Knowledge Compilation Meets Database Theory : Compiling Queries to Decision Diagrams*

Abhay Jha
Computer Science and Engineering
University of Washington, WA, USA
abhaykj@cs.washington.edu

Dan Suciu
Computer Science and Engineering
University of Washington, WA, USA
suciu@cs.washington.edu

ABSTRACT

The goal of *Knowledge Compilation* is to represent a Boolean expression in a format in which it can answer a range of *online-queries* in PTIME. The online-query of main interest to us is *model counting*, because of its application to query evaluation on probabilistic databases, but other online-queries can be supported as well such as testing for equivalence, testing for implication, etc. In this paper we study the following problem. Given a database query q, decide whether its lineage can be compiled efficiently into a given target language. We consider four target languages, of strictly increasing expressive power(when the size of compilation is constrained to be polynomial in the input size): Read-Once Boolean formulae, OBDD, FBDD and d-DNNF. For each target, we study the class of database queries that admit polynomial size representation: these queries can also be evaluated in PTIME over probabilistic databases. When queries are restricted to conjunctive queries without self-joins, it was known that these four classes collapse to the class of hierarchical queries, which is also the class of PTIME queries over probabilistic databases. Our main result in this paper is that, in the case of Unions of Conjunctive Queries (UCQ), these classes form a strict hierarchy. Thus, unlike conjunctive queries without self-joins, the expressive power of UCQ differs considerably w.r.t. these target compilation languages. Moreover, we give a complete characterization of the first two target languages, based on the query's syntax.

Categories and Subject Descriptors

H.2.3 [**DATABASE MANAGEMENT**]: Languages—*Query Languages*; F.1.1 [**COMPUTATION BY ABSTRACT DEVICES**]: Models of Computation; G.3 [**Probability and statistics**]: Probabilistic Algorithms

General Terms

Algorithms, Management, Theory

*This work was supported by IIS-0713576 and IIS-0627585.

ICDT 2011, March 21–23, 2011, Uppsala, Sweden.

Keywords

Probabilistic databases, Knowledge compilation, Binary Decision Diagrams, OBDD, FBDD, d-DNNF

1. INTRODUCTION

The goal of *Knowledge compilation* [6, 13, 27] is to represent a Boolean expression in a *format* in which it can answer a range of problems, also called *online-queries*, in PTIME. Typical problems are satisfiability, validity, implication, model counting, substitution with constants, substitution with functions. For example, the *model counting problem* asks for the number of satisfying assignments to a Boolean expression; the more general *probability computation problem* asks for the probability of that expression being true, if every variable is true/false independently with some probability. If one compiles the Boolean expression into (say) an *FBDD*, then the model counting problem and the probability computation problem can be solved in linear time in the size of the *FBDD*. Different compilation languages can solve efficiently different classes of problems, in time polynomial in the size of compiled expression. This motivates the need to know if an expression can be compiled into a small-sized or *compact* representation in a given language.

The *provenance* of a query on a relational database is an expression that describes how the answer was derived from the tuples in the database [17]. In this paper, we are interested in the flavor of provenance called PosBool in [25] (see also [18]), which we will refer to as *lineage*. The lineage is a Boolean expression over Boolean variables corresponding to tuples in the input database. Our goal in this paper is to identify queries whose lineage admits a *compact* compilation. Our main motivation comes from (but is not limited to) probabilistic databases, where the problem is the following: given a query and a probabilistic database (i.e. each tuple has a given probability), compute the probability of each query answer [10]. If the lineage has been compiled into a compact format that supports the probability computation, then one can compute the output probabilities efficiently. In this paper we study queries whose lineage always admits a compact compilation, on any database instance. We are only interested in the data complexity i.e. we assume the query size to be a constant. Our query language is that of unions of conjunctive queries, *UCQ*, and, as usual, we restrict our discussion to Boolean queries.

We consider four compilation targets. For each target T, we denote $UCQ(T)$ the class of *UCQ* queries whose lineage admits a compact compilation in T for all input databases.

Query	Syntactic properties	Membership in $UCQ(T)$, where T is				
		RO	*OBDD*	*FBDD*	*UCQ*	*P*
$q_1 = R(x_1)S(x_1, y_1) \vee S(x_2, y_2)T(x_2)$	inversion-free; read-once	yes	yes	yes	yes	yes
$q_2 = R(x_1), S(x_1, y_1), S(x_2, y_2), T(x_2)$	inversion-free	no	yes	yes	yes	yes
$q_V = R(x_1), S(x_1, y_1)$ $\vee S(x_2, y_2), T(y_2)$ $\vee R(x_3), T(y_3)$	has inversion; all lattice points have separators	no	no	yes	yes	yes
$q_W =$ $(R(x_1), S_1(x_1, y_1) \vee S_2(x_2, y_2), S_3(x_2, y_2))$ $\wedge (R(x_3), S_1(x_3, y_3) \vee S_3(x_4, y_4), T(y_4))$ $\wedge (S_1(x_5, y_5), S_2(x_5, y_5) \vee S_3(x_6, y_6), T(y_6))$	lattice point $\hat{0}$ has no separator but is erasable	no	no	no	yes	yes
q_9 in Fig. 1	lattice point $\hat{0}$ has no separator and has $\mu = 0$ and is non-erasable	no	no	no	?	yes [11]
$h_1 = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$	lattice point $\hat{0}$ has no separator and has $\mu \neq 0$	no	no	no	no	no* [8, 11]

Table 1: Several representative queries. All queries are hierarchical, and have the additional syntactic properties shown. $\hat{0}$ denotes the minimal element of the query's CNF-lattice; μ its Mobius function. Queries q_2, q_V, q_W separate the corresponding classes. We conjecture that q_9 separates $UCQ(UCQ)$ from $UCQ(P)$. * h_1 separates $UCQ(P)$ from UCQ, assuming $FP \neq \#P$.

The first target are Read Once formulas, *RO*. A Boolean expression is *RO* if it can be written using the connectors $\wedge$, $\vee$, $\neg$ in such a way that every input variable is used only once. Read-once formulas admit an elegant characterization due to Gurvich [19] (see [16]). Thus, $UCQ(RO)$ is the class of queries q such that for every input database, the lineage of q on that database is a read-once formula. The second and third targets are Ordered and Free BDD. A *Binary Decision Diagram*[1], BDD, is a rooted DAG where each internal node is labeled with a variable and has two outgoing edges labeled 0 and 1, and each sink node is labeled either 0 or 1. A BDD can be used to compute the value of the Boolean expression: starting at the root node, at each variable node follow the 0 or the 1 edge according to the variable's value, stop after reaching a sink node, and return its label. A BDD is *free* (hence *FBDD*) if any path from the root to a sink node reads every variable at most once. An *FBDD* is *ordered* (hence *OBDD*) if there exists a total order on the Boolean variables s.t. any path from the root to a sink node reads the variables in this order (it may skip some variables). Thus, $UCQ(OBDD)$ and $UCQ(FBDD)$ denote the class of queries q s.t. that for any database instance D, the lineage of q on D admits an *OBDD* (*FBDD*) of polynomial size in D. Finally, our fourth target are d-DNNF, introduced by Darwiche [12] (see also [13]), which are DAGs whose leaves are labeled with Boolean variables or their negation, and internal nodes are labeled either an independent-$\wedge$ (where the two children must have distinct sets of Boolean variables), or with disjoint-$\vee$ (where the two children must be exclusive Boolean formulas). $UCQ(UCQ)$ represents the class of queries whose lineage admits a polynomial size d-DNNF, for any input database.

In addition to these four classes defined by a compilation target, we also consider $UCQ(P)$, the class of queries q with the property that, for every probabilistic database D, the probability of q on D can be computed in PTIME in the size of D. It follows from known results that these five classes form an increasing hierarchy: $UCQ(RO) \subseteq UCQ(OBDD) \subseteq UCQ(FBDD) \subseteq UCQ(UCQ) \subseteq UCQ(P)$.

Dalvi and Suciu [9, 10] have studied the evaluation problem over probabilistic databases for conjunctive queries without self-joins, denoted here CQ^-, and showed that the class of queries computable in PTIME, $CQ^-(P)$, consists precisely of *hierarchical queries* (reviewed in Sect. 2). Olteanu and Huang [20] have shown a remarkable result: that for any hierarchical query, its lineage is a read-once formula. In other words, they explained that the reason why hierarchical queries can be computed in PTIME is because their lineage is read once. This immediately implies (assuming $FP \neq \#P$) that the following five classes collapse: $CQ^-(RO) = CQ^-(OBDD) = CQ^-(FBDD) = CQ^-(UCQ) = CQ^-(P)$.

In this paper we show that, on unions of conjunctive queries (UCQ), these classes no longer collapse. In fact they form a strict hierarchy: $UCQ(RO) \subsetneq UCQ(OBDD) \subsetneq UCQ(FBDD) \subsetneq UCQ(UCQ) \subseteq UCQ(P)$. This means that the reason why certain queries can be computed in PTIME over probabilistic database is no longer their read-onceness, or any other efficient compilation method. (We were not able to separate $UCQ(UCQ)$ from $UCQ(P)$ but we conjecture that they are also separated); instead, each notion of efficiency is distinct. We refer to Table 1 to discuss our

[1]BDD are also known as Branching Program(BP) in the literature

results.

Our results make use of three syntactic properties of a query, called *inversion* [8], *separator* [11], and *hierarchical* queries [10], reviewed in Sect. 2. The following strict implications hold: inversion-free implies existence of separators at all *levels*, which implies the query is hierarchical.

