, \ldots, S_k$. The potential tuples in these relations are represented by Boolean variables $R(i)$, $T(j)$, $S(i, j)$, and $S_\ell(i, j)$, respectively, where $i, j \in [n]$ and $\ell \in [k]$.[8] Lineages represent propositional formulas that are generated by grounding a given first-order formula (query) on a database instance and capture

[8]In a probabilistic database, in general, different relations can have different number of tuples and different ranges of the values of attributes. However, we assume that the active domain of all attributes ranges from 1 to n without loss of generality, which preserves polynomial-time query evaluation for a given query (in terms of data complexity). For positive results, that is, polynomial-time algorithms, we can plug in any probability value for each of these tuples to generate an arbitrary probability instance (for missing tuples, the probability will be zero). For negative results, this gives a class of database instances for the class of queries for which the existing model counting algorithms will provably take exponential time. This is unlike other constrained scenarios like symmetric databases [Jaeger and Van den Broeck 2012; Beame et al. 2015], where all the tuples in the same relation are required to have the same probability value.

the all possible combinations of input tuples that make the Boolean query true on the database instance.

Consider the following set of $k+1$ Boolean queries $\mathbf{h_k} = (h_{k0}, \ldots, h_{kk})$, where:

$$\begin{aligned} h_{k0} &= \exists x_0 \exists y_0 \; R(x_0) \wedge S_1(x_0, y_0) \\ h_{k\ell} &= \exists x_\ell \exists y_\ell \; S_\ell(x_\ell, y_\ell) \wedge S_{\ell+1}(x_\ell, y_\ell) \qquad \forall \ell \in [k-1] \\ h_{kk} &= \exists x_k \exists y_k \; S_k(x_k, y_k) \wedge T(y_k). \end{aligned}$$

Then the corresponding lineages are as follows:

$$H_{k0} = \bigvee_{i,j \in [n]} R(i) S_1(i,j), \qquad H_{kk} = \bigvee_{i,j \in [n]} S_k(i,j) T(j),$$
$$H_{k\ell} = \bigvee_{i,j \in [n]} S_\ell(i,j) S_{\ell+1}(i,j) \qquad \forall \ell \in [k-1].$$

We define $\mathbf{H_k} = (H_{k0}, \ldots, H_{kk})$. Two well-studied queries [Dalvi and Suciu 2012] that we will consider in this section are given below:

Query h_k, for $k \geq 1$: h_k is a disjunction on the queries in $\mathbf{h_k}$: $h_k = h_{k0} \vee h_{k1} \vee \cdots \vee h_{kk}$. The lineage H_k of h_k is given by $H_k = H_{k0} \vee H_{k1} \vee \cdots \vee H_{kk}$.

Query h_0: h_0 uses a single relation symbol S in addition to R and T: $h_0 = \exists x \exists y \; R(x) \wedge S(x,y) \wedge T(y)$. The lineage H_0 of h_0 is given by $H_0 = \bigvee_{i,j \in [n]} R(i) S(i,j) T(j)$.

Lower Bounds on FBDDs for Queries h_0, h_k. Jha and Suciu [2013] previously showed that every FBDD for the lineage H_1 of h_1 has size $2^{\Omega(\log^2 n)}$, but this is insufficient to derive any lower bound using our simulation. Our first result improves this to an exponential lower bound, not just for H_1 but also for H_0 and for all H_k for $k > 1$:

THEOREM 3.6. *For every $n > 0$, any FBDD for H_0 or for H_k for $k \geq 1$ has $\geq (2^n - 1)/n$ nodes.*

It is known that weighted model counting for both H_0 and H_k is #P-hard [Dalvi and Suciu 2012]. However, the lower bounds we show on these FBDD sizes are absolute (independent of any complexity theoretic assumption) and do not rely on the #P-hardness of the associated weighted model counting problems. We give the proof in Section 5. This improved bound is critical for proving the overall lower-bound result in this article (Theorem 3.10).

While we do not need h_0 and H_0 in the rest of the article, we include it in Theorem 3.6 because it is obtained in a fashion similar to that for H_k and substantially improves on the $2^{\Omega(\sqrt{n})}$ lower bound for H_0 derived from Theorem 3.2. Our new lower bound improves this to the nearly optimal $(2^n - 1)/n$.

We also note that our stronger lower bounds for H_1 give instances of bipartite 2-DNF formulas that are simpler to describe than those of Bollig and Wegener [1998], but yield as good a lower bound on FBDD sizes in terms of the number of variables and even better bounds as a function of the number of terms: p in Ψ_{E_n} is analogous to n in H_1 formulas and Ψ_{E_n} has p^3 terms, versus only $2n^2$ for H_1.

Lower Bounds for FBDDs for Queries Over $\mathbf{h_k}$. Theorem 3.6 gives a lower bound on h_k, which is simply the logical OR of the queries in $\mathbf{h_k}$. Theorem 3.9 below generalizes this result by allowing queries that are *arbitrary functions* of queries in $\mathbf{h_k}$.

Let $f(\mathbf{X}) = f(X_0, X_1, \ldots, X_k)$ be an arbitrary Boolean function on $k+1$ Boolean variables $\mathbf{X} = (X_0, \ldots, X_k)$ and Q the Boolean query $Q = f(h_{k0}, h_{k1}, \ldots, h_{kk})$. Clearly, the lineage of Q is $f(\mathbf{H_k}) = f(H_{k0}, H_{k1}, \ldots, H_{kk})$.

Example 3.7. If $f(X_0, X_1, \ldots, X_k) = \bigvee_{\ell=0}^{k} X_\ell$, then we get query $h_k = \bigvee_{\ell=0}^{k} h_{k\ell}$; its lineage is $H_k = \bigvee_{\ell=0}^{k} H_{k\ell}$.

Any function f has an equivalent form that includes all $k+1$ variables, for example, by adding redundant variables $f(\mathbf{X}) \wedge [\bigvee_{\ell=0}^{k} X_\ell \vee \neg X_\ell]$. To disallow such redundancy, we require that the function *depends on* all $k+1$ variables defined as follows:

Definition 3.8. The function f *depends on* a variable X_ℓ, $\ell \in \{0, \ldots, k\}$, if there is an assignment $\mu_{-\ell}$ on the rest of the variables $\mathbf{X} \setminus \{X_\ell\}$ such that $f[X_\ell, \mu_{-\ell}] = X_\ell$ or $\neg X_\ell$.

THEOREM 3.9. *If f depends on all $k+1$ variables $X_0, \ldots, X_k$, then any FBDD $\mathcal{F}$ with N nodes for the lineage of $Q = f(h_{k0}, \ldots, h_{kk})$ can be converted into a multi-output FBDD $\mathcal{F}'$ for $(H_{k0}, \ldots, H_{kk})$ with $O(k2^k n^3 N)$ nodes. It follows that $N = 2^{\Omega(n)}$.*

To see that $N = 2^{\Omega(n)}$, note that the multi-output FBDD $\mathcal{F}'$ can be immediately converted into an FBDD computing H_k with the same number of nodes, and, therefore, if $k \leq \alpha n$ for some constant $\alpha < 1$, then $\mathcal{F}$ has at least $2^{\Omega(n)}$ nodes by Theorem 3.6.

We prove the theorem in Section 6. The condition that f depends on all variables is necessary.

If Q does not depend on any one of the queries in $\mathbf{h_k}$, then the theorem fails. For example, if $Q \equiv h_{k0}$, then Q admits an OBDD of size $O(n)$, while any multi-output FBDD for $(H_{k0}, \ldots, H_{kk})$ has size $\geq (2^n - 1)/n$ by Theorem 3.6. More generally, if Q does not depend on some query h_{ki}, then its lineage has a very simple and easy to construct FBDD of size linear in n [Jha and Suciu 2011], which uses row-major order for one side of the cut and column-major order for the other side; it can then be used to count Q in polynomial time (see Section 6).

Theorem 3.9 extends prior work in several ways. First, it is the first result showing exponential lower bounds on FBDDs for a large class of queries. Prior to Theorem 3.9, the only known lower bound was the quasipolynomial lower bound for h_1 [Jha and Suciu 2013]. Second, although a conversion of an FBDD for a specific query Q_W (described later in this section) into one for h_1 was given in Jha and Suciu [2013], this conversion did not extend to other queries. While we were inspired by that proof, the techniques we use in Theorem 3.9 are considerably more powerful and use new ideas that can be of independent interest to show lower bounds on the size of FBDDs in general.

Lower Bounds for Model Counting Algorithms for Queries Over $\mathbf{h_k}$. Theorems 3.1, 3.6, and 3.9 together prove the following lower bound result:

THEOREM 3.10. *If Q is a Boolean combination of the queries in $\mathbf{h_k}$ that depends on all $k+1$ queries in $\mathbf{h_k}$, then any DLDD (and therefore any decision-DNNF) for the lineage Θ of Q has size $2^{\Omega(\sqrt{n})}$.*

PROOF. Let N be the size of a DLDD for Q. By Theorem 3.1, Q has an FBDD of size $N2^{\log^2 N}$. By Theorem 3.9, $N2^{\log^2 N} = 2^{\Omega(n)}$, implying that N is $2^{\Omega(\sqrt{n})}$. □

As we discussed previously, since current propositional exact weighted model counting algorithms (extended with negation to handle DNFs) without loss of generality yield DLDDs of size at most their running time, we immediately obtain:

COROLLARY 3.11. *All current propositional exact model counting algorithms require running time $2^{\Omega(\sqrt{n})}$ to perform weighted model counting for any query Q that is a Boolean combination of the queries in $\mathbf{h_k}$ and depends on all $k+1$ queries in $\mathbf{h_k}$.*

A Dichotomy Theorem for a Class of Queries. We extend the lower bound in Theorem 3.9 by proving a dichotomy theorem for a slightly more general class of queries:

Any query in this class either has a polynomial-time model counting algorithm or all existing DPLL-based model counting algorithms require exponential time. To define the class of queries, we consider the $k+1$ queries in $\mathbf{h_k}$ and the following additional $k+2$ queries $\mathbf{b_k}$: $b_0 = \exists u_0 R(u_0)$, $b_\ell = \exists u_\ell \exists v_\ell S_\ell(u_\ell, v_\ell)$, $\forall \ell \in [k]$, $b_{k+1} = \exists v_{k+1} T(v_{k+1})$. These queries have the following lineages on the same domain of size n: $B_0 = \bigvee_{i\in[n]} R(i)$, $B_\ell = \bigvee_{i,j\in[n]} S_\ell(i,j)$, $\forall \ell \in [k]$, and $B_{k+1} = \bigvee_{j\in[n]} T(j)$, respectively.

Let $g(X_0, \ldots, X_k, Y_0, \ldots, Y_{k+1})$ be a Boolean function on $2k+3$ variables. Consider the query $Q = g(h_{k0}, \ldots, h_{kk}, b_0, \ldots, b_{k+1}) = g(\mathbf{h_k}, \mathbf{b_k})$, and its lineage $g(H_k, B_k)$. Let $g(\mathbf{X}, \mathbf{1}) = g(X_0, \ldots, X_k, 1, \ldots, 1)$. Then the following dichotomy holds where n is the size of the domain:

THEOREM 3.12.

(*1*) *If $g(\mathbf{X}, \mathbf{1})$ depends on all $k+1$ variables $X_0, \ldots, X_k$, then any DLDD for the lineage of Q has size $2^{\Omega(n)}$.*
(*2*) *Otherwise, there exists an FBDD for the lineage of Q of size $n^{O(1)}$, and the FBDD can be constructed in $n^{O(1)}$ time.*

We prove Theorem 3.12 in Section 7.

3.3. Separating Propositional and Lifted Model Counting

Theorem 3.10, when applied to query h_k, $k \geq 1$, is not surprising: #P-hardness of h_k makes it unlikely to have an efficient model counting algorithm. However, there are many other query combinations over $\mathbf{h_k}$ where weighted model counting can be done in polynomial time using lifted inference methods. This separates the grounded from the lifted inference algorithms.

Consider the case when $Q = f(\mathbf{h_k})$ and f is a monotone Boolean formula $f(X_0, \ldots, X_k)$, and thus Q is a UCQ query. Here the cases when weighted model counting for Q can be done in polynomial time are entirely determined by the structure of the query expression[9] f, and we review it here briefly following Suciu et al. [2011].

To check if weighted model counting for $Q = f(\mathbf{h_k})$ is computable in polynomial time, write f as a CNF formula, $f = \bigwedge_i C_i$, where each (positive) clause C_i is a set of propositional variables $C_i \subseteq \{X_0, \ldots, X_k\}$. Define the lattice $(L, \leq)$, where L contains all subsets $u \subseteq \mathbf{X}$ that are a union of clauses C_i, and the order relation is given by $u \leq v$ if $u \supseteq v$. The maximal element of the lattice is $\emptyset$, (we denote it $\hat{1}$), while the minimal element is $\mathbf{X}$ (we denote it $\hat{0}$). The Möbius function on the lattice L, $\mu : L \times L \to \mathbf{R}$, is defined as $\mu(u,u) = 1$ and $\mu(u,v) = -\sum_{u<w\leq v} \mu(w,v)$ [Stanley 1997]. The following holds [Suciu et al. 2011]: If $\mu(\hat{0}, \hat{1}) = 0$, then weighted model counting for Q can be done in time polynomial in n (using in the inclusion/exclusion formula on the CNF); if $\mu(\hat{0}, \hat{1}) \neq 0$, then the weighted model counting problem for Q is #P-hard.

Example 3.13. Here we give examples of easy and hard queries. A more practical example is given in Example 3.15.

—For a trivial example, $h_k = h_{k0} \vee \cdots \vee h_{kk}$ has a single clause, and hence its lattice has exactly two elements $\hat{0}$ and $\hat{1}$, and $\mu(\hat{0}, \hat{1}) = -1$, hence h_k is #P-hard.
—Two more interesting examples for $k = 3$:

$$f_W = (X_0 \vee X_2) \wedge (X_0 \vee X_3) \wedge (X_1 \vee X_3)$$
$$f_9 = (X_0 \vee X_3) \wedge (X_1 \vee X_3) \wedge (X_2 \vee X_3) \wedge (X_0 \wedge X_1 \wedge X_2).$$

[9]The propositional formula f describes the query expression Q and should not be confused with the propositional grounding of Q on the instance $R(i), S_1(i,j), \ldots, S_k(i,j), T(j)$; $\ell \in [1, k-1]$, $i, j \in [n]$.

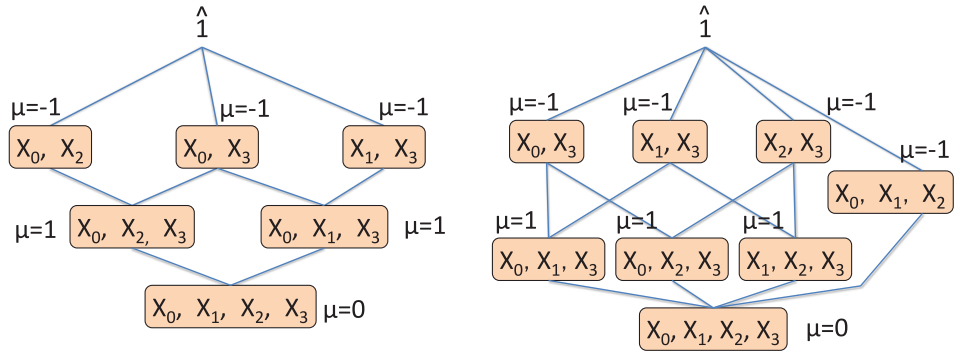


Fig. 4. The lattices for (a) f_W, (b) f_9.

Their lattices, shown in Figure 4, satisfy $\mu(\hat{0}, \hat{1}) = 0$ (in the figure, μ at each node u denotes $\mu(u, \hat{1})$). Therefore, for the queries $Q_W = f_W(h_{30}, h_{31}, h_{32}, h_{33})$ and $Q_9 = f_9(h_{30}, h_{31}, h_{32}, h_{33})$, weighted model counting can be done in polynomial time. For example, to compute the probability of Q_W, we apply the inclusion/exclusion formula on the query expression and get

$$\begin{aligned}\Pr[Q_W] = \; & \Pr[h_{30} \vee h_{32}] + \Pr[h_{30} \vee h_{33}] + \Pr[h_{31} \vee h_{33}] \\ & - \Pr[h_{30} \vee h_{32} \vee h_{33}] - \Pr[h_{30} \vee h_{31} \vee h_{33}] \\ & - \Pr[h_{30} \vee h_{31} \vee h_{32} \vee h_{33}] + \Pr[h_{30} \vee h_{31} \vee h_{32} \vee h_{33}]\end{aligned}$$

While computing $\Pr[h_{30} \vee h_{31} \vee h_{32} \vee h_{33}]$ is #P-hard (because this query is h_3), the two occurrences of this term cancel out, and for all remaining terms one can compute the probability in polynomial time in n (since each misses at least one term $h_{30}, h_{31}, h_{32}, h_{33}$). Thus, weighted model counting can be done in polynomial time for Q_W (similarly for Q_9) at the query expression level.

