Model Counting of Query Expressions:
Limitations of Propositional Methods

Paul Beame    Jerry Li    Sudeepa Roy    Dan Suciu

University of Washington, Seattle, WA
{beame, jerryli, sudeepa, suciu}@cs.washington.edu

Abstract

Query evaluation in tuple-independent probabilistic databases is the problem of computing the probability of an answer to a query given independent probabilities of the individual tuples in a database instance. There are two main approaches to this problem: (1) in grounded inference one first obtains the lineage for the query and database instance as a Boolean formula, then performs weighted model counting on the lineage (i.e., computes the probability of the lineage given probabilities of its independent Boolean variables); (2) in methods known as lifted inference or extensional query evaluation, one exploits the high-level structure of the query as a first-order formula. Although it is 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 paper we show such a formal separation for the first time.

We exhibit a class of queries for which model counting can be done in polynomial time using extensional query evaluation, whereas the algorithms used in state-of-the-art exact model counters on their lineages provably require exponential time. Our lower bounds on the running times of these exact model counters follow from new exponential size lower bounds on the kinds of $d$-DNNF representations of the lineages that these model counters (either explicitly or implicitly) produce. Though some of these queries have been studied before, no non-trivial lower bounds on the sizes of these representations for these queries were previously known.

1 Introduction

Model counting is the problem of computing the number, $\#\Phi$, of satisfying assignments of a Boolean formula $\Phi$. In this paper 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) [24], there have been major advances in practical algorithms that compute exact, weighted model counts for many relatively complex formulas. Exact model counting for propositional formulas (see [12] for a survey) are based on extensions of backtracking search using the DPLL family of algorithms [11, 10] that were originally designed for satisfiability search.

The study of (weighted) model counting in this paper is motivated by query evaluation in tuple-independent probabilistic databases (see, e.g., [22]): given a (fixed) Boolean query $Q$, and independent probabilities of the individual tuples in a probabilistic database $D$, compute $\Pr[Q(D)]$, the probability that $Q$ is true on a random instance of $D$. Here the propositional formula $\Phi$ is the grounding of the first-order formula representing the query $Q$ on database $D$, and is called the lineage. We call this the grounded inference approach: first compute $\Phi$, then perform model counting on $\Phi$ at the propositional level to compute $\Pr[Q(D)]$.

The mismatch between the high level representation as a first-order formula and the low level of propositional inference was noted early on, and has given rise to various techniques that operate at the first-order level, which are collectively called lifted inference in statistical relational models (see [15] for an extensive
discussion), or extensional query evaluation in probabilistic databases [22]. These methods exploit the high level structure of the first-order formula for the query.

It is widely believed that lifted inference, or extensional query evaluation, is strictly more powerful than grounded inference that does not take advantage of the high level structure. While there have been examples in other contexts where provable separations have been shown (e.g., [19]), 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 queries and prove a formal statement which implies that grounded inference in the form of current propositional model counting algorithms requires exponential time on any member in the class. On the other hand, model counting for each query in this class can be done in polynomial time in the size of the domain using the inclusion/exclusion formula applied to the query expression [5] [22], which is a form of extensional query evaluation, or lifted inference.

We first review the state of the art for propositional model counting algorithms that are based on extensions of DPLL family of algorithms [11] [10]. Extensions include caching the results of solved sub-problems [17], dynamically decomposing residual formulas into components (RelSat [2]) and caching their counts ([1]), and applying dynamic component caching together with conflict-directed clause learning (CDCL) to further prune the search (Cachet [20] and sharpSAT [23]). In addition to DPLL-style algorithms that compute the counts on the fly, model counting has been addressed through a complementary approach, known as knowledge compilation, which converts 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 [7] [8] [14] [18]. Efficiency for knowledge compilation depends both on the size of the representation and the time required to construct it. These two approaches are quite related. As noted in c2d [14] (based on component caching) and Dsharp [18] (based on sharpSAT), the traces of all the DPLL-style methods yield knowledge compilation algorithms that can produce what are known as decision-DNNF representations [13] [14], a syntactic subclass of d-DNNF representations [8] [9].

Indeed, all the methods for exact model counting surveyed in [12] (and all others of which we are aware) can be converted to knowledge compilation algorithms that produce decision-DNNF representations, without any significant increase in their running time. A decision-DNNF is a rooted DAG where each node either tests a variable Z and has two outgoing edges corresponding to Z = 0 or Z = 1, or is an AND-node with two sub-DAGs that do not test any variable in common; these naturally correspond to the two types of operations in any modern DPLL-style algorithm: Shannon expansion on a variable Z, or partitioning the formula into two disconnected components. We will refer to the DPLL-style algorithms and knowledge compilation methods used in the state-of-the-art model counters as decision-DNNF-based model counting algorithms.

Therefore, we obtain our lower bounds on propositional model counting algorithms by showing that, for every query in the class we define, every decision-DNNF for the lineage of that query is exponentially large in the size of the domain; this implies that all current model counting algorithms will run in exponential time on the propositional grounding of that query.

Our Contributions. We first consider a well-studied family of simple queries whose associated (weighted) model counting problems are known to be #P-hard [0]. These queries have lineages that are simple 2-DNF formulas. We prove exponential lower bounds of the form \(2^{Ω(√n)}\) on the sizes of decision-DNNF representations of these lineages, which is the first non-trivial decision-DNNF lower bounds for these queries (Theorem 3.1).

We obtain these lower bounds by first proving that any FBDD representation for any of these formulas requires at least \(2^{n−1}/n\) size for a domain of size n; then using our recent result [3] showing that every decision-DNNF of size N can be converted into an FBDD of size at most \(N2^{log^2 N}\). (FBDDs, free binary decision diagrams, also known as read-once branching programs, are a restricted subclass of decision-DNNFs without any AND-nodes). A much weaker lower bound of \(2^{Ω(log^2 n)}\) was shown previously [16] for the FBDD size of one of these queries, but this is insufficient to yield any decision-DNNF lower bound using our translation.

We also strengthen the conversion from [3] to one with the same complexity that applies to a new, more general class of representations than decision-DNNFs (Theorem 2.1). We call this class decomposable logic decision diagrams (DLDDs). DLDDs can better represent ways of using decision-DNNF-based algorithms...
for query lineages, which naturally are given as DNF formulas rather than the CNF formulas that are the natural input for DPLL algorithms.

Given that model counting for the queries considered in Theorem 3.1 is \#P-hard even using lifted methods, it should not be surprising that we do not have efficient algorithms for model counting for these queries. So Theorem 3.1 does not lead to a separation between lifted and grounded inference. The separation we derive follows by substantially generalizing the class of queries for which the lower bounds on the sizes of FBDDs and DLDDs hold. Each of the queries in Theorem 3.1 is a disjunction of a number of elementary queries. We strengthen our lower bounds by showing that, if we generalize the disjunction operation to any Boolean combination of the same elementary queries such that the query \( Q \) built this way depends on all of the elementary queries, then the exponential lower bounds also hold for \( Q \) (Theorem 3.3). To prove this surprising result we show that every FBDD for such a Boolean combination can, with a small increase in size, be converted into an FBDD that simultaneously represents the lineages of all of its constituent elementary queries, and therefore into an FBDD for the disjunction query for which the previous lower bounds hold.

To obtain the separation between the decision-DNNF-based algorithms for model counting and lifted inference on a query \( Q \), we apply a characterization given in [6] for Union of Conjunctive Queries (UCQ), which gives a specific property (entirely) based on the structure of \( Q \) that allows exact model counting in time polynomial in the size of the database. This yields a \( 2^{\Omega(\sqrt{n})} \) versus \( n^{\Omega(1)} \) separation between these propositional and lifted methods for weighted model counting for a wide variety of such queries \( Q \) (Theorem 3.7).

Roadmap. We discuss some useful knowledge compilation representations in Section 2. In Section 3 we describe our main results which are proved in the following Sections 4 and 5. We discuss related issues in Section 8.

## 2 Background

In this section we review the knowledge compilation representations used in the rest of the paper.

### FBDDs.

An FBDD\(^1\) is a rooted directed acyclic graph (DAG) \( F \) that computes \( m \) Boolean functions \( \Phi = (\Phi_1, \ldots, \Phi_m) \). \( F \) 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 \( \{0, 1\}^m \). Every path from the root to some sink node may test a Boolean variable \( X \) at most once. For each assignment \( \theta \) on all the Boolean variables, \( \Phi[\theta] = (\Phi_1[\theta], \ldots, \Phi_m[\theta]) = L \), where \( L \) is the label of the unique sink node reachable by following the path defined by \( \theta \). The size of the FBDD \( F \) is the number nodes in \( F \). Typically \( m = 1 \), but we will also consider FBDDs \( F \) with \( m > 1 \) and call \( F \) a multi-output FBDD.

For every node \( u \), the sub-DAG of \( F \) rooted at \( u \), denoted \( F_u \), computes \( m \) Boolean functions \( \Phi_u \) defined as follows. If \( u \) is a decision node labeled with \( X \) and has children \( u_0, u_1 \) for 0- and 1-edge respectively, then \( \Phi_u = (\neg X)\Phi_{u_0} \lor X \Phi_{u_1} \); if \( u \) is a sink node labeled \( L \in \{0, 1\}^m \), then \( \Phi_u = L \). \( F \) computes \( \Phi = \Phi_r \) where \( r \) is the root. 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: \( \Pr[\Phi_u] = (1 - p(X)) \Pr[\Phi_{u_0}] + p(X) \Pr[\Phi_{u_1}] \).

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 the no-op node.

### Decision-DNNFs.

A decision-DNNF\(^2\) \( D \) generalizes an FBDD allowing decomposable AND-nodes in addition to decision-nodes, i.e., any AND-node \( u \) must satisfy the restriction that, for its two children \( u_1, u_2 \), the sub-DAGs \( D_{u_1} \) and \( D_{u_2} \) do not mention any common Boolean variables. The function \( \Phi_u \) is defined as \( \Phi_u = \Phi_{u_1} \land \Phi_{u_2} \), and its probability is computed as \( \Pr[\Phi_u] = \Pr[\Phi_{u_1}] \cdot \Pr[\Phi_{u_2}] \). In a decision-DNNF,

\(^1\)FBDDs are also known as Read Once Branching Programs.

\(^2\)A decision-DNNF is a special case of both an AND-FBDD (the AND nodes have no restriction) [25] and a d-DNNF [8], which are different kinds of circuits used in knowledge compilation. see [3] for a discussion.
similar to FBDDs, any Boolean variable can be tested at most once along any path from the root to any sink.

**DLDDs.** In this paper we introduce *Decomposable Decision Logic Diagrams* or DLDDs by further generalizing a decision-DNNF. A DLDD can also have NOT-nodes $u$ having a unique child $u_1$, and decomposable OR-, XOR-, and EQUIV-nodes similar to decomposable AND-nodes\(^3\) (i) for a NOT-node, $\Phi_u = \neg \Phi_{u_1}$, and $\Pr[\Phi_u] = 1 - \Pr[\Phi_{u_1}]$; (ii) for an OR-node, $\Phi_u = \Phi_{u_1} \lor \Phi_{u_2}$, and $\Pr[\Phi_u] = 1 - (1 - \Pr[\Phi_{u_1}]) \cdot (1 - \Pr[\Phi_{u_2}])$; (iii) for an XOR-node, $\Phi_u = \Phi_{u_1} \land \neg \Phi_{u_2} \lor \neg \Phi_{u_1} \land \Phi_{u_2}$, and (iv) for an EQUIV-node, $\Phi_u = \Phi_{u_1} \land \Phi_{u_2} \lor \neg \Phi_{u_1} \land \neg \Phi_{u_2}$ (again $\Pr[\Phi_u]$ can easily be computed from $\Pr[\Phi_{u_1}], \Pr[\Phi_{u_2}]$). Hence the probability of the formula can still be computed in time linear in $D$.