We give a complete characterization of $UCQ(RO)$ and $UCQ(OBDD)$. $UCQ(OBDD)$ coincides with inversion-free queries. $UCQ(RO)$ coincides with queries that are both inversion-free and can be written using $\wedge, \vee, \exists$ such that every relation symbol occurs only once. For example, q_1 in Table 1 can be written as $\exists x.((R(x) \vee T(x)) \wedge \exists y.(S(x,y)))$: here each symbol R, S, T occurs only once and, since q_1 is also inversion-free, it follows that it is in $UCQ(RO)$. Note that the characterization of $UCQ(RO)$ is unrelated to Gurvich's characterization of read-once Boolean expressions [19, 16], or to the algorithm for checking read-once-ness in [22]: these results are about the lineage, our result is about the query.

For $UCQ(FBDD)$ and $UCQ(UCQ)$, we only give sufficient conditions by making use of the *CNF-lattice* associated to a query (introduced in [11]), where each lattice element x is labeled by a subquery, denoted $\lambda(x)$. A sufficient condition for a query to be in $UCQ(FBDD)$ is for every lattice element to have a separator and to satisfy some additional condition. A sufficient condition for $UCQ(UCQ)$ is that every lattice element must have a separator, *except* those lattice elements that can be erased (a notion we define in Sect. 6). For comparison, the necessary and sufficient condition for $UCQ(P)$ is that every lattice element must have a separator, *except* those lattice elements where the Mobius function is 0 ($\mu = 0$) [11]. If an element can be erased, then its Mobius function is 0, but the converse is not true, as illustrated by q_9 in Table 1. We conjecture that q_9 is not in $UCQ(UCQ)$.

The most difficult results in this paper are the separation results $UCQ(OBDD) \subsetneq UCQ(FBDD) \subsetneq UCQ(UCQ)$; they are separated by the queries q_V and q_W respectively in Table 1. The lineage for queries in UCQ is a *simple* Boolean expression: it is monotone, and has a DNF expression of polynomial size. In this sense, our separation results make important contributions to the general separation problem of polynomial-size *OBDD*, *FBDD*, and d-DNNF. Early lower bounds for *FBDD* were for non-monotone formulas, with exponential size DNFs. The first "simple" Boolean formula shown to have exponential *FBDD* was given by Gál in [14], followed by a "very simple" formula given by Bollig and Wegener [1]. The latter is of importance to us, because that formula is precisely the lineage of the non-hierarchical query $R(x), S(x,y), T(y)$, and it implies that all non-hierarchical queries have exponential size *FBDD*, but says nothing about hierarchical queries (q_W is hierarchical).

The lineage of the query q_V, that we use for the first major separation $UCQ(OBDD) \subsetneq UCQ(FBDD)$ is, to the best of our knowledge, the first "simple" Boolean formula separating polynomial-size *OBDD* from *FBDD*. Previous Boolean formulas separating the two classes are non-monotone, and do not have polynomial size DNFs. The classic example is the Weighted Bit Addressing problem (WBA), defined as $F(X_1, \ldots, X_n) = X_{\sum_{i=1,n} X_i}$ (where $X_0 = 0$). Bryant [5] has shown that has it has no polynomial size *OBDD*, while Gergov and Meinel [15] and independently Sieling and Wegener [23] have shown that WBA has a polynomial sized *FBDD*. More examples are given in [26]. Our characterization of $UCQ(OBDD)$ and $UCQ(FBDD)$ allows one to give a class of simple boolean expressions that separate polynomial-size *OBDD* from *FBDD*.

The lineage of the query q_W that we use for our second major separation $UCQ(FBDD) \subsetneq UCQ(UCQ)$ is also, to the best of our knowledge, the first "simple" Boolean formula separating polynomial-size *FBDD* from d-DNNF. The previous separation relies on a result due to Bollig and Wegener [2]: they give an example of two Boolean formulas Φ_1, Φ_2 that have polynomial size *OBDD*, $\Phi_1 \wedge \Phi_2 \equiv \texttt{false}$, yet $\Phi_1 \vee \Phi_2$ cannot have polynomial size *FBDD*. Hence $\Phi_1 \vee \Phi_2$ separates d-DNNF from *FBDD*.

Finally, we note that no lower bounds for d-DNNFs are presently known, except for formulas whose probability computation problem is hard for #P. In particular, we leave open the question whether $UCQ(UCQ) \subsetneq UCQ(P)$. However, our algorithm in Sect. 6 suggests how d-DNNFs may be constructed for general queries, which further suggests that this is not possible for q_9. We conjecture that q_9 is not in $UCQ(UCQ)$, and, hence, that its lineage has no polynomial size d-DNNF.

The paper is organized as follows. We give the basic definitions and review the relevant results in [11] in Sect. 2, then discuss read-once, *OBDD*, *FBDD*, and d-DNNF in Sect. 3, Sect. 4, Sect. 5, Sect. 6. We conclude in Sect. 7. The missing proofs can be found in the full version of this paper.

2. BACKGROUND AND DEFINITIONS

In this paper we discuss *unions of conjunctive queries* (UCQ), which are expressions defined by the following grammar:

$$Q ::= R(\bar{x}) \mid \exists x.Q_1 \mid Q_1 \wedge Q_2 \mid Q_1 \vee Q_2 \qquad (1)$$

$R(\bar{x})$ is a relational atom with variables and/or constants, whose relation symbol R is from a fixed vocabulary. We replace $\wedge$ with comma, and drop $\exists$, when no confusion arises: for example we write $R(x), S(x)$ for $\exists x.(R(x) \wedge S(x))$.

A *query* is an expression as defined by Eq. 1, up to logical equivalence. We consider only Boolean queries in this paper. A *conjunctive query* (CQ) is a query that can be written without $\vee$. Given two conjunctive queries q, q', the logical implication $q \Rightarrow q'$ holds iff there exists a homomorphism $q' \rightarrow q$ [7].

Let D be a database instance. Denote X_t a distinct Boolean variable for each tuple $t \in D$. Let Q be a UCQ. The *lineage* of Q on D is the Boolean expression Φ_Q^D, or simply Φ_Q if D is understood from the context, defined inductively as follows, where $ADom(D)$ denotes the active domain of the database instance:

$$\Phi_{R(\bar{a})} = X_{R(\bar{a})} \quad \Phi_{\exists x.Q} = \bigvee_{a \in ADom(D)} \Phi_{Q[a/x]} \qquad (2)$$

$$\Phi_{Q_1 \wedge Q_2} = \Phi_{Q_1} \wedge \Phi_{Q_2} \qquad \Phi_{Q_1 \vee Q_2} = \Phi_{Q_1} \vee \Phi_{Q_2} \qquad (3)$$

The *query evaluation problem on probabilistic databases* is the following. Given numbers $p(t) \in [0,1]$, compute the probability that the formula Φ_Q is equal to 1, if each Boolean variable X_t is set to 1 independently, with probability $p(t)$; the resulting probability is denoted $P(Q) = P(\Phi_Q^D)$.

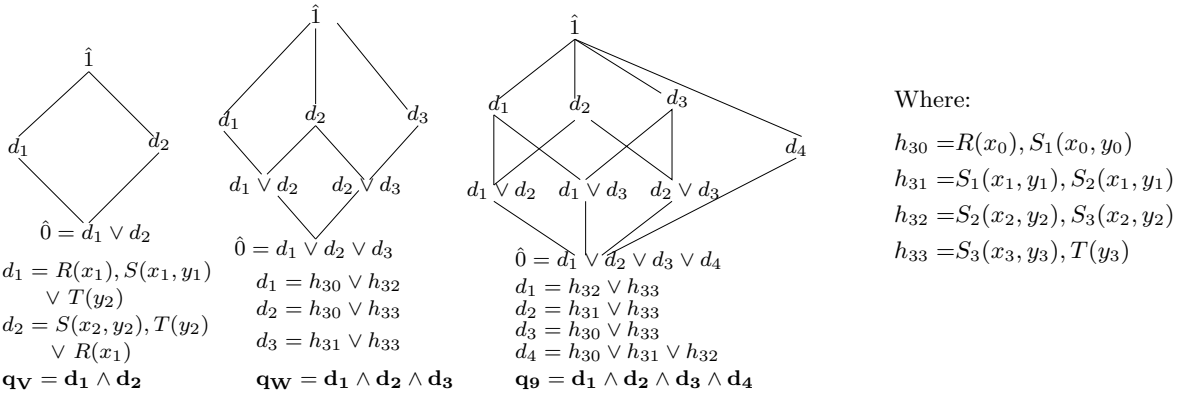

Figure 1: CNF Lattices for the queries q_V, q_W, and q_9. In the lattices for q_W and q_9, $\mu(\hat{0}, \hat{1}) = 0$; in all other cases, $\mu(x, \hat{1}) \neq 0$. In q_W the element $\hat{0}$ is erasable (Def. 6.4); in q_9, the element $\hat{0}$ is not erasable.

DEFINITION 2.1. *$UCQ(P)$ is the class of UCQ queries Q s.t. for any probabilistic database D, the probability $P(Q)$ can be computed in PTIME in the size of D.*

A complete characterization of the class $UCQ(P)$ was given in [11]. We review here, since we will reuse many of those concepts, and also present some new results that we need in this paper. We start with some basics:

- A *component*, c, is a conjunctive query that is *connected* if whenever $c \equiv q_1 \wedge q_2$ then either $c \equiv q_1$ or $c \equiv q_2$. If c has no constants, then an equivalent definition is: whenever q_1, q_2 are two conjunctive queries without constants and $q_1 \wedge q_2 \Rightarrow c$ then either $q_1 \Rightarrow c$ or $q_2 \Rightarrow c$.
- Every *conjunctive query* can be written as a conjunction of components, $q = c_1, c_2, \ldots, c_k$. If q has no constants, then the implication $q \Rightarrow q'$ holds iff $\forall j.\exists i$ s.t. $c_i \Rightarrow c'_j$.
- A *disjunctive query* is a disjunction of components, $d = c_1 \vee \ldots \vee c_k$. An implication $d \Rightarrow d'$ holds iff $\forall i.\exists j$ s.t. $c_i \Rightarrow c'_j$.
- A *UCQ* in DNF is an expression of the form $Q = q_1 \vee \ldots \vee q_m$. An implication $Q \Rightarrow Q'$ holds iff $\forall i.\exists j$ s.t. $q_i \Rightarrow q'_j$.
- A *UCQ* in CNF is an expression of the form $Q = d_1 \wedge \ldots \wedge d_m$. If no constants are used in the queries, then the implication $Q \Rightarrow Q'$ holds iff $\forall j.\exists i$ s.t. $d_i \Rightarrow d_j$.