On the other hand, Theorem 3.10 proves that, if we ground Q_W or Q_9 first, then any DPLL-based model counting algorithm will take exponential time on the lineage. This leads to the main separation result of this article:

THEOREM 3.14. *Let $Q = f(\mathbf{h_k})$ be any monotone, Boolean combination of the queries in $\mathbf{h_k}$ that depends on all $k+1$ queries in $\mathbf{h_k}$ such that $\mu(\hat{0}, \hat{1}) = 0$. Then weighted model counting for Q can be done in time polynomial in n, whereas all existing DPLL-based propositional algorithms for model counting require exponential time on the lineage.*

PROOF. From Theorem 3.10, since Q is a Boolean combination of the queries in $\mathbf{h_k}$ that depends on all $k+1$ queries in $\mathbf{h_k}$, then any DLDD (and therefore any decision-DNNF) for the lineage Θ of Q has size $2^{\Omega(\sqrt{n})}$. This gives the same lower bound on the running time of all existing DPLL-based propositional algorithms (which produce a decision-DNNF as a trace) since the running time must be at least as large as the size of the trace. On the other hand, by Theorem 4.21 of Dalvi and Suciu [2012], since Q is monotone and satisfies $\mu(\hat{0}, \hat{1}) = 0$, it has a polynomial-time lifted inference algorithm for query evaluation. □

Example 3.15. In the spirit of Example 2.6, consider the following extended vocabulary of a knowledge base:

```
Blogger(name)
Friend(name1, name2)
Manager(name1, name2)
Family(name1, name2)
Tweeter(name)
```

Consider the following partial constraints:

$C_0 = \forall x \forall y\ \texttt{Friend}(x, y) \Rightarrow \texttt{Blogger}(x)$ "Friends are bloggers"
$C_1 = \forall x \forall y\ \neg\texttt{Friend(x,y)} \vee \neg\texttt{Manager}(x, y)$ "Managers are not friends"
$C_2 = \forall x \forall y\ \neg\texttt{Manager(x,y)} \vee \neg\texttt{Family}(x, y)$ "Managers are not family"
$C_3 = \forall x \forall y\ \texttt{Family}(x, y) \Rightarrow \texttt{Tweeter}(y)$ "Family members are tweeters"

Such constraints arrive, for example, by converting Markov Logic Networks [Domingos and Lowd 2009] to tuple-independent probabilistic databases, see, for example, Gribkoff and Suciu [2016]. Consider the following complex constraint:

$$C = (C_0 \wedge C_2) \vee (C_0 \wedge C_3) \vee (C_1 \wedge C_3).$$

It says that: "either all friends are bloggers and managers are not family members, or all friends are bloggers and all family members are tweeters, or all managers are not friends and family members are tweeters." It is straightforward to extend the definition of lineage (Definition 2.5) to universal quantifiers, negations, and implications, in order to compute the lineage of the constraint C. Then we can run a DPLL-based algorithm to compute the probability $\Pr[C]$. Theorem 3.14 can be used to show that such an algorithm takes exponential time, while $\Pr[C]$ can be computed in polynomial time in the size of the probabilistic database. To see this, it suffices to define the following relations:

$$\begin{aligned} R(x) &= \neg\texttt{Blogger(x)} \\ S_1(x, y) &= \texttt{Friend}(x, y) \\ S_2(x, y) &= \texttt{Manager}(x, y) \\ S_3(x, y) &= \texttt{Family}(x, y) \\ T(y) &= \neg\texttt{Tweeter}(y). \end{aligned}$$

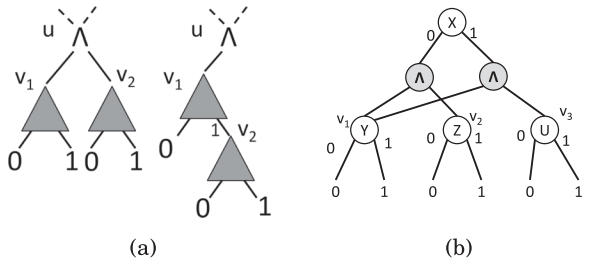
Negation is defined over the active domain of the database, that is, $R(x)$ is true on all constants x where $\texttt{Blogger}(x)$ is false. Then the negation $\neg C$ is precisely the query Q_W in Example 3.13, because $\neg C_0 = \exists x \exists y\ R(x) \wedge S_1(x, y) = h_{03}$, $\neg C_1 = \exists x \exists y\ S_1(x, y) \wedge S_2(x, y) = h_{13}$, and so on.

4. LOWER BOUNDS FOR KNOWLEDGE REPRESENTATIONS

In this section, we prove Theorem 3.1 by describing a construction to convert a DLDD $\mathcal{D}$ to an FBDD $\mathcal{F}$. Recall that for a node u in $\mathcal{D}$ or $\mathcal{F}$, the sub-DAGs rooted at u are denoted by $\mathcal{D}_u$ and $\mathcal{F}_u$, respectively.

4.1. Main Ideas

To construct $\mathcal{F}$ we must remove all OP-nodes in $\mathcal{D}$ and replace them with decision nodes. Assume that $\mathcal{D}$ has a single OP-node, which is an AND-node u with two children, v_1, v_2. We need to replace this node with an FBDD for the expression $\Phi_{v_1} \wedge \Phi_{v_2}$. Both $\mathcal{D}_{v_1}$ and $\mathcal{D}_{v_2}$ are already FBDDs, and then we can remove the AND-node easily as illustrated



(a) (b)

Fig. 5. (a) Basic construction for converting a decision-DNNF into an FBDD (b) where it fails.

in Figure 5(a): Stack $\mathcal{D}_{v_1}$ over $\mathcal{D}_{v_2}$, and redirect all 1-sink nodes in $\mathcal{D}_{v_1}$ to the root of $\mathcal{D}_{v_2}$. Clearly, this computes the same AND function; moreover, it satisfies the read-once condition required for FBDD, because $\mathcal{D}_{v_1}, \mathcal{D}_{v_2}$ do not have any common variable. This idea can be extended to the case when the OP-node is an OR-node by redirecting all 0-sink nodes from $\mathcal{D}_{v_1}$ to the root of $\mathcal{D}_{v_2}$. If this construction worked in general, then any DLDD could be converted into an FBDD of at most twice the size.

But, in general, this simple idea fails, as can be seen on the simple decision-DNNF in Figure 5(b) (it computes $(\neg X)YZ \vee XYU$). To compute the first AND node, we need to stack $\mathcal{D}_{v_1}$ over $\mathcal{D}_{v_2}$ and to compute the second AND node we need to stack $\mathcal{D}_{v_1}$ over $\mathcal{D}_{v_3}$: This creates a conflict for redirecting the 1-sink node in $\mathcal{D}_{v_1}$ to v_2 or to v_3.[10] To get around that, we use two ideas. The first idea is to make copies of some subgraphs. For example, if we make two copies of $\mathcal{D}_{v_1}$, we call them $\mathcal{D}_{v_1}$ and $\mathcal{D}'_{v_1}$, and then we can compute the first AND-node by stacking $\mathcal{D}_{v_1}$ over $\mathcal{D}_{v_2}$ and compute the second AND-node by stacking $\mathcal{D}'_{v_1}$ over $\mathcal{D}_{v_3}$, and the conflict is resolved. The second idea is to reorder the children of the AND-nodes to limit the exponential blowup due to copying. We present the details next.

4.2. The Construction of $\mathcal{F}$

We will convert $\mathcal{D}$ into an FBDD $\mathcal{F}$ augmented with NOT-nodes and *NO-OP nodes* (unlabeled nodes having only one outgoing edge). We call such an object FBDD$^\neg$. First, we state a simple observation:

PROPOSITION 4.1. *Any FBDD$^\neg$ can be converted into an equivalent FBDD by at most doubling the number of nodes (and the conversion can be computed in linear time in the size of the FBDD$^\neg$).*

PROOF. The idea is to use double-rail logic. Fix a FBDD$^\neg$ $\mathcal{F}$. Make another copy of $\mathcal{F}$, say, $\mathcal{F}'$, which is exactly the same as $\mathcal{F}$ except the 0/1 labels of all sink nodes are flipped. Therefore, not only does $\mathcal{F}'$ compute the negation of the function computed by $\mathcal{F}$, but also every node u' in $\mathcal{F}'$ computes the negation of the function computed by the corresponding node u in $\mathcal{F}$ and vice versa. We make a combined FBDD from the union of $\mathcal{F}$ and $\mathcal{F}'$ as follows: For every NOT-node u in $\mathcal{F}$ with unique child v in $\mathcal{F}$ and the corresponding NOT-node u' and child v' in $\mathcal{F}'$, reconnect u to v' and u' to v and change both u and u' to NO-OP nodes. The root will be the root of $\mathcal{F}$. Finally, for each NO-OP node u, replace u by its unique child v until all NO-OP nodes have been removed. (Any node not reachable from the root can also be removed.) □

For each node u of a DLDD $\mathcal{D}$, let M_u be the number of binary OP-nodes in the subgraph $\mathcal{D}_u$. If u is an OP-node, then we have $M_u = 1 + M_{v_1} + M_{v_2}$, because, by

[10] In this particular example, one could stack $\mathcal{D}_{v_2}$ and $\mathcal{D}_{v_3}$ over $\mathcal{D}_{v_1}$ and avoid the conflict, but, in general, $\mathcal{D}_{v_2}, \mathcal{D}_{v_3}$ may have conflicts with other subgraphs.

definition, the two DAGs $\mathcal{D}_{v_1}$ and $\mathcal{D}_{v_2}$ are disjoint; we will always assume that $M_{v_1} \leq M_{v_2}$ (otherwise we swap the two children of the OP-node u), and this implies that $M_u \geq 2M_{v_1} + 1$. We classify the edges of the DLDD into three categories: (u, v) is a *light edge* if u is an OP-node and v its first child, (u, v) is a *heavy edge* if u is an OP-node and v is its second child, and (u, v) is a *neutral edge* if u is a decision node or NOT-node. We always have $M_u \geq M_v$, while for a light edge we have $M_u \geq 2M_v + 1$.

THEOREM 4.2. *For any DLDD $\mathcal{D}$ with at most N nodes, M binary OP-nodes, if there are at most L light edges on any path in $\mathcal{D}$ from the root to a sink, then there exists an equivalent FBDD$^\neg$ $\mathcal{F}$ computing the same function as $\mathcal{D}$, with at most NM^L nodes. Furthermore, if $\mathcal{D}$ is positive, then $\mathcal{F}$ is an FBDD. Moreover, $\mathcal{F}$ can be constructed in time linear in its size.*

Theorem 3.1 immediately follows from Proposition 4.1 and Theorem 4.2 and the fact that $M + 1 \leq N$ and $L \leq \log_2 M \leq \log_2 N$. Indeed, consider any path in $\mathcal{D}$ with L light edges, $(u_1, v_1), (u_2, v_2), \ldots, (u_L, v_L)$. We have $M_{u_i} \geq 2M_{v_i} + 1$ and $M_{v_i} \geq M_{u_{i+1}}$ for all i, and we also have $M \geq M_{u_1}$ and $M_{v_L} \geq 0$, which implies $M \geq 2^L - 1$ (by induction on L). Therefore, $2^L \leq M + 1 \leq N$ (because $\mathcal{D}$ has at least one node that is not an OP-node), and $NM^L = N2^{L \log M} \leq N2^{\log^2 N}$, proving the claim.

In the rest of this section, we prove Theorem 4.2. Fix the DLDD $\mathcal{D}$. Let u denote a node in $\mathcal{D}$ and P denote a path from the root to u. Let $s(P)$ be the set of light edges on the path P, and let $S(u)$ consist of the sets $s(P)$ for all paths from the root to u, formally:

$$s(P) = \{(v, w) \mid (v, w) \text{ is a light edge in } P\}$$
$$S(u) = \{s(P) \mid P \text{ is a path from the root to } u\}.$$

We view the set of light edges $s = s(P)$ as a sequence of edges ordered by their occurrences in P (from the root to u). This order is independent of P: If $s = s(P) = s(P')$ where P, P' are two paths from the root to the node u, then any light edges that occur in both P and P' must occur in the same order, since $\mathcal{D}$ is acyclic.

We will assume, without loss of generality, that $\mathcal{D}$ has exactly two sink nodes, namely 0 and 1. We define $\mathcal{F}$ formally. Its nodes are pairs (u, s) where u is a node in $\mathcal{D}$, $s \in S(u)$. The root node is $(\texttt{root}(\mathcal{D}), \emptyset)$. We define the edges and node labels of $\mathcal{F}$ as follows: The edges in $\mathcal{F}$ and are of three types:

Type 1: For each light edge $e = (u, v)$ in $\mathcal{D}$, and every $s \in S(u)$, add the edge $((u, s), (v, s \cup \{e\}))$ to $\mathcal{F}$. Label the node (u, s) as a NO-OP node.

Type 2: For every neutral edge (u, v) in $\mathcal{D}$, and every $s \in S(u)$, add the edge $((u, s), (v, s))$ to $\mathcal{F}$. Label the node (u, s) as u was labeled in $\mathcal{D}$; that is, (u, s) is precisely the same kind of node that u was in $\mathcal{D}$.

Type 3: For each of the two sink nodes $b \in \{0, 1\}$ in $\mathcal{D}$, and every non-empty $s \in S(b)$, add one edge of type 3 from (b, s) as follows. Let $e = (u, v_1)$ be the last edge in the set s and g_u be the operator of the OP-node u. Let v_2 be the heavy child of u. Then the function $g_u(b, Y)$ is Y, $\neg Y$, or a constant b', and we construct the edge accordingly, in each case as follows:

(1) If $g_u(b, Y) = Y$, then add the edge $((b, s), (v_2, s - \{e\}))$ and label (b, s) a NO-OP node.
(2) If $g_u(b, Y) = \neg Y$, then add the edge $((b, s), (v_2, s - \{e\}))$ and label (b, s) a NOT-node.
(3) If $g_u(b, Y) = b' \in \{0, 1\}$, then add the edge $((b, s), (b', s - \{e\}))$ and label (b, s) a NO-OP node.

Finally, for each $b \in \{0, 1\}$, $(b, \emptyset)$ is the unique sink node with label b. Note that if $\mathcal{D}$ is positive, then there are no NOT-nodes in $\mathcal{D}$ and $g_u(b, Y) \neq \neg Y$ for all b and all nodes u, so $\mathcal{F}$ does not have any NOT-nodes.

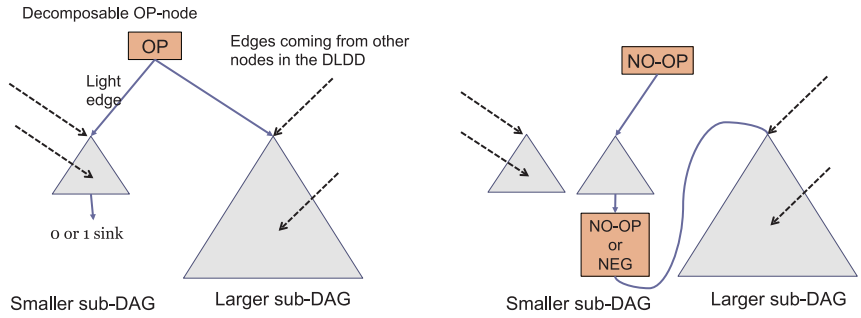


Fig. 6. Illustration of conversion of DLDD to FBDD$^\neg$: (a) A decomposable OP-node with smaller and larger subgraphs and (b) new connection to larger subgraph with NO-OP or NOT node (see text).

It is easy to see that $\mathcal{F}$ is a well-defined fully labeled directed graph with decision, NO-OP, and NOT-nodes and two sinks, since every OP-node in $\mathcal{D}$ has a light out-edge and every other non-sink node has a neutral out-edge. Also, the out-degrees of these nodes match the requirements for their labels. It remains to show that $\mathcal{F}$ is acyclic and satisfies the read-once property, is equivalent to $\mathcal{F}$, and has at most NM^L nodes.

The intuition behind the sets/sequences s associated with each node of $\mathcal{F}$ is that they allow us to keep track of what can be seen as an iterative copying process in the conversion from the DLDD to the FBDD$^\neg$. Each additional light edge in s can be used to make a fresh copy of the descendants of the left child v_1 of an OP-node u. To illustrate this, suppose that $\mathcal{D}$ has a single OP-node u with two children v_1, v_2, and let $e = (u, v_1)$ be the light edge. Suppose that there are other neutral edges into $\mathcal{D}_{v_1}$. Then $\mathcal{F}$ contains two copies of the subgraph $\mathcal{D}_{v_1}$, one with nodes labeled $(w, \{e\})$, and the other with nodes labeled $(w, \emptyset)$. Any 1-sink node in the first copy becomes a NO-OP or NOT-node in $\mathcal{F}$ and is connected to v_2 (similarly to Figure 5(a)). The OP-node becomes a NO-OP node with out-degree one. The sink nodes and neutral edges to the second copy remain as before. This copying process is repeated in $\mathcal{D}_{v_1}$. The first step for this process is illustrated in Figure 6.

4.3. Proof of Theorem 4.2

Theorem 4.2 follows from the following three lemmas:

LEMMA 4.3. *$\mathcal{F}$ has at most NM^L nodes.*

PROOF. Recall that all nodes in $\mathcal{F}$ are of the form (u, s). There are N possible choices for the node u, and at most M^L possible choices for the set s, because $|s| \leq L$ (since every path has $\leq L$ light edges), and M is the number of light edges. □

LEMMA 4.4. *$\mathcal{F}$ is a correct FBDD$^\neg$.*

LEMMA 4.5. *$\mathcal{F}$ computes the same function as $\mathcal{D}$.*

PROOF OF LEMMA 4.4. As we have already noted, the node labels and edges of $\mathcal{F}$ are locally consistent with it being an FBDD$^\neg$. It remains to prove that $\mathcal{F}$ is a DAG and it satisfies the read-once condition of being an FBDD$^\neg$. Since NO-OP and NOT nodes have out-degree 1, these properties follow immediately from the following claim:

CLAIM. If u is a decision node in $\mathcal{D}$ labeled with a variable X, and there exists a non-empty path (with at least one edge) between the nodes (u, s) and (v, s') in $\mathcal{F}$, then the variable X does not occur in $\mathcal{D}_v$.

Indeed, the claim implies that $\mathcal{F}$ is acyclic, because any cycle in $\mathcal{F}$ implies a non-empty path from some node (u, s) to itself, and, obviously, $X \in \mathcal{D}_u$, contradicting the claim. It also implies that every path in $\mathcal{F}$ is read-once: If a path tests a variable X twice, once at (u, s) and once at (u_1, s_1), then $X \in \mathcal{D}_{u_1}$, contradicting the claim. It remains to prove the claim.

Suppose, to the contrary, that there exists a node (u, s) such that u is labeled with X and there exists a path from (u, s) to (v, s') in $\mathcal{F}$ such that X occurs in $\mathcal{D}_v$. Choose v such that $\mathcal{D}_v$ is maximal; in other words, there is no path from (u, s) to some (v', s'') such that $\mathcal{D}_v \subset \mathcal{D}_{v'}$ (in the latter case replace v with v': we still have that X occurs in $\mathcal{D}_{v'}$). Consider the last edge on the path from (u, s) to (v, s') in $\mathcal{F}$:

$$(u, s), \ldots, (w, s''), (v, s'). \tag{2}$$

Observe that (w, v) is not an edge in $\mathcal{D}$ since $\mathcal{D}_v$ is maximal and since (u, v) is not an edge in $\mathcal{D}$ by the read-once property of $\mathcal{D}$; therefore, the edge from (w, s'') to (v, s') is of Type 3. It cannot be of type 3.c, because, in that case, v is a sink node and thus $X \notin \mathcal{D}_v$. Thus, it is of type 3.a or 3.b, and, hence, there exists an OP-node z with children v_1, v, and our last edge is of the form $(w, s' \cup \{e\}), (v, s')$, where $e = (z, v_1)$ the light edge from z. We claim that $e \notin s$; that is, it is not present at the beginning of the path in Equation (2). Otherwise, if $e \in s$, then there exists a path from v_1 to u (recall that s consists of light edges on some path from the root to u), which means that the variable X occurs both in $\mathcal{D}_{v_1}$ (because it is the label of the node u) and in $\mathcal{D}_v$ (by assumption, X occurs in $\mathcal{D}_v$), but this contradicts the fact that z is a decomposable binary operation (Definition 2.4), meaning that its two children do not share any common variables. This proves $e \notin s$. On the other hand, $e \in s''$. Now consider the first node on the path in Equation (2) where e is introduced. It can only be an edge of the form $(z, s_1), (v_1, s_1 \cup \{e\})$. But now we have a path from (u, s) to (z, s_1) with $X \in \mathcal{D}_z \supset \mathcal{D}_v$, contradicting the maximality of v. This proves the claim.