**Conversion of a DLDD into an equivalent FBDD.** The trace of any DPLL-based algorithm with caching and components is a decision-DNNF. Therefore any lower bound on the size of decision-DNNFs represents a lower bound on the running time of modern model counting algorithms. We have proven recently the first lower bounds on decision-DNNFs \(^3\). However, model counting algorithms were designed for CNF expressions: for example, the component analysis partitions the clauses into two disconnected components (without common variables), then computes the probability as $\Pr[\Phi_u]$. Hence the probability of the formula can still be computed in time linear in $D$.

**Theorem 2.1.** For any DLDD $D$ with $N$ nodes there exists an equivalent FBDD $F$ computing the same formula as $D$, with at most $N2\log^2 N$ nodes (at most quasi-polynomial increase in size).

In \(^3\) we have proven a similar result with the same bound for decision-DNNFs; now we strengthen it to DLDDs. The proof of Theorem 2.1 appears in Section 7.

### 3 Main Results

Here we formally state our main results and discuss their implications, and defer the proofs to the following sections. But first we introduce some elementary queries that work as building blocks for the class of queries considered in these results \(^6\)\(^10\).

Let $[n]$ denote the set $\{1, \ldots, n\}$. Fix $k > 0$ and consider the following set of $k + 1$ Boolean queries $h_k = (h_{k0}, \ldots, h_{kk})$, where

\[
\begin{align*}
  h_{k0} &= \exists x_0 \exists y_0 \ R(x_0) \land S_1(x_0, y_0) \\
  h_{k\ell} &= \exists x_\ell \exists y_\ell \ S_\ell(x_\ell, y_\ell) \land S_{\ell+1}(x_\ell, y_\ell) \quad \forall \ell \in [k-1] \\
  h_{kk} &= \exists x_k \exists y_k \ S_k(x_k, y_k) \land T(y_k)
\end{align*}
\]

Fix a domain size $n > 0$; for each $i, j \in [n]$, let $R(i), S_1(i, j), \ldots, S_k(i, j), T(j)$ be Boolean variables representing potential tuples in the database. Then the corresponding *lineages*, the associated Boolean expressions for these queries are \(^4\)

\[
\begin{align*}
  H_{k0} &= \bigvee_{i,j \in [n]} R(i)S_1(i,j), \\
  H_{kk} &= \bigvee_{i,j \in [n]} S_k(i,j)T(j), \\
  H_{k\ell} &= \bigvee_{i,j \in [n]} S_\ell(i,j)S_{\ell+1}(i,j) \quad \forall \ell \in [k-1]
\end{align*}
\]

We define $H_k = (H_{k0}, \ldots, H_{kk})$. Two well-studied queries \(^6\) that we will consider in this section are given below:

\(^3\)These four nodes along with NOT-nodes can capture all possible non-constant functions on two Boolean variables

\(^4\)For simplicity, conjunctions in Boolean formulas are represented as products.
Query $h_k$: $h_k$ is a disjunction on the queries in $h_k$: $h_k = h_{k0} \lor h_{k1} \lor \cdots \lor h_{kk}$. The lineage $H_k$ of $h_k$ is given by $H_k = H_{k0} \lor H_{k1} \lor \cdots \lor H_{kk}$.

Query $h_0$: Also we define $h_0$ that uses a single relation symbol $S$ in addition to $R$ and $T$: $h_0 = \exists x \exists y \ R(x) \land S(x, y) \land T(y)$. $S$ is defined on Boolean variables $S(i, j)$, $i, j \in [n]$, and therefore the lineage $H_0$ of $h_0$ is $H_0 = \bigvee_{i,j\in [n]} R(i)S(i, j)T(j)$.

Lower bounds on FBDDs for queries $h_0$, $h_k$

Jha and Suciu [16] previously showed that every FBDD for the lineage $H_1$ of $h_1$ has size $2^{\Omega(\log^2 n)}$. Our first result improves this to an exponential lower bound, not just for $H_1$ but also for $H_0$ and all $H_k$ for $k > 1$:

**Theorem 3.1.** For every $n > 0$, any FBDD for $H_0$ or $H_k$ for $k \geq 1$ has $\geq 2^{(n^2 - 1)/n}$ nodes.

It is known that weighted model counting for both $H_1$ and $H_k$ is #P-hard [6]. 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 4. This improved bound is critical for proving the overall lower bound result in this paper (Theorem 3.4).

While we do not need $h_0$ and $H_0$ in the rest of the paper, we include it in Theorem 3.1 because it is obtained in a fashion similar to that for $H_k$ and substantially improves on a $2^{\Omega(\sqrt{n})}$ lower bound for $H_0$ from our previous work [3] which was based on a result by of Bollig and Wegener [4]. Our new lower bound improves this to the nearly optimal $2^{(n^2 - 1)/n}$.

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 [4] but yield as good a lower bound on FBDD sizes in terms of variables and even better bounds as a function of their number of terms.

Lower bounds for FBDDs for queries over $h_k$

Theorem 3.1 gives a lower bound on $h_k$, which is simply the logical OR of the queries in $h_k$. Theorem 3.3 below generalizes this result by allowing queries that are arbitrary functions of queries in $h_k$.

Let $f(X) = f(x_0, x_1, \cdots, x_k)$ be an arbitrary Boolean function on $k + 1$ Boolean variables $X = (x_0, \cdots, x_k)$, and $Q$ the Boolean query $Q = f(h_{k0}, h_{k1}, \cdots, h_{kk})$. Clearly, the lineage of $Q$ is $H_k = f(H_{k0}, H_{k1}, \cdots, H_{kk})$.

**Example 3.2.** If $f(x_0, x_1, \cdots, x_k) = \bigvee_{\ell=0}^k x_{\ell}$, we get query $h_k = \bigvee_{\ell=0}^k h_{k_{\ell}}$; its lineage is $H_k = \bigvee_{\ell=0}^k H_{k\ell}$.

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

**Theorem 3.3.** If $f$ depends on all $k + 1$ variables $x_0, \cdots, x_k$, then any FBDD $F$ with $N$ nodes for the lineage of $Q = f(h_{k0}, \cdots, h_{kk})$ can be converted into a multi-output FBDD for $(H_{k0}, \cdots, H_{kk})$ with $O(k2^k n^3 N)$ nodes. In particular, for $k \leq \alpha n$ for any constant $\alpha < 1$, $F$ requires at least $2^{O(n)}$ nodes.

We prove the theorem in Section 5. The condition that $f$ depends on all variables is necessary (see Sections 4 and 5): if $Q$ does not depend on any one of the queries in $h_k$, then its lineage has an FBDD of size linear in the number of Boolean variables.

---

5Bollig and Wegener defined a set $E_a \subseteq [n] \times [n]$ for which any FBDD for the formula $\bigvee_{(i,j) \in E_a} R(i)T(j)$ requires size $2^{n^{\Omega(\sqrt{n})}}$ which obviously implies the same lower bound for $H_0$. The set $E_a$ is given as follows: Assume that $n = p^a$ where $p$ is a prime number. Each number $0 \leq i < n$ can be uniquely written as $i = a + bp$ where $0 \leq a, b < p$. Then: $E_a = \{ (i+1, j+1) | i = a + bp, j = c + dp, c \equiv (a + bd) \mod p \}$.

6In the formulas of [4], $p$ is analogous to $n$ in our formulas and theirs have $p^3$ terms, versus only $2n^2$ for our formulas.
other queries. While we were inspired by that proof, the techniques we use in Theorem 3.3 are considerably more powerful, and uses new ideas which can be of independent interest to show lower bounds on the size of FBDDs in general.

We also extend the lower bound in Theorem 3.3 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 decision-DNNF-based model counting algorithms require exponential time. The details appear in Section 6.

Lower Bounds for Model Counting Algorithms for Queries over \( h_k \)

Theorems 2.1, 3.1, and 3.3 together prove the following lower bound result:

**Theorem 3.4.** If \( Q \) is a Boolean combination of the queries in \( h_k \) that depends on all \( k + 1 \) queries in \( h_k \), then any DLDD (and therefore any decision-DNNF) for the lineage \( \Theta \) of \( Q \) has size \( 2^{\Omega(\sqrt{n})} \).

**Proof.** Let \( N \) be the size of a DLDD for \( Q \). By Theorem 2.1, \( Q \) has an FBDD of size \( N 2^{\log^2 N} \). By Theorem 3.3, \( H_1 \) has an FBDD for size \( 2^{O(\log^2 N)} \), which has to be \( 2^{\Omega(n)} \) by Theorem 3.1 implying that \( N \) is \( 2^{\Omega(\sqrt{n})} \).

Notice that in order to prove Theorem 3.4, we needed the strong exponential lower bound on the size of an FBDD for \( H_k \) in Theorem 3.1: the prior quasipolynomial lower bound was not sufficient because of the quasipolynomial increase in size in Theorem 2.1 moving from DLDDs to FBDDs.

Since, as we discussed previously, 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 3.5.** 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 \( h_k \) and depends on all \( k + 1 \) queries in \( h_k \).

Propositional versus Lifted Model Counting

Theorem 3.4 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 \( h_k \) for which lifted methods taking advantage of the high-level structure yield polynomial-time model counting and therefore outperform current propositional techniques.

Consider the case when \( Q = f(h_0) \) 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 \( f \), and we review it here briefly following [22].

To check if weighted model counting for \( Q \) 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 \( u \subseteq X \) 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 \( \hat{0} \)), while the minimal element is \( X \) (we denote it \( \hat{1} \)). The Möbius function on the lattice \( L, \mu : L \times L \to R \), is defined as \( \mu(u, v) = 1 \) and \( \mu(u, v) = -\sum_{u < w \leq v} \mu(w, v) \) [24]. The following holds [22]: if \( \mu(0, 1) = 0 \), then weighted model counting for \( Q \) can be done in time polynomial in \( n \) (using in the inclusion/exclusion formula on the CNF); if \( \mu(0, 1) \neq 0 \), then the weighted model counting problem for \( Q \) is \#P-hard.

**Example 3.6.** Here we give examples of easy and hard queries:

- For a trivial example, \( h_k = h_{k0} \lor \cdots \lor h_{kk} \) has a single clause, hence its lattice has exactly two elements \( \emptyset \) and \( 1 \), and \( \mu(0, 1) = -1 \), hence \( h_k \) is \#P-hard.

\[ \text{Footnote}\] The propositional formula \( f \) describes the query expression \( Q \), and should not be confused with the propositional grounding of \( Q \) on the instance \( R(i), S_1(i, j), \ldots, S_k(i, j), T(j) \); \( i \in [1, k - 1], j \in [n] \).
In this section we prove Theorem 3.1 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 $\Phi$ be a Boolean formula. A prime implicant (or minterm) of $\Phi$ is a term $T$ such that $T \Rightarrow \Phi$ and no proper subterm of $T$ implies $\Phi$. 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 $X \lor YZ \lor YU \lor W$.

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.

**Theorem 3.7** (Main Result). If $Q$ is a Boolean combination of the queries in $h_k$ that depends on all $k+1$ queries in $h_k$ 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 decision-DNNF-based propositional algorithms for model counting require exponential time on the lineage.

4 Exponential lower bounds for all $H_k$

In this section we prove Theorem 3.1 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.