The containment condition for DNF is due to Sagiv and Yannakakis [21]. The containment condition for CNF is from [11], and only holds if the queries have no constants: for example $R(x, a), S(a, z) \Rightarrow R(x, y), S(y, z)$ (where a is a constant), but neither $R(x, a) \not\Rightarrow R(x, y), S(y, z)$ nor $S(a, z) \not\Rightarrow R(x, y), S(y, z)$.

Following [11] we first perform the following transformations on the query. They preserve the lineage of the query and hence membership in $UCQ(P)$ and all the classes considered in this paper.

Remove constants Every query with constants is rewritten into an equivalent query without constants, over an extended vocabulary. For example, $R(x, a), S(x) \vee R(x, y), T(x)$ is rewritten as $R_1(x), S(x) \vee R_1(x), T(x) \vee R_2(x, y), T(y)$, where $R_1(x) = \pi_x(\sigma_{y=a}(R(x, y)))$ and $R_2(x, y) = \sigma_{y \neq a}(R(x, y))$.

Ranking Assume an ordered domain. A query is *ranked* if it remains consistent after adding all predicates of the form $x < y$, for all pairs of variables x, y that co-occur in some atom, with x occuring before y. For example, $R(x, y), R(y, z), R(x, z)$ is ranked ($x < y \wedge y < z \wedge x < z$ is consistent), while $R(x, y), R(y, x)$ is not ranked ($x < y \wedge y < x$ is inconsistent), and $R(x, x, y)$ is not ranked ($x < x \wedge x < y$ is inconsistent). Every query is rewritten into an equivalent, ranked query, over an extended vocabulary. We give here the main intuition by illustrating with $q = R(x, y), R(y, x)$, and refer to [11] for further details. Denoting $R_1(x) = \pi_x(\sigma_{x=y}(R(x, y)))$, $R_2(x, y) = \sigma_{x<y}(R)$, $R_3(y, x) = \pi_{yx}(\sigma_{x>y}(R))$, we rewrite the query as $R_1(x_1) \vee R_2(x_2, y_2), R_3(x_2, y_2)$. The new query is ranked.

The reason for the first transformation is to ensure that the implication criteria for CNF expressions holds. As a consequence, every *UCQ* has a unique, minimal representation in DNF, and a unique, minimal representation in CNF. The reason for the second transformation will become clear below. We will assume throughout the paper that a CNF or DNF expression of a query is minimized.

The first step in characterizing $UCQ(P)$ is to describe a class of disjunctive queries that are hard for #P, using the notion of a *separator*. Consider a query, and a subexpression of the form $\exists w.Q$ (see grammar Eq. 1): the *scope* of the variable w is the subexpression Q.

DEFINITION 2.2. *A variable w is called a* root variable *if it occurs in all atoms in its scope.*

For a simple illustration, consider $\exists x.\exists y.R(x) \wedge S(x, y)$. Then x is a root variable, but y is not. However, we can write the query equivalently as $\exists x.R(x) \wedge (\exists y.S(x, y))$: now both x and y are root variables.

DEFINITION 2.3. *A disjunctive query d has a separator if $d \equiv \exists w.Q$, w is a root variable, and for every two atoms g, g' with the same relational symbol, w occurs in the same*

position in g and in g'. The variable w is called a separator variable.

THEOREM 2.4. *[11] Let d be a ranked disjunctive query s.t. each component has at least one variable. If d has no separator, then d is hard for #P.*

If d has any component without variables then it has no separator: for example $d = R() \vee S(x)$ has no separator because there is no root variable in $R()$. Evey disjunctive query can be written as $d = d_0 \vee d'$ where d_0 contains all components without variables and d' contains all components with variables. Since d_0 and d' are independent probabilistic events, computing $P(d)$ reduces to computing $P(d')$. This is the reason why the theorem focuses only on the latter. Note that the theorem holds only if the query is ranked: for a counter-example, $R(x,y), R(y,x)$ has no separator, yet is in $UCQ(P)$ (this follows from the ranking shown above, and from Theorem 2.7 below); this is the reason why we rank queries.

Conversely, if d has a separator, $d = \exists w.Q$, then its probability can be computed as $P(d) = 1 - \prod_i (1 - P(Q[a_i/w]))$, where $a_1, \ldots, a_n$ is the active domain of the database, because the events $Q[a_1/w], \ldots, Q[a_n/w]$ are independent. Furthemore, this can be computed efficiently, provided that each query $Q[a_i/w]$ is in $UCQ(P)$. Although we disallowed constants in queries, the expression $Q[a_i/w]$ is OK because all occurrences of a relational symbol have the constant a in the same position; we simply remove a from all atoms, renaming all relational symbols, and decreasing their arity by 1.

EXAMPLE 2.5. *Query q_1 in Table 1 has a separator, because*[2] $q_1 \equiv \exists w.(R(w), S(w,y_1) \vee S(w,y_2), T(w))$. *We can compute its probability as $P(q_1) = 1 - \prod_i (1 - P(R(a_i), S(a_i,y_1) \vee S(a_i,y_2), T(a_i)))$. Query h_1, on the other hand, does not have a separator: if we write it as $\exists w.(R(w), S(w,y_1) \vee S(w,y_2), T(y_2))$ then w is not a root variable, and if we write it as $\exists w.(R(w), S(w,y_1) \vee S(x_2,w), T(w))$ then w occurs on different positions in $S(w,y_1)$ and $S(x_2,w)$. Therefore, h_1 is hard for #P.*

Consider a UCQ in CNF: $Q = d_1 \wedge \ldots \wedge d_k$. For each subset $s \subseteq [k]$ denote $d_s = \bigvee_{i \in s} d_i$. The inclusion/exclusion formula gives us $P(Q) = -\sum_{s \neq \emptyset} (-1)^{|s|} P(d_s)$ and, therefore, if all d_s are in $UCQ(P)$ (in particular, they have separators), then so is Q. The formula is exponential in the size of the query, but this does not affect data complexity. However, the condition $d_s \in UCQ(P)$ is not necessary for all s: some terms in the inclusion/exclusion formula may cancel out, and Q may be in $UCQ(P)$ even if some disjunctive queries d_s are hard.

To characterize precisely when Q is in $UCQ(P)$, [11] defines the *CNF lattice* $(L, \leq)$ for Q. Each element $x \in L$ corresponds to a distinct disjunctive query, denoted $\lambda(x) = d_s$, for some $s \subseteq [k]$, up to logical equivalence; that is, if $d_{s_1} \equiv d_{s_2}$ then they correspond to the same element in $x \in L$. The order relation $\leq$ is reversed logical implication: $x \leq y$ iff $\lambda(y) \Rightarrow \lambda(x)$.

The maximal element in the lattice is denoted $\hat{1}$, and corresponds to $d_\emptyset \equiv$ `false`: all other elements correspond to non-trivial disjunctive queries d_s. The minimal element of the lattice is denoted $\hat{0}$, and corresponds to $\lambda(\hat{0}) = d_1 \vee \ldots \vee d_k$. Three examples are shown in Fig. 1. The *Mobius function* of a lattice $(L, \leq)$ is the function $\mu : L \times L \to \mathbf{Z}$ defined by $\mu(x,x) = 1$, $\mu(x,y) = -\sum_{x < z \leq y} \mu(z,y)$, and $\mu(x,y) = 0$ whenever $x \not\leq y$. Mobius' inversion formula applied to $P(Q)$ is: $P(Q) = -\sum_{x < \hat{1}} \mu(x,\hat{1}) P(\lambda(x))$. Now it becomes obvious that we only need to compute $P(d_s)$ for those queries for which $\mu(x,\hat{1}) \neq 0$. This justifies:

DEFINITION 2.6 (SAFE QUERIES). *[11] (1) Let $Q = d_1 \wedge \ldots \wedge d_k$, and $k \geq 2$. Then Q is* safe *if for every element x in its CNF lattice, if $\mu(x,\hat{1}) \neq 0$, then the disjunctive query $\lambda(x)$ is safe (recursively). (2) Let $d = d_0 \vee d_1$, be a disjunctive query where d_0 contains all components without variables, and d_1 contains all components with at least one variable. Then d is safe if d_1 has a separator w and $d_1[a/w]$ is safe (recursively), for a constant a.*

The characterization of $UCQ(P)$ is:

THEOREM 2.7. *[11] Any safe query is in $UCQ(P)$. Any unsafe query is hard for #P.*

The first part of the theorem follows from our discussion so far. The second part is proven in [11] by using Theorem 2.4.

This completes the characterization of $UCQ(P)$ from [11]. We still need to introduce two more notions that we use in rest of the paper: hierarchical queries and inversion-free queries.