Proof of Lemma 4.5. Denote by $F(\mathcal{D})$ the function whose input is a DLDD $\mathcal{D}$ and that returns the FBDD$^{\neg}$ $\mathcal{F}$ described above. Recall that for every node $u \in \mathcal{D}$, Φ_u denotes the function computed by the sub-DAG of $\mathcal{D}$ rooted at u. Similarly, for a node $u' = (u, s)$ in $\mathcal{F}$, let $\Psi_{u'}$ denote the function computed by the sub-DAG of $\mathcal{F}$ rooted at u'. To prove Lemma 4.5, we need to show that $\Phi_r = \Psi_{(r,\emptyset)}$, where r is the root node in $\mathcal{D}$. For that we will establish a stronger relationship between the functions computed at the nodes in $\mathcal{D}$ and in $\mathcal{F}$ by proving inductive properties of our formal construction.

We start by observing a simple property of the functions computed in $\mathcal{F}$:

Proposition 4.6. *Let $\mathcal{D}$ be any DLDD and $\mathcal{F} = F(\mathcal{D})$. (1) If u is a decision node in $\mathcal{D}$ testing the variable X, and with children v_0, v_1, then the function computed by $\mathcal{F}$ at the node (u, s) is as follows:*

$$\Psi_{(u,s)} = (\neg X)\Psi_{(v_0,s)} \vee X\Psi_{(v_1,s)}.$$

Similarly, if u is a NOT-node with child v, then $\Psi_{(u,s)} = \neg\Psi_{(v,s)}$. (2) Let $b \in \{0, 1\}$ be a sink node in $\mathcal{D}$ such that $u' = (b, s)$ is not a sink node in $\mathcal{F}$; then u' has unique child v', where (u', v') is an edge of Type 3. Let $e = (u, v_1)$ be the last edge in s, and let g_u the binary operator labeling u. Then, the function computed by $\mathcal{F}$ at u' is as follows:

$$\Psi_{u'} = g_u(b, \Psi_{v'}).$$

Proof. Equation (1) follows immediately from the construction of $\mathcal{F}$, because these node types do not change when we move from $\mathcal{D}$ to $\mathcal{F}$. We prove Equation (2) by examining the same cases as in the Type 3 edges and labels:

(1) If $g_u(b, Y) = Y$, then the node u' is a NO-OP, hence $\Psi_{u'} = \Psi_{v'} = g(b, \Psi_{v'})$.
(2) If $g_u(b, Y) = \neg Y$, then the node u' is a NOT-node and $\Psi_{u'} = \neg\Psi_{v'} = g(b, \Psi_{v'})$

(3) If $g_u(b, Y) = b' \in \{0, 1\}$, then u' is again a NO-OP and we reuse the argument in (a). □

To prove Lemma 4.5, we will apply Proposition 4.6 and the definition of the mapping F to prove the following claim.

CLAIM. For each node u in $\mathcal{D}$, if we write $\mathcal{F}^u = F(\mathcal{D}_u)$ and let Ψ^u denote the function mapping each node of $\mathcal{F}^u$ to the Boolean function it computes, then

$$\Psi^u_{(u,\emptyset)} = \Phi_u. \tag{3}$$

Lemma 4.5 follows immediately by applying the claim to the root node r of $\mathcal{D}$. We need to use the different functions Ψ^u because, in general, $F(\mathcal{D}_u)$ is not a subset of $F(\mathcal{D})$ (which is $F(\mathcal{D}_r)$) since the sets of light edges s associated with nodes in $F(\mathcal{D})$ will be longer than those in $F(\mathcal{D}_u)$. We prove Equation(3) by induction on the size of $\mathcal{D}_u$ and consider four cases, depending on the type of u:

Sink node. If $u \in \{0, 1\}$ is a sink node in $\mathcal{D}$, then, by construction, $\mathcal{F}^u = F(\mathcal{D}_u)$ consists of a single sink node $(u, \emptyset)$. Both $\Psi^u_{(u,\emptyset)}$ and Φ_u are the constant $u \in \{0, 1\}$, proving the claim.

Decision node. Let X be the variable labeling u and v_0, v_1 its two children. By the definition of $\mathcal{F}^u = F(\mathcal{D}_u)$, the node $(u, \emptyset)$ is labeled X and has two children $(v_0, \emptyset)$ and $(v_1, \emptyset)$. Here, the FBDD $\mathcal{F}^u_{(v_0,\emptyset)}$ is the same as $\mathcal{F}^{v_0} = F(\mathcal{D}_{v_0})$, and, similarly, $\mathcal{F}^u_{(v_1,\emptyset)}$ is the same as $\mathcal{F}^{v_1} = F(\mathcal{D}_{v_1})$, so we can apply the induction hypothesis to obtain $\Psi^u_{(v_0,\emptyset)} = \Psi^{v_0}_{(v_0,\emptyset)} = \Phi_{v_0}$ and $\Psi^u_{(v_1,\emptyset)} = \Psi^{v_1}_{(v_0,\emptyset)} = \Phi_{v_1}$. Applying Proposition 4.6 (1) to $\mathcal{D}_u$, we obtain $\Psi^u_{(u,\emptyset)} = (\neg X)\Psi^u_{(v_0,\emptyset)} \vee X\Psi^u_{(v_1,\emptyset)}$, which is equal to $(\neg X)\Phi_{v_0} \vee X\Phi_{v_1} = \Phi_u$, proving the claim.

NOT-node. If u is a NOT-node of $\mathcal{D}$ with child v, then again $\mathcal{F}^u_{(v,\emptyset)}$ is the same as $\mathcal{F}^v = F(\mathcal{D}_v)$ and so, by induction, $\Psi^u_{(v,\emptyset)} = \Psi^v_{(v,\emptyset)} = \Phi_v$. Applying Proposition 4.6 (1) to $\mathcal{D}_u$, we obtain $\Psi^u_{(u,\emptyset)} = \neg\Psi^u_{(v,\emptyset)} = \neg\Phi_v = \Phi_u$.

OP-node. Let $g_u : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$ be the operator labeling the node u, and let u_1 and u_2 be its two children. By definition, we have $\Phi_u = g_u(\Phi_{u_1}, \Phi_{u_2})$, and, by induction, we have $\Psi^{u_1}_{(u_1,\emptyset)} = \Phi_{u_1}$ and $\Psi^{u_2}_{(u_2,\emptyset)} = \Phi_{u_2}$.

At this point, we examine the connection between $\mathcal{F}^u$ and $\mathcal{F}^{u_1}, \mathcal{F}^{u_2}$. In essence, $\mathcal{F}^u$ is obtained by re-wiring some sink nodes in $\mathcal{F}^{u_1}$ to the root node in $\mathcal{F}^{u_2}$. More precisely, $\mathcal{F}^u$ is obtained as follows from $\mathcal{F}^{u_1}, \mathcal{F}^{u_2}$. (a) It has a new root node $(u, \emptyset)$; (b) if $e = (u, u_1)$ is the light edge at u, then all nodes in $\mathcal{F}^{u_1}$ are renamed from (w, s') to $(w, \{e\} \cup s')$ (with e ordered first); the root $(u, \emptyset)$ of $\mathcal{F}^u$ has a single outgoing edge to the former root of $\mathcal{F}^{u_1}$, now renamed to $(u_1, \{e\})$; (c) for every sink node $(b, \emptyset)$ in $\mathcal{F}^{u_1}$, where $b \in \{0, 1\}$, $\mathcal{F}^u$ has a new edge of type-3 from $u' = (b, \{e\})$ to some node v' (either the root of $\mathcal{F}^{u_2}$ which is $(u_2, \emptyset)$, or a new sink node, depending on whether the edge is of type 3.a, 3.b, or 3.c). Here, Proposition 4.6 (2) immediately implies that $\Psi^u_{u'} = g_u(b, \Psi^u_{v'})$. We also have (d) $\mathcal{F}^u_{(u_2,\emptyset)} = \mathcal{F}^{u_2}$ and hence, when $v' = (u_2, \emptyset)$, $\Psi^u_{v'} = \Psi^{u_2}_{(u_2,\emptyset)}$. This immediately implies that $\Psi^u_{u'} = g_u(b, \Psi^{u_2}_{(u_2,\emptyset)})$ since in the only case (3.c) that $v' \neq (u_2, \emptyset)$, $g_u(b, \cdot)$ does not depend on its second argument.

We claim that the function $\Psi^u_{(u,\emptyset)}$ computed by $\mathcal{F}^u$ is $g_u(\Psi^{u_1}_{(u_1,\emptyset)}, \Psi^{u_2}_{(u_2,\emptyset)})$: This completes the proof of Equation (3) because the induction hypothesis yields $\Psi^{u_1}_{(u_1,\emptyset)} = \Phi_{u_1}$ and $\Psi^{u_2}_{(u_2,\emptyset)} = \Phi_{u_2}$, and we have $g_u(\Phi_{u_1}, \Phi_{u_2}) = \Phi_u$. To prove our claim, we will show that the two functions have the same values for any truth assignment θ; in other words:

$$\Psi^u_{(u,\emptyset)}[\theta] = g_u\left(\Psi^{u_1}_{(u_1,\emptyset)}, \Psi^{u_2}_{(u_2,\emptyset)}\right)[\theta]. \tag{4}$$

The left-hand side of Equation (4) is obtained by following the path θ in the FBDD $\mathcal{F}^u$. This path starts at the root; follows the edge from $(u, \emptyset)$ to $(u_1, \{e\})$, leading to the root of a copy of $\mathcal{F}^{u_1}$ in which every node has light edge e added to the start of the sequence of light edges in its name; and then follows a path within this modified $\mathcal{F}^{u_1}$, up to some former sink node $(b, \emptyset)$ of $\mathcal{F}^{u_1}$, which is now $u' = (b, \{e\})$. Therefore, since the copy of the unmodified $\mathcal{F}^{u_1}$ must also have contained this path, we have the following:

$$\begin{aligned} \Psi^{u_1}_{(u_1,\emptyset)}[\theta] &= b \\ \Psi^{u}_{(u,\emptyset)}[\theta] &= \Psi^{u}_{u'}[\theta]. \end{aligned}$$

On the other hand, we have shown above that $\Psi^u_{u'} = g_u(b, \Psi^{u_2}_{(u_2,\emptyset)})$, and, in particular:

$$\Psi^u_{u'}[\theta] = g_u\big(b, \Psi^{u_2}_{(u_2,\emptyset)}\big)[\theta].$$

By combining these three equalities, we obtain Equation (4).

5. EXPONENTIAL LOWER BOUNDS FOR ALL H_k

In this section, we prove Theorem 3.6, which gives lower bounds on the sizes of FBDDs computing all H_k. We find it convenient to prove these bounds assuming a natural property of FBDDs. We show that we can ensure this property with only minimal change in FBDD size, yielding our claimed lower bounds.

Let Φ be a Boolean formula. A *prime implicant* (or *minterm*) of Φ is a term T such that $T \Rightarrow \Phi$ and no proper subterm of T implies Φ. If T involves k variables, then we call it a *k-prime implicant*. 1-prime implicants are also known as *unit variables*. For example, X and W are unit in $X \vee YZ \vee YU \vee W$.

The following definition is motivated by the *unit clause rule* in DPLL algorithms, which are primarily designed for satisfiability of CNF formulas. If there is any clause consisting of a single variable or its negation (a unit clause), then DPLL immediately sets such variables, one after another, since their value is forced.

Definition 5.1. Let $\mathcal{F}$ be an FBDD for a Boolean function Φ. Call a node u in $\mathcal{F}$ a *unit node* if the variable tested at u is a unit in Φ_u and a *decision node* otherwise. We say that $\mathcal{F}$ follows the *unit rule* if, for every node u, if Φ_u has a unit, then u is a unit node.

In the special case that Φ is a monotone formula, we can apply a transformation in order to convert any FBDD $\mathcal{F}$ for Φ into one that follows the unit rule and is not much larger than $\mathcal{F}$.

We say that two variables X, Y of Φ are *neighbors* if there exists a partial assignment θ such that Y is not a unit in $\Phi[\theta]$ and is a unit in $\Phi[\theta \cup \{X = 1\}]$. If Φ is monotone, then the neighbor relation is symmetric, and X, Y are neighbors iff they co-occur in some minterm of Φ. We define the *degree* of a variable X to be the number of neighbors of X and write $\Delta(\Phi)$ for the maximum degree of any variable in Φ. In Section 5.2, we prove the following:

LEMMA 5.2. *If Φ is a monotone formula with FBDD $\mathcal{F}$ of size N, then Φ has an FBDD of size at most $\Delta(\Phi) \cdot N$ that follows the unit rule.*

Since H_k obviously has degree at most n (for variables $R(i)$ and $T(j)$), we obtain the following corollary.

COROLLARY 5.3. *If H_k has an FBDD of size N, then H_k has an FBDD of size at most nN that follows the unit rule.*

Now Theorem 3.6 is an immediate consequence of Corollary 5.3 together with the following lemma.

LEMMA 5.4. *Every FBDD $\mathcal{F}$ for H_k that follows the unit rule has size $\geq 2^n - 1$.*

In the rest of this section, we prove Lemma 5.4 and 5.2.

5.1. Proof of Lemma 5.4

For simplicity, we first use a variant of a standard method to extend $\mathcal{F}$ to another FBDD $\hat{\mathcal{F}}$ that computes the same function but is a little easier to deal with. $\hat{\mathcal{F}}$ will be obtained from $\mathcal{F}$ using different choices of the set $\mathcal{X}$ in the following definition, depending on the value of k.

Definition 5.5. Let $\mathcal{F}$ be an FBDD defined on a variable set that contains $\mathcal{X}$. For each node u in $\mathcal{F}$, let $\mathcal{X}_u$ be the set of variables in $\mathcal{X}$ that appear as labels on paths from the root to u. Define $\mathcal{F}^{\mathcal{X}}$ as follows: The nodes of $\mathcal{F}^{\mathcal{X}}$ include the nodes of $\mathcal{F}$ plus some extra dummy nodes: For every edge (u, v) of $\mathcal{F}$, if u tests variable X and (u, v) is labeled by $b \in \{0, 1\}$, then

—if $\mathcal{X}_v = \mathcal{X}_u$ or $\mathcal{X}_v = \mathcal{X}_u \cup \{X\}$, then $\mathcal{F}^{\mathcal{X}}$ has edge (u, v) exactly as $\mathcal{F}$ does;
—otherwise, let $\{X_1, \ldots, X_a\} = \mathcal{X}_v \setminus (\mathcal{X}_u \cup \{X\})$. $\mathcal{F}^{\mathcal{X}}$ has dummy nodes $(u, v, 1), \ldots, (u, v, a)$ between u and v, and there is an edge from u to $(u, v, 1)$ labeled b, where node (u, v, i) tests X_i and has both out-edges pointing to $(u, v, i + 1)$ for $i < a$ and pointing to v for (u, v, a). Further, define $\mathcal{X}_{(u,v,i)} = \mathcal{X}_u \cup \{X_1, \ldots, X_{i-1}\}$.

The following is immediate.

PROPOSITION 5.6. *For any set $\mathcal{X}$ contained in the variable set for $\mathcal{F}$, $\mathcal{F}^{\mathcal{X}}$ is an FBDD that computes the same function as $\mathcal{F}$ does, and $\mathcal{F}^{\mathcal{X}}$ satisfies the unit rule if and only if $\mathcal{F}$ does. Further, for every node u in $\mathcal{F}^{\mathcal{X}}$, every path in $\mathcal{F}^{\mathcal{X}}$ from the root to u tests precisely the same subset $\mathcal{X}_u$ of the variables in $\mathcal{X}$ (possibly testing additional variables outside $\mathcal{X}$).*

Let $\mathcal{F}$ compute H_k. For $k = 0$, let $\hat{\mathcal{F}} = \mathcal{F}$. For k odd, let $\hat{\mathcal{F}} = \mathcal{F}^{\mathcal{X}}$ for $\mathcal{X} = \{R(1), \ldots, R(n), T(1), \ldots, T(n)\}$. Finally, for $k > 0$ even, let $\hat{\mathcal{F}} = \mathcal{F}^{\mathcal{X}}$, where $\mathcal{X} = \{R(i), T(j), S_1(i, j) \mid i, j \in [n]\}$. We will prove the lower bound by considering a special class of paths in $\hat{\mathcal{F}}$ that end at nodes of $\mathcal{F}$; we call these *admissible* paths. We give such a definition and prove two properties: There are at least $2^n - 1$ distinct admissible paths, and no two admissible paths can lead to the same node. These two properties prove the lemma. We begin by defining a set of admissible total assignments on which we will base our definition of admissible paths.

Definition 5.7. Let k be a non-negative integer. The set $\mathcal{A}_k$ of admissible assignments consists of all total assignments θ to the variables of H_k satisfying the following:

(1) If $k = 0$, for every $i, j \in [n]$, θ assigns values so $S(i, j) = \neg(R(i) \wedge T(j))$.
(2) If k is odd, then for all $i, j \in [n]$, θ assigns values so
    —$S_\ell(i, j) = \neg(R(i) \vee T(j))$ for ℓ odd, and
    —$S_\ell(i, j) = R(i) \vee T(j)$ for ℓ even.
(3) If $k > 0$ is even, then for all $i, j \in [n]$, θ assigns values so
    —$S_1(i, j) = \neg R(i)$,
    —$S_\ell(i, j) = \neg(\neg R(i) \vee T(j))$ for ℓ even, and
    —$S_\ell(i, j) = \neg R(i) \vee T(j)$ for $\ell > 1$ odd.

Figure 7 illustrates admissible assignments for $k = 0$, 5, and 4.