Figure 1: The lattices for (a) $f_W$, (b) $f_9$.

- Two more interesting examples for $k = 3$:

$f_W = (X_0 \lor X_2) \land (X_0 \lor X_3) \land (X_1 \lor X_3)$

$f_9 = (X_0 \lor X_3) \land (X_1 \lor X_3) \land (X_2 \lor X_3) \land (X_0 \land X_1 \land X_2)$

Their lattices, shown in Figure 1, satisfy $\mu(\hat{0}, \hat{1}) = 0$, therefore weighted model counting for $Q_W = f_W(h_{30}, h_{31}, h_{32}, h_{33})$ and $Q_9 = f_9(h_{30}, h_{31}, h_{32}, h_{33})$ 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 $Pr[Q_W] =$

$Pr[h_{30} \lor h_{32}] + Pr[h_{30} \lor h_{33}] + Pr[h_{31} \lor h_{33}]$

$- Pr[h_{30} \lor h_{32} \lor h_{33}] - Pr[h_{30} \lor h_{31} \lor h_{33}]$

$- Pr[h_{30} \lor h_{31} \lor h_{32} \lor h_{33}] + Pr[h_{30} \lor h_{31} \lor h_{32} \lor h_{33}]$

While computing $Pr[h_{30} \lor h_{31} \lor h_{32} \lor h_{33}]$ 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 3.4 proves that, if we ground $Q_W$ or $Q_9$ first, then any decision-DNNF-based model counting algorithm will take exponential time on the lineage. This leads to the main separation result of this paper:

**Theorem 3.7** (Main Result). If $Q$ is a Boolean combination of the queries in $h_k$ that depends on all $k + 1$ queries in $h_k$ 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 decision-DNNF-based propositional algorithms for model counting require exponential time on the lineage.
Definition 4.1. Let $F$ be an FBDD for a Boolean function $\Phi$. Call a node $u$ in $F$ a unit node if $\Phi_u$ has a unit, and a decision node otherwise. We say that $F$ follows the unit rule if for every unit node $u$ the variable tested at $u$ is a unit.

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

For a variable $X$ of $\Phi$, define the degree of $X$ in $\Phi$ to be the maximum over all partial assignments $\theta$ of the number of unit of $\Phi[\theta \cup \{X=1\}]$ that are not units of $\Phi[\theta]$. (If $\Phi$ is a DNF formula then the degree of $X$ is at most the number of distinct variables to co-occur in terms with $X$.) Write $\Delta(\Phi)$ for the maximum degree of any variable in $\Phi$. In section 4.1 we prove the following:

Lemma 4.2. If $\Phi$ is a monotone formula with FBDD $F$ of size $N$, then $\Phi$ 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 4.3. 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 3.1 is an immediate consequence of Corollary 4.3 together with the following lemma.

Lemma 4.4. Every FBDD $F$ for $H_k$ that follows the unit rule has size $\geq 2^{(n-1)}$.

The proof of Lemma 4.4 follows using a general technique in which one defines a notion of admissible paths in $F$. We will give such a definition and show that no two admissible paths in $F$ can lead to the same node of $F$ since they must correspond to different subfunctions of $H_k$. We will further show that every admissible path branches off from other admissible paths at least $n-1$ times, guaranteeing that $F$ must contain a complete binary tree of distinct nodes of depth $n-1$ (in which edges may have been stretched to partial paths).

For the remainder of this section we fix some FBDD $F$ for $H_k$ that follows the unit rule. Given a path $P$ in $F$, let $\text{Row}(P)$ be the set of $i \in [n]$ so that $P$ tests $R(i)$ at a decision node or there are $\ell$ and $j$ so that $P$ tests $S_\ell(i, j)$ at a decision node; similarly, let $\text{Col}(P)$ be the set of $j \in [n]$ which $P$ tests $T(j)$ at a decision node or there is some $\ell$ and $i$ so that $P$ tests $S_\ell(i, j)$ at a decision node. Let $P$ be the set of partial paths $P$ starting at the root and ending at a (non-leaf) decision node so that both $|\text{Row}(P)| < n$ and $|\text{Col}(P)| < n$ but any extension of $P$ has either $|\text{Row}(P)| = n$ or $|\text{Col}(P)| = n$.

The following is an easy observation.

Lemma 4.5. For all $k \geq 0$, if $P_1, P_2 \in P$ with $H_k[P_1] = H_k[P_2]$ then the two paths test the same set of $R$ and $T$ variables and must assign those tested the same values.

Proof. Suppose that there is some $R(i)$ such that $P_1$ assigns $R(i)$ value $b \in \{0, 1\}$ that $P_2$ does not. $H_k[P_2] = H_k[P_1]$ does not depend on $R(i)$ so we can assume without loss of generality that $P_2$ assigns $R(i)$ value $1-b$. Suppose without loss of generality that $P_1$ sets $R(i)$ to 1 and $P_2$ sets $R(i)$ to 0.

First consider the case $k = 0$. Let $j_1 \in [n] - \text{Col}(P_1)$. Since $P_2$ sets $R(i)$ to 0, $H_0[P_2]$ does not depend on $S(i, j_1)$ but $H_0[P_1]$ sets neither $T(j_1)$ nor $S(i', j_1)$ for any $i'$, so it does depend on $S(i, j_1)$, a contradiction.

Now suppose that $k \geq 1$. Let $j_2 \in [n] - \text{Col}(P_2)$. As $F$ follows the unit rule, this implies that $P_1$ sets $S_1(i, j) = 0$ for all $j$, which in particular implies that $H_k[P_1]$, and thus $H_k[P_2]$, does not depend on $S_1(i, j)$. However, since $j_2 \notin \text{Col}(P_2)$, all terms of $H_k[i]$ for $\ell \in [k]$ involving indices $(i', j_2)$ are unset for every $i'$, which implies that $H_k[P_2]$ depends on $S_1(i, j_2)$, a contradiction. The case when the difference is $T(j)$ is analogous.

We will first prove Lemma 4.4 for $k = 2m + 1$ odd. The cases when $k > 0$ is even as well as when $k = 0$ use almost identical techniques; the proofs in these cases are given in the appendix.

Definition 4.6. Let $P$ be a partial path through $F$ starting at the root. It is admissible if for all $i, j$, it is consistent with one of the four following assignments:

8
resulted in a forbidden path, and call the assignment \( P \) of \( \Phi \). The following proposition will be useful because it implies that for monotone formulas, setting units cannot create additional units.

**Proposition 4.8.** If \( \Phi \) is a monotone function and \( W \) is a variable in \( \Phi \), then \( U(\Phi|W=0) \subseteq U(\Phi) \).

| \( i \) | \( S_1(i,j) \) | \( S_2(i,j) \) | \( S_3(i,j) \) | \( S_4(i,j) \) | \( S_5(i,j) \) | \( T(j) \) |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |

Table 1: The patterns for admissible paths for \( k = 5 \).

1. \( R(i) = T(j) = 0 \) and \( S_2(i,j) = 0 \) for all \( \ell \) odd and \( S_\ell(i,j) = 1 \) for all \( \ell \) even,
2. \( R(i) = T(j) = 1 \) and \( S_2(i,j) = 1 \) for all \( \ell \) even and \( S_\ell(i,j) = 0 \) for all \( \ell \) odd,
3. \( R(i) = 0, T(j) = 1 \) and \( S_2(i,j) = 1 \) for all \( \ell \) even and \( S_\ell(i,j) = 0 \) for all \( \ell \) odd, or
4. \( R(i) = 1, T(j) = 0 \) and \( S_2(i,j) = 1 \) for all \( \ell \) even and \( S_\ell(i,j) = 0 \) for all \( \ell \) odd.

\( P \) is forbidden if it is not admissible. Let \( A \subset \mathcal{P} \) be the set of admissible paths in \( \mathcal{P} \). (See Table 1 for the case \( k = 5 \)).

**Lemma 4.7.** If \( P_1, P_2 \in A \) are distinct then \( H_k[P_1] \neq H_k[P_2] \).

**Proof.** Suppose \( P_1, P_2 \in A \) are distinct with \( H_k[P_1] = H_k[P_2] = F \). Let \( u \) be the first node at which \( P_1 \) and \( P_2 \) diverge, and assume without loss of generality that \( P_1 \) takes the 0-edge and \( P_2 \) takes the 1-edge. Notice that \( u \) must be a decision node. By \( \text{Lemma 4.5} \), if \( u \) is a decision node, the node \( u \) cannot test a \( R(i) \) or \( T(j) \) variable so it must test \( S_\ell(i,j) \) for some \( i, j \). Assume that \( \ell \) is even (the case when \( \ell \) is odd is symmetrical; switch the roles of \( P_1 \) and \( P_2 \)). Then \( F \) does not contain the prime implicant \( S_\ell(i,j)S_{\ell+1}(i,j) \) and does not contain any units, but along \( P_2 \) the variable \( S_\ell(i,j) = 1 \), so \( S_{\ell+1}(i,j) = 0 \) along \( P_2 \). This implies that \( F \) does not contain the prime implicant \( S_{\ell+1}(i,j)S_{\ell+2}(i,j) \) but since \( P_1 \) cannot set \( S_{\ell+1}(i,j) = 0 \) as otherwise it would be forbidden, this implies that \( P_1 \) sets \( S_{\ell+2}(i,j) = 0 \). Inductively, we conclude that \( S_{\ell+2p}(i,j) = 0 \) is set to zero on \( P_1 \) and \( S_{\ell+1+2p}(i,j) \) is set to zero on \( P_2 \) for all non-negative integers \( p \leq (k - \ell - 1)/2 \). In particular, \( S_k(i,j) = 0 \) along \( P_2 \), so the prime implicant \( S_k(i,j)T(j) \) does not appear in \( F \); as \( F \) has no units, \( T(j) \) must be set to zero in \( P_1 \), as otherwise it would be forbidden. Doing the same procedure but inducting downwards, we also conclude that \( R(i) = 0 \) in \( P_1 \) and \( S_1(i,j) = 0 \) in \( P_2 \). However, by \( \text{Lemma 4.5} \), this implies that \( R(i) = T(j) = 0 \) in \( P_2 \), and since \( S_1(i,j) = S_k(i,j) = 0 \) we conclude that \( P_2 \) is forbidden, which is a contradiction. 

of Lemma 4.4. By Lemma 4.7, it suffices to count how many paths are in \( A \), because each such path must correspond to a unique node in the FBDD. We show that there are at least \( 2^{n-1} \) such paths. For any path \( P \in \mathcal{A} \), call an assignment at a decision node \( u \) along \( P \) forced if taking the opposite assignment would have resulted in a forbidden path, and call the assignment unforced otherwise. We claim that there are at least \( n - 1 \) unforced assignments along any path \( P \in \mathcal{P} \). Since some extension of \( P \) either sets some variable in all rows or in all columns, \( P \) itself must have either \( |\text{Row}(P)| = n - 1 \) or \( |\text{Col}(P)| = n - 1 \). Without loss of generality assume that \( |\text{Row}(P)| = n - 1 \). Then the patterns of admissible paths ensure that, for each \( i \in \text{Row}(P) \), the first decision node \( u \) along \( P \) testing a variable either of the form \( R(i) \) or \( S_1(i,j) \) for some \( \ell \) and \( j \) must be unforced (see Table 1). So there must be at least \( n - 1 \) unforced assignments.