Hierarchical queries Let q be a conjunctive query, and denote $at(x)$ the set of atoms containing the variable $x \in Vars(q)$. We say that q is *hierarchical* if for any two variables x, y, we have $at(x) \subseteq at(y)$ or $at(x) \supseteq at(y)$, or $at(x) \cap at(y) = \emptyset$. A UCQ query Q is *hierarchical* if it is the union of hierarchical conjunctive queries. We give an alternative definition next:

DEFINITION 2.8. *Let Q be a query expression given by the grammar Eq. 1. We say that it is a* hierarchical expression *if every variable is a root variable.*

It is easy to check that a query is hierarchical iff it can be written as a hierarchical expression. For example, the query $R(x,y), S(x,z)$ is hierarchical, because it can be written as $\exists x.(\exists y.R(x,y) \wedge \exists z.S(x,z))$. Examples of non-hierarchical queries are $R(x), S(x,y), T(y)$ and $R(x,y), R(y,z), R(x,z)$. The following is easy to prove :

PROPOSITION 2.9. *If Q is safe, then it is hierarchical.*

In particular, if Q has separators at all levels (meaning at each point in the lattice, and recursively), then it is hierarchical. The converse is not true: for example h_1 in Table 1 is hierarchical, but unsafe. Thus, all non-hierarchical queries are #P-hard, but the converse fails for UCQ queries (it holds for CQ^- queries). We treat non-hierarchical queries separately when proving hardness of $FBDD$ queries.

Inversions We use inversion-free queries, introduced in [8], to characterize $UCQ(RO)$ and $UCQ(OBDD)$. Let $Q = q_1 \vee \ldots \vee q_k$ be a query in DNF. The *unification graph* G has as nodes all pairs of variables (x,y) that co-occur in some atom, and has an edge between (x,y) and (x',y') if : suppose x, y co-occur in g , x', y' co-occur in g', then g and g' are over the same relation symbol and x, y appear at the same positions in g as x', y' in g'. (In other words, g and g' are unifiable, and the unification equates $x = x'$ and $y = y'$.). Given $x, y \in Vars(q_i)$, denote $x \succ y$ if $at(x) \not\subseteq at(y)$.

[2]We omitted the inner quantifiers $\exists y_1$ and $\exists y_2$.

DEFINITION 2.10 (INVERSION). *[8] An inversion in Q is a path of length ≥ 0 in G from a node (x, y) to a node (x', y') s.t. $x \succ y$ and $x' \prec y'$. If no such path exists, we say Q is inversion-free.*

If a query is non-hierarchical then it has an inversion. Indeed, let x, y be two variables s.t. $at(x) \cap at(y) \neq \emptyset$ and neither of the two sets $at(x)$, $at(y)$ contains the other. Then we have $x \succ y$ and $x \prec y$, and the empty path at (x, y) is an inversion. The converse fails: h_1 in Table 1 is hierarchical, yet has an inversion, from (x_1, y_1) to (x_2, y_2).

We give now an alternative, syntactic characterization of an inversion-free query, which we need later. Consider a query expression Q given by the grammar Eq. 1. Let g be an atom in Q, over the relation symbol R of arity k; thus g contains k distinct variables. Assume the existential quantifiers of these k variables are in the following order: $\exists x_1, \exists x_2, \ldots, \exists x_k$. In other words, each variable x_{i+1} is within the scope of x_i. Define π_g to be the permutation for which $g = R(x_{\pi_g(1)}, \ldots, x_{\pi_g(k)})$.

DEFINITION 2.11. *A query expression Q given by the grammar Eq. 1 is an* inversion-free expression *if it is a hierarchical expression, and for any two atoms g_1, g_2 with the same relational symbol, $\pi_{g_1} = \pi_{g_2}$.*

If Q is a hierarchical expression and R a relational symbol, then we write π_R for the common permutation π_g of all atoms g with symbol R.

PROPOSITION 2.12. *Q is inversion free iff it can be written as an inversion-free expression.*

For example, an inversion-free expression for q_1 in Table 1 is $\exists x_1.R(x_1), \exists y_1.S(x_1, y_1) \vee \exists x_2.T(x_2), \exists y_2.S(x_2, y_2)$: in both S-atoms the existential variables x_i, y_i are introduced in the same order, for $i = 1, 2$. On the other hand, the query h_1 has an inversion: if we write it hierarchically as $\exists x_1.R(x_1), \exists y_1.S(x_1, y_1) \vee \exists y_2.T(y_2).\exists x_2.S(x_2, y_2)$, then the variables in $S(x_2, y_2)$ are introduced in a different order from those of $S(x_1, y_1)$.

We end with a simple remark. If d is a disjunctive query that is inversion free, then it has a separator. Indeed, write $d = \bigvee_i c_i$, and write each component as a hierarchical expression, $c_i = \exists x_i.Q_i$. Re-write d as $\exists w.(\bigvee_i Q_i[w/x_i])$. Then w is a separator variable: it oviously occurs in all atoms, and in every atom with relation symbol R, it must occur in position $\pi_R(1)$.

3. QUERIES WITH READ-ONCE LINEAGE

A Boolean expression Φ is *read once* (RO) if it can be written using the connectors $\vee, \wedge, \neg$ such that every Boolean variable occurs at most once. We consider only positive Boolean expressions in this paper, and therefore will use only $\vee$ and $\wedge$. The probability of a read-once Boolean expression can be computed in linear time, because of independence: $P(\Phi_1 \wedge \Phi_2) = P(\Phi_1) \cdot P(\Phi_2)$ and $P(\Phi_1 \vee \Phi_2) = 1 - (1 - P(\Phi_1))(1 - P(\Phi_2))$; this justifies our interest in this class of expressions. In this section we characterize the queries that have read-once lineages. An elegant characterization of read-once Boolean expressions was given by Gurvich [19] (see [16]), but we will not use that characterization. Note that our characterization is of *queries*, while Gurvich's characterization is of *Boolean expressions*.

DEFINITION 3.1. *$UCQ(RO)$ is the class of queries Q s.t. for every database instance D, the lineage of Q on D is a read once Boolean expression.*

Recall that CQ^- denotes the set of conjunctive queries without self-joins. Dalvi and Suciu [9, 10] showed that $CQ^-(P)$ is precisely the class of hierarchical queries. Olteanu and Huang [20] showed that all hierarchical queries in CQ^- have read-once lineages, implying $CQ^-(RO) = CQ^-(P) =$ "hierarchical queries". In this section we characterize the class $UCQ(RO)$.

DEFINITION 3.2. *Let Q be a query expression given by the grammar Eq. 1. We say that Q is* hierarchical-read-once *if it is hierarchical (see Def. 2.8), and every relational symbol occurs at most once. A query is hierarchical-read-once if it is equivalent to a hierarchical-read-once expression.*

Obviously, every hierarchical CQ^- query is also hierarchical-read-once; our definition is more interesting when applied to UCQ. The following is a necessary condition for hierarchical-read-once-ness:

PROPOSITION 3.3. *If Q is a hierarchical read-once expression then it is also an inversion-free expression.*

The proof is immediate, since no two distinct atoms in Q may refer to the same relational symbol, hence the condition $\pi_{g_1} = \pi_{g_2}$ is satisfied vacuously.

For a simple example, consider query q_1 in Table 1. It is equivalent to the expression $\exists x.(R(x) \vee T(x)) \wedge \exists y.S(x, y)$, which is both hierarchical and read-once. Notice that in the definition we require Q to be be at the same time hierarchical and read-once. Sometimes we can achieve these two goals separately, but not simultaneously: for example $h_1 = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$ is hierarchical, and can also be written as $\exists x.\exists y.(R(x) \vee T(y)) \wedge S(x, y)$, which is read-once. Since h_1 has an inversion, by Prop. 3.3 it cannot be written simultaneously as a hierarchical and read-once expression.

THEOREM 3.4. *$Q \in UCQ(RO)$ iff it is hierarchical-read-once.*

The "if" direction is a straightforward extension of the technique used in [20] to prove that hierarchical queries in CQ^- are read-once. For the "only-if", we construct one database instance D that is "large enough" (depending only on the query), and prove the following: if Q's lineage on D is read-once, then Q is hierarchical-read-once.

It is decidable if a given query Q is hierarchical-read-once, because for a fixed vocabulary there are only finitely many hierarchical-read-once expressions: simply iterate over all of them and check equivalence to Q. This implies that it is decidable whether $Q \in UCQ(RO)$. For example, one can check that q_2 in Table 1 is not in $UCQ(RO)$, by enumerating all hierarchical-read-once expressions over the vocabulary R, S, T; we will return to q_2 in the next section.

4. QUERIES AND OBDD

OBDD were introduced by Bryant [3] and studied extensively in the context of model checking and knowledge representation. A good survey can be found in [27]; we give here a quick overview. A BDD, is a rooted DAG with

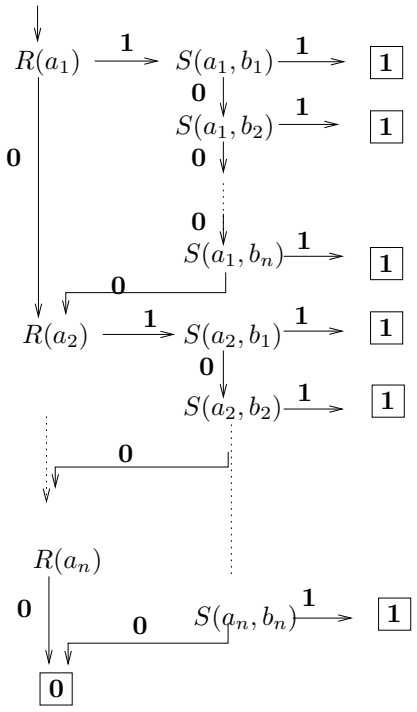

Figure 2: *OBDD* **for the query** $R(x), S(x, y)$**(cf. [20])**

two kinds of nodes. A *sink node* or *output node* is a node without any outgoing edges, which is labeled either 0 or 1. An *inner node*, *decision node*, or *branching node* is labeled with a Boolean variable X and has two outgoing edges, labeled 0 and 1 respectively. Every node u uniquely defines a Boolean expression Φ_u as follows: $\Phi_u = \texttt{false}$ and $\Phi_u = \texttt{true}$ for a sink node labeled 0 or 1 respectively, and $\Phi_u = \neg X \wedge \Phi_{u_0} \vee X \wedge \Phi_{u_1}$ for an inner node labeled with X and with successors u_0, u_1 respectively. The BDD represents a Boolean expression $\Phi : \Phi \equiv \Phi_u$ where u is the root of the BDD. A *Free BDD*, or *FBDD* is one in which every path from the root to a sink node contains any variable X at most once. Given an *FBDD* that represents Φ, one can compute the probability $P(\Phi)$ in time linear in the size of the *FBDD*: this justifies our interest in *FBDD*.