$R(i)$	$S(i,j)$	$T(j)$
0	1	0
0	1	1
1	0	1
1	1	0

$R(i)$	$S_1(i,j)$	$S_2(i,j)$	$S_3(i,j)$	$S_4(i,j)$	$S_5(i,j)$	$T(j)$
0	1	0	1	0	1	0
0	0	1	0	1	0	1
1	0	1	0	1	0	0
1	0	1	0	1	0	1

$R(i)$	$S_1(i,j)$	$S_2(i,j)$	$S_3(i,j)$	$S_4(i,j)$	$T(j)$
0	1	0	1	0	0
0	1	0	1	0	1
1	0	1	0	1	0
1	0	0	1	0	1

Fig. 7. The patterns for admissible assignments for $k = 0$, $k = 5$, and $k = 4$.

Definition 5.8. Let $i_0, j_0 \in [n]$. For $k = 0$ define the set $\mathcal{R}^k_{i_0}$ of *variables in row* i_0 to be $R(i_0), S(i_0, j)$ for all $j \in [n]$; for $k \geq 1$ define $\mathcal{R}^k_{i_0}$ to be all $R(i_0), S_1(i_0, j), \ldots, S_k(i_0, j)$ for all $j \in [n]$. For $k = 0$ define the set $\mathcal{C}^k_{j_0}$ of *variables in column* j_0 to be to be $S(i, j_0), T(j_0)$ for all $i \in [n]$. For k odd define the set $\mathcal{C}^k_{j_0}$ to be $S_1(i, j_0), \ldots, S_k(i, j_0), T(j_0)$ for all $i \in [n]$ and for $k > 0$ even, define the set $\mathcal{C}^k_{j_0}$ to be $S_2(i, j_0), \ldots, S_k(i, j_0), T(j_0)$ for all $i \in [n]$. Finally, for each k define the set $\mathcal{T}^k_{i_0 j_0}$ of *variables in the* (i_0, j_0) *transversal* by $\mathcal{T}^0_{i_0 j_0} = \{R(i_0), S(i_0, j_0), T(j_0)\}$ and $T^k_{i_0 j_0} = \{R(i_0), S_1(i_0, j_0), \ldots, S_k(i_0, j_0), T(j_0)\}$ for $k > 0$.

Note that we omit $S_1(i, j_0)$ for all i from the variables in column j_0 when $k > 0$ is even. This is natural because among all admissible assignments, each of these variables is equal to the negation of $R(i)$ and hence does not depend on the column j_0.

PROPOSITION 5.9.

(1) *For every integer* $k \geq 0$*, and every* $\theta \in \mathcal{A}_k$*, we have* $H_k[\theta] = 0$.
(2) *For fixed* $k > 0$ *and each pair of values* $b_R, b_T \in \{0, 1\}$*, there is precisely one sequence of values* $S_1(i, j), \ldots S_k(i, j)$ *that is agreed on by all extensions of* $R(i) = b_R$, $T(j) = b_T$ *in* $\mathcal{A}_k$*. The same is true for* $S(i, j)$ *when* $k = 0$.
(3) *For* k *odd or* $k = 0$*, the admissible assignments are symmetric with respect to rows and columns.*
(4) *For* $k > 0$ *even, an assignment* θ *to* $(R, S_1, \ldots, S_k, T)$ *is admissible if and only if*
   —$R(i) = \neg S_1(i, j)$ *for all* $i, j \in [n]$*, and*
   —*the unique assignment* θ_{k-1} *to* $(R', S'_1, \ldots, S'_{k-1}, T')$ *given by* $R'(i) = S_1(i, j)$, $S'_1(i, j) = S_2(i, j), \ldots, S'_{k-1}(i, j) = S_k(i, j), T'(j) = T(j)$ *is admissible.*
(5) *For any integer* k *and any* $i, j \in [n]$*,*
   (a) *for any assignment* $\theta \in \mathcal{A}_k$*, there is an assignment* $\theta' \in \mathcal{A}_k$ *that agrees with* θ *everywhere except possibly in the variables in* $\mathcal{C}^k_j$ *and has the opposite value of* $T(j)$ *from* θ.

(*b*) *for any assignment* $\theta \in \mathcal{A}_k$*, there is an assignment* $\theta' \in \mathcal{A}_k$ *that agrees with* θ *everywhere except in the variables in* $\mathcal{R}_i^k$ *and has the opposite value of* $R(i)$ *from* θ*.*

(*c*) *for every assignment* $\theta \in \mathcal{A}_k$ *and every variable* X *in* $\mathcal{T}_{ij}^k$*, there is an assignment* $\theta'' \in \mathcal{A}_k$ *that agrees with* θ *everywhere except in the variables in* $\mathcal{R}_i^k \cup \mathcal{C}_j^k$ *that has the opposite value for* X*.*

Proof. Parts (1), (2), (3), and (4) are immediate by inspection. We show part (5) by cases based on the value of k.

First, suppose that $k = 0$: For sub-part (5a), we create θ' from θ by flipping the value of $T(j)$ and flipping the value of $S(i, j)$ for every $i \in [n]$ for which $R(i) = 1$. For sub-part (5b), we create θ' from θ by flipping the value of $R(i)$ and flipping the value of $S(i, j)$ for every $j \in [n]$ for which $T(j) = 1$. Finally, for sub-part (5c), by parts (5a) and (5b) we can already do this for $R(i)$ and $T(j)$. To obtain θ'' that flips $S(i, j)$, if θ sets either $R(i)$ or $T(j)$ to 1 then the corresponding θ' from flipping the other one from sub-parts (5b) or (5a) will flip $S(i, j)$. If θ sets $R(i) = T(j) = 0$, then we first apply (5b) to get θ' having $R(i) = 1$ and then apply (5a) to get θ'' that flips $S(i, j)$.

Now suppose that k is odd: For sub-part (5a), we create θ' from θ by flipping the value of $T(j)$ and flipping the values of $S_1(i, j), \ldots, S_k(i, j)$ for every $i \in [n]$ for which $R(i) = 0$. Sub-part (5b) is symmetric—we create θ' from θ by flipping the value of $R(i)$ and flipping the values of $S_1(i, j), \ldots, S_k(i, j)$ for every $j \in [n]$ for which $T(j) = 0$. Finally, for sub-part (5c), by parts (5a) and (5b) we can already do this for $R(i)$ and $T(j)$; whenever θ sets at least one of $R(i) = 0$ or $T(j) = 0$, we can also do this for all of $S_1(i, j), \ldots, S_k(i, j)$ by flipping the other one. If $R(i) = T(j) = 1$, then we first apply (5b) to get θ' having $R(i) = 0$ and then apply (5a) to get θ'' that flips each of $S_1(i, j), \ldots, S_k(i, j)$.

Finally, we handle the case that $k > 0$ is even. Using part (4) we let θ_{k-1} be the translation of assignment θ given by $R', S_1', \ldots, S_{k-1}', T'$. It is easy to see that applying (5a), (5b), and (5c) to θ_{k-1} to yield θ_{k-1}' or θ_{k-1}'' when transformed back to length k by part (4) yields the modified elements of $\mathcal{A}_k$ with the required properties since no $S_1(i, j)$ is in $\mathcal{C}_j^k$. □

Definition 5.10. Let π be a partial assignment to the variables of H_k. We view π as a set of assignments to individual variables and so write $\pi \subseteq \pi'$ iff partial assignment π' extends π. We write $\pi \parallel \pi'$ iff π and π' are consistent partial assignments. If $\pi \parallel \pi'$ are consistent partial assignments, then we write $\pi \cap \pi'$ for the partial assignment where they agree. If π is a partial assignment consistent with some total assignment in $\mathcal{A}_k$, then we define $\pi^* = \bigcap_{\theta \in \mathcal{A}_k,\ \theta \parallel \pi} \theta$, the partial assignment *forced by* π.

Note that by definition $\pi \subseteq \pi^*$.

Proposition 5.11. *Let* $k > 0$ *and* $i, j \in [n]$.

(*1*) *Suppose that* k *is odd and* π^* *sets* $S_\ell(i, j)$ *for some* $\ell \in [k]$*. If* π *does not set any variable in* $\mathcal{R}_i^k$*, then* π^* *sets* $T(j) = 1$*; if* π *does not set any variable in* $\mathcal{C}_j^k$*, then* π^* *sets* $R(i) = 1$*.*

(*2*) *Suppose that* k *is even and* π^* *sets* $S_\ell(i, j)$ *for some* ℓ *with* $2 \le \ell \le k$*. If* π *does not set any variable in* $\mathcal{R}_i^k$*, then* π^* *sets* $T(j) = 1$*; if* π *does not set any variable in* $\mathcal{C}_j^k$*, then* π^* *sets* $R(i) = 0$ *and* $S_1(i, j) = 1$*.*

Proof. (1) First, consider k odd and suppose that π does not set any variable in $\mathcal{R}_i^k$. By Definition 5.7, the fact that π^* sets $S_\ell(i, j)$ is equivalent to π^* fixing the value of $R(i) \vee T(j)$. Suppose that π^* does not set $T(j) = 1$. Then there is some assignment θ in

$\mathcal{A}_k$ consistent with π^* such that θ sets $T(j) = 0$. By Proposition 5.9(5b) there is another assignment θ' in $\mathcal{A}_k$ that agrees with θ everywhere except in the variables in $\mathcal{R}_i^k$ and flips the value of $R(i)$. Since π does not set any variable in $\mathcal{R}_i^k$ and θ is consistent with π, θ' is also consistent with π. Therefore θ' is also consistent with π^*. However, θ and θ' both set $T(j) = 0$ but differ on the value of $R(i)$, so π^* does fix the value of $R(i) \vee T(j)$, a contradiction.

By the above case together with Proposition 5.9(3), the fact that π does not set any variable in $\mathcal{C}_j^k$ implies that π^* sets $R(i) = 1$.

(2) Next consider $k > 0$ even and suppose that π does not set any variable in $\mathcal{C}_j^k$. By Definition 5.7, the fact that π^* sets $S_\ell(i, j)$ for $2 \leq \ell \leq k$ is equivalent to π^* fixing the value of $\neg R(i) \vee T(j)$. Suppose that π^* does not set $R(i) = 0$. Then there is some assignment θ in $\mathcal{A}_k$ consistent with π^* such that θ sets $R(i) = 1$. Again, as in (1), by Proposition 5.9(5a) there is another assignment θ' in $\mathcal{A}_k$ that agrees with θ everywhere except in the variables in $\mathcal{C}_j^k$ and flips the value of $T(j)$. Since π does not set any variable in $\mathcal{C}_j^k$ and θ is consistent with π, θ' is also consistent with π. Therefore, θ' is also consistent with π^*. However, θ and θ' both set $R(i) = 1$ but differ on the value of $R(i)$, so π^* does fix the value of $\neg R(i) \vee T(j)$, a contradiction. Therefore π^* must set $R(i) = 0$ and, by Definition 5.7, π^* must also set $S_1(i, j) = 1$.

The case when π does not set any variable in $\mathcal{R}_i^k$ is essentially identical to the case spelled out in (1), except that the function leading to the contradiction is $\neg R(i) \vee T(j)$ rather than $R(i) \vee T(j)$. □

Definition 5.12. Let P be path in FBDD $\hat{\mathcal{F}}$ and u be a node in P. Identify P with the partial assignment its edges define. Let P_u be the partial assignment defined by the prefix of P ending at u and X_u be the variable tested at node u.

Suppose now that P is consistent with some assignment in $\mathcal{A}_k$. We say that $X_u = b$ is *forced* in P iff P_u^* sets X_u to b; otherwise, we say that $X_u = b$ is a *free* assignment to X_u. Let core(P) be the partial assignment that is the union of all free assignments in P.

PROPOSITION 5.13. *If P is consistent with some assignment in $\mathcal{A}_k$, then* $(\text{core}(P))^* = P^*$.

Note that the definition of core(P) depends on the order that variables are tested in P. Moreover, core(P) may not even be a minimal set of free assignments along P; for example, for odd k, a path P that first sets $S_1(i, j) = 0$ and then sets $T(j) = 1$ will have both variable assignments in core(P), but the variable assignment $T(j) = 1$ forces $S_1(i, j) = 0$.

Definition 5.14. Suppose that $\mathcal{F}$ computes H_k and follows the unit rule, and let $\hat{\mathcal{F}}$ be its extension as defined above (following Proposition 5.6). A path P in $\hat{\mathcal{F}}$ is called *admissible* iff it ends at a node of $\mathcal{F}$ that is not a unit node and is consistent with some assignment in $\mathcal{A}_k$ and $|\text{core}(P)| \leq n - 1$. Thus, in particular, every admissible path is consistent with some admissible total assignment.

Lemma 5.4 follows from two lemmas, which we state here and prove in the rest of the section.

LEMMA 5.15. *Let P be a admissible path in $\hat{\mathcal{F}}$. If $|\text{core}(P)| < n - 1$, then there are two distinct admissible paths P' and P'' extending P such that $|\text{core}(P')| = |\text{core}(P'')| = |\text{core}(P)| + 1$.*

Since the empty path from the root of $\hat{\mathcal{F}}$ to itself is admissible, the following Corollary is immediate by induction.

COROLLARY 5.16. *There are at least $2^n - 1$ admissible paths in $\hat{\mathcal{F}}$.*

The second lemma is as follows:

LEMMA 5.17. *If P_0, P_1 are admissible paths in $\hat{\mathcal{F}}$ ending at some node v (of $\mathcal{F}$), then $P_0 = P_1$.*

Corollary 5.16 and Lemma 5.17 immediately imply Lemma 5.4.

We now prove Lemmas 5.15 and 5.17. To prove Lemma 5.15 we first show the following two lemmas:

LEMMA 5.18. *Let P be an admissible path in FBDD $\hat{\mathcal{F}}$ that computes H_k. Then there is some $i_0, j_0 \in [n]$ such that no variable in $\mathcal{R}^k_{i_0}$ or $\mathcal{C}^k_{j_0}$ is set by* core(P). *For any such i_0, j_0, P^* does not set any variables in $\mathcal{T}^k_{i_0,j_0}$. In particular, $H_k[P^*]$ is not a constant function.*

PROOF. Since $|\text{core}(P)| \leq n-1$, there must be some pair (i_0, j_0) such that core(P) does not set any variable in $\mathcal{R}^k_{i_0} \cup \mathcal{C}^k_{j_0}$. Let (i_0, j_0) be any such pair. By Proposition 5.9(5c), for any $X \in \mathcal{T}^k_{i_0 j_0}$ there are two assignments θ and θ' in $\mathcal{A}_k$ that agree with core(P) but have different values for X so (core(P))* does not set any variable in $\mathcal{T}^k_{i_0 j_0}$. Hence by Proposition 5.13, P^* does not set any variable in $\mathcal{T}^k_{i_0 j_0}$. We can extend P^* by setting variables in $\mathcal{T}^k_{i_0 j_0}$ to 1 to make H_k evaluate to 1. On the other hand, since P^* is consistent with an assignment in $\mathcal{A}_k$, by Proposition 5.9, H_k also evaluates to 0 on some extension of P^* and hence $H_k[P^*]$ is not constant. □

LEMMA 5.19. *Suppose that $\hat{\mathcal{F}}$ computes H_k. Let P be an admissible path in $\hat{\mathcal{F}}$. All of the assignments of* core(P) *are set at nodes of $\mathcal{F}$ and none of the variable assignments in* core(P) *involves the unit rule.*

PROOF. All unit rule assignments are forced since H_k evaluates to 0 on all elements of $\mathcal{A}_k$. There is nothing else to prove for $k = 0$ since in that case $\hat{\mathcal{F}} = \mathcal{F}$. For the other cases, we need to show that every assignment in P at a dummy node of $\hat{\mathcal{F}}$ is forced.

Case k is odd. In this case, the dummy node must test either $R(i)$ or $T(j)$. First, suppose that variable $R(i)$ is tested at a dummy node (u, v, ℓ) of $\hat{\mathcal{F}}$. By definition, the prefix $P_1 = P_{(u,v,1)}$ of P is admissible. Also, by definition, $H_k[P_1]$ does not depend on $R(i)$. Since P_1 is admissible, by Lemma 5.18 there are $i_0, j_0 \in [n]$ such that core(P_1) does not set any variable in $\mathcal{C}^k_{j_0}$ and P_1^* does not set any variable in $\mathcal{T}^k_{i_0 j_0}$; in particular, P_1^* does not set $T(j_0)$. Now since P_1 does not set $R(i)$ but $H_k[P_1]$ does not depend on $R(i)$, to avoid having the prime implicant $R(i)S_1(i, j_0)$, P_1 must set $S_1(i, j_0) = 0$, and hence (core(P_1))* sets $S_1(i, j_0) = 0$. Therefore, by Proposition 5.11(1) applied to core(P_1), we obtain that P_1^* sets $R(i) = 1$, and hence the value of $R(i)$ is forced in P, as required. By Proposition 5.9(3), the case when the variable tested at the dummy node is $T(j)$ is completely dual and $T(j) = 1$ is also forced.

Case $k > 0$ is even. Now there are three possibilities for the variable tested at the dummy node: some $R(i)$, some $T(j)$, or some $S_1(i, J)$. Suppose that $R(i)$ is tested at a dummy node (u, v, ℓ) and let $P_1 = P_{(u,v,1)}$. For any fixed $j \in [n]$, if P_1 sets $S_1(i, j)$ then P_1^* sets $R(i) = \neg S_1(i, j)$ and hence the value of $R(i)$ is forced in P; on the other hand, if P_1 does not set $S_1(i, j)$, then $R(i)S_1(i, j)$ is a 2-prime implicant of $H_k[P_1]$, which contradicts the fact that $H_k[P_1]$ does not depend on $R(i)$ since $R(i)$ labels dummy node (u, v, ℓ).

Similarly, if $S_1(i, j)$ is tested at dummy node (u, v, ℓ), then if P_1 sets $R(i)$, then P_1^* sets $S_1(i, j) = \neg R(i)$; otherwise, $R(i)S_1(i, j)$ is a 2-prime implicant of $H_k[P_1]$, which contradicts the fact that $H_k[P_1]$ does not depend on $S_1(i, j)$ since it is tested at (u, v, ℓ). Again, $S_1(i, j)$ is forced on P.