We now define an injection from \( \{0,1\}^{n-1} \) to \( \mathcal{A} \), as follows: map each sequence of bits \( (a_1, \ldots, a_{n-1}) \) to the unique path \( P \in \mathcal{A} \) that at its \( i \)-th unforced decision takes the \( a_i \)-edge for \( i \leq n-1 \), takes the 1-edge at all unforced decisions after its first \( n-1 \) unforced decisions, makes all forced decisions as required, and at each unit node takes the 0 branch. The existence of such an injection implies that \( |\mathcal{A}| \geq 2^{n-1} \), as claimed.

4.1 Proof of Lemma 4.2

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

**Proposition 4.8.** If \( \Phi \) is a monotone function and \( W \) is a variable in \( \Phi \), then \( U(\Phi|W=0) \subseteq U(\Phi) \).
Let $\mathcal{F} = (V, E)$ be an FBDD for a monotone formula $\Phi$, 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 4.8 any edge $e$ for which $U(e)$ is non-empty must be labeled 1 in $\mathcal{F}$.

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

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

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 $\mathcal{F}'$ will be the $i$-th element of $U(e)$ under the ordering $\pi$; 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 4.2 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 4.9.** 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 $\Phi_u$. Therefore $\Phi_u = W \lor \Phi_u'$ for some $\Phi_u'$. Since $\mathcal{F}$ is an FBDD and $W$ labels $w$, $W$ is not set on the path from $v$ to $w$, hence $\Phi_w = W \lor \Phi''_w$ for some formula $\Phi''_w$. 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. \hfill $\square$

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

**Proposition 4.10.** If there is a path from $u$ to $v$ in $\mathcal{F}$ and $X \in U(\Phi_u)$ then either $X \in U(\Phi_v)$, or $\Phi_v = 1$, or $X$ is queried on the path from $u$ to $v$ and hence $\Phi_v$ does not depend on $X$.

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

Taken together with the size bound for our construction, the following lemma immediately implies Lemma 4.2.

**Lemma 4.11.** Let $\Phi$ be monotone and computed by FBDD $\mathcal{F}$. Then $\mathcal{F}'$ is an FBDD for $\Phi$ that follows the unit rule.

**Proof.** We first show that $\mathcal{F}'$ is an FBDD, namely, every root-leaf path $P$ in $\mathcal{F}'$ queries 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 descendent node $v$, the formula $\Phi_v$ no longer depends on $X$, hence $X \not\in U(e)$. 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 $F'$. 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 $X \notin U(v_2)$ where there is a path from $v_1$ to $v_2$ in $F$. By Proposition 4.10 this implies that $F_{u_2}$ does not depend on $X$ which contradicts the requirement that $X \in U(v_2)$ since $v_2$ is a child of $v_2$.

By construction, $F'$ obviously follows the unit rule. It remains to prove that $F'$ computes $\Phi$. We show something slightly stronger: For any function $F$, define $\bar{F}$ to be $F(F(\bar{F})=0)$, in which all variables in $U(F)$ are set to 0. We claim by induction that for all nodes $v \in V$, if $\theta'$ labels a path in $F'$ from the root to $v$, then $\Psi[\theta'] = \Phi_{\theta'}$, and $\theta' = \theta \cup (U(\Phi_v)=0)$ for some $\theta$ that labels a path in $F$ from the root to $v$. This trivially is true for the root. If it is true for the output nodes, then $F'$ correctly computes $\Psi$ 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 $\theta'$ from the root to $v$ in $F'$ for which $u$ is the last vertex in $V$ on $\theta'$. 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 $F$ is $W$. We have 3 cases: If $e = (u, v) \in E$ is a 1-edge then $\Phi_v = \Phi_u[W=1]$. Every path $\theta'$ from the root to $v$ through $u$ is of the form $\theta' = \theta \cup \{W=1\} \cup \{U(e)=0\}$ for some $\theta$ that labels a path from the root to $u$ in $F$. (This is true even if $U(e)$ is empty.) By induction, $\Psi[\theta] = \Phi_u[U(\Phi_u)=0]$ and by definition $U(\Phi_v) = U(\Phi_u) \cup U(e)$ so $\Psi[\theta'] = \Phi_u[U(\Phi_u)=0] = \Phi_u[U(\Phi_v)=0]$ as required. If $e = (u, v) \in E$ is a 0-edge of $F$ then $\Phi_v = \Phi_u[W=0]$. If $u$ became a no-op vertex in $F'$ then there was some ancestor $w$ of $u$ at which $W$ became a unit of $\Phi_w$. Since $F'$ is an FBDD, it does not query $W$ between that ancestor and $u$. By Proposition 4.10 either $W \in U(\Phi_u)$ or $\Phi_u = 1$. In the latter case, $\Phi_v = \Phi_u = 1$ and the correctness for $u$ implies that for $v$. In the former case, $U(\Phi_u) = U(\Phi_v) \cup \{W\}$ and $\Phi_u[W] = \Phi_v[W]$ and again the correctness for $\Phi_v$ implies that for $\Phi_v$. In the case that $u$ does not become a no-op vertex, $W$ is not a unit of $\Phi_u$ 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 $\Phi_u[U(\Phi_u)=0]$ yields $\Phi_v[U(\Phi_v)=0] = \Phi_v$ as required. Therefore the statements holds for all possible paths from the root to $v$. 

5 Lower bounds for Boolean combinations over $H_k$

In this section we prove Theorem 3.3. Throughout this section we fix $f(X) = f(X_0, \ldots, X_k)$, a Boolean function that depends on all variables $X$, and a domain size $n > 0$. To prove Theorem 3.3 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 $H_k = (H_{k0}, H_{k1}, \ldots, H_{kk})$ with at most an $O(k^2 n^3)$ increase in size. The proof is constructive. Theorem 3.3 then follows immediately using Theorem 3.1 since any FBDD for $H_k$ yields an FBDD for $H_k$ of the same size.

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

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

In order to produce the FBDD $F'$ for $H_k$ from $F$ computing $\Psi = f(H_k)$, we would like to ensure that every internal node $v$ of $F'$ 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 $H_{k\ell}(v) \equiv H_{k\ell}[P]$ for all $\ell$. Since we will not easily be able to characterize its paths we find it convenient to define this property not only with respect to paths of $F'$ but for formulas $\Phi_v$ with respect to arbitrary partial assignments $\theta$. We use the term transparent to describe this property since it ensures that the value of $\Phi_v$ automatically also reveals the values for all $H_{k\ell}(v)$.

Definition 5.1. Fix $\Psi = f(H_{k0}, \ldots, H_{kk})$. A formula $\Phi$ that is a restriction of $\Psi$ is called transparent if there exist $k+1$ formulas $\varphi_0, \ldots, \varphi_k$ such that, for every partial assignment $\theta$, if $\Phi = \Psi[\theta]$, then $H_{k0}[\theta] = \varphi_0$, ..., $H_{kk}[\theta] = \varphi_k$. We say that $\Phi$ defines $\varphi_0, \ldots, \varphi_k$.

In other words, assuming that $\Phi$ is derived from $\Psi$ as $\Phi = \Psi[\theta]$ for some partial assignment $\theta$, then $\Phi$ is transparent if the formulas $H_{k0}[\theta], \ldots, H_{kk}[\theta]$ are uniquely defined; i.e., are independent of $\theta$. Equivalently, for any two assignments $\theta, \theta'$, if $\Psi[\theta] = \Psi[\theta'] = \Phi$, then for all $0 \leq \ell \leq k$, $H_{k\ell}[\theta] = H_{k\ell}[\theta']$. 

11
Example 5.2. Let $k = 3$ and $f = X_0 \lor X_1 \lor X_2 \lor X_3$. Given a domain size $n > 0$, the formula $\Phi$ is:

$$\bigvee_{i,j} R(i) S_i(j) \lor \bigvee_{i,j} S_i(j) S_2(i,j) \lor \bigvee_{i,j} S_2(i,j) S_3(i,j) \lor \bigvee_{i,j} S_3(i,j) T(j)$$

$H_{30}, \ldots, H_{33}$ denote each of the four disjunctions above. Let $\Phi = R(3) S_1(3,7) \lor S_1(3,7) S_2(3,7)$. There are many partial substitutions $\theta$ for which $\Phi = \Psi[\theta]$; for example, $\theta$ 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 $\theta$ such that $\Phi = \Psi[\theta]$, we have:

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

Therefore, $\Phi$ 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 $\theta$ that produces $\Phi$ with either $R(3) = 0$, $S_2(3,7) = 1$, or $R(3) = 1$, $S_2(3,7) = 0$, or $R(3) = S_2(3,7) = 1$, and these lead to different residual formulas for $H_{30}$ and $H_{31}$ (namely 0 and $S_1(3,7)$, or $S_1(3,7)$ and 0, or $S_1(3,7)$ and $S_1(3,7)$).

In order to convert an FBDD $F$ for $\Psi = f(H_k)$ into a multi-output FBDD for $H_k = (H_{k0}, \ldots, H_{kk})$, we will try to modify it so that 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 $F_\theta$ for $H_k[\theta]$ 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 $H_k$ and $H_k$ might not have such small FBDDs is the tension between the $R(i) S_i(j)$ terms, which give a preference for reading entries in row-major order and the $S_i(j) T(j)$ terms, which suggest column-major order, together with the intermediate $S_i(j) S_{\ell+1}(i,j)$ 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 5.3. Let $\theta$ be a partial assignment to $\text{Var}(H_k)$.

- A transversal in $\theta$ is a pair of indices $(i,j)$ such that $R(i) S_i(j)$ is a prime implicant of $H_{k0}[\theta]$, $S_i(j) T(j)$ is a prime implicant of $H_{kk}[\theta]$, and $S_i(j) S_{\ell+1}(i,j)$ 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 $\theta$ such that $\Phi = \Psi[\theta]$ and $\theta$ has no transversals.

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

Lemma 5.4. Let $\theta$ be a partial assignment to $\text{Var}(H_k)$. If $\theta$ 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 $t = 0$ ($\theta$ has no transversals) then there exists a small OBDD that computes $H_k[\theta]$.

Let $G_\theta$ be the following undirected graph. The nodes are the variables $\text{Var}(H_k)$, and the edges are pairs of variables $(Z, Z')$ such that $Z' \lor Z'$ is a 2-prime implicant in $H_{k\ell}[\theta]$ for some $\ell$. Since $\theta$ has no transversals, all nodes $R(i)$ are disconnected from all nodes $T(j)$. In particular, there exists a partition $\text{Var}(H_k) = Z \cup Z''$ such that all $R(i)'s$ are in $Z'$, all $T(j)'s$ are in $Z''$ and every $H_{k\ell}[\theta]$ can written as $\varphi'_\ell \lor \varphi''_\ell$ where $\text{Var}(\varphi'_\ell) \subset Z'$ and $\text{Var}(\varphi''_\ell) \subset Z''$; in particular, $\varphi_0' = \varphi_0'' = \emptyset$. 

12
Define row-major order of the variables \( \text{Var}(H_k) = \{ T(1), \ldots, T(n) \} \) by:

\[
R(1), S_1(1, 1), \ldots, S_k(1, 1), S_1(1, 2), \ldots, S_k(1, n), \\
R(2), S_1(2, 1), \ldots, S_k(2, 1), S_1(2, 2), \ldots, S_k(2, n), \\
\vdots \\
R(n), S_1(n, 1), \ldots, S_k(n, 1), S_1(n, 2), \ldots, S_k(n, n)
\]

Let \( \pi' \) be the restriction of the row-major order to the variables in \( Z' \). Similarly, let \( \pi'' \) be the restriction to \( Z'' \) of the corresponding column-major order of the variables that omits the \( R(i) \)'s, and places the \( T(j) \) before all variables \( S_t(i, j) \). We build a multi-output OBDD using the order \( \pi = (\pi', \pi'') \) for \( (H_{k,0}, \ldots, H_{k,k}) \).