While it is trivial to construct a large *FBDD* for Φ (e.g. as a tree of size 2^n that checks exhaustively all n variables $X_1, \ldots, X_n$), it is not trivial at all to construct a compact *FBDD*. To simplify the construction problem, Bryant [4] introduced the notion of *Ordered BDD*, *OBDD*, which is an *FBDD* such that there exists a total order Π on the set of variables s.t. on each path from the root to a sink, the variables $X_1, \ldots, X_n$ are tested in the order Π (variables may be skipped). One also writes Π-*OBDD*, to emphasize that the *OBDD* has order Π. Therefore, the *OBDD* construction problem has been reduced to the problem of finding a variable order Π.

Every read-once formula Φ admits an *OBDD* whose size is linear in Φ, by an inductive argument: if $\Phi = \Phi_1 \wedge \Phi_2$ first construct *OBDD*s for Φ_1 and Φ_2, and replace every sink-node labeled 1 in Φ_1 with (an edge to) the root of Φ_2; for $\Phi_1 \vee \Phi_2$, replace every sink-node labeled 0 in Φ_1 with the root of Φ_2.

Definition 4.1. *$UCQ(OBDD)$ is the class of queries Q s.t. for every database D, the lineage of Q on D has an OBDD of size polynomial in the database size.*

We show an example in Fig. 2. In this section we prove the following:

Theorem 4.2. *$Q \in UCQ(OBDD)$ iff it is inversion-free.*

We have seen that q_2 from Table 1 is not read-once. However, $q_2 \in UCQ(OBDD)$, because it is inversion-free, therefore we obtain the following separation:

Proposition 4.3. *$q_2 \in UCQ(OBDD) - UCQ(RO)$*

The significance of this result is the following. Olteanu and Huang [20] showed that all hierarchical queries in CQ^- have an *OBDD* whose size is linear in that of the database, proving that $CQ^-(RO) = CQ^-(OBDD)$. Our proposition shows that these classes no longer collapse over UCQ.

We also note that all inversion-free queries are hierarchical (Sect. 2), therefore any non-hierarchical query is not in $UCQ(OBDD)$.

In the remainder of the section we prove Theorem 4.2, in two stages : first showing that inversion-free formulae have polynomial size *OBDD*, and then that those with inversion have exponential size *OBDD*.

4.1 Tractable Queries

Given an *OBDD* of Φ over variables $\bar{x}$ with variable order Π, the width at level k, $k \leq n$ is the number of distinct subformulae that result after checking first k variables in the order Π, i.e. $|\{\Phi_{x_{\pi(1)} \cdots x_{\pi(k)} = \bar{b}} \mid \bar{b} \in \{0,1\}^k\}|$. The width of an *OBDD* is the maximum width at any level. If the number of variables is n, and width w, then a trivial upper bound on the size of the *OBDD* is nw. In what follows, we give a variable ordering for inversion-free queries under which the width is always constant(exponential in query size) and hence the size of *OBDD* is linear.

We also need to define the notion of shared BDD. A *shared* BDD for a set of formulas $\Phi_1, \Phi_2, \ldots, \Phi_m$ is a BDD where the sink nodes are labeled with $\{0,1\}^m$ i.e. they give the valuation for each of the Φ_i,$1 \leq i \leq m$. This means a node reached by following the assignments $\bar{x}$ from the root can be thought of as representing a set of subformulae $\Phi_{1\bar{x}}, \Phi_{2\bar{x}}, \ldots, \Phi_{k\bar{x}}$. Shared BDD evaluate a set of formulae simultaneously: this enables us to compute any combination function of the formulae. So, for instance, one can derive the *OBDD* of $\Phi_1 \otimes \Phi_2$ for any boolean operation $\otimes$ from the shared *OBDD* for Φ_1, Φ_2

The following is a well-known lemma for *OBDD* synthesis.

Lemma 4.4. *(cf. [27]) Let Φ_1, Φ_2 be two boolean functions and consider a fixed variable order Π. If there exists Π-OBDDs of width w_1, w_2 for Φ_1, Φ_2 respectively, then there exists a shared Π-OBDD of width $w_1 w_2$ for Φ_1, Φ_2.*

Proposition 4.5. *If Q is inversion-free, then for every database D its lineage has an OBDD with width $w = 2^g$, where g is the number of atoms in the query. Therefore, the size of the OBDD is linear in the size of the database.*

We give a simple proof, using Lemma 4.4, that constructs the *OBDD* inductively on the hierarchical expression for Q : the resulting *OBDD* has size $O(|D|)$.

PROOF. Consider a hierarchical expression for Q, and let π_R be the permutation associated to the symbol R (Def. 2.11). Let D be a database, and assume that its active domain $ADom(D)$ is an ordered domain. We start by defining a linear order Π on all tuples in D. Fix any linear order on the relational symbols, $R_1 < R_2 < \ldots$. We add all relation symbols to $ADom(D)$, placing them at the beginning of the order. We associate to each tuple in D a string in $(ADom(D))^*$, as follows: tuple $R(a_{\pi_R(1)}, a_{\pi_R(2)}, \ldots, a_{\pi_R(k)})$ is associated to the string $a_1 a_2 \ldots a_k R$. That is, the first element is the constant on the root attribute position; the second element is the constant on the attribute position corresponding to a quantifier depth 2, etc. We add the relation name at the end. Next, we order the Boolean variables in the lineage expression Φ_Q^D lexicographically by their string, and denote Π the resulting order. We prove that Π-*OBDD* has width $w = 2^g$, inductively on the structure of the inversion-free expression Q. If $Q = Q_1 \vee / \wedge Q_2$ then we use Lemma 4.4. If $Q = \exists x.Q_1$, then $\Phi_Q = \bigvee_{a \in ADom(D)} \Phi_{Q[a/x]}$. Let the active domain consists of $a_1 < a_2 < \ldots < a_n$, in this order. The *OBDD*s for $\Phi_{Q[a_1/x]}, \ldots, \Phi_{Q[a_n/x]}$ are over disjoint sets of Boolean variables (because x is a root variable); assume that their width is w. The *OBDD* for Φ_Q consists of their union, where we redirect the 0 sink nodes of $\Phi_{Q[a_i/x]}$ to the root node of $\Phi_{Q[a_{i+1}/x]}$: the width is still w. The *OBDD* of a single ground atom, say $R(\bar{a})$, has width only 2. This completes the proof. □

COROLLARY 4.6. *If a set of components $c_1, c_2, \ldots, c_m$ is inversion-free, then for every database D, they have a shared-OBDD with size linear in the size of the database.*

4.2 Hard Queries

For $k \geq 1$, define the following queries (see also Fig. 1):

$$\begin{aligned} h_{k0} &= R(x_0), S_1(x_0, y_0) \\ h_{ki} &= S_i(x_i, y_i), S_{i+1}(x_i, y_i) \qquad i = 1, k-1 \\ h_{kk} &= S_k(x_k, y_k), T(y_k) \end{aligned}$$

Denote $h_k = \bigvee_{i=0,k} h_{ki}$. The queries h_k were shown in [8, 11] to be hard for #P and are used to prove the hardness of a much larger class of unsafe queries. We show here that they have a remarkable property w.r.t. *OBDD*: if the same variable order Π is used to compute all queries h_{k0}, h_{k1}, ..., h_{kk}, then at least one of these $k+1$ *OBDD*s has exponential size. Note that each query is inversion-free, hence it admits an efficient *OBDD*, e.g. Fig. 2 illustrates h_{k0}: what we prove is that there is no common order under which all have an efficient *OBDD*. This tool is quite powerful, allowing us to give a rather simple proof that queries with inversion have exponential size *OBDD* (Prop. 4.8). There is no analogous tool for proving #P-hardness: all queries h_{ki} are in PTIME, for $i = 0, k$, and this tells us nothing about the larger query where they occur.

The *complete bipartite graph* of size n is the following database D over the vocabulary of h_k: relation R has n tuples $R(a_1), \ldots, R(a_n)$, relation T has n tuples $T(b_1), \ldots, T(b_n)$, and each relation S_i has n^2 tuples $S_i(a_j, b_l)$, for $i = 1, k$, and $j, l = 1, n$.

PROPOSITION 4.7. *Let D be the complete bipartite graph of size n, and fix any ordering Π on the corresponding Boolean variables. For any $i = 0, k$, let n_i be the size of some Π-OBDD for the lineage of h_{ki} on D. Then $\sum_{i=0}^{k} n_i > k \cdot 2^{\frac{n}{2k}}$.*

PROOF. Denote the Boolean variables associated to the tuples $R(a_i)$, $i = 1, n$ with $X_1, X_2, \ldots$; those associated to the tuples $S_p(a_i, b_j)$ with Z_{ij}^p; and those associated to the tuples $T(b_j)$ with Y_j. We will refer generically to any variable as v_i, and assume the order Π is $v_1, v_2, \ldots$ Denote Φ_{kp} the lineage of h_{kp} on D; by assumption, we have Π-*OBDD* for each of them. Assume w.l.o.g. that each *OBDD* is complete i.e. every path from root to sink contains every variable exactly once.