Finally, suppose that the label of dummy node (u, v, ℓ) is $T(j)$. Since P_1 is admissible, by Lemma 5.18 there are $i_0, j_0 \in [n]$ such that core(P_1) does not set any variable in

$\mathcal{R}^k_{i_0}$ and P_1^* does not set any variable in $\mathcal{T}^k_{i_0 j_0}$; in particular, P_1^* does not set $R(i_0)$. Now since $(\text{core}(P_1))^*$ does not set $T(j)$ and $H_k[P_1]$ does not depend on $T(j)$, P_1 must set $S_k(i, j) = 0$, and hence $(\text{core}(P_1))^*$ must set $S_k(i, j) = 0$. Applying Proposition 5.11(2) to $\text{core}(P_1)$, we obtain that P_1^* sets $T(j) = 1$ and thus the value of $T(j)$ is forced in P. □

These two lemmas immediately let us prove Lemma 5.15.

Proof of Lemma 5.15. Let P be an admissible path in $\hat{\mathcal{F}}$ and let v be the node of $\mathcal{F}$ at which $\hat{\mathcal{F}}$ ends. By Lemma 5.18, v is not an output node and hence it tests a variable X'. If the value of X' is set in P^*, then we extend P following that value, and the value H_k is not constant so the path can be continued. By Lemma 5.19, we can repeat following forced values until a a node w of $\mathcal{F}$ is reached that is not a unit node and does not test a variable whose value is set in P^*. The resulting path to w will be admissible by definition and will have the same core as $\text{core}(P)$. By Lemma 5.18, w cannot be a leaf node of $\mathcal{F}$ and so must test a variable X that is not set in P^*. The admissible path P' will extend the path from w using edge 0, and P'' will extend it using edge 1. Both are consistent with assignments in $\mathcal{A}_k$ by definition. To make P' and P'' admissible, we follow P' and P'' through any forced values at dummy nodes or unit nodes of $\hat{\mathcal{F}}$ until they next reach a non-unit node of $\mathcal{F}$. Lemma 5.19 ensures that such a node will be reached. Observe that $\text{core}(P') = \text{core}(P) \cup \{X = 0\}$ and $\text{core}(P'') = \text{core}(P) \cup \{X = 1\}$ so the core sizes are precisely 1 more than $|\text{core}(P)|$ and hence at most $n-1$ as required. □

In the rest of this section, we prove Lemma 5.17.

Proof of Lemma 5.17. Since admissible path P_0 and P_1 in $\hat{\mathcal{F}}$ end at the same node v of $\mathcal{F}$ computing H_k, they have $H_k[P_0] = H_k[P_1]$. Assume that $P_0 \neq P_1$. Then P_0 and P_1 must diverge at some node of $\hat{\mathcal{F}}$, so there is some variable X set by both of them on which P_0 and P_1 differ.

Case $k = 0$. There are three sub-cases: $X = R(i)$, $X = S(i, j)$, or $X = T(j)$ for some $i, j \in [n]$.

Suppose, first, that P_0 sets $R(i) = 0$ and P_1^* sets $R(i) = 1$. (Note that this is weaker than assuming that P_1 sets $R(i) = 1$ but we will find the generality useful.) Since P_1 is admissible, by Lemma 5.18 there is some $i_0, j_0 \in [n]$ such that $\text{core}(P_1)$ does not set any variable in $\mathcal{R}^k_{i_0}$ or $\mathcal{C}^k_{j_0}$, and P_1^* does not set $T(j_0)$ (which is in $\mathcal{T}^k_{i_0 j_0}$). Now if $S(i, j_0)$ is not set to 0 by P_1, then $H_k[P_1]$, which cannot have any 1-prime implicants, would have a prime implicant, either $R(i)S(i, j_0)T(j_0)$ or $S(i, j_0)T(j_0)$, that is not a prime implicant of $H_k[P_0]$, which is a contradiction. Hence P_1 sets $S(i, j_0) = 0$. But then P_1^* sets $R(i) = 1$ and $S(i, j_0) = 0$, which says that P_1^* sets $T(j_0) = 1$ by admissibility, which is a contradiction to $T(j_0)$ being unset by P_1^*. The subcase when P_0 sets $T(j) = 0$ and P_1 sets $T(j) = 1$ is completely dual to the case for R using Proposition 5.9 (3) and $R(i_0)$ instead of $T(j_0)$.

Finally, suppose that P_0 sets $S(i, j) = 0$ and P_1 sets $S(i, j) = 1$. Therefore, $R(i)T(j)$ is not a 2-prime implicant of $H_k[P_0]$. Also, by the definition of $\mathcal{A}_0$, since P_0 is admissible, P_0^* sets $R(i) = 1$ and $T(j) = 1$. For P_1, since $H_k[P_0] = H_k[P_1]$, we must have P_1 set $R(i) = 0$ or $T(j) = 0$, because otherwise $H_k[P_1]$ has a 2-prime implicant $R(i)T(j)$. Since both $R(i)$ and $T(j)$ are set to 1 by P_0^*, we obtain a contradiction using the previous subcases: if P_1 sets $R(i) = 0$, then we use the first subcase; and if P_1 sets $T(j) = 0$, then we use the second.

Case k is odd. We now consider the case that k is odd: First, suppose that P_0 sets $R(i) = 0$ and P_1 sets $R(i) = 1$. Since P_1 is admissible, by Lemma 5.18 there is some $i_0, j_0 \in [n]$ such that $\text{core}(P_0)$ does not set any variable in $\mathcal{C}^k_{j_0}$ and P_0^* does not set $T(j_0)$ (which is in $\mathcal{T}^k_{i_0 j_0}$). Note that by Proposition 5.11(1) applied to $\text{core}(P_0)$, P_0 also cannot set

any of $S_1(i, j_0), \ldots, S_k(i, j_0)$ since it sets $R(i) = 0$ and core(P_0) does not set any element in $\mathcal{C}^k_{j_0}$. Also, since P_1 is admissible and P_1 sets $R(i) = 1$, P_1 must set $S_1(i, j_0) = 0$ by the unit rule. Now if $k = 1$, then this means that $S_1(i, j_0)T(j_0)$ is a 2-prime implicant of $H_k[P_0]$ but this would be a contradiction since no term containing $S_1(i, j_0)$ can be a 2-prime implicant of $H_k[P_1]$; if $k \geq 3$, then $S_1(i, j_0)S_2(i, j_0)$ is a 2-prime implicant of $H_k[P_1]$, again yielding a contradiction.

Again, using Proposition 5.9 (3), the subcase when P_0^* sets $T(j) = 0$ and P_1 sets $T(j) = 1$ is completely dual to the case for $R(i)$.

Finally, suppose that P_0 sets $S_\ell(i, j) = 0$ and P_1 sets $S_\ell(i, j) = 1$ for some $\ell \in [k]$. We now use the fact that both paths end at a decision node, and so no units remain in $H_k[P_0] = H_k[P_1]$. Observe that since P_1 sets $S_\ell(i, j) = 1$, by the unit rule P_1 also sets $S_{\ell-1}(i, j) = 0$ if $\ell > 1$ and $S_{\ell+1}(i, j) = 0$ if $\ell < k$. If $\ell = 1$, then P_1 must set $R(i) = 0$ and if $\ell = k$, then P_1 must set $T(j) = 0$. Therefore there is no 2-prime implicant containing these variables in $H_k[P_1]$. On the other hand, P_0^* sets the corresponding variables to 1. Since $H_k[P_0] = H_k[P_1]$ cannot be identically 1, and cannot have any 2-prime implicants containing $S_{\ell-1}(i, j)$ or $S_{\ell+1}(i, j)$, then P_0 must set $S_{\ell-2}(i, j) = 0$ if $\ell > 2$ and $S_{\ell+2}(i, j) = 0$ if $\ell < k - 1$. If $\ell = 2$ or $\ell = k - 1$, then P_0 must set $R(i) = 0$ or $T(j) = 0$, respectively. Applying this argument inductively, we ensure that

—$P_{\ell \bmod 2}$ sets $R(i) = 0, S_2(i, j) = 0, \ldots, S_{k-1}(i, j) = 0, T(j) = 0$ and
—$P_{(\ell+1) \bmod 2}$ sets $S_1(i, j) = 0, S_3(i, j) = 0, \ldots, S_k(i, j) = 0$.

But since both P_0 and P_1 arrive at the same node of $\hat{\mathcal{F}}$, $P_{(\ell+1) \bmod 2}$ must also set $R(i)$ and $T(j)$, and since it is not consistent with the same assignment to $S_1(i, j), \ldots, S_k(i, j)$ given by $P_{\ell \bmod 2}$, it must set at least one of $R(i)$ or $T(j)$ to 1, yielding a contradiction based on one of the previous two subcases.

Case $k > 0$ is even. This time we also need a fourth subcase for when P_0 sets $S_1(i, j) = 0$ and P_1 sets $S_1(i, j) = 1$. By the correspondence with the case of $k - 1$ on $R', S'_1, \ldots, S'_{k-1}, T'$ given by Proposition 5.9 (4), we obtain a contradiction using the reasoning for $k - 1$ with paths P'_0 setting $R'(i) = 0$ and P'_1 setting $R'(i) = 1$.

If instead P_0 sets $R(i) = 1$ and P_1 sets $R(i) = 0$, then the unit rule ensures that P_0 sets $S_1(i, j) = 0$, which in turn means that P_1 sets $S_1(i, j)$ by the properties of $\hat{\mathcal{F}}$, and this must be to 1 since P_1 is admissible. We now can apply the previous case.

If P_0 sets $T(j) = 0$ and P_1 sets $T(j) = 1$, then we could use some of the symmetry of R', S', T' and some extra work, but instead we argue directly: Since P_0 is admissible, by Lemma 5.18 there is some $i_0, j_0 \in [n]$ such that core(P_0) does not set any element of $R^k_{i_0}$ and $P_0^* = (\text{core}(P_0))^*$ does not set any element of $\mathcal{T}^k_{i_0 j_0}$; in particular, it does not set $R(i_0)$. Hence it also does not set $S_1(i_0, j)$ since any admissible assignment that sets one of $R(i_0)$, $S_1(i_0, j)$ must set the other to the opposite value. By Proposition 5.11(2) applied to core(P_0), P_0 cannot set any of $S_2(i_0, j), \ldots, S_k(i_0, j)$ since it sets $T(j) = 0$ and core(P_0) does not set any element of $\mathcal{R}^k_{i_0}$. By the unit rule P_1 must set $S_k(i_0, j) = 0$. But then $S_{k-1}(i_0, j)S_k(i_0, j)$ would not be a 2-prime implicant of $H_k[P_1]$ but would be a 2-prime implicant of $H_k[P_0]$, a contradiction.

Finally, suppose that P_0 sets $S_\ell(i, j) = 0$ and P_1 sets $S_\ell(i, j) = 1$ for some ℓ with $2 \leq \ell \leq k$. We now use the fact that no units remain in $H_k[P_0] = H_k[P_1]$. Observe that since P_1 sets $S_\ell(i, j) = 1$, by the unit rule P_1 also sets $S_{\ell-1}(i, j) = 0$ and $S_{\ell+1}(i, j) = 0$ if $\ell < k$. If $\ell = k$, then P_1 must set $T(j) = 0$. Therefore there is no 2-prime implicant containing these variables in $H_k[P_1]$. On the other hand, if $2 < \ell < k$, P_0^* sets the corresponding variables to 1. Since $H_k[P_0] = H_k[P_1]$ cannot be identically 1, and cannot have any 2-prime implicants containing $S_{\ell-1}(i, j)$ or $S_{\ell+1}(i, j)$, then P_0 must set $S_{\ell-2}(i, j) = 0$ if $\ell > 3$ and $S_{\ell+2}(i, j) = 0$ if $\ell < k - 1$. If $\ell = 3$ or $\ell = k - 1$, then P_0 must set $S_1(i, j) = 0$ or $T(j) = 0$, respectively. Applying this argument inductively, we ensure that

—$P_{\ell \bmod 2}$ sets $S_2(i,j) = 0, \ldots, S_k(i,j) = 0$ and
—$P_{(\ell+1) \bmod 2}$ sets $S_1(i,j) = 0, S_3(i,j) = 0, \ldots, S_{k-1}(i,j) = 0, T(j) = 0$.

But since both P_0 and P_1 arrive at the same node of $\hat{\mathcal{F}}$, $P_{\ell \bmod 2}$ must also set $S_1(i,j)$ and $T(j)$, and since it is not consistent with the same assignment to $S_2(i,j), \ldots, S_k(i,j)$ given by $P_{(\ell+1) \bmod 2}$, it must set at least one of $S_1(i,j)$ or $T(j)$ to 1, yielding a contradiction based on one of the previous subcases. □

5.2. Proof of Lemma 5.2

We begin with a simple property of monotone functions. For a formula Φ, let $U(\Phi)$ denote the set of units in Φ. The following proposition will be useful because it implies that for monotone formulas, setting units to 0 cannot create additional units.

PROPOSITION 5.20. *If Φ is a monotone function and W is a variable in Φ, then $U(\Phi[W = 0]) \subseteq U(\Phi)$.*

Let $\mathcal{F} = (V, E)$ be an FBDD for a monotone formula Φ, where V and E, respectively, denote the nodes and edges of $\mathcal{F}$. For every edge $e = (u, v) \in E$, define $U(e) = U(\Phi_v) - U(\Phi_u)$. Observe that, by Proposition 5.20, any edge e for which $U(e)$ is non-empty must be labeled 1 in $\mathcal{F}$.

Fix some canonical ordering π on the variables of Φ. Define the following transformation on $\mathcal{F}$ to produce an FBDD $\mathcal{F}'$ for Φ that follows the unit rule: The set of nodes V' of $\mathcal{F}'$ is given by:

$$V' = V \cup \{(e,i) \mid e = (u,v) \in E, u \in V,\ 1 \le i \le |U(e)|\}.$$

The other details of $\mathcal{F}'$ are given as follows:

—For $e = (u, v) \in E$, the new vertices $(e, 1), \ldots, (e, |U(e)|)$ will appear in sequence on a path from u to v that replaces the edge e. (If $U(e)$ is empty, then the original edge e remains.)
—Edge $(u, (e, 1))$ in $\mathcal{F}'$ will have label 1, which is the label that e has in $\mathcal{F}$.
—The variable labeling each new vertex (e, i) in V' will be the ith element of $U(e)$ under the ordering π; we denote this variable by $Z_{e,i}$.
—The 1-edge out of each new vertex (e, i) will lead to the 1-sink. The 0-edge will lead to the next vertex on the newly created path.
—For a vertex $w \in V$ labeled by a variable W, if W appears in $U(e)$ for any edge $e = (u, v)$ such that there is a path in $\mathcal{F}$ from v to w, then the node w becomes a no-op node in $\mathcal{F}'$, namely its labeling variable W is removed, its 1-outedge is removed, and its 0-outedge is retained with no label. Otherwise, w keeps the variable label W as in $\mathcal{F}$ and its outedges remain the same in $\mathcal{F}'$.

The size bound required for Lemma 5.2 is immediate by construction since the degree of a variable upper-bounds the number of new units that setting it can create. However, in order for this construction to be well defined we need to ensure that the conversion to no-op nodes does not conflict with the conversion of edges to paths of units.

PROPOSITION 5.21. *If the variable W labeling w is in $U(e)$ for some edge $e = (u, v)$ for which there is a path from v to w, then the outedges e' of w have $U(e') = \emptyset$.*

PROOF. The assumption implies that W is a unit of some Φ_v. Therefore $\Phi_v = W \vee \Phi'_v$ for some Φ'_v. Since $\mathcal{F}$ is an FBDD and W labels w, W is not set on the path from v to w, and hence $\Phi_w = W \vee \Phi''$ for some formula Φ''. A 0-outedge e_0 from w always has $U(e_0) = \emptyset$ and the 1-outedge $e_1 = (w, w')$ of w sets W to 1, and hence $\Phi_{w'} = 1$, which implies that $U(e_1)$ is also empty. □

The following simple proposition is useful in reasoning about the correctness of our construction.

PROPOSITION 5.22. *If there is a path from u to v in $\mathcal{F}$ and $X \in U(\Phi_u)$, then $X \in U(\Phi_v)$, $\Phi_v = 1$, or X is tested on the path from u to v, and hence Φ_v does not depend on X.*

PROOF. $X \in U(\Phi_u)$ implies that $\Phi_u = X \vee F$ for some monotone formula F. If X is set on the path from u to v, then Φ_v does not depend on X; otherwise $\Phi_v = X \vee F'$ for some monotone formula F' and either X is a prime implicant or F' is the constant 1 and hence $\Phi_v = 1$. □

Taken together with the size bound for our construction, the following lemma immediately implies Lemma 5.2.

LEMMA 5.23. *Let Φ be monotone and computed by FBDD $\mathcal{F}$. Then $\mathcal{F}'$ is an FBDD for Φ that follows the unit rule.*

PROOF. We first show that $\mathcal{F}'$ is an FBDD, namely, every root-leaf path P in $\mathcal{F}'$ tests each variable at most once. P contains old nodes $u \in V$ and new nodes (e, i). Suppose that a variable X is tested twice along a path. Clearly, the two tests cannot be done by old nodes since $\mathcal{F}$ is an FBDD. It cannot be tested by an old node u and later by a new node (e, i), because, once tested by u, for any descendant node v, the formula Φ_v no longer depends on X, and hence $X \notin U(e)$. It cannot be first tested by a new node (e, i) and then later by an old node u since the test at the old node would have been removed and converted to a no-op by the last item in the construction of $\mathcal{F}'$. Finally, suppose that the two tests are done by two new nodes (e_1, i), and (e_2, j) on P, where we write $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$. Then we must have $X \in U(v_1)$ and $X \notin U(u_2)$ where there is a path from v_1 to u_2 in $\mathcal{F}$. By Proposition 5.22, this implies that Φ_{u_2} does not depend on X, which contradicts the requirement that $X \in U(v_2)$ since v_2 is a child of u_2.