In the first part using order \( \pi' \) it will compute each \( \varphi'_t \) term in parallel in width \( O(2^k) \) and in the second part it will continue by including the additional terms from \( \varphi''_t \) using order \( \pi'' \). Observe that, except for the \( R(i)S_1(i, j) \) terms, each of the variables in the 2-prime implicants in \( \varphi'_t \) appear consecutively in \( \pi' \). 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 \( \varphi'_t \), remembered value of \( R(i) \), and remembered value of the immediately preceding queried variable. In the part using order \( \pi'' \), the remembered value of \( T(j) \) is used instead of the remembered value of \( R(i) \). The size of \( \mathcal{F}' \) is \( O(k2^kn^2) \) since there are \( kn^2 + 2n \) variables in total in \( \text{Var}(H_k) \).

For general \( t \), let \( I \) and \( J \) be the sets of rows and columns, respectively, of the transversals \((i,j)\) in \( \theta \). Since \( \theta \) 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 \( \theta \) has \( i \in I \). Notice that if we set all \( R(i) \) variables with \( i \in I \) in an assignment \( \theta' \) then the assignment \( \theta \cup \theta' \) has no transversals, and thus, by the above construction, \( H_k[\theta'] \) can be computed efficiently by a multi-output OBDD. Therefore, construct the FBDD which 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 \( \theta \) is that its existence ensures that each \( H_{k,t} \) is a non-trivial function of its remaining inputs; more transversals will in fact ensure that less about each \( H_{k,t} \) disappears. We will see the following: if there are at least some small number of independent transversals for \( \theta \) (three suffice), then we can use the fact that \( f \) depends on all inputs to ensure that \( \Psi[\theta] = f(H_k)[\theta] \) will be transparent provided one additional condition holds: there is no variable which we can set to kill off all transversals in \( \theta \) at once.

If we didn’t have this additional condition, then the construction of \( \mathcal{F}' \) for \( H_k \) would be simple: We would just use Lemma 5.4 at all nodes \( v \) of \( \mathcal{F} \) at which all assignments \( \theta \) for which \( \Phi_v = \Psi[\theta] \) do not have enough transversals to ensure transparency of \( \Phi_v \).

Failure of the additional condition is somewhat reminiscent of the situation with setting units in Section 4. This failure means that there is some variable we can set to kill off all transversals in \( \theta \) at once, which by Lemma 5.4 means that along the branch corresponding to that setting one can get an easy computation of \( H_k \) (not quite as simple as fixing the value of the formula to 1 by setting units as in Section 4 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 \( H_k \)-units.

Proposition 5.5. Let \( \Phi = \Psi[\theta] \) for some \( \theta \) with \( t \) independent transversals and \( \theta' = \theta \cup \{ W=b \} \) for \( b \in \{0, 1\} \).

The number of independent transversals in \( \theta' \) is in \( \{t-1, t\} \) if \( b = 0 \) and is in \( \{0, t-1, t\} \) if \( b = 1 \).

Definition 5.6. We say that a variable \( Z \) is an \( H_k \)-unit for the formula \( \Phi \) if \( \Phi[Z=1] \) is transversal-free but \( \Phi \) is not. We let \( U_k(\Phi) \) denote the set of \( H_k \)-units of \( \Phi \), and we say that \( \Phi \) is \( H_k \)-unit-free if \( U_k(\Phi) = \emptyset \).

In Section 5.1 we will prove the following, which makes our intuitive claim precise.

Lemma 5.7. Let \( \Psi = f(H_k) \) where \( f \) depends on all its inputs. Suppose that there exists a \( \theta \) with at least 3 independent transversals such that \( \Psi[\theta] = \Phi \). If \( \Phi \) is \( H_k \)-unit-free then \( \Phi \) is transparent.
We will still need to deal with the situation when $\Phi$ has any $H_k$-units along with multiple independent transversals. Our strategy will be simple: whenever we encounter an edge in $F$ on which an $H_k$-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 $H_k$-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 will be useful to understand how $H_k$-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 $H_k$-unit for $\Psi[\theta]$, because setting $Z = 1$ we ensure that $H_{k\ell}[\theta] \cup \{Z = 1\} = 1$, wiping out all transversals. The following lemma, which we prove in Section 5.1, shows a converse of this statement under the assumption that $\theta$ has at least 4 independent transversals.

**Lemma 5.8.** Let $\Psi = f(H_k)$ where $f$ depends on all its inputs. If $\Phi = \Psi[\theta]$ for some partial assignment $\theta$ that has at least 4 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 $H_k$ have 2-prime implicants rather than units on the terms involving $(i, j)$, Lemma 5.8 immediately implies the following:

**Corollary 5.9.** If $\Phi = \Psi[\theta]$ for some partial assignment $\theta$, then no $H_k$-unit of $\Phi$ is in the prime implicants indexed by any transversal of $\theta$.

Since the formulas in $H_k$ are monotone, by Lemma 5.8 and Proposition 4.8 if created by setting a variable to 1. Hence, if $\Phi$ has at least 4 independent transversals, then setting all $H_k$-units in $\Phi$ to 0 in turn yields a formula that still has at least 4 independent transversals (by Corollary 5.9) and is $H_k$-unit-free (by Lemma 5.8), and hence transparent (by Lemma 5.7 and Proposition 4.8).

We now describe the procedure for building a multi-output FBDD $F'$ computing $H_k$: Start with the FBDD $F$ for $\Psi$ and let $V$ and $E$ be, respectively, the vertices and edges of $F$. Let $V_4 \subseteq V$ be the set of nodes $v \in V$ such that $F_v = \Psi[\theta]$ for some assignment $\theta$ that has at least 4 independent transversals. By Proposition 5.5 $V_4$ is closed under predecessors (ancestors) in $F$; let $E_4$ be the set of edges in $F$ whose endpoints are both in $V_4$. The following is immediate from Proposition 5.5 and the definition of $V_4$.

**Proposition 5.10.** 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 $\theta$ with precisely 4 independent transversals such that $F_v = \Psi[\theta]$, 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 will apply a similar construction to that of Section 4.1 to the subgraph of $F$ 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 $H_k$-units created along edge $e$. There are two differences from the argument in Section 4.1: (1) we will only apply the construction to edges in $E_4$ and will build the rest of $F'$ independently of $F$, and (2) unlike setting ordinary units to 1, in which the corresponding FBDD edges simply point to the 1-sink, each setting of an $H_k$-unit to 1 only guarantees that the resulting formula is transversal-free; moreover the transversal-free formulas resulting from different settings may be different. The details are as follows (see Figure 2 in Subsection 5.1):

- 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 $F'$ will have label 1, which is the label that $e$ has in $F$; denote the variable tested at $u$ by $W$.
- The variable labeling each new vertex $(e, i)$ will be the $i$-th 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 out of each new vertex $(e, i)$ will lead to a distinct new node $(u, i)$ of $F'$. Since $(u, v) \in E_4$ there is some partial assignment $\theta$ such that $\Phi_u = \Psi[\theta]$, $\Phi_v = \Psi[\theta, W=1]$, and $\theta \cup \{W=1\}$ has at least 4 transversals; for definiteness we will pick...
the lexicographically first such assignment. Define the partial assignment
\[ \theta(u, i) = \theta \cup \{ W=1 \} \cup \{ U_k(\Phi_v)=0 \} \cup \{ Z_{c,1}=0, \ldots, Z_{c,i-1}=0, Z_{c,i}=1 \}, \]
to be the assignment that sets all \( H_k \)-units in \( \Phi_v \) to 0 along with the first \( i-1 \) of the \( H_k \)-units created by setting \( W \) to 1. The sub-dag of \( F' \) rooted at \( (u, i) \) will be the size \( O(k2^k n^2) \) FBDD for \( H_k(\theta(u, i)) \) constructed in [Lemma 5.4].

- For any node \( w \in V_4 \), whose 0-child is in \( V_1 \), such that \( w \) is labeled by a variable \( W \) that was an \( H_k \)-unit of \( \Phi_v \) 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 \( \theta \) be a partial assignment such that \( \Phi_v = \Psi[\theta] \) and \( \theta \) has precisely 4 independent transversals, as guaranteed by Proposition 5.10. Make \( v \) the root of the size \( O(k2^k n^2) \) FBDD for \( H_k(\theta') \) constructed in [Lemma 5.4] where \( \theta' = \theta \cup \{ U_k(\Phi_v) = 0 \} \).

- All other labeled edges of \( F \) between nodes of \( E_4 \) are included in \( F' \).

The fact that this is well-defined follows similarly to Proposition 4.9.

**Lemma 5.11.** \( F' \) as constructed above is a multi-output FBDD computing \( H_k \) that has size at most \( O(k2^k n^3) \) times the size of \( F \).

**Proof.** We first analyze the size of \( F' \): 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 the 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 \( F \). Also, the fact that \( F' \) is an FBDD follows similarly to the proof in [Lemma 4.11].

If \( \Phi_v \) is the function computed in \( F \) at node \( v \) for all \( v \in V_4 \), then we show by induction that for every partial assignment \( \theta' \) reaching \( v \) in \( F' \) \( \Psi[\theta'] = \Phi_v[U_k(\Phi_v)=0] \) and \( \theta' = \theta \cup \{ U_k(\Phi_v)=0 \} \) for some partial assignment \( \theta \) such that \( \Phi_v = \Psi[\theta] \). It is trivially true of the root. The argument is similar to that for [Lemma 4.11].

We now see why this is enough. Since \( v \in V_4 \), \( \Phi_v[U_k(\Phi_v)=0] \) is \( H_k \)-unit-free and has at least 4 transversals, and so it is transparent by Lemma 5.7. It remains to observe that (i) each multi-output FBDD attached directly to any node \( v \in V_4 \) used a restriction \( \theta \) of \( \Psi \) that would lead to that node in \( F' \), which, because \( \Psi[\theta] \) is transparent, implies that its leaves correctly compute the values of \( H_k \), and (ii) the same holds for the restriction leading to node \( (u, i) \) with parent \( \epsilon \), namely, the restriction used to build the multi-output FBDD consists of a restriction \( \theta \) that in \( F' \) would reach node \( u \in V_4 \) and for which \( \Psi[\theta] \) is transparent, together with the assignment \( \{ W=1 \} \cup \{ Z_{c,1}=0, \ldots, Z_{c,i-1}=0, Z_{c,i}=1 \} \) which follows the unique path from \( u \) to \( (u, i) \). Again this implies that its leaves correctly compute the values of \( H_k \).

### 5.1 Proofs of Lemmas 5.7 and 5.8

All formulas in \( H_k \) 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 \( P_{0, i, j} = R(i)S_{\ell}(i, j), P_{k, i, j} = S_{\ell}(i, j)T(j) \) 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 5.12.** Let \( \Psi = f(H_k) \) for some function \( f \) that depends on all its inputs. Suppose \( \theta \) 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 \( \theta \) of both transversals, the neighboring prime implicants of the prime implicant, \( P_{\ell, i, j} \) of \( H_\ell \) are either unassigned or set to 0 by \( \theta \). Then there exists a partial assignment \( \mu \) to all the variables of \( \text{Var}(\Psi[\theta]) \), 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 \( \Psi[\theta] \cup \mu = f(P_{\ell, i, j}[\theta]) \). Moreover, the choice of \( \mu \) depends on \( \Psi[\theta] \) (as well as the indices \( \ell, i, i_0, i_1, j_0, j_1 \)) but not on any other aspect of \( \theta \).
Figure 2: Given an FBDD $\mathcal{F}$ for $\Psi = f(H_k)$ in (a), apply the conversion to produce $\mathcal{F}'$ for $H_k$ as in (b), with detail for unit propagation in (c) in case that setting $W = 1$ produces new $H_k$-units.