In any *OBDD* of a Boolean expression Φ, the number of nodes at level h (i.e. after first h variables $v_1 \ldots v_h$ have been eliminated) is the size of the set $\{\Phi[(v_1 \ldots v_h) = \bar{b}] \mid \bar{b} \in \{0,1\}^h\}$. This is because every distinct subformula will result in a new separate node. A standard technique in proving lower bounds on the size of *OBDD* is to find a level where the number of distinct formulae must be exponential. This immediately gives the same exponential lower bound on the size of *OBDD* for that ordering.

For any level h, denote h_1, h_2 the number of $\mathbf{X}$, and of $\mathbf{Y}$ variables respectively in the initial sequence $v_1, v_2, \ldots, v_h$ of Π. Define h to be the first level for which $h_1 + h_2 = n$. Denote $\mathbf{X}^{set} = \{X_i \mid X_i \in \{v_1, \ldots, v_h\}\}$ and $\mathbf{X}^{unset} = \mathbf{X} \setminus \mathbf{X}^{set}$, and similarly $\mathbf{Y}^{set}$, $\mathbf{Y}^{unset}$, $\mathbf{Z}^{set}$, $\mathbf{Z}^{unset}$. W.l.o.g. assume $h_1 \geq n/2$.

Consider the *OBDD* for $\Phi_{k0} = \bigvee_{ij} X_i Z_{ij}^1$. Suppose there exists j s.t. $\forall i. X_i \in \mathbf{X}^{set} \Rightarrow Z_{ij}^1 \in \mathbf{Z}^{unset}$; then for each assignment $\bar{b}$ to $\mathbf{X}^{set}$, we get a different subformula $\Phi_{k0}[\mathbf{X}^{set} = \bar{b}]$. Since the number of such formulae is $2^{h_1} \geq 2^{n/2}$, we obtain $n_0 > 2^{n/2}$, which proves the claim. Hence we can assume there is no such j. This means $\forall j$, $\exists i$ s.t. $X_i \in \mathbf{X}^{set}$ and $Z_{ij}^1 \in \mathbf{Z}^{set}$.

Define S to be a set of pairs (i, j) as follows. For each j s.t. $Y_j \in \mathbf{Y}^{unset}$, choose some i s.t. $Z_{ij}^1 \in \mathbf{Z}^{set}$: then include (i, j) in S. Note that the cardinality of S is $n - h_2 = h_1$.

For each $p = 1, \ldots, k-1$, denote C_p the subset of S consisting of indices (i, j) s.t. $Z_{ij}^1, \ldots, Z_{ij}^p \in \mathbf{Z}^{set}$ and $Z_{ij}^{p+1} \in \mathbf{Z}^{unset}$; and let $C_k = S - \bigcup_{p=1,k-1} C_p$. Thus, $C_1, \ldots, C_k$ forms a partition of S. Denoting $c_1, \ldots, c_k$ their cardinalities we have $c_1 + \ldots + c_k = h_1$.

Next, for each $p = 1, \ldots, k-1$, consider the *OBDD* for $\Phi_{kp} = \bigvee_{ij} Z_{ij}^p Z_{ij}^{p+1}$. Forall $(i, j) \in C_p$ we have $Z_{ij}^p \in \mathbf{Z}^{set}$ and $Z_{ij}^{p+1} \in \mathbf{Z}^{unset}$. Each assignment of the former variables leads to a different expression over the latter variables: hence there are at least 2^{c_p} distinct expressions, therefore the number of nodes in this *OBDD* is $n_p \geq 2^{c_p}$.

Finally, consider the *OBDD* for $\Phi_{kk} = \bigvee_{ij} Z_{ij}^k Y_j$. Forall $(i, j) \in C_k$ we have $Z_{ij}^k \in \mathbf{Z}^{set}$ and $Y_j \in \mathbf{Y}^{unset}$. Using the same arugment, we obtain $n_k \geq 2^{c_k}$.

Putting everything together we obtain:

$$\begin{aligned} \sum_{i=1,k} n_i &\geq \sum_{i=1,k} 2^{c_i} \geq k 2^{\frac{\sum_i c_i}{k}} \\ &= k 2^{\frac{h_1}{k}} > k 2^{\frac{n}{2k}} \end{aligned}$$

Notice that n_0 does not appear above, but we used it in order to construct the set S. This proves our claim. □

PROPOSITION 4.8. *Let Q be a query, and suppose it has an inversion of length $k > 0$. Let D_0 be a complete bipartite graph of size n (i.e. a database over the vocabulary of h_k). Then there exists a database D for Q s.t. $|D| = O(|D_0|)$ and any OBDD for Q has size $\Omega(k 2^{n/2k})$.*

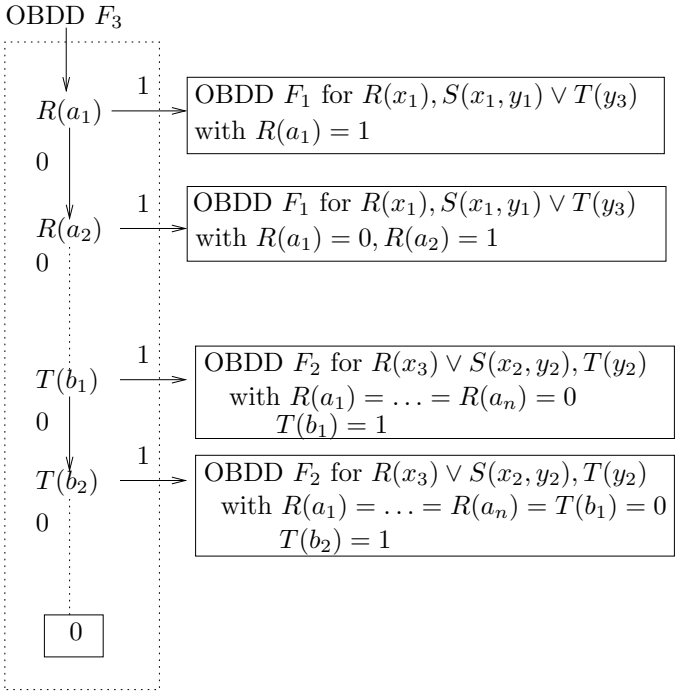

Figure 3: *FBDD* **for the query** q_V

We explain here the main idea. We use the inversion of length k to construct a database D that mimics the query h_k over a complete bipartite graph. Assuming an *OBDD* for Q on this database, we show that one can set the Boolean variables to 0 or 1, to obtain a lineage for each h_{ki}. What is interesting is that this construction *cannot* be used to prove #P-hardness of Q by reduction from h_k: in other words, Q over D is not equivalent to h_k over D_0. But we make Q equivalent to each h_{ki}, and by Prop. 4.7 this is sufficient to prove that Q has a no compact *OBDD*.

If Q has an inversion of length 0, then it is non-hierarchical and as we discuss later in Theorem 5.8 $Q \notin UCQ(FBDD)$, and hence $Q \notin UCQ(OBDD)$ either.

5. QUERIES AND FBDD

We now turn to *FBDD*, also known as Read-Once Branching Programs. Unlike *OBDD*, here we no longer require the same variable order on different paths. *FBDD* are known to be strictly more expressive than *OBDD* over *arbitrary* (non-monotone) Boolean expressions, for example the Weighted Bit Addressing problem admits polynomial sized *FBDD*, but no polynomial size *OBDD* [5, 15, 23]. On the other hand, to the best of our knowledge no monotone formula was known to separate these two classes. Moreover, over conjunctive queries without self-joins, *FBDD* are no more expressive than *OBDD*, since the latter already capture $CQ^-(P)$. In this section we show that *FBDD* are strictly more expressive than *OBDD* over *UCQ*. In particular, we give a simple (!) monotone Boolean expression that has a polynomial size *FBDD*, but no *OBDD*.

DEFINITION 5.1. *$UCQ(FBDD)$ is the class of queries Q s.t. for any database D, the lineage of Q on D has an FBDD of size polynomial in the database size.*

Clearly $UCQ(OBDD) \subseteq UCQ(FBDD)$: we prove now that the inclusion is strict, using a simple example.

EXAMPLE 5.2. *Consider q_V in Table 1. This query has an inversion between $S(x_1, y_1)$ and $S(x_2, y_2)$, hence it does not admit a compact OBDD. We show how to construct a compact FBDD. Write it in CNF:*

$$q_V = (R(x_1), S(x_1, y_1) \vee T(y_3)) \wedge (S(x_2, y_2), T(y_2) \vee R(x_3)) = d_1 \wedge d_2$$

Its CNF lattice is shown in Fig. 1. The minimal element of the lattice is:

$$d_3 = d_1 \vee d_2 = R(x_3) \vee T(x_3)$$

Each of d_1, d_2, d_3 is inversion-free, hence they have OBDDs, denote them F_1, F_2, F_3. Of course, F_1 and F_2 use different variable orderings and cannot be combined into an OBDD for q_V. Consider the database given by the bipartite graph (Sect. 4) and assume the following order on the active domain: $a_1 < \ldots < a_n < b_1 < \ldots < b_n$. Our FBDD starts by computing d_3. If $d_3 = 0$, then $q_V = 0$; this is a sink node. If $d_3 = 1$, then, depending on which sink node in F_3 we have reached, either $d_1 = 1$ or $d_2 = 1$, and we need continue with either F_2 or of F_1 respectively. This way, no path goes through both F_1 and F_2. Note that the FBDD is not ordered, since some paths use the order in F_1, others that in F_2. Fig. 3 illustrates the construction;

Thus:

PROPOSITION 5.3. *$q_V \in UCQ(FBDD) - UCQ(OBDD)$.*

The significance of this result is the following. The lineage of q_V is, to the best of our knowledge, the first "simple" Boolean expression (i.e. monotone, and with polynomial size DNF) that has a polynomial size *FBDD* but no polynomial size *OBDD*. Previous examples separating these classes where Weighted Bit Addressing problem (WBA) [5, 15, 23], and other examples given in [26], and these were not "simple". Our result also constrasts *UCQ* to CQ^-: for the latter it follows from [20] that $CQ^-(OBDD) = CQ^-(FBDD)$.