By construction, $\mathcal{F}'$ obviously follows the unit rule. It remains to prove that $\mathcal{F}'$ computes Φ. We show something slightly stronger: For any function F, define F^- to be $F[U(F) = 0]$, in which all variables in $U(F)$ are set to 0. We claim by induction that for all nodes of $v \in V$, if θ' labels a path in $\mathcal{F}'$ from the root to v, then $\Psi[\theta'] = \Phi_v^-$, and $\theta' = \theta \cup \{U(\Phi_v) = 0\}$ for some θ that labels a path in $\mathcal{F}$ from the root to v. This trivially is true for the root. If it is true for the output nodes, then $\mathcal{F}'$ correctly computes Ψ since constant functions have no units. Let $v \in V$ and suppose that this is true for all vertices u such that there is some path θ' from the root to v in $\mathcal{F}'$ for which u is the last vertex in V on θ'. By the construction, for each such u, there must be an edge $e = (u, v) \in E$. Suppose that the variable tested at u in $\mathcal{F}$ is W. We have three cases: If $e = (u, v) \in E$ is a 1-edge, then $\Phi_v = \Phi_u[W = 1]$. Every path θ' from the root to v through u is of the form $\theta' = \theta \cup \{W = 1\} \cup \{U(e) = 0\}$ for some θ that labels a path from the root to u in $\mathcal{F}'$. (This is true even if $U(e)$ is empty.) By induction, $\Psi[\theta] = \Phi_u^- = \Phi_u[U(\Phi_u) = 0]$ and, by definition, $U(\Phi_v) = U(\Phi_u) \cup U(e)$ so $\Psi[\theta'] = \Phi_u[W = 1 \cup \{U(\Phi_v) = 0\}] = \Phi_v[U(\Phi_v) = 0]$, as required. If $e = (u, v) \in E$ is a 0-edge of $\mathcal{F}$, then $\Phi_v = \Phi_u[W = 0]$. If u became a no-op vertex in $\mathcal{F}'$, then there was some ancestor w of u at which W became a unit of Φ_w. Since $\mathcal{F}$ is an FBDD, it does not test W between that ancestor and u. By Proposition 5.22, either $W \in U(\Phi_u)$ or $\Phi_u = 1$. In the latter subcase, $\Phi_v = \Phi_u = 1$, and the correctness for u implies that for v. In the former case, $U(\Phi_u) = U(\Phi_v) \cup \{W\}$ and $\Phi_u^- = \Phi_v^-$, and, again, the correctness for Φ_u^- implies that for Φ_v^-. In the case that u does not become a no-op vertex, W is not a unit of Φ_u so $U(\Phi_u) = U(\Phi_v)$, and the fact that all paths to u yield $\Phi_u[U(\Phi_u) = 0] = \Phi_u[U(\Phi_v) = 0]$ for all paths to v that pass through u as the previous vertex in V, The last edge to v adds the extra $W = 0$ constraint. Adding this constraint to $\Phi_u[U(\Phi_v) = 0]$ yields $\Phi_v[U(\Phi_v) = 0] = \Phi_v^-$, as required. Therefore the statements holds for all possible paths from the root to v. □

6. LOWER BOUNDS FOR BOOLEAN COMBINATIONS OVER H_K

In this section, we prove Theorem 3.9. Throughout this section we fix $f(\mathbf{X}) = f(X_0, \ldots, X_k)$, a Boolean function that depends on all variables $\mathbf{X}$, and a domain size $n > 0$. To prove Theorem 3.9, we first prove that any FBDD for the lineage of the query $Q = f(h_{k0}, \ldots, h_{kk})$ can be converted into a multi-output FBDD for all of $\mathbf{H_k} = (H_{k0}, H_{k1}, \ldots, H_{kk})$ with at most an $O(k2^k n^3)$ increase in size. The proof is constructive. Theorem 3.9 then follows immediately using Theorem 3.6 since any multi-output FBDD for $\mathbf{H_k}$ yields an FBDD for H_k of the same size.

Recall that $H_{k\ell}$ denotes the lineage of $h_{k\ell}$ and let $\Psi = f(\mathbf{H_k}) = f(H_{k0}, \ldots, H_{kk})$ be the lineage of Q.

If $\mathcal{F}$ is an FBDD for $\Psi = f(\mathbf{H_k})$, then let Φ_u denote the Boolean function computed at the node u; thus $\Psi = \Phi_r$, where r is the root node of $\mathcal{F}$. By the correctness of $\mathcal{F}$, all paths P leading to u have the property that $\Psi[P] = \Phi_u$.

In order to produce the multi-output FBDD $\mathcal{F}'$ for $\mathbf{H_k}$ from $\mathcal{F}$ computing $\Psi = f(\mathbf{H_k})$, we would like to ensure that every internal node v of $\mathcal{F}'$ has the property that all paths P leading to v not only are consistent with the same residual function $\Phi_v = \Psi[P]$, but they also all agree on the residual values of $H_{k\ell}(v) = H_{k\ell}[P]$ for all ℓ. Since we are not easily able to characterize its paths, we find it convenient to define this property not only with respect to paths of $\mathcal{F}'$ but also for formulas Φ_v with respect to arbitrary partial assignments θ. We use the term *transparent* to describe the property that the value of Φ_v automatically also reveals the values for all $H_{k\ell}(v)$. Call a formula Φ a *restriction* of Ψ if $\Phi = \Psi[\theta]$ for some partial assignment θ.

Definition 6.1. Fix $\Psi = f(H_{k0}, \ldots, H_{kk})$. A formula Φ that is a restriction of Ψ is called *transparent* if there exist $k+1$ formulas $\varphi_0, \ldots, \varphi_k$ such that, for every partial assignment θ, if $\Phi = \Psi[\theta]$, then $H_{k0}[\theta] = \varphi_0, \ldots, H_{kk}[\theta] = \varphi_k$. We say that Φ *defines* $\varphi_0, \ldots, \varphi_k$.

In other words, assuming that Φ is a restriction of Ψ, $\Phi = \Psi[\theta]$ for some partial assignment θ, then Φ is transparent if the formulas $H_{k0}[\theta], \ldots, H_{kk}[\theta]$ are uniquely defined, that is, are independent of θ. Equivalently, for any two assignments θ, θ', if $\Psi[\theta] = \Psi[\theta'] = \Phi$, then for all $0 \le \ell \le k$, $H_{k\ell}[\theta] = H_{k\ell}[\theta']$.

Example 6.2. Let $k = 3$ and $f = X_0 \vee X_1 \vee X_2 \vee X_3$. Given a domain size $n > 0$, the formula Ψ is as follows:

$$\Psi = \bigvee_{i,j} R(i)S_1(i,j) \vee \bigvee_{i,j} S_1(i,j)S_2(i,j) \bigvee_{i,j} S_2(i,j)S_3(i,j) \vee \bigvee_{i,j} S_3(i,j)T(j).$$

$H_{30}, \ldots, H_{33}$ denote each of the four disjunctions above. Let $\Phi = R(3)S_1(3,7) \vee S_1(3,7)S_2(3,7)$. There are many partial substitutions θ for which $\Phi = \Psi[\theta]$: For example, θ may set to 0 all variables with index $\neq (3,7)$, and also set $S_3(3,7) = T(7) = 0$, or it could set $S_3(3,7) = 0$, $T(7) = 1$. There are many more choices for variables with index $\neq (3,7)$. However, one can check that, for any θ such that $\Phi = \Psi[\theta]$, we have the following:

$$H_{30}[\theta] = R(3)S_1(3,7) \qquad H_{31}[\theta] = S_1(3,7)S_2(3,7)$$
$$H_{32}[\theta] = 0 \qquad H_{33}[\theta] = 0.$$

Therefore, Φ is *transparent*. On the other hand, consider $\Phi' = S_1(3,7)$. This formula is no longer transparent, because it can be obtained by extending any θ that produces Φ with $R(3) = 0$, $S_2(3,7) = 1$, $R(3) = 1$, $S_2(3,7) = 0$, or $R(3) = S_2(3,7) = 1$, and these

lead to different residual formulas for H_{30} and H_{31} (namely 0 and $S_1(3,7)$, $S_1(3,7)$ and 0, or $S_1(3,7)$ and $S_1(3,7)$).

In order to convert an FBDD $\mathcal{F}$ for $\Psi = f(\mathbf{H_k})$ into a multi-output FBDD for $\mathbf{H_k} = (H_{k0}, \ldots, H_{kk})$, we will try to modify it so the formulas defined by the restrictions reaching its nodes become transparent without much of an increase in the FBDD size. To do this, we will add new intermediate nodes at which the formulas may not be transparent but we will be able to reason about its computations based on the nodes where the formulas are transparent.

Observe that if we know that $\Phi_v = \Psi[\theta]$ is transparent and we have a small multi-output FBDD $\mathcal{F}_\theta$ for $\mathbf{H_k}[\theta]$, then we can simply append that small FBDD at node v to finish the job and ignore what the original FBDD did below v. Intuitively, the reason that $\mathbf{H_k}$ and H_k might not have such small FBDDs is the tension between the $R(i)S_1(i,j)$ terms, which gives a preference for reading entries in row-major order and the $S_k(i,j)T(j)$ terms, which suggest column-major order, together with the intermediate $S_\ell(i,j)S_{\ell+1}(i,j)$ terms that link these two conflicting preferences. If all of those links are broken, then it turns out that there is no conflict in the variable order and the difficulty disappears. This motivates the following definition, which we will use to make this intuitive idea precise.

Definition 6.3. Let θ be a partial assignment to $Var(\mathbf{H_k})$.

—A *transversal* in θ is a pair of indices (i,j) such that $R(i)S_1(i,j)$ is a prime implicant of $H_{k0}[\theta]$, $S_k(i,j)T(j)$ is a prime implicant of $H_{kk}[\theta]$, and $S_\ell(i,j)S_{\ell+1}(i,j)$ is a prime implicant of $H_{k\ell}[\theta]$ for all $\ell \in [k-1]$.
—Call two pairs of indices (or transversals) $(i_1,j_1),(i_2,j_2)$ *independent* if $i_1 \neq i_2$ and $j_1 \neq j_2$.
—A Boolean formula is called *transversal-free* if there *exists* a θ such that $\Phi = \Psi[\theta]$ and θ has no transversals.

We now see that assignments without transversals, or even those with few independent transversals, yield small FBDDs.

LEMMA 6.4. *Let θ be a partial assignment to $Var(\mathbf{H_k})$. If θ has at most t independent transversals, then there exists a multi-output FBDD for $(H_{k0}[\theta], \ldots, H_{kk}[\theta])$ of size $O(k2^{k+t}n^2)$.*

PROOF. We first show that if $t = 0$ (θ has no transversals), then there exists a small OBDD that computes $\mathbf{H_k}[\theta]$.

Let G_θ be the following undirected graph. The nodes are the variables $Var(\mathbf{H_k})$, and the edges are pairs of variables (Z, Z') such that ZZ' is a 2-prime implicant in $H_{k\ell}[\theta]$ for some ℓ. Since θ has no transversals, all nodes $R(i)$ are disconnected from all nodes $T(j)$. In particular, there exists a partition $Var(\mathbf{H_k}) = \mathbf{Z}' \cup \mathbf{Z}''$ such that all $R(i)$'s are in $\mathbf{Z}'$, all $T(j)$'s are in $\mathbf{Z}''$, and every $H_{k\ell}[\theta]$ can written as $\varphi'_\ell \vee \varphi''_\ell$, where $Var(\varphi'_\ell) \subset \mathbf{Z}'$ and $Var(\varphi''_\ell) \subset \mathbf{Z}''$; in particular, $\varphi''_0 = \varphi'_k = 0$.

Define *row-major order* of the variables in the set $Var(\mathbf{H_k}) - \{T(1), \ldots, T(n)\}$ by

$$R(1), S_1(1,1), \ldots, S_k(1,1), S_1(1,2), \ldots, S_k(1,n),$$
$$R(2), S_1(2,1), \ldots, S_k(2,1), S_1(2,2), \ldots, S_k(2,n),$$
$$\ldots$$
$$R(n), S_1(n,1), \ldots, S_k(n,1), S_1(n,2), \ldots, S_k(n,n).$$

Let π' be the restriction of the row-major order to the variables in $\mathbf{Z}'$. Similarly, let π'' be the restriction to $\mathbf{Z}''$ of the corresponding column-major order of the variables

that omits the $R(i)$'s and places the $T(j)$ before all variables $S_\ell(i,j)$. We build a multi-output OBDD using the order $\pi = (\pi', \pi'')$ for $(H_{k0}, \ldots, H_{kk})$. In the first part using order π', it will compute each φ'_ℓ term in parallel in width $O(2^k)$ and in the second part it will continue by including the additional terms from φ''_ℓ using order π''. Observe that, except for the $R(i)S_1(i,j)$ terms, each of the variables in the 2-prime implicants in φ'_ℓ appear consecutively in π'. Each level of the OBDD will have at most 2^{k+3} nodes, one for each tuple consisting of a vector of values of the partially computed values for the $k+1$ functions φ'_ℓ, remembered value of $R(i)$, and remembered value of the immediately preceding queried variable. In the part using order π'', the remembered value of $T(j)$ is used instead of the remembered value of $R(i)$. The size of $\mathcal{F}'$ is $O(k2^k n^2)$ since there are $kn^2 + 2n$ variables in total in $Var(\mathbf{H_k})$.

For general t, let I and J be the sets of rows and columns, respectively, of the transversals (i,j) in θ. Since θ has at most t independent transversals, the smaller of I and J has size at most t. Suppose that this smaller set is I; the case when J is smaller is analogous. In this case, every transversal (i,j) of θ has $i \in I$. Notice that if we set all $R(i)$ variables with $i \in I$ in an assignment θ', then the assignment $\theta \cup \theta'$ has no transversals, and, thus, by the above construction, $\mathbf{H_k}[\theta']$ can be computed efficiently by a multi-output OBDD. Therefore, construct the FBDD that first exhaustively tests all possible settings of these at most t variables in a complete tree of depth t, then at each leaf node of the tree, attaches the OBDD constructed above. □

A nice property of a single transversal for θ is that its existence ensures that each $H_{k\ell}$ is a non-trivial function of its remaining inputs; more transversals will in fact ensure that less about each $H_{k\ell}$ disappears. We will see the following: If there are at least some small number of independent transversals for θ (three suffice), then we can use the fact that f depends on all inputs to ensure that $\Psi[\theta] = f(\mathbf{H_k})[\theta]$ will be transparent provided one additional condition holds: There is no variable that we can set to kill off all transversals in θ at once.

If we did not have this additional condition, then the construction of $\mathcal{F}'$ for $\mathbf{H_k}$ would be simple: We would just use Lemma 6.4 at all nodes v of $\mathcal{F}$ at which all assignments θ for which $\Phi_v = \Psi[\theta]$ do not have enough transversals to ensure transparency of Φ_v.

Failure of the additional condition is somewhat reminiscent of the situation with setting units in Section 5: This failure means that there is some variable we can set to kill off all transversals in θ at once, which by Lemma 6.4 means that along the branch corresponding to that setting one can get an easy computation of $\mathbf{H_k}$ (not quite as simple as fixing the value of the formula to 1 by setting units as in Section 5 but still easy). It is not hard to see, and implied by the proposition below, which is easy to verify, that the only way to kill off multiple independent transversals at once is to set such a variable to 1. By analogy we call such variables $\mathbf{H_k}$-*units*.

PROPOSITION 6.5. *Let $\Phi = \Psi[\theta]$ for some θ with t independent transversals and $\theta' = \theta \cup \{W = b\}$ for $b \in \{0,1\}$. The number of independent transversals in θ' is in $\{t-1, t\}$ if $b = 0$ and is in $\{0, t-1, t\}$ if $b = 1$.*

Definition 6.6. We say that a variable Z is an $\mathbf{H_k}$-*unit* for the formula Φ if $\Phi[Z=1]$ is transversal free but Φ is not. We let $U_k(\Phi)$ denote the set of $\mathbf{H_k}$-units of Φ, and we say that Φ is $\mathbf{H_k}$-*unit-free* if $U_k(\Phi) = \emptyset$.

The following lemma makes our intuitive claim precise; the proof of Lemma 6.7 appears in the appendix.

LEMMA 6.7. *Let $\Psi = f(\mathbf{H_k})$ where f depends on all its inputs. Suppose that there exists a θ with at least three independent transversals such that $\Psi[\theta] = \Phi$. If Φ is $\mathbf{H_k}$-unit-free, then Φ is transparent.*

We still need to deal with the situation when Φ has any $\mathbf{H_k}$-units along with multiple independent transversals. Our strategy is simple: Whenever we encounter an edge in $\mathcal{F}$ on which an $\mathbf{H_k}$-unit is created (possibly more than one at once) and the resulting formula has sufficiently many transversals, then, just as with the unit rule, we immediately test these $\mathbf{H_k}$-units, one at a time, each one after the previous one has been set to 0 (since the branch where it is set to 1 has an easy computation remaining).

In order to analyze this strategy properly, it is useful to understand how $\mathbf{H_k}$-units can arise. Observe, that if $\Phi = \Psi[\theta]$ and Z is a unit for some $H_{k\ell}[\theta]$, for $0 \leq \ell \leq k$, then Z is an $\mathbf{H_k}$-unit for $\Psi[\theta]$, because by setting $Z = 1$ we ensure that $H_{k\ell}[\theta \cup \{Z = 1\}] = 1$, wiping out all transversals. The following lemma shows a converse of this statement under the assumption that θ has at least four independent transversals.

LEMMA 6.8. *Let $\Psi = f(\mathbf{H_k})$ where f depends on all its inputs. If $\Phi = \Psi[\theta]$ for some partial assignment θ that has at least four independent transversals, then $U_k(\Phi) = \bigcup_{\ell \in \{0,\ldots,k\}} U(H_{k\ell}[\theta])$.*

Since a transversal (i, j) requires that all elements of $\mathbf{H_k}$ have 2-prime implicants rather than units on the terms involving (i, j), Lemma 6.8 immediately implies the following:

COROLLARY 6.9. *If $\Phi = \Psi[\theta]$ for some partial assignment θ, then no $\mathbf{H_k}$-unit of Φ is in the prime implicants indexed by any transversal of θ.*

Since the formulas in $\mathbf{H_k}$ are monotone, by Lemma 6.8 and Proposition 5.20, units are created by setting a variable to 1. Hence, if Φ has at least four independent transversals, then setting all $\mathbf{H_k}$-units in Φ to 0 in turn yields a formula that still has at least four independent transversals (by Corollary 6.9) and is $\mathbf{H_k}$-unit-free (by Lemma 6.8) and hence transparent (by Lemma 6.7 and Proposition 5.20).

We now describe the procedure for building a multi-output FBDD $\mathcal{F}'$ computing $\mathbf{H_k}$: Start with the FBDD $\mathcal{F}$ for Ψ and let V and E be, respectively, the vertices and edges of $\mathcal{F}$. Let $V_4 \subseteq V$ be the set of nodes $v \in V$ such that $\Phi_v = \Psi[\theta]$ for some assignment θ that has at least four independent transversals. By Proposition 6.5, V_4 is closed under predecessors (ancestors) in $\mathcal{F}$; let E_4 be the set of edges in $\mathcal{F}$ whose endpoints are both in V_4. The following is immediate from Proposition 6.5 and the definition of V_4.