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 $\ell$ there exists an assignment $\mu_\ell : X - \{X_\ell\} \rightarrow \{0,1\}$, such that $f_\ell(X_\ell) \equiv f[\mu_\ell]$ is a function that depends on $X_\ell$: i.e., $f_\ell(X) = X_\ell$ or $f_\ell(X) = \neg X_\ell$.

We define assignment $\mu$ so that it sets the remaining variables to force $H_{km}$ to equal $\mu_\ell(X_m)$ for all $m \neq \ell$ and force $H_{kl}$ to equal $P_{\ell,i,j}[\theta]$. In order to force some $H_{km}$ to $1$, $\mu$ 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 $\text{Ones}_\ell = \{m \mid \mu_\ell(X_m) = 1\}$ and order the elements of $\text{Ones}_\ell$ as $m_1 < m_2 < \cdots$, and define $\text{Ones}_\ell^b = \{m_r \mid b = r \mod 2\}$ for $b \in \{0,1\}$. For $b \in \{0,1\}$, define $\mu$ to set the variables of the $(i_b,j_b)$ prime implicant of $H_{km}$ to $1$ for every $m \in \text{Ones}_\ell^b$. This will force $H_{km}[\theta \cup \mu] = 1$, for all $m \in \text{Ones}_\ell$. Let $\mu$ 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 $\mu$ ensures that
for $b \in \{0, 1\}$, if the $(i_b, j_b)$ prime implicant of $H_{km}$ is set to 1 by $\mu$, then its neighboring prime implicants are forced to 0 by $\mu$. In fact, $H_{km}[\theta \cup \mu] = 0$ for all $m \notin One_{k} \cup \ell$ since each neighboring prime implicant to $P_{i,j}[\theta]$ will have one variable set as in $\theta$ and the other set to 0 and all other prime implicants are either set to 0 by $\theta$ or by $\mu$. Finally, the same property is true of every prime implicant of $H_{kl}$ except for $P_{i,j}[\theta]$. It remains to observe that $\Psi[\theta \cup \mu] = f(H_{k}[\theta \cup \mu]) = f[\mu_{\ell}(P_{i,j}[\theta])] = f_{\ell}(P_{i,j}[\theta])$ and $\mu$ only depended on $\theta$ through the value of $\Psi[\theta]$, as required.

Notice that the lemma fails if $\theta$ has only 1 transversal:

**Example 5.13.** For a counterexample, consider $f(X_{01}, X_{11}, X_{2}) = X_{0}X_{2} \lor X_{1}$. Suppose that $\theta$ sets all variables in $Z$ 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, $\theta$, has the transversal $(3, 7)$. However, $\Psi[\theta]$ is:

$$f(H_{03}[\theta], H_{13}[\theta], H_{32}[\theta], H_{33}[\theta]) = f(R(3)S_{1}(3, 7), S_{1}(3, 7)S_{2}(3, 7), S_{2}(3, 7)T(7)) = R(3)S_{1}(3, 7)S_{2}(3, 7)T(7) \lor S_{1}(3, 7)S_{2}(3, 7) = S_{1}(3, 7)S_{2}(3, 7)$$

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

We immediately obtain the following two corollaries:

**Corollary 5.14.** If $\theta$ 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, all $P_{i,j}[\theta]$ are 2-prime implicants in their respective $H_{kl}[\theta]$.

**Corollary 5.15.** If $\theta$ has at least 3 independent transversals then, every partial assignment $\theta'$ such that $\Psi[\theta] = \Psi[\theta']$ has the same set of transversals as $\theta$.

**Proof.** We prove that every transversal of $\theta$ is a transversal of $\theta'$: this implies that $\theta'$ has at least 3 independent transversals, and therefore the converse holds too (every transversal of $\theta$ is a transversal of $\theta$). Let $\Phi = \Psi[\theta] = \Psi[\theta']$. Let $(i, j)$ be a transversal for $\theta$. Since $\theta$ has at least 3 transversals, by Corollary 5.14, $\Phi$ depends on all variables of the transversal $(i, j)$. It follows that $\theta'$ cannot set any of these variables. For $\ell \in [k - 1]$, the 2-prime implicants within each $H_{kl}$ 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 $H_{kl}[\theta']$ contains the 2-prime implicant $S_{k}(i, j)S_{k+1}(i, j)$.

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 $\theta'$. 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 $\theta$ 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 $\Phi$ of $\Psi$ has $k$ independent transversals if there exists an assignment $\theta$ so that $\Phi = \Psi[\theta]$ and $\theta$ has $k$ independent transversals; by the above, this is equivalent to saying that all $\theta$ with $\Phi = \Psi[\theta]$ have $k$ independent transversals.
Proof of Lemma 5.7

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

Suppose the contrary: then there exist two partial assignments \( \theta, \theta' \) such that \( \Phi = \Psi[\theta] = \Psi[\theta'] \) and \( \Phi \) has at least 3 independent traversals, but for some \( \ell \), \( H_{k\ell}[\theta] \neq H_{k\ell}[\theta'] \). Observe that if any \( H_{k\ell}[\theta] \) or \( H_{k\ell}[\theta'] \) were the constant 1, then \( \Phi \) would be transversal-free contradicting the fact that it has 3 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 traversals, contradicting the assumption that \( \Phi \) is \( H \)-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, \( \Phi \) implies that all variables of all prime implicants are in \( var(\Phi) \).

Assume w.l.o.g. that \( P_{\ell,i,j} \) is a 2-prime implicant of \( H_{k\ell}[\theta] \) that is not a prime implicant of \( H_{k\ell}[\theta'] \); i.e., \( P_{\ell,i,j}[\theta] = P_{\ell,i,j}[\theta'] = 0 \). Let \((i_0, j_0)\) and \((i_1, j_1)\) be two independent transversals for \( \theta \) (which are also transversals for \( \theta' \) by Corollary 5.15) that are also independent of \((i, j)\). Since in both \( \theta \) and \( \theta' \), the neighboring implicants of \( P_{\ell,i,j} \) either remain as 2-prime implicants or are set to 0 in \( \theta \) and \( \theta' \) (though not necessarily the same in both), we can apply Lemma 5.12 to both \( \theta \) and \( \theta' \) to obtain \( \mu \) and \( \mu' \). By the conclusion of Lemma 5.12, \( \Phi[\mu] = \Psi[\theta \cup \mu] = f(P_{\ell,i,j}) \) which is either \( P_{\ell,i,j} \) or \( \neg P_{\ell,i,j} \) but \( \Phi[\mu'] = \Psi[\theta \cup \mu'] = f_i(0) \) which is neither of the two. However, since the assignments \( \mu \) and \( \mu' \) depended only on the indices involved and \( \Phi \), we conclude that \( \mu = \mu' \) which is a contradiction.

The requirement that \( \Phi \) be unit-free is necessary for Lemma 5.7 to hold: a simple example is given by the formula \( \Phi' = S_1(3, 7) \) in Example 5.2 which is not transparent. The formula remains non-transparent even if we expand it with three independent transversals, e.g. \( S_1(3, 7) \lor [R(4) S_1(4, 4) \lor \ldots \lor S_3(4, 4) T(4)] \lor [R(5) S_1(5, 5) \lor \ldots] \lor [R(6) S_1(6, 6) \lor \ldots] \), for the same reasons given in the example.

Proof of Lemma 5.8

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

If \( Z \in U(H_{k\ell}[\theta]) \) for some \( \ell \), 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 \( Z \notin \bigcup_{\ell \in \{1, \ldots, k\}} U(H_{k\ell}[\theta]) \). 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 \( \geq 4 \) independent transversals in \( \theta \), so \( \theta \cup \{Z = 1\} \) has at least 3 (independent) transversals. Therefore, by Corollary 5.15 every partial assignment \( \theta' \) such that \( \Psi[\theta'] = \Phi[Z = 1] = \Psi[\theta \cup \{Z = 1\}] \) has at least 3 independent transversals. This implies that \( \Phi[Z = 1] \) is not transversal-free and hence \( Z \notin U_k(\Phi) \).

6 A Dichotomy Theorem for Efficient Propositional Model Counting

In this section we present 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.

To define the class of queries, we consider the \( k + 1 \) queries in \( b_k \) defined in Section 3 and the following additional \( k + 2 \) queries \( b_k \): \( b_0 = \exists u_0 R(u_0) \), \( b_k = \exists u_k \forall v_k S_i(u_k, v_k), \forall \ell \in [k] \), \( b_{k+1} = \exists v_{k+1} T(v_{k+1}) \). These queries have the following lineages on the same domain of size \( n \): \( B_0 = \bigvee_{i \in [n]} R(i) \), \( B_{k} = \bigvee_{i,j \in [n]} S_i(j) \), \( \forall \ell \in [k] \), and \( B_{k+1} = \bigvee_{j \in [n]} T(j) \) respectively. Note that we consider data complexity, so we assume that the query is fixed (i.e., \( k \) is a constant).
Theorem 6.1. (a) If \( g(\mathbf{X}, 1) \) depends on all \( k+1 \) variables \( X_0, \ldots, X_k \), then any DLDD for the lineage of \( g(\mathbf{X}, 1) \) is exponential in size.

(b) Otherwise, there exists an FBDD for the lineage of \( g(\mathbf{X}, 1) \) of size \( n^{O(1)} \), and the FBDD can be constructed in \( n^{O(1)} \) time.

The first part of Theorem 6.1 extends Theorem 3.3 where \( f(\mathbf{X}) = g(\mathbf{X}, 1) \).

Example 6.2. We illustrate with three examples.

- \( g = (X_0 \lor B_2) \land (B_0 \lor X_1) \) where \( k = 1 \). Then \( g(\mathbf{X}, 1) = 1 \): it does not depend on \( X_0, X_1 \), and therefore, the lineage has a poly-size FBDD.

- \( g = X_0 \land (X_1 \lor B_3) \land (X_1 \lor B_5) \land (X_2 \lor X_3 \lor X_4 \lor X_5) \) where \( k = 5 \). Then \( g(\mathbf{X}, 1) = X_0 \land (X_2 \lor X_3 \lor X_4 \lor X_5) \): it does not depend on \( X_1 \), and the lineage has a poly-size FBDD.

- \( g = (X_0 \lor X_1) \land (X_1 \lor B_3) \land (X_2 \lor X_3) \) where \( k = 3 \). Then \( g(\mathbf{X}, 1) = (X_0 \lor X_1) \land (X_2 \lor X_3) \): it depends on all of \( X_0, \ldots, X_3 \), and therefore every DLDD for the lineage has exponential size.

In prior work \[16\] a sufficient condition was given 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 6.1.

Proof. Proof of (a)

Suppose \( f(X_0, \ldots, X_k) = g(\mathbf{X}, 1) = g(X_0, \ldots, X_k, 1, \ldots, 1) \) depends on all variables \( X_0, \ldots, X_k \). Let \( F \) 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 \( F \) to an FBDD \( F' \) for query \( Q' = f(h_{k0}, \ldots, h_{kk}) \) over the domain of size \( n \); further \( F \) and \( F' \) will have the same size. By Theorem 3.3, the size of \( F \) is \( 2^{\Omega(n)} \); therefore the size of \( F' \) is \( 2^{\Omega(n)} = 2^{\Omega(n')} \).