In the reminder of this section we will give a partial characterization of $UCQ(FBDD)$, by providing a sufficient condition, and a necessary condition for membership. We start with the sufficient condition.

DEFINITION 5.4. *Let $d = \bigvee c_i$ and $d' = \bigvee c'_j$ be two disjunctive queries, s.t. the logical implication $d' \Rightarrow d$ holds. We say that d* dominates *d' if for every component c'_j in d' and for every atom g in c'_j one of the following conditions hold: (a) the relation symbol of g does not occur in d, or (b) there exists a component c_i and a homomorphism $c_i \to c'_j$ whose image contains g.*

In Example 5.2, d_3 dominates d_1: if one considers the component $R(x_1), S(x_1, y_1)$ in d_1, then the atom $R(x_1)$ is the image of a homomorphism, while the atom $S(x_1, y_1)$ does not occur at all in d_3. Similarly d_3 dominates d_2.

In analogy to the definition of *safe queries* Def. 2.6 we define here *rf-safe queries*[3]:

[3] *r* is for restricted, since we don't have a full characterization yet

DEFINITION 5.5. *(1) Let $Q = d_1 \wedge \ldots \wedge d_k$, and $k \geq 2$. Then Q is* rf-safe *if for every element x in its CNF lattice the disjunctive query $\lambda(x)$ is rf-safe, and for every two lattice elements $x \leq y$, $\lambda(x)$ dominates $\lambda(y)$. (2) Let $d = d_0 \vee d_1$, be a disjunctive query, where d_0 contains all components c_i without variables, and d_1 contains all components c_i with at least one variable. Then d is rf-safe if d_1 has a separator w and $d_1[a/w]$ is rf-safe, for a constant a.*

For example, query q_V is rf-safe, since d_3 dominates both d_1 and d_2. Our sufficient characterization of $UCQ(FBDD)$ is:

THEOREM 5.6. *Every rf-safe query is in $UCQ(FBDD)$.*

But this is not a complete characterization. The following query q_T, is not rf-safe, but one can construct a polynomial-size *FBDD* for it.

$$q_T = (T_1(x), A(x), V(x,y,z) \vee T_3(z), C(z) \vee T_2(y), B(y), D(y)),$$
$$(T_2(y), B(y), V(x,y,z) \vee T_1(x), A(x) \vee T_3(z), C(z)),$$
$$(T_3(z), C(z), V(x,y,z) \vee T_1(x), A(x) \vee T_2(y), B(y), D(y))$$

Next, we present our separation result. Recall the query q_W from Table 1. We prove here:

THEOREM 5.7. $q_W \notin UCQ(FBDD)$.

We will return to this query in the next section.

Our hardness result for *FBDD* is more limited in scope than that for *OBDD*; in particular it says nothing about non-hierarchical queries. This, however, follows from a very strong result by Bollig&Wegener[1]. They showed that, for arbitrary large n, there exists a bipartite graph G s.t. the formula $\Phi = \bigvee_{(i,j)\in G} X_i Y_j$ has no polynomial size *FBDD*[4]. This immediately implies that the query $Q = R(x), S(x,y), T(y)$ is not in $UCQ(FBDD)$, because from any *FBDD* for Q on the complete, bipartited graph one can obtain and *FBDD* for Φ by setting all variables $X_{S(i,j)} = 1$ for $(i,j) \in G$ and setting $X_{S(i,j)} = 0$ for $(i,j) \notin G$. In particular, this implies:

THEOREM 5.8. *(cf. [1]) If Q is non-hierarchical, then $Q \notin UCQ(FBDD)$.*

6. QUERIES AND D-DNNFS

d-DNNFs were introduced by Darwiche [12]; a good survey is [13], we review them here briefly. A *Negation Normal Form* is a rooted DAG, internal nodes are labeled with $\vee$ or $\wedge$, and leaves are labeled with either a Boolean variable X or its negation $\neg X$. Each node x in an NNF represents a Boolean expression Φ_x, and the NNF is said to represent Φ_z, where z is the root node. A *Decomposable NNF*, or DNNF, is one where for every $\wedge$ node, the expressions of its children are over disjoint sets of Boolean variables. A *Deterministic DNNF*, or d-DNNF is a DNNF where for every $\vee$ node, the expressions of its children are mutually exclusive. Given a d-DNNF one can compute its probability in polynomial time, by applying the rules $P(\Phi_x \wedge \Phi_y) = P(\Phi_x)P(\Phi_y)$ and $P(\Phi_x \vee \Phi_y) = P(\Phi_x) + P(\Phi_y)$ (and similarly for nodes with out-degree greater than 2); this justifies our interest in d-DNNF. Any *FBDD* of size n can be converted to an d-DNNF of size $5n$ [13]: for every node x in the *FBDD*, testing a variable X, write its formula as $(\neg X) \wedge \Phi_y \vee X \wedge \Phi_z$, where y and z are the 0-child and the 1-child: obviously, the $\vee$ is "deterministic", and the $\wedge$'s are "decomposable".

It is open whether d-DNNFs are closed under negation [13, pp. 14]; NNFs are obviously closed under negation, but the d-DNNFs impose asymmetric restrictions on $\wedge$ and $\vee$, so by switching them during negation, the resulting NNF is no longer a d-DNNF. For that reason, we extend here d-DNNF's with $\neg$-nodes, and denote the result d-DNNF$^\neg$: probability computation can still be done in polynomial time on an d-DNNF$^\neg$.

DEFINITION 6.1. *$UCQ(UCQ)$ is the class of queries Q s.t. for any database D, the negation of Q's lineage on D has a dDNNF of size polynomial in the database size. $UCQ(UCQ^\neg)$ is the class of queries Q whose lineage on a database D admits a d-DNNF$^\neg$ whose size is polynomial in D.*

$UCQ(FBDD) \subseteq UCQ(UCQ) \subseteq UCQ(UCQ^\neg) \subseteq UCQ(P)$; we prove now that the first inclusion is strict, by using an example.

EXAMPLE 6.2. *Consider $q_W = d_1 \wedge d_2 \wedge d_3$ in Fig. 1. Denote the three lower points in the lattice as:*

$$d_{12} = d_1 \vee d_2$$
$$d_{23} = d_2 \vee d_3$$
$$d_{123} = d_1 \vee d_2 \vee d_3$$

We have seen in Theorem 5.7 that q_W does not have a polynomial size FBDD. On the other hand, this query is in $UCQ(P)$, because $\mu(\hat{0}, \hat{1}) = 0$: to compute the probability we only need the other 5 points in the lattice (which form a W, hence the name). Thus, this query is right at the border of $UCQ(FBDD)$ and $UCQ(P)$, an interesting study for d-DNNFs.

We construct a compact d-DNNF for $\neg q_W$, by writing:

$$\neg q_W = \neg d_2 \vee (d_2 \wedge \neg d_1) \vee (d_2 \wedge \neg d_3) \quad (4)$$

Both $\vee$'s are "deterministic": clearly $\neg d_2$ is disjoint from the other two, and the last two queries are disjoint because of the implication $d_2 \Rightarrow d_1 \vee d_3$ (this can be seen from Fig. 1). On the other hand, each of the three smaller queries in Eq. 4 has an OBDD. $\neg d_2$ has an OBDD because d_2 is inversion-free; $d_2 \wedge \neg d_1$ has an OBDD because there is no inversion between d_2 and d_1, hence they have OBDDs using the same variable order, and we can synthesize an OBDD for $d_2 \wedge \neg d_1$ using Lemma 4.4.

PROPOSITION 6.3. $q_W \in UCQ(UCQ) - UCQ(FBDD)$.

The significance of this result is the following. This is, to the best of our knowledge, the first example of a "simple" Boolean expression (meaning monotone, and with a polynomial size DNF) that has a d-DNNF but with no *FBDD*. The previous separation of *FBDD* and d-DNNFs is based on a result by Bollig and Wegener [2], which we review briefly. Consider a Boolean matrix of variables X_{ij}. Let Φ_1 denote

[4]Their graph is the following: fix $n = p^2$ where p is a prime number. Then $G = \{(a + bp, c + dp) \mid c \equiv (a + bd) \mod p\}$.

the formula "there are an even number of 1's and there is a row consisting only of 1's". Let Φ_2 denote the formula "there are an odd number of 1's and there is a column consisting only of 1's"; [2] show that $\Phi_1 \vee \Phi_2$ does not have a polynomial size *FBDD*. However, this formula has a polynomial size d-DNNF, because each of Φ_1, Φ_2 that have polynomial size *OBDD*s and $\Phi_1 \wedge \Phi_2 \equiv$ `false`. Note, however, that these formulas are non-monotone and have exponential size DNF's (they are not in AC^0). By contrast, the lineage of q_W is monotone, has polynomial size DNF, and separates *FBDD* from d-DNNF.

In the rest of the section, we give a sufficient criteria for a query Q to be in $UCQ(UCQ^\neg)$, which is quite interesting because it explains the border between d-DNNF's and PTIME in terms of lattice-theoretic concepts. For that purpose we adapt the "incomplete algorithm based on conditioning", which was described in [11] in order to illustrate the power of the Mobius' inversion formula approach for capturing all of $UCQ(P)$. We adapt here the algorithm to $UCQ^\neg$.

Given a query $Q = d_1 \wedge \ldots \wedge d_m$, if $m = 1$ then Q is a disjunctive query; in this case it must have a separator (otherwise it is #P-hard (Theorem 2.4)), $d_1 = \exists w.Q_1$ and we write: $\neg Q = \bigwedge_{a \in ADom(D)} d_1[a/w]$. The $\wedge$ operator is "decomposable", i.e. its children are independent.