PROPOSITION 6.10. *If $v \in V_4$ but some child of v is not in V_4, then either or both of the following hold: (i) there is an assignment θ with precisely four independent transversals such that $\Phi_v = \Psi[\theta]$ or (ii) the variable Z tested at v is in $U_k(\Phi_v)$ and the 0-child of v is in V_4.*

We apply a similar construction to that of Section 5.2 to the subgraph of $\mathcal{F}$ on V_4. For $e = (u, v)$, define $U_k(e) = U_k(\Phi_v) - U_k(\Phi_u)$ to be the set of new $\mathbf{H_k}$-units created along edge e. There are two differences from the argument in Section 5.2: (1) we only apply the construction to edges in E_4 and build the rest of $\mathcal{F}'$ independently of $\mathcal{F}$ and (2) unlike setting ordinary units to 1, in which the corresponding FBDD edges simply point to the 1-sink, each setting of an $\mathbf{H_k}$-unit to 1 only guarantees that the resulting formula is transversal free; moreover the transversal-free formulas resulting from different settings may differ. The details are as follows (see Figure 8).

—For every $e = (u, v) \in E_4$ such that $U_k(e)$ is non-empty (and, hence, the 0-child of u is also in V_4), add new vertices $(e, 1), \ldots, (e, |U_k(e)|)$ and replace e with a path from u to v having the new vertices in order as internal vertices.
—Edge $(u, (e, 1))$ in $\mathcal{F}'$ will have label 1, which is the label that e has in $\mathcal{F}$; denote the variable tested at u by W.

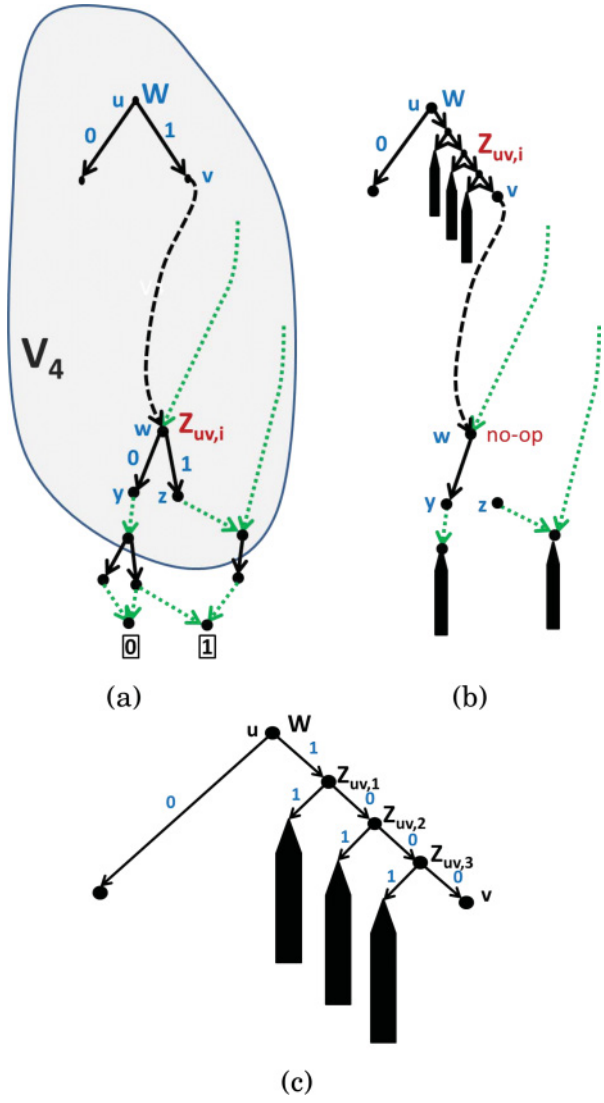


Fig. 8. Given an FBDD $\mathcal{F}$ for $\Psi = f(\mathbf{H_k})$ in (a), apply the conversion to produce $\mathcal{F}'$ for $\mathbf{H_k}$ as in (b), with detail for unit propagation in (c) in case that setting $W = 1$ produces new $\mathbf{H_k}$-units.

—The variable labeling each new vertex (e, i) will be the ith element of $U_k(e)$ under some fixed ordering of variables; we denote this variable by $Z_{e,i}$.
—The 0-edge out of each new vertex (e, i) will lead to the next vertex on the newly created path. However, unlike the simple situation with ordinary units, the 1-edge from each new vertex (e, i) will lead to a distinct new node (u, i) of $\mathcal{F}'$. Since $(u, v) \in E_4$, there is some partial assignment θ such that $\Phi_u = \Psi[\theta]$, $\Phi_v = \Psi[\theta, W = 1]$, and $\theta \cup \{W = 1\}$ has at least four transversals; for definiteness, we will pick the lexicographically first such assignment. Define the partial assignment

$$\theta(u, i) = \theta \cup \{W = 1\} \cup \{U_k(\Phi_u) = 0\} \cup \{Z_{e,1} = 0, \ldots, Z_{e,i-1} = 0, Z_{e,i} = 1\}$$

to be the assignment that sets all $\mathbf{H_k}$-units in Φ_u to 0 along with the first $i-1$ of the $\mathbf{H_k}$-units created by setting W to 1. The sub-DAG of $\mathcal{F}'$ rooted at (u,i) will be the size $O(k2^k n^2)$ FBDD for $\mathbf{H_k}[\theta(u,i)]$ constructed in Lemma 6.4.

—For any node $w \in V_4$, whose 0-child is in V_4, such that w is labeled by a variable W that was an $\mathbf{H_k}$-unit of Φ_v for some ancestor v of w, convert w to a no-op node pointing to its 0-child; that is, remove its variable label and its 1-outedge and retain its 0-outedge with its labeling removed.

—For any node $v \in V_4$ with a child that is not in V_4 and to which the previous condition did not apply, let θ be a partial assignment such that $\Phi_v = \Psi[\theta]$ and θ has precisely four independent transversals, as guaranteed by Proposition 6.10, make v the root of the size $O(k2^k n^2)$ FBDD for $\mathbf{H_k}[\theta']$ constructed in Lemma 6.4 where $\theta' = \theta \cup \{U_k(\Phi_v) = 0\}$.

—All other labeled edges of $\mathcal{F}$ between nodes of E_4 are included in $\mathcal{F}'$.

The fact that this is well-defined follows similarly to Proposition 5.21.

LEMMA 6.11. *$\mathcal{F}'$ as constructed above is a multi-output FBDD computing $\mathbf{H_k}$ that has size at most $O(k2^k n^3)$ times the size of $\mathcal{F}$.*

PROOF. We first analyze the size of $\mathcal{F}'$: As in the analysis for computing H_k, some nodes u have one added unit-setting path of length at most n and each node on the path of at the extremities of V_4 has a new added FBDD of size $O(k2^k n^2)$, yielding only $O(k2^k n^3)$ new nodes per node of $\mathcal{F}$. Also, the fact that $\mathcal{F}'$ is an FBDD follows similarly to the proof in Lemma 5.23.

If Φ_v is the function computed in $\mathcal{F}$ at node v for all $v \in V_4$, then we show by induction that for every partial assignment θ' reaching v in $\mathcal{F}'$, $\Psi[\theta'] = \Phi_v[U_k(\Phi_v) = 0]$ and $\theta' = \theta \cup \{U_k(\Phi_v) = 0\}$ for some partial assignment θ such that $\Phi_v = \Psi[\theta]$. It is trivially true of the root. The argument is similar to that for Lemma 5.23.

We now see why this is enough. Since $v \in V_4$, $\Phi_v[U_k(\Phi_v) = 0]$ is $\mathbf{H_k}$-unit free and has at least four transversals, and so it is transparent by Lemma 6.7. It remains to observe that (i) each multi-output FBDD attached directly to any node $v \in V_4$ used a restriction θ of Ψ that would lead to that node in $\mathcal{F}'$, which, because $\Psi[\theta]$ is transparent, implies that its leaves correctly compute the values of $\mathbf{H_k}$, and (ii) the same holds for the restriction leading to node (u,i) with parent (e,i), namely, the restriction used to build the multi-output FBDD consists of a restriction θ that in $\mathcal{F}'$ would reach node $u \in V_4$ and for which $\Psi[\theta]$ is transparent, together with the assignment $\{W = 1\} \cup \{Z_{e,1} = 0, \ldots, Z_{e,i-1} = 0, Z_{e,i} = 1\}$, which follows the unique path from u to (u,i). Again, this implies that its leaves correctly compute the values of $\mathbf{H_k}$. □

Now it remains to proof Lemmas 6.7 and 6.8. All formulas in $\mathbf{H_k}$ are 2-DNF formulas and, for every $(i,j) \in [n]^2$, each has a unique 2-prime implicant $P_{\ell,i,j}$ indexed by (i,j), where $P_{0,i,j} = R(i)S_1(i,j)$, $P_{k,i,j} = S_k(i,j)T(j)$, and $P_{\ell,i,j} = S_\ell(i,j)S_{\ell+1}(i,j)$ for $\ell \in [k-1]$. We say that two of their 2-prime implicants, one from $H_{k\ell}$ and one from $H_{k(\ell+1)}$, are *neighbors* if they share a variable and hence have the same index (i,j). Observe that each prime implicant in $H_{k\ell}$ has two neighbors if $\ell \in [k-1]$ and one neighbor if $\ell \in \{0,k\}$.

The key technical lemma is the following:

LEMMA 6.12. *Let $\Psi = f(\mathbf{H_k})$ for some function f that depends on all its inputs. Suppose θ is a partial assignment with two independent transversals (i_0,j_0) and (i_1,j_1). Suppose that for some (i,j) independent of of both transversals, the neighboring prime implicants of the prime implicant, $P_{\ell,i,j}$, of $H_{k\ell}$ are either unassigned or set to 0 by θ. Then there exists a partial assignment μ to all the variables of $Var(\Psi[\theta])$, except those in $P_{\ell,i,j}$, and to all variables of the transversals, (i_0,j_0) and (i_1,j_1), such that*

$\Psi[\theta \cup \mu] = f_\ell(P_{\ell,i,j}[\theta])$. *Moreover, the choice of* μ *depends on* $\Psi[\theta]$ *(as well as the indices* $\ell, i, i_0, i_1, j, j_0, j_1$*) but not on any other aspect of* θ.

The lemma still holds when we merely assume that the three pairs of indices are distinct; however, we do not allow it here since it would complicate the proof without any advantage with respect to our applications of it.

PROOF. Recall that since f depends on all its inputs, for every ℓ there exists an assignment $\mu_\ell : \mathbf{X} - \{X_\ell\} \to \{0, 1\}$, such that $f_\ell(X_\ell) = f[\mu_\ell]$ is a function that depends on X_ℓ: that is, $f_\ell(\mathbf{X}) = X_\ell$ or $f_\ell(\mathbf{X}) = \neg X_\ell$.

We define assignment μ so it sets the remaining variables to force H_{km} to equal $\mu_\ell(X_m)$ for all $m \neq \ell$ and force $H_{k\ell}$ to equal $P_{\ell,i,j}[\theta]$. In order to force some H_{km} to 1 μ may need to set two variables to 1 that may appear in neighboring prime implicants. In order to avoid incidentally forcing any of those neighboring prime implicants to 1, when forcing the H_{km} to 1, we use the variables in the two transversals alternately. We now give the formal details.

Let $Ones_\ell = \{m \mid \mu_\ell(X_m) = 1\}$ and order the elements of $Ones_\ell$ as $m_1 < m_2 < \cdots$, and define $Ones_\ell^b = \{m_r \mid b = r \mod 2\}$ for $b \in \{0, 1\}$. For $b \in \{0, 1\}$, define μ to set the variables of the (i_b, j_b) prime implicant of H_{km} to 1 for every $m \in Ones_\ell^b$. This will force $H_{km}[\theta \cup \mu] = 1$ for all $m \in Ones_\ell$. Let μ set all other variables in the transversals (i_0, j_0) and (i_1, k_1) as well as all variables of $\Psi[\theta]$, except for those in $P_{\ell,i,j}$, to 0. In particular, the alternation between how the 1's are forced in the definition of μ ensures that for $b \in \{0, 1\}$, if the (i_b, j_b) prime implicant of H_{km} is set to 1 by μ, then its neighboring prime implicants are forced to 0 by μ. In fact, $H_{km}[\theta \cup \mu] = 0$ for all $m \notin Ones_\ell \cup \{\ell\}$ since each neighboring prime implicant to $P_{\ell,i,j}[\theta]$ will have one variable set as in θ and the other set to 0 and all other prime implicants are either set to 0 by θ or by μ. Finally, the same property is true of every prime implicant of $H_{k\ell}$ except for $P_{\ell,i,j}[\theta]$. It remains to observe that $\Psi[\theta \cup \mu] = f(\mathbf{H_k}[\theta \cup \mu]) = f[\mu_\ell](P_{\ell,i,j}[\theta]) = f_\ell(P_{\ell,i,j}[\theta])$ and μ only depended on θ through the value of $\Psi[\theta]$, as required. □

Notice that the lemma fails if θ has only 1 transversal:

Example 6.13. For a counterexample, consider $f(X_0, X_1, X_2) = X_0X_2 \vee X_1$. Suppose that θ sets all variables in $\mathbf{Z}$ to 0, except for the variables with indices $(i, j) = (3, 7)$, which remain unset:

$$R(3), S_1(3, 7), S_2(3, 7), T(7).$$

Thus, θ, has the transversal $(3, 7)$. However, $\Psi[\theta]$ is

$$\begin{aligned} & f(H_{30}[\theta], H_{31}[\theta], H_{32}[\theta], H_{33}[\theta]) \\ = \; & f(R(3)S_1(3, 7), S_1(3, 7)S_2(3, 7), S_2(3, 7)T(7)) \\ = \; & R(3)S_1(3, 7)S_2(3, 7)T(7) \vee S_1(3, 7)S_2(3, 7) \\ = \; & S_1(3, 7)S_2(3, 7), \end{aligned}$$

and hence it does not depend on $R(3)$ or $T(7)$.

We immediately obtain the following two corollaries:

COROLLARY 6.14. *If* θ *has at least 3 distinct transversals, then all variables in its transversals are in* $Var(\Psi[\theta])$.

PROOF. This follows immediately since if (i, j) is a transversal, then all $P_{\ell,i,j}[\theta]$ are 2-prime implicants in their respective $H_{k\ell}[\theta]$. □

COROLLARY 6.15. *If* θ *has at least 3 independent transversals, then every partial assignment* θ' *such that* $\Psi[\theta] = \Psi[\theta']$ *has the same set of transversals as* θ.

PROOF. We prove that every transversal of θ is a transversal of θ': This implies that θ' has at least three independent transversals, and therefore the converse holds, too (every transversal of θ' is a transversal of θ). Let $\Phi = \Psi[\theta] = \Psi[\theta']$. Let (i, j) be a transversal for θ. Since θ has at least three transversals, by Corollary 6.14, Φ depends on all variables of the transversal (i, j). It follows that θ' cannot set any of these variables. For $\ell \in [k-1]$, the 2-prime implicants within each $H_{k\ell}$ are disjoint from each other and, hence, if the variables are unset, then each such 2-prime implicant remains. Thus, for each $\ell \in [k-1]$ the Boolean function $H_{k\ell}[\theta']$ contains the 2-prime implicant $S_\ell(i, j)S_{\ell+1}(i, j)$.

It remains to prove that $H_{k0}[\theta']$ and $H_{kk}[\theta']$ each contain the 2-prime implicants on (i, j), $R(i)S_1(i, j)$, or $S_k(i, j)T(j)$, which are unset by θ'. To show this, we must rule out $R(i)$ or $T(j)$ absorbing them. We do this for $R(i)$; the case for $T(j)$ is analogous. Suppose, to the contrary, that $H_{k0}[\theta' \cup \{R(i) = 1\}] = 1$. If this is the case, then $\Psi[\theta' \cup \{R(i) = 1\}] = \Phi[R(i) = 1]$ does not depend on any of the R variables. However, since θ has (i, j) as a transversal as well as, in particular, another (independent) transversal (i', j') with $i' \neq i$, $H_{k0}[\theta]$ contains $R(i)S_1(i, j)$ and $R(i')S_1(i', j')$ as 2-prime implicants. It follows that $H_{k0}[\theta \cup \{R(i) = 1\}]$ depends on $R(i')$ and hence $\Psi[\theta \cup \{R(i) = 1\}] = \Phi[R(i) = 1]$ depends on $R(i')$, contradicting our earlier derivation that it did not depend on any R variables. □

Thus, for $k \geq 3$, the property of having k-independent transversals is a property of the subformula and not of an assignment, and for the rest of the section we will say that a restriction Φ of Ψ has k-independent transversals if there exists an assignment θ so $\Phi = \Psi[\theta]$ and θ has k independent transversals; by the above, this is equivalent to saying that all θ with $\Phi = \Psi[\theta]$ have k-independent transversals.

Proof of Lemma 6.7

We use the above to prove our lemma that are formulas are transparent if they are unit free and have sufficiently many independent transversals.

PROOF OF LEMMA 6.7. Suppose the contrary: Then there exist two partial assignments θ, θ' such that $\Phi = \Psi[\theta] = \Psi[\theta']$ and Φ has at least three independent traversals, but for some ℓ, $H_{k\ell}[\theta] \neq H_{k\ell}[\theta']$. Observe that if any $H_{k\ell}[\theta]$ or $H_{k\ell}[\theta']$ were the constant 1, then Φ would be transversal free, contradicting the fact that it has three independent traversals. Also observe that if any $H_{k\ell}[\theta]$ or $H_{k\ell}[\theta']$ contained a 1-prime-implicant (unit) Z, then setting $Z = 1$ would set the corresponding $H_{k\ell}$ to 1, which would eliminate all of its transversals, contradicting the assumption that Φ is $\mathbf{H_k}$-unit free. Therefore, all prime implicants of $H_{k\ell}[\theta]$ and $H_{k\ell}[\theta']$ are 2-prime implicants. Since all prime implicants in $H_{k\ell}[\theta]$ and $H_{k\ell}[\theta']$ are 2-prime implicants, Lemma 6.12 implies that all variables of all prime implicants are in $Var(\Phi)$.