To convert \( F \) to \( F' \), modify \( F \) by setting the following value\[9\] where \( n_1 = n + 1 \) and \( n_2 = n + 2 \):

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

Note that the modified FBDD \( F' \) 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 \( n \).

---

8 This is \( Q_U \) in \[16\] for which a poly-size FBDD was shown.

9 This means: 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.
Proof of (b)

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 5.3) and therefore it has a shared OBDD \( F_s \) of size \( O(n) \) for formulas \( h_{k0}, \ldots, h_{kk} \) (see the proof of Lemma 5.4 which constructs the FBDD in poly-time for a fixed \( k \)).

Further, if for any \( \ell \), \( b_\ell = 0 \), 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 \( F_{0,\ell} \) 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 \( F \) 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 \( F \) 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 \( F_s \) or \( F_{0,\ell} \), \( \ell \in [k] \) described above.

The layers of \( F \) 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 \( \ell \), \( 1 \leq \ell \leq k \): Test for \( b_\ell \) Each sub-tree in the layer \( \ell \) 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 point to a unique FBDD \( (F_s \) or \( F_{0,\ell} \) for some \( \ell \in [k] \)) in the last layer.

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 \( b_0, \ldots, b_k \), i.e. for each of them we know if it is 0 or 1, and (ii) we know which variables have been tested. In the FBDDs \( F_s \) or \( F_{0,\ell} \) 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 \( F \) (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 \( \ell \), \( b_\ell = 0 \), then both \( h_{\ell-1} = h_\ell = 0 \). Consider the least \( \ell \) such that \( h_\ell = 0 \) and connect this edge to the shared FBDD \( F_{0,\ell} \) substituting the values of the variables that have already been tested.

(2) If for all \( \ell \), \( b_\ell = 1 \), then connect the edge to the shared FBDD \( F_s \) 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 \( F \) (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 \( F \) is \( O(n) \times [O(n^2)]^k \times O(n) \times O(n) = n^{O(1)} \). The completes the proof of part (b) of the theorem.
7 A Quasi-polynomial Conversion from DLDDs to FBDDs

Here we prove Theorem 2.1 that any DLDD $D$ with $N$ nodes can be converted into an equivalent FBDD $F$ computing the same formula as $D$, where $F$ has at most $N^{2 \log^{*} N}$ nodes. A DLDD can have these types of nodes: (i) decision-nodes and (ii) 0- and 1-sinks as in FBDDs; (iii) NOT-nodes having a single child; decomposable (iv) AND-, (v) OR-, (vi) XOR-, and (vii) EQUIV-nodes $u$ with two children $u_1, u_2$ such that the sub-DAGs $D_{u_1}$ and $D_{u_2}$ do not share any common Boolean variables. Recall that $\Phi_u$ denotes the (vector) subformula of the sub-DAG of $D$ at a node $u \in D$; $D$ finally computes the formula $\Phi = \Phi_r$ where $r$ is the root of $D$.

**Step 1: DLDD $D$ for $\Phi$ to DLDD $O$ for $\neg \Phi$.** As a first step, we convert the DLDD $D$ into another DAG $O$ such that $O$ has the same DAG structure as $D$ but the types of the nodes are changed as follows:

1. A 1-sink in $D$ becomes a 0-sink in $O$ and vice versa.
2. An AND-node in $D$ becomes an OR-node in $O$ and vice versa.
3. An XOR-node in $D$ becomes an EQUIV-node in $O$ and vice versa.
4. Decision nodes and NOT-nodes remain unchanged.

For any node $u \in D$, we will denote the corresponding node in $O$ by $u'$, the subformula at $u$ by $\Phi_u$ (as before), and the subformula at $u' \in O$ by $\Psi_u$. The following proposition shows that $O$ computes $\neg \Phi$.

**Proposition 7.1.** For all nodes $u \in D$ and the corresponding $u' \in O$, $\Psi_u = \neg \Phi_u$.

**Proof.** We prove the claim by an induction on a reverse topological order of the nodes $u$ in $D$ (a node is processed only after all its descendants are processed) that $\Psi_u = \neg \Phi_u$. The hypothesis holds for the 0-sinks and 1-sinks by construction. Suppose the hypothesis holds up to some step and consider the next node $u$ in the reverse topological order.

(a) If $u'$ is a NOT-node in $O$ ($u$ is a NOT-node in $D$), $\Psi_u = \neg \Psi_{u_1} = \neg (\neg \Phi_{u_1})$ (by induction hypothesis)

(b) If $u'$ is a decomposable OR-node in $O$ ($u$ is a decomposable AND-node in $D$) with children $u_1, u_2$, then $\Psi_u = \Psi_{u_1} \vee \Psi_{u_2} = (\neg \Phi_{u_1}) \vee (\neg \Phi_{u_2})$ (by induction hypothesis) $= \neg \Phi_{u_1} \wedge \Phi_{u_2}$.

(c) If $u'$ is a decomposable AND-node in $O$ and $u$ is a decomposable OR-node in $D$ is similar to the above case.

(d) If $u'$ is a decomposable XOR-node in $O$ ($u$ is a decomposable EQUIV-node in $D$) with children $u_1, u_2$, then $\Psi_u = \Psi_{u_1} \cdot \neg \Psi_{u_2} \vee \neg \Psi_{u_1} \cdot \Psi_{u_2} = (\neg \Phi_{u_1}) \cdot \adult(\neg \Phi_{u_2}) \vee (\neg \Phi_{u_1}) \cdot \Phi_{u_2}$ (by induction hypothesis)

(e) If $u'$ is a decomposable EQUIV-node in $O$ and $u$ is a decomposable XOR-node in $D$ is similar to the above case.

(f) If $u'$ is a decision-node in $O$ ($u$ is also a decision-node in $D$) with two children $u_0, u_1$ on 0- and 1-branch respectively, then $\Psi_u = (\neg x)\Psi_{u_0} \vee x\Psi_{u_1} = (\neg x)(\neg \Phi_{u_0}) \vee x(\neg \Phi_{u_1})$ (by induction hypothesis)

$= (\neg x)(\neg \Phi_{x = 0}) \vee x(\neg \Phi_{x = 1}) = \neg \Phi_u$.

\[
\square
\]

**Step 2: Combining DLDDs $D$ and $O$ into a DLDD $P$ that does not have any NOT-node $P$** has the union of nodes of $D$ and $O$. If $u \in D$ is not a NOT-node (i.e., $u' \in O$ is also not a NOT-node), then the connections to the child or children of $u$ and $u'$ are retained in $P$. Let $u \in D$ be a NOT-node with child
FBDDs. (which must be decision-nodes) are unchanged and they ensure that the base case holds. 

When we process \( u \) nodes, so that they can be removed altogether later. We process the nodes in reverse topological order.

This conversion increases the number of nodes by a factor of 2. For any node \( u \in \mathcal{P} \), let \( \Theta_u \) be the subformula at \( u \).

**Proposition 7.2.** \( \Theta_u = \Phi_u \) if \( u \in \mathcal{D} \), and \( \Theta_u = \Psi_u = \neg \Phi_u \) for the corresponding \( u' \in \mathcal{O} \).

**Proof.** We prove this by induction on the nodes in \( \mathcal{D} \) in reverse topological order. The base case holds for the 0- and 1-sinks. The hypothesis holds by construction for any decision-node, or decomposable AND-, OR-XOR-, or EQUIV-node \( u \in \mathcal{P} \). Let \( u \in \mathcal{D} \) be a NOT node which is a \texttt{no-op} node in \( \mathcal{P} \). Then \( \Theta_u = \Psi_u = \neg \Phi_u = \Phi_u \). Similarly \( \Theta_u = \Psi_u \) if \( u \in \mathcal{O} \) is a NOT-node.

**Step 3:** DAG \( \mathcal{P} \) with decomposable-AND and -OR nodes to FBDD \( \mathcal{F} \) with quasipolynomial blow-up. This conversion is similar to the one in [3] where we showed that a decision-DNNF can be converted to an equivalent FBDD with at most quasipolynomial increase in size. The DAG \( \mathcal{P} \) can have additional decomposable-OR nodes in addition to decision-nodes and decomposable-AND nodes in a decision-DNNF. The same construction works here except the correctness proof that both the DAGs \( \mathcal{P} \) and \( \mathcal{F} \) compute the same formula. Evaluating the function on a given assignment in \( \mathcal{P} \) is different than a decision-DNNF. We need to check that

- both branches of a decomposable AND-node evaluate to 1;
- at least one branch of a decomposable OR-node evaluates to 1;
- exactly one branch of a decomposable XOR-node evaluates to 1; and
- either both or none of the branches of a decomposable EQUIV-node evaluates to 1.

Here we give a simple construction and proof of correctness.

For any decomposable (AND-, OR-, XOR-, or EQUIV-) node \( u \in \mathcal{P} \) with two children \( u_1 \) and \( u_2 \), wlog. let the sub-DAG \( \mathcal{F}_{u_1} \) has fewer number of nodes (lighter sub-DAG) than \( \mathcal{P}_{u_2} \) (heavier sub-DAG). For any such node \( u \), recursively (top-down) create a private copy of the lighter sub-DAG \( L_u = \mathcal{P}_{u_1} \) such that any path from the root of \( \mathcal{P} \) to any node \( v \in L_u \) goes through \( u \). It is not hard to see that this replication leads to at most a quasi-polynomial increase in size (see [3] for a detailed analysis). Let this intermediate DAG be \( \mathcal{P}' \).

Then we reconnect some of the edges in \( \mathcal{P}' \) to replace the decomposable-AND and -OR nodes by \texttt{no-op} nodes, so that they can be removed altogether later. We process the nodes in reverse topological order. When we process \( u \), we maintain the invariant that the sub-DAG at \( u \) is an FBDD, i.e., does not have any decomposable nodes but they still compute the same function as in \( \mathcal{P} \) or \( \mathcal{P}' \). The sinks and their parents (which must be decision-nodes) are unchanged and they ensure that the base case holds.

Suppose we are at a decomposable node \( u \), its sub-DAGs \( \mathcal{P}_{u_1}' \) and \( \mathcal{P}_{u_2}' \) have been already changed to FBDDs \( \mathcal{F}_{u_1} \) and \( \mathcal{F}_{u_2} \) respectively. Further, \( \mathcal{F}_{u_1} \) is a private copy for \( u \).

- If \( u \) is an AND-node (in \( \mathcal{D} \) or \( \mathcal{O} \)), then we add all 1-sinks of \( \mathcal{F}_{u_1} \) to the root of \( \mathcal{F}_{u_2} \) (note that the root of \( \mathcal{F}_{u_2} \) is a copy of the original node \( u_2 \in \mathcal{P} \)) to simulate the AND-function.
- If \( u \) is an OR node (in \( \mathcal{D} \) or \( \mathcal{O} \)), we connect the 0-sinks in of \( \mathcal{F}_{u_1} \) to the root of \( \mathcal{F}_{u_2} \) to simulate the OR-function.
• Suppose $u$ is an XOR-node in $D$ with light child $u_1$ and heavy child $u_2$. Then there is an EQUIV-node $u' \in \mathcal{O}$ with light child $u'_1$ and heavy child $u'_2$. Therefore, the sub-DAGs, $P_{u_1}'$ and $P_{u_1}'$ are private sub-DAGs of $u$ and $u'$.