The interesting case is $m \geq 2$. Let $(L, \leq)$ be the CNF lattice: its co-atoms are $d_1, \ldots, d_m$, and every element is a meet of co-atoms. Choose a subset $\Gamma \subseteq L$ (to be described below), and define the query $C = \bigwedge_{x \in \Gamma} \lambda(x)$. Let $u_1, \ldots, u_k$ be the minimal elements of the set $\{z \mid z \in L, \neg(\exists x \in \Gamma. z \leq x)\}$. For each $i = 1, k$ denote the query $Q_i = \bigwedge_{u_i \leq x} \lambda(x)$: this is a conjunction of disjunctive queries, and it suffices to take only those x that are coatoms. Write Q as:

$$\begin{aligned}\neg Q =& \neg C \vee (C \wedge \neg Q)\\ =& \neg C \vee (C \wedge \neg Q_1) \vee \ldots (C \wedge \neg Q_k)\end{aligned}$$

All $\vee$'s are "deterministic" (i.e. disjoint), because the u_i's are minimal elements. Next, write:

$$\begin{aligned}C \wedge \neg Q_i =& \neg((\neg C \wedge \neg Q_i) \vee Q_i)\\ =& \neg(\neg(C \vee Q_i) \vee \neg(\neg Q_i))\end{aligned}$$

Here, too, the outermost $\vee$ is "deterministic", which can be seen easily in the first line. After this we continue to compute the d-DNNF$^\neg$, recursively, for $\neg C$, $\neg Q_i$, and $\neg(C \vee Q_i)$, for $i = 1, k$. Each of these queries is the negation of a *UCQ*. Furthermore, the CNF lattice of that *UCQ* is a meet-sublattice of L. Indeed, $Q_i = \wedge d_j$ is the disjunction of a subset of the co-atoms $d_1, \ldots, d_k$; $C = \wedge c_p$ is the disjunction of a subset of queries c_p in the lattice (not necessarily coatoms); and, by writing $C \vee Q_i$ as $\bigwedge(d_j \vee c_p)$ we see that it, too, is the disjunction of queries in L (since $d_j \vee c_p$ is their meet in L). Thus, if we choose Γ s.t. $k \geq 2$: then all these lattices are strict subsets of L and we are guaranteed to make progress, eventually reaching $m = 1$.

One strategy is to choose $\Gamma = \{\hat{0}\}$: then $u_1, \ldots, u_k$ are all atoms of the lattice, and we have $k \geq 2$ (since in our lattice $\hat{0}$ is the meet of atoms). But, with this strategy we eventually reach every query in the lattice: in some cases, some of these queries are hard (as was d_{123} in Example 6.2), and they do not have an d-DNNF$^\neg$. In those cases we need a different strategy.

In general we have a set $Z \subseteq L$ of elements whose queries are hard: we will choose Γ such that, eventually, all elements in Z are removed from all sublattices. In Example 6.2, $Z = \{d_{123}\}$ and we have chosen $\Gamma = \{d_2\}$: hence $C = q_2$, $Q_1 = d_1, Q_2 = d_3$, $C \wedge Q_1 = d_{12}$, $C \wedge Q_2 = d_{23}$, and all five lattices have a single co-atom, none touches the toxic d_{123}. We need a strategy to choose Γ for a general Z. Here, it helps to notice that, by Theorem 2.7, we must have $\mu(z, \hat{1}) = 0$ for all $z \in Z$ (otherwise there is no compact d-DNNF$^\neg$). We set as goal to choose Γ such that, in all sublattices (for C, Q_i, and $C \vee Q_i$), if that sublattice contains some $z \in Z$, then $\mu(z, \hat{1}) = 0$ for all $z \in Z$. This ensures that all Z's will be eliminated: when we reach a lattice with a single coatom x, then $\mu(x, \hat{1}) = -1$ and we are guaranteed $x \notin Z$. The following procedure from [11] ensures this:

$$\begin{aligned}ZA &= \{a \mid a \text{ covers some element } z \in Z\}\\ E &= \{\hat{1}\} \cup \texttt{join-closure}(Z \cup ZA)\\ \Gamma &= \texttt{co-atoms}(E)\end{aligned}$$

DEFINITION 6.4. *Let $(L, \leq)$ be a lattice, and $Z \subseteq L$. We say that Z* is erasable *in L if either $Z = \emptyset$ or, denoting $L_0 =$ `meet-closure`(Γ), $Z \cap L_0$ is erasable in L_0.*

If Z is erasable, then forall $z \in Z$, $\mu(z, \hat{1}) = 0$. Indeed, in any lattice where $\hat{0}$ is not the meet of co-atoms, $\mu(\hat{0}, \hat{1}) = 0$ [24]: if we ever eliminate z, then z is not the meet of co-atoms in $[z, \hat{1}]$, hence $\mu(z, \hat{1}) = 0$. However, the converse is false: in the lattice of q_9 in Fig. 1, taking $Z = \{\hat{0}\}$, we have $ZA =$ all four atoms, $E = L$, $\Gamma =$ all four co-atoms, and its meet-closure is L.

This justifies the following definition of *d-safe queries*, analogous to *safe queries* Def. 2.6.

DEFINITION 6.5. *(1) Let $Q = d_1 \wedge \ldots \wedge d_k$, and $k \geq 2$. Then Q is* d-safe *if there exists a subset Z of its CNF lattice s.t. Z is erasable in L, and for every $x \in L - Z$, the query $\lambda(x)$ is d-safe. (2) Let $d = c_1 \vee \ldots \vee c_k$ be a disjunctive query, and let $d = d_0 \cup d_1$, where d_0 contains all components c_i without variables, and d_1 contains all components c_i with at least one variable. Then d is d-safe if d_1 has a separator w and $d_1[a/w]$ is d-safe.*

THEOREM 6.6. *If Q is d-safe, then it is in $UCQ(UCQ^\neg)$.*

Thus, the distinction between d-DNNFs and PTIME over *UCQ* boils down to the distinction between an erasable element of a lattice, and an element z for which $\mu(z, \hat{1}) = 0$. The former implies the latter, but the converse fails, as illustrated by q_9 in Table 1: q_9 is in $UCQ(P)$, but is not d-safe, because its minimal element $\hat{0}$ is labeled with h_3 (a hard query), and is not erasable (see Fig. 1). We conjecture that q_9 does not have a polynomial size d-DNNF$^\neg$.

7. CONCLUSION

We have studied the problem of compiling the query lineage into compact representations. We considered four compilation targets: read-once, *OBDD*, *FBDD*, and d-DNNF. We showed that over the query language of unions of conjunctive queries, these four classes form a strict hierarchy. For the first two classes we gave a complete characterization based on the query's syntax. For the last two classes we gave sufficient characterizations.

Our two main separation results (between $UCQ(OBDD)$ and $UCQ(FBDD)$, and between $UCQ(FBDD)$ and $UCQ(UCQ)$)

are the first examples of "simple" Boolean expressions (meaning: monontone, and with polynomial size DNFs) that separate those two classes.

We leave three open problems: complete characterizations of *FBDD* and d-DNNF, and separation of the latter from PTIME. Also, as future work, it would be interesting to investigate compact representations of lineages in other semirings described in [17].

8. REFERENCES

[1] B. Bollig and I. Wegener. A very simple function that requires exponential-size read-once branching programs. In *Information Processing Letters 66*, pages 53–57, 1998.

[2] B. Bollig and I. Wegener. Complexity theoretical results on partitioned (nondeterministic) binary decision diagrams. *Theory of Computing Systems*, 32:487–503, 1999. 10.1007/s002240000128.

[3] R. E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, pages 688–694, 1985.

[4] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[5] R. E. Bryant. On the complexity of vlsi implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. Comput.*, 40(2):205–213, 1991.

[6] M. Cadoli and F. M. Donini. A survey on knowledge compilation. *AI Commun.*, 10(3,4):137–150, 1997.

[7] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of 9th ACM Symposium on Thoery of Computing*, pages 77–90, Boulder, Colorado, May 1977.

[8] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[9] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDBJ*, 16(4):523–544, 2007.

[10] N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, New York, NY, USA, 2007. ACM Press.

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

[12] A. Darwiche. On the tractable counting of theory models and its application to belief revision and truth maintenance. *CoRR*, cs.AI/0003044, 2000.

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

[14] A. Gál. A simple function that requires exponential size read-once branching programs. *Information Processing Letters*, 62(1):13 – 16, 1997.

[15] J. Gergov and C. Meinel. Efficient boolean manipulation with obdd's can be extended to fbdd's. *IEEE Trans. Computers*, 43(10):1197–1209, 1994.

[16] M. C. Golumbic, A. Mintz, and U. Rotics. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. *Discrete Applied Mathematics*, 154(10):1465–1477, 2006.

[17] T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[18] T. J. Green. Containment of conjunctive queries on annotated relations. In *ICDT*, pages 296–309, 2009.

[19] V. Gurvich. Repetition-free boolean functions. *Uspekhi Mat. Nauk*, 32:183–184, 1977.

[20] D. Olteanu and J. Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, pages 326–340, 2008.

[21] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27:633–655, 1980.

[22] P. Sen, A. Deshpande, and L. Getoor. Read-once functions and query evaluation in probabilistic databases. In *VLDB*, 2010.

[23] D. Sieling and I. Wegener. Graph driven bdds - a new data structure for boolean functions. *Theor. Comput. Sci.*, 141(1&2):283–310, 1995.

[24] R. P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1997.

[25] V. Tannen. Provenance for database transformations. In *EDBT*, page 1, 2010.

[26] I. Wegener. *Branching programs and binary decision diagrams: theory and applications*. SIAM, 2000.

[27] I. Wegener. BDDs–design, analysis, complexity, and applications. *Discrete Applied Mathematics*, 138(1-2):229–251, 2004.