Assume without loss of generality that $P_{\ell,i,j}$ is a 2-prime implicant of $H_{k\ell}[\theta]$ that is not a prime implicant of $H_{k\ell}[\theta']$; that is, $P_{\ell,i,j}[\theta] = P_{\ell,i,j}$ but $P_{\ell,i,j}[\theta'] = 0$. Let (i_0, j_0) and (i_1, j_1) be two independent transversals for θ (which are also transversals for θ' by Corollary 6.15) that are also independent of (i, j). Since in both θ and θ' the neighboring implicants of $P_{\ell,i,j}$ either remain as 2-prime implicants or are set to 0 in θ and θ' (though not necessarily the same in both), we can apply Lemma 6.12 to both θ and θ' to obtain μ and μ'. By the conclusion of Lemma 6.12, $\Phi[\mu] = \Psi[\theta \cup \mu] = f_\ell(P_{\ell,i,j})$ which is either $P_{\ell,i,j}$ or $\neg P_{\ell,i,j}$ but $\Phi[\mu'] = \Psi[\theta' \cup \mu'] = f_\ell(0)$, which is neither of the two. However, since the assignments μ and μ' depended only on the indices involved and Φ, we conclude that $\mu = \mu'$ which is a contradiction. □

The requirement that Φ be unit free is necessary for Lemma 6.7 to hold: A simple example is given by the formula $\Phi' = S_1(3, 7)$ in Example 6.2, which is not transparent.

The formula remains non-transparent even if we expand it with three independent transversals, for example, $S_1(3,7) \vee [R(4)S_1(4,4) \vee \ldots \vee S_3(4,4)T(4)] \vee [R(5)S_1(5,5) \vee \ldots] \vee [R(6)S_1(6,6) \vee \ldots]$, for the same reasons given in the example.

Proof of Lemma 6.8

Finally, we use the above properties to prove our characterization of $\mathbf{H_k}$-units in case there are sufficiently many independent transversals.

PROOF OF LEMMA 6.8. If $Z \in U(H_{k\ell}[\theta])$ for some ℓ. then, by definition, $H_{k\ell}[\theta \cup \{Z=1\}] = 1$, and therefore $\Psi[\theta \cup \{Z=1\}] = \Phi(\{Z=1\})$ is transversal free; it follows that $Z \in U_k(\Phi)$.

Conversely, suppose that $Z \notin \bigcup_{\ell \in \{0,\ldots,k\}} U(H_{k\ell}[\theta])$. In particular, none of the (monotone) formulas $H_{k\ell}[\theta \cup \{Z=1\}]$ is the constant 1. The assignment $Z=1$ can eliminate at most one of the ≥ 4 independent transversals in θ, so $\theta \cup \{Z=1\}$ has at least three (independent) transversals. Therefore, by Corollary 6.15, *every* partial assignment θ' such that $\Psi[\theta'] = \Phi[Z=1] = \Psi[\theta \cup \{Z=1\}]$ has at least three independent transversals. This implies that $\Phi[Z=1]$ is not transversal-free and hence $Z \notin U_k(\Phi)$. □

7. A DICHOTOMY THEOREM FOR EFFICIENT PROPOSITIONAL MODEL COUNTING

In this section, we present the proof of Theorem 3.12 that provides a characterization for a restricted class of queries for the existence of efficient (current) model counting algorithms on the propositional formulas. For this class, either all DLDDs require exponential size (therefore all modern model counting algorithms take exponential time), or we can construct a poly-size FBDD in polynomial time (data complexity), leading to a polynomial-time model counting algorithm.

The first part of Theorem 3.12 extends Theorem 3.9, where $f(\mathbf{X}) = g(\mathbf{X}, \mathbf{1})$.

Example 7.1. We illustrate with three examples as follows:

—$g = (X_0 \vee B_2) \wedge (B_0 \vee X_1)$, where $k=1$. Then $g(\mathbf{X}, \mathbf{1}) = 1$: It does not depend on X_0, X_1, and, therefore, the lineage has a poly-size FBDD.
—$g = X_0 \wedge (X_1 \vee B_3) \wedge (X_1 \vee B_5) \wedge (X_2 \vee X_3 \vee X_4 \vee X_5)$, where $k=5$. Then $g(\mathbf{X}, \mathbf{1}) = X_0 \wedge (X_2 \vee X_3 \vee X_4 \vee X_5)$: It does not depend on X_1, and the lineage has a poly-size FBDD.[11]
—$g = (X_0 \vee X_1) \wedge (X_1 \vee B_3) \wedge (X_2 \vee X_3)$, where $k=3$. Then $g(\mathbf{X}, \mathbf{1}) = (X_0 \vee X_1) \wedge (X_2 \vee X_3)$: It depends on all of $X_0, \ldots, X_3$, and therefore every DLDD for the lineage has exponential size.

Jha and Suciu [2013] gave a sufficient condition under which a UCQ is guaranteed to have a polynomial size FBDD. Our result here is novel in that it represents a necessary and sufficient condition, albeit for a very restricted fragment of UCQ.

Next we prove Theorem 3.12.

PROOF.

Proof of (1)

Suppose $f(X_0, \ldots, X_k) = g(\mathbf{X}, \mathbf{1}) = g(X_0, \ldots, X_k, 1, \ldots, 1)$ depends on all variables $X_0, \ldots, X_k$. Let $\mathcal{F}$ be an FBDD for the query $Q = g(h_{k0}, \ldots, h_{kk}, b_0, \ldots, b_{k+1})$, over the domain of size $n' = n+2$. We will convert $\mathcal{F}$ to an FBDD $\mathcal{F}'$ for query $Q' = f(h_{k0}, \ldots, h_{kk})$ over the domain of size n; further, $\mathcal{F}$ and $\mathcal{F}'$ will have the same size. By Theorem 3.9, the size of $\mathcal{F}'$ is $2^{\Omega(n)}$; therefore the size of $\mathcal{F}'$ is $2^{\Omega(n)} = 2^{\Omega(n')}$.

[11]This is Q_V in Jha and Suciu [2013] for which a poly-size FBDD was shown.

To convert $\mathcal{F}$ to $\mathcal{F}'$, modify $\mathcal{F}$ by setting the following values,[12] where $n_1 = n + 1$ and $n_2 = n + 2$:

$$\begin{aligned} R(n_1) = 1, R(n_2) = 0 & \\ S_\ell(n_2, n_2) = 1 & \quad \text{if } \ell \text{ is odd} \\ S_\ell(n_1, n_1) = 1 & \quad \text{if } \ell \text{ is even} \\ S_\ell(i, j) = 0 & \quad \forall \ell \in [k] \\ \forall \text{ other } (i, j) \in \{n_1, n_2\} \times \{n_1, n_2\} & \\ T(n_1) = 1, T(n_2) = 0 & \quad \text{if } k \text{ is odd} \\ T(n_2) = 1, T(n_1) = 0 & \quad \text{if } k \text{ is even.} \end{aligned}$$

Note that the modified FBDD $\mathcal{F}'$ computes the query $Q' = f(h_{k0}, \ldots, h_{kk})$ over a domain of size n: All queries $b_0, \ldots, b_{k+1}$ become `true` under the partial assignment above, while the lineages for $h_{k0}, \ldots, h_{kk}$ over domain $n' = n+2$ becomes their lineage over the domain of size n.

Proof of (2)

For the converse, assume that $f(X_0, \ldots, X_k) = g(X_0, \ldots, X_k, 1, \ldots, 1)$ does not depend on X_s. Denote $Q' = f(h_{k0}, \ldots, h_{kk})$: Its lineage is *transversal-free* (Definition 6.3) and therefore it has a shared OBDD $\mathcal{F}_s$ of size $O(n)$ for formulas $h_{k0}, \ldots, h_{kk}$ (see the proof of Proposition 6.4, which constructs the FBDD in poly-time for a fixed k).

Further, if for any ℓ, $b_\ell = 0$, then both $h_{\ell-1} = h_\ell = 0$, hence the lineage of the residual formula $f(X_0, \ldots, X_k)$ is transversal free. Therefore, the residual formula has a shared OBDD$\mathcal{F}_{0,\ell}$ of size $O(n)$ for $h_{k0}, \ldots, h_{kk}$ obtained by traversing the variables $R(i), S_1(i, j), \ldots, S_{\ell-1}(i, j)$ in row-major order and traversing the variables $S_{\ell+1}(i, j), \ldots, T(j)$ in column-major order.

We now describe the FBDD $\mathcal{F}$ for $Q = g(h_{k0}, \ldots, h_{kk}, b_0, \ldots, b_{k+1})$, which will have $k + 3$ layers: 0 to $k + 2$. (1) Layers 0 to $k + 1$ of $\mathcal{F}$ will have a tree structure; the $k + 2$ queries $b_0, \ldots, b_{k+1}$ are tested in these layers one after one. (As an optimization, the FBDD only needs to test those queries on which the function F depends.) (2) In the $k + 2$th layer, there will be copies of FBDDs $\mathcal{F}_s$ or $\mathcal{F}_{0,\ell}$, $\ell \in [k]$ described above.

The layers of $\mathcal{F}$ are described below:

Layer 0: Test b_0. Test the variables $R(1), R(2), \ldots, R(n)$ in an arbitrary order: For each node $R(i)$, its 0-child is $R(i + 1)$ and its 1-child is a root of a unique subtree at the next layer. The 0-child of the last node $R(n)$ is also leads to a unique subtree at the next layer. The total number of edges to the next layer is $n + 1$.

Layer ℓ, $1 \le \ell \le k$: Test for b_ℓ. Each sub-tree in the layer ℓ tests the variables $S_\ell(1, 1), \ldots, S_\ell(n, n)$ in an arbitrary order (e.g., row-major): For each node $S_\ell(i, j)$ its 0-child is the next variable in this order, and each 1-child is a root of a unique subtree at the next layer. The 0-child of the last node in this order is also a unique subtree at the next layer. The total number of edges from each of these subtrees to the next layer is $n^2 + 1$.

Layer $k + 1$: Test for b_{k+1}. Test the variables $T(1), \ldots, T(n)$ in an arbitrary order: The 0-child of $T(i)$ is $T(i + 1)$. All 1-children plus the 0-child of the last node points to a unique FBDD ($\mathcal{F}_s$ or $\mathcal{F}_{0,\ell}$ for some $\ell in [k]$) in the last layer.

[12]This means the following: Replace a node testing one of the variables Z mentioned above by a no-op node, whose unique child is either the 0-child or the 1-child of Z, according to the assignment.

Before we describe the last $k+2$-th layer, we note that each of the outputs from the $k+1$-th layer encode two pieces of information: (i) the values of all queries $b_0, \ldots, b_k$, that is, for each of them we know if it is 0 or 1, and (ii) we know which variables have been tested. In the FBDDs $\mathcal{F}_s$ or $\mathcal{F}_{0,\ell}$s in the last layer, we set the values of these variables according to the test earlier (replace the variable by a no-op node having a unique child based on its 0- or 1-value) to ensure that every variable is tested at most once in $\mathcal{F}$ (as is the case with the first $k+2$ layers).

Layer $k+2$: Shared FBDDs for $h_{k0}, \ldots, h_{kk}$. Consider any edge to layer $k+2$ from layer $k+1$. (1) If for any ℓ, $b_\ell = 0$, then both $query_{\ell-1} = h_\ell = 0$. Consider the least ℓ such that $h_\ell = 0$ and connect this edge to the shared FBDD $\mathcal{F}_{0,\ell}$ substituting the values of the variables that have already been tested.
(2) If for all ℓ, $b_\ell = 1$, then connect the edge to the shared FBDD $\mathcal{F}_s$ substituting the values of the variables that have already been tested.

Computing the function $g(h_0, \ldots, h_k, b_0, \ldots, b_{k+1})$. Now at the sinks of the FBDD $\mathcal{F}$ (sinks of the FBDDs in $k+2$-th layer), we know the values of the lineages for all queries $b_0, \ldots, b_{k+1}$ (from 0 to $k+1$-th layers) as well as for the queries $h_0, \ldots, h_k$ (from the $k+2$-th layer). Therefore, the value of the lineage of the query $Q = g(h_0, \ldots, h_k, b_0, \ldots, b_{k+1}$ can be easily computed.

The total number of nodes in the FBDD $\mathcal{F}$ is $O(n) \times [O(n^2)]^k \times O(n) \times O(n) = n^{O(1)}$. The completes the proof of part (2) of the theorem.

8. DISCUSSION

We have proved that decision-DNNFs and their generalization DLDDs can be efficiently converted into equivalent FBDDs that are at most quasipolynomially larger. As a consequence, known lower bounds for FBDDs imply lower bounds for decision-DNNFs and thus (a) exponential separations of the representational power of decision-DNNFs from that of both d-DNNFs and AND-FBDDs and (b) lower bounds on the running time of any algorithm that, either explicitly or implicitly, produces a decision-DNNF, including the current generation of exact model counting algorithms. Further, using these lower bounds, we proved exponential separations between lifted model counting using extensional query evaluation and state-of-the-art propositional methods for exact model counting. Our results were obtained by proving exponential lower bounds on the sizes of the decision-DNNF representations even for queries that can be evaluated in polynomial time.

Some natural questions arise as follows: Is there some other, more powerful, syntactic subclass of d-DNNFs that is useful for exact model counting? On the other hand, are there model-counting algorithms using DLDDs that are more efficient than exact model-counting using decision-DNNFs? In light of our lower bounds for probabilistic databases, it would be interesting to prove a dichotomy, classifying queries into those for which any decision-DNNF-based model counting algorithm takes exponential time and those for which such algorithms run in polynomial time. In this article, we showed such a dichotomy for a very restricted class of queries. A dichotomy for general model counting is known for the broader query class UCQ [Dalvi and Suciu 2012] that classifies queries as either #P-hard or solvable in polynomial time. Our separation results show that this same dichotomy does not extend to decision-DNNF-based algorithms; is there some other general dichotomy that can be shown for this class of algorithms? Finally, we entirely focused on exact model counting, whereas one can explore the limits of *approximate* model counting [Gomes et al. 2009], in general, and with respect to query evaluation in probabilistic databases.

REFERENCES

2014. The SDD Package: Version 1.1.1. Retrieved January 31, 2014, from http://reasoning.cs.ucla.edu/sdd/.

Sheldon B. Akers. 1978. Binary decision diagrams. *IEEE Trans. Comput.* 27, 6 (1978), 509–516.

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. 2003. Algorithms and complexity results for #SAT and Bayesian inference. In *FOCS*. 340–351.

Roberto J. Bayardo, Jr., and J. D. Pehoushek. 2000. Counting models using connected components. In *AAAI*. 157–162.

Paul Beame, Russell Impagliazzo, Toniann Pitassi, and Nathan Segerlind. 2010. Formula caching in DPLL. *ACM Trans. Comput. Theory* 1, 3 (2010).

Paul Beame, Henry A. Kautz, and Ashish Sabharwal. 2004. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res.* 22 (2004), 319–351.

Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. 2013. Lower bounds for exact model counting and applications in probabilistic databases. In *UAI*.

Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. 2014. Counting of query expressions: Limitations of propositional methods. In *ICDT*. 177–188.

Paul Beame and Vincent Liew. 2015. New limits for knowledge compilation and applications to exact model counting. In *UAI*. 131–140.

Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu. 2015. Symmetric weighted first-order model counting. In *PODS*. 313–328.

Eli Ben-Sasson and Avi Wigderson. 2001. Short proofs are narrow—Resolution made simple. *J. ACM* 48, 2 (Mar. 2001), 149–169.

Beate Bollig and Ingo Wegener. 1998. A very simple function that requires exponential size read-once branching programs. *Inf. Process. Lett.* 66, 2 (April 1998), 53–57.

Randal E. Bryant. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* 35, 8 (1986), 677–691.

Nilesh N. Dalvi and Dan Suciu. 2012. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM* 59, 6 (2012), 30.

Adnan Darwiche. 2001a. Decomposable negation normal form. *J. ACM* 48, 4 (2001), 608–647.

Adnan Darwiche. 2001b. On the tractable counting of theory models and its application to truth maintenance and belief revision. *J. Appl. Non-Class. Logic.* 11, 1–2 (2001), 11–34.

Adnan Darwiche. 2011. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. 819–826.

Adnan Darwiche and Pierre Marquis. 2002. A knowledge compilation map. *J. Artif. Int. Res.* 17, 1 (Sept. 2002), 229–264.

Martin Davis, George Logemann, and Donald Loveland. 1962. A machine program for theorem-proving. *Commun. ACM* 5, 7 (1962), 394–397.

Martin Davis and Hilary Putnam. 1960. A computing procedure for quantification theory. *J. ACM* 7, 3 (1960), 201–215.

Pedro Domingos and Daniel Lowd. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*.

Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2009. Model counting. In *Handbook of Satisfiability*. IOS Press, 633–654.

Eric Gribkoff and Dan Suciu. 2016. SlimShot: In-database probabilistic inference for knowledge bases. *Proc. VLDB* 9, 7 (2016), 552–563.

Jinbo Huang and Adnan Darwiche. 2005. DPLL with a trace: From SAT to knowledge compilation. In *IJCAI*. 156–162.

Jinbo Huang and Adnan Darwiche. 2007. The language of search. *J. Artif. Intell. Res.* 29 (2007), 191–219.

Manfred Jaeger and Guy Van den Broeck. 2012. Liftability of probabilistic inference: Upper and lower bounds. In *Proceedings of the 2nd International Workshop on Statistical Relational AI*.

Abhay Kumar Jha and Dan Suciu. 2011. Knowledge compilation meets database theory: Compiling queries to decision diagrams. In *ICDT*. 162–173.

Abhay Kumar Jha and Dan Suciu. 2013. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.* 52, 3 (2013), 403–440.

Stephen M. Majercik and Michael L. Littman. 1998. Using caching to solve larger probabilistic planning problems. In *AAAI*. 954–959.

William Joseph Masek. 1976. *A Fast Algorithm for the String Editing Problem and Decision Graph Complexity*. Master's thesis, MIT.

Christian Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. 2012. Dsharp: Fast d-DNNF compilation with sharpSAT. In *Canadian AI*. 356–361.

Igor Razgon. 2016. On the read-once property of branching programs and CNFs of bounded treewidth. *Algorithmica* 75, 2 (2016), 277–294.

Ashish Sabharwal. 2009. SymChaff: Exploiting symmetry in a structure-aware satisfiability solver. *Constraints* 14, 4 (2009), 478–505.

Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. 2004. Combining component caching and clause learning for effective model counting. In *SAT*.

Richard P. Stanley. 1997. *Enumerative Combinatorics*. Cambridge University Press.

Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases*. Morgan & Claypool.

Marc Thurley. 2006. sharpSAT: Counting models with advanced component caching and implicit BCP. In *SAT*. 424–429.

Leslie G. Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.

Guy Van den Broeck. 2011. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011.* 1386–1394.

Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. 2014. Skolemization for weighted first-order model counting. In *KR*.

Ingo Wegener. 2000. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM, Philadelphia, PA.

Received February 2015; revised May 2016; accepted August 2016