To simulate the XOR-function $\Phi_u = \Phi_{u_1} \cdot \neg \Phi_{u_2} \lor (\neg \Phi_{u_1}) \cdot \Phi_{u_2}$, we (i) connect all the 1-sinks of $F_{u_1}'$ (private) to $F_{u_2}'$ (shared child of $u'$), (ii) delete the edge $(u', u_2')$, (iii) connect all the 0-sinks of $P_{u_1}'$ (private) to $P_{u_2}'$ (shared), and (iv) delete the edge $(u, u_2)$. Since $F_{u_1}'$ computes $\Psi_{u_1} = \neg \Phi_{u_1}$ and $F_{u_1}'$ computes $\Psi_{u_2} = \neg \Phi_{u_2}$, these connections correctly simulate XOR-function at $u$.

Similarly, to simulate the EQUIV-function at $u'$, $\Psi_u = \Psi_{u_1} \cdot \Psi_{u_2} \lor (\neg \Psi_{u_1}) \cdot (\neg \Psi_{u_2})$, we (i) connect all the 1-sinks of $F_{u_1}'$ (private) to $F_{u_2}'$ (shared), and (ii) connect all the 0-sinks of $F_{u_1}'$ (private) to $F_{u_2}'$ (shared child of $u$).

• The case when $u$ is an EQUIV-node and $u'$ is an XOR-node is similar to the above case.

**Proposition 7.3.** The sub-DAG $F_u$ for every node $u$ (and in particular the full DAG at the root) created by the above construction is an FBDD.

**Proof.** Since the sub-DAGs at $u_1$ and $u_2$ (in $D, \mathcal{O}, P$ and $P'$) did not share any variable, every variable is still tested at most once along any path in the resulting DAG $F_u$. To see that the DAG structure is retained and no cycles are created, orient the edges from $u$ to lighter and heavier sub-DAGs such that they are the left and right child respectively. Also place $u' \in \mathcal{O}$ and its children $u_1', u_2'$ in $F$ such that both $u, u'$ appear at left of both $u_2, u_2'$. Then the original connections in $P'$ are top-down, and the new connections are from left to right, which cannot create a cycle. \qed

Therefore, after the conversion, we get an FBDD $F$ with at most quasi-polynomial increase in size and computing the same formula as the original DLDD $D$.

### 8 Discussion

In this paper 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 implied by those proposition methods even for queries that can be evaluated in polynomial time. We also introduced DLDDs, which generalize decision-DNNFs while retaining their good algorithmic properties for model counting. Though our query lower bounds apply equally to their DLDD representations, DLDDs may prove to be better than decision-DNNFs in other scenarios.

In light of our lower bounds, 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 paper 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 [6] which 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?
References

[1] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for \#sat and bayesian inference. In FOCS, pages 340–351, 2003.

[2] R. J. Bayardo, Jr., and J. D. Pehoushek. Counting models using connected components. In AAAI, pages 157–162, 2000.

[3] P. Beame, J. Li, S. Roy, and D. Suciu. Lower bounds for exact model counting and applications in probabilistic databases. In UAI, 2013.

[4] B. Bollig and I. Wegener. A very simple function that requires exponential size read-once branching programs. Inf. Process. Lett., 66(2):53–57, Apr. 1998.

[5] N. N. Dalvi, K. Schnaitter, and D. Suciu. Computing query probability with incidence algebras. In PODS, pages 203–214, 2010.

[6] N. N. Dalvi and D. Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. J. ACM, 59(6):30, 2012.

[7] A. Darwiche. Decomposable negation normal form. J. ACM, 48(4):608–647, 2001.

[8] A. Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. Journal of Applied Non-Classical Logics, 11(1-2):11–34, 2001.

[9] A. Darwiche and P. Marquis. A knowledge compilation map. J. Artif. Int. Res., 17(1):229–264, Sept. 2002.

[10] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. Commun. ACM, 5(7):394–397, 1962.

[11] M. Davis and H. Putnam. A computing procedure for quantification theory. J. ACM, 7(3):201–215, 1960.

[12] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In Handbook of Satisfiability, pages 633–654. IOS Press, 2009.

[13] J. Huang and A. Darwiche. Dpll with a trace: From sat to knowledge compilation. In IJCAI, pages 156–162, 2005.

[14] J. Huang and A. Darwiche. The language of search. JAIR, 29:191–219, 2007.

[15] M. Jaeger and G. V. den Broeck. Liftability of probabilistic inference: Upper and lower bounds. In Proceedings of the 2nd International Workshop on Statistical Relational AI, 2012.

[16] A. K. Jha and D. Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. Theory Comput. Syst., 52(3):403–440, 2013.

[17] S. M. Majercik and M. L. Littman. Using caching to solve larger probabilistic planning problems. In AAAI, pages 954–959, 1998.

[18] C. Muise, S. A. McIlraith, J. C. Beck, and E. I. Hsu. Dsharp: fast d-dnnf compilation with sharpsat. In Canadian AI, pages 356–361, 2012.

[19] A. Sabharwal. Symchaff: exploiting symmetry in a structure-aware satisfiability solver. Constraints, 14(4):478–505, 2009.

[20] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In SAT, 2004.
[21] R. P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1997.

[22] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[23] M. Thurley. sharpsat: counting models with advanced component caching and implicit bcp. In *SAT*, pages 424–429, 2006.

[24] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 1979.

[25] I. Wegener. *Branching programs and binary decision diagrams: theory and applications*. SIAM, Philadelphia, PA, USA, 2000.
Appendix

Omitted Proofs from Section 4

Here we present proofs for Lemma 4.4 in the case where \( k \) is even, including when \( k = 0 \). For any partial path \( P \) through an FBDD for \( H_k \), let \( \text{Row}(P), \text{Col}(P) \), and \( P \) be the same entities as before. The techniques required for the case when \( k = 0 \) are different, so we will present it separately from the case when \( k \) is even and nonzero. In either case, it will suffice to come up with a definition of admissible paths in \( P \) so that

(1) Lemma 4.7 holds for all admissible paths, and

(2) for all \( i, j \), the first decision node \( u \) at which a admissible path tests a variable of the form \( R(i), S(i, j) \)

or \( T(j) \) is unforced, using terminology introduced in the proof of Lemma 4.4.

Then it is readily checked that using the same proof structure as given in Section 4 will yield a complete proof.

We will first prove hardness for \( H_0 \). Our definition of a admissible path is as follows:

**Definition .1.** A path \( P \in P \) in a FBDD computing \( H_0 \) is forbidden if there exist \( i, j \) so that \( R(i) = S(i, j) = 0 \) or \( S(i, j) = T(j) = 0 \) along the path. It is admissible otherwise.

That this definition follows property (2) is easily verified. We now demonstrate property (1). This will complete the proof for \( k = 0 \).

**Lemma .2.** Lemma 4.7 holds when \( k = 0 \) with this definition of admissible paths.

**Proof.** Suppose \( P_1, P_2 \) are two distinct admissible paths in \( P \) with \( H_0|P_1| = H_0|P_2| = F \). Let \( u \) be the first node (it must be a decision node) where \( P_1 \) and \( P_2 \) diverge, then by Lemma 4.5 it must be of the form \( S(i, j) \). WLOG assume \( P_1 \) sets \( S(i, j) = 0 \) and \( P_2 \) sets \( S(i, j) = 1 \). Then the 3-prime implicant \( R(i)S(i, j)T(j) \) cannot appear in \( F \), nor can \( R(i)T(j) \) and there are no units at all, so in particular \( P_2 \) must set either \( R(i) = 0 \) or \( T(j) = 0 \) which implies that \( P_1 \) is forbidden.

We will now prove hardness for \( H_k \) for \( k = 2m \) even and nonzero. Here our definition of a admissible path is as follows:

**Definition .3.** A partial path \( P \in P \) in a FBDD computing \( H_k \) for \( k \) even and nonzero is admissible if for all \( i, j \) it is consistent with one of the four following assignments:

1. \( R(i) = 1, T(j) = 0 \) and \( S_l(i, j) = 0 \) for all \( \ell \) odd and \( S_l(i, j) = 1 \) for all \( \ell \) even
2. \( R(i) = 0, T(j) = 1 \) and \( S_l(i, j) = 1 \) for all \( \ell \) odd and \( S_l(i, j) = 0 \) for all \( \ell \) even
3. \( R(i) = T(j) = 0 \) and \( S_l(i, j) = 1 \) for all \( l \) odd and \( S_l(i, j) = 0 \) for all \( l \) even
4. \( R(i) = T(j) = 1 \) and \( S_l(i, j) = S_l(i, j) = 0 \) for \( l \) even and \( S_l(i, j) = 1 \) for \( \ell > 1 \) odd.

\( P \) is forbidden if it is not admissible.

Again that this definition follows property (2) is easily verified, and it suffices to demonstrate property (1).

**Lemma .4.** Lemma 4.7 holds when \( k > 0 \) and even with this definition of admissible paths.

**Proof.** Suppose \( P_1, P_2 \) are two distinct admissible paths in \( P \) with \( H_k|P_1| = H_k|P_2| = F \). Let \( u \) be the first (decision) node where \( P_1 \) and \( P_2 \) diverge. Then by Lemma 4.5 it must be of the form \( S_l(i, j) \) for some \( l \). WLOG assume \( P_1 \) sets \( S_l(i, j) = 0 \) and \( P_2 \) sets \( S_l(i, j) = 1 \). First consider the case when \( \ell = 1 \). Then since the FBDD follows the unit clause rule we conclude that \( P_2 \) and hence \( P_1 \) sets \( R(i) = 0 \) which implies \( P_1 \) is forbidden, which is a contradiction. Hence \( \ell \neq 1 \).

26
Now let \( l > 1 \). Then the prime implicant \( S_{\ell-1}(i,j)S_\ell(i,j) \) is missing from \( H_k[P_1] = F \); as no 1-prime implicants can exist, this implies that \( S_{\ell-1}(i,j) = 0 \) in \( P_2 \). As long as \( \ell - 1 > 1 \), by the definition of admissible paths we have \( P_1 \) cannot set \( S_{\ell-1}(i,j) = 0 \) so reversing roles and inducting downwards we conclude that either \( P_1 \) sets \( S_2(i,j) = 0 \) and \( P_2 \) cannot set \( S_2(i,j) \) to 0, or vice versa. WLOG assume that the former case occurs. This implies that \( P_2 \) must set \( S_1(i,j) = 0 \), and by the same logic as above, this implies \( P_1 \) cannot set \( R(i) \) to 0 and sets \( S_1(i,j) = S_2(i,j) = 0 \). Therefore \( P_1 \) can potentially be an admissible path only by case 4 above. Then going upward from \( S_\ell(i,j) \) to \( T(j) \), assuming \( \ell > 1 \) is odd (and \( k \) is even), \( S_k(i,j) = 0 \) on \( P_2 \) and \( S_{k-1}(i,j) = 0 \) on \( P_1 \). Therefore, \( P_1 \) cannot set \( S_k(i,j) \) to 0 and must set \( T(j) \) to 0, which violates the condition in case 4. Now consider an even \( \ell > 1 \). If \( k = 2 \), the only scenario is \( \ell = k = 2 \). Then \( S_k(i,j) = 0 \) on \( P_1 \), and \( S_k(i,j) = 1 \) on \( P_2 \) by assumption, therefore \( T(j) \) must be 0 on \( P_2 \) and also on \( P_1 \), which violates case 4. If \( k > 2 \) and \( \ell \) is even, \( S_k(i,j) = 0 \) on \( P_1 \), and \( S_{k-1}(i,j) = 0 \) on \( P_2 \), and since \( k > 2 \), \( S_k(i,j) \) cannot be set to 0 on \( P_2 \), so \( T(j) = 0 \) on \( P_2 \), and therefore on \( P_1 \), which violates case 4. \( \Box \)