Theory Comput Syst (2013) 52:403–440
DOI 10.1007/s00224-012-9392-5

Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams

Abhay Jha · Dan Suciu

Published online: 6 March 2012

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 restricted to be polynomial in the data 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 with respect to these target compilation languages. Moreover, we give a complete characterization of the first two target languages, based on the query's syntax.

Keywords Probabilistic databases · Knowledge compilation · Binary decision diagrams · OBDD · FBDD · d-DNNF

A. Jha (✉) · D. Suciu
University of Washington, Seattle, WA, USA
e-mail: abhaykj@cs.washington.edu

D. Suciu
e-mail: suciu@cs.washington.edu

1 Introduction

The goal of *Knowledge Compilation* [6, 13, 27] is to represent a Boolean expression in a *language* 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, and 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 [13]. 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 to be fixed (and, in particular its size is 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 by *UCQ*(T) the class of *UCQ* queries whose lineage admits a compact compilation in T for all input databases: the precise definition of the term "compact" depends on the compilation target, but usually means that the target has polynomial size. There are two different ways of defining a compact compilation: (i) *Uniform*: A compact compilation can be found in polynomial time, (ii) *Non-Uniform*: A compact compilation exists, but there are no restrictions on how to find it. The first interpretation is more strict, but makes more practical sense if one is looking for tractable algorithms for compilation; all our upper bounds are for uniform compilation. The second interpretation is more useful in complexity theory, since the results show the expressibility, and limitation of different models of compilation/computation; all our lower bounds are for non-uniform compilation. Unless stated otherwise, we always assume that the compilation is uniform, except in Sect. 7 where we focus exclusively on non-uniform compilation.

Our first target is the class of *read-once expressions*, denoted *RO*. A read-once Boolean expression is an expression consisting of $\wedge$, $\vee$, $\neg$ operators in such a way that every input variable is used only once. A read-once Boolean formula is one that can be represented by a read-once expression. 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. Our second and third targets are *Ordered* and *Free Binary Decision Diagram*. 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 represents a Boolean function, as follows. Given an assignment to the Boolean variables, the value of the function is obtained by traversing the BDD starting at the root node, and at each internal node following either the 0 or the 1 edge, according to the value of that node's variable. The unique sink node reached at the end of the traversal gives the value (0 or 1) of the Boolean function under that assignment. 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, one can construct an *OBDD* (*FBDD*) for the lineage of q on D, in time polynomial in D; in particular, the resulting OBDD or FBDD also has size polynomial in D. Finally, our fourth target is the language of *deterministic-Decomposable Negation Normal Form*(d-DNNF) introduced by Darwiche [12] (see also [13]), which are DAGs whose leaves are labeled with literals (Boolean variables or negated Boolean variables), 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). We also allow for one more type of internal node: not ($\neg$). $UCQ(dDNNF)$ represents the class of queries s.t. one can construct a d-DNNF of its lineage in PTIME 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: $UCQ(RO) \subseteq UCQ(OBDD) \subseteq UCQ(FBDD) \subseteq UCQ(dDNNF) \subseteq UCQ(P)$.

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, over unions of conjunctive queries (UCQ), these classes no longer collapse. In fact they form a strict hierarchy:

$$UCQ(RO) \subsetneq UCQ(OBDD) \subsetneq UCQ(FBDD) \subsetneq UCQ(dDNNF) \subseteq UCQ(P)$$

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

Table 1 Several representative queries defined in Table 2. 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*(*dDNNF*) from $UCQ(P)$. *: assuming $FP \neq \#P$; $^?$: conjectured

Query	Syntactic properties	Membership in $UCQ(T)$, where T is				
		RO	*OBDD*	*FBDD*	*dDNNF*	P
q_1	inversion-free, read-once	yes	yes	yes	yes	yes
q_2	inversion-free	no	yes	yes	yes	yes
q_V	has inversion, all lattice points have separators	no	no	yes	yes	yes
q_W	lattice point $\hat{0}$ has no separator but is erasable	no	no	no	yes	yes
q_9	lattice point $\hat{0}$ has no separator and has $\mu = 0$ and is non-erasable	no	no	no$^?$	no$^?$	yes [11]
h_1	lattice point $\hat{0}$ has no separator and has $\mu \neq 0$	no	no	no	no*	no* [8, 11]

Table 2 Important queries used throughout the paper

$$\begin{aligned}
h_{30} &= R(x_0), S_1(x_0, y_0)\\
h_{31} &= S_1(x_1, y_1), S_2(x_1, y_1)\\
h_{32} &= S_2(x_2, y_2), S_3(x_2, y_2)\\
h_{33} &= S_3(x_3, y_3), T(y_3)\\
h_1 &= R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)\\
q_1 &= R(x_1), S(x_1, y_1) \vee T(x_2), S(x_2, y_2)\\
q_2 &= R(x_1), S(x_1, y_1), S(x_2, y_2), T(x_2)\\
q_V &= h_1 \vee R(x_3), T(y_3)\\
q_W &= (h_{30} \vee h_{32}) \wedge (h_{30} \vee h_{33}) \wedge (h_{31} \vee h_{33})\\
q_9 &= (h_{30} \vee h_{33}) \wedge (h_{31} \vee h_{33}) \wedge (h_{32} \vee h_{33}) \wedge (h_{30} \vee h_{31} \vee h_{32})
\end{aligned}$$

This means that the reason why certain queries can be computed in PTIME over probabilistic databases is no longer their read-once-ness, or any other efficient compilation method (We were not able to separate *UCQ*(*dDNNF*) 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 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*). First, *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, consider the query q_1 in Table 1, $q_1 = \exists x_1.\exists y_1.R(x_1), S(x_1, y_1) \vee \exists x_2.\exists y_2.T(x_2), S(x_2, y_2)$; in this paper we drop existential quantifiers when they are clear from the context, and write the query as $q_1 = R(x_1), S(x_1, y_1) \vee T(x_2), S(x_2, y_2)$. The query can also 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 our characterization of *UCQ*(*RO*) is unrelated to Gurvich's characterization of read-once Boolean expressions [16, 19], or to algorithms for checking read-once-ness in [21, 23]: these results apply to the Boolean formula, while our results apply directly to the query.

For *UCQ*(*FBDD*) and *UCQ*(*dDNNF*), 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 certain additional conditions (see Sect. 6). A sufficient condition for *UCQ*(*dDNNF*) 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*(*dDNNF*).

The most difficult results in this paper are the separation results *UCQ*(*OBDD*) $\subsetneq$ *UCQ*(*FBDD*) $\subsetneq$ *UCQ*(*dDNNF*); they are separated by the queries q_V and q_W respectively in Table 1. In each case we prove that the query does not belong to the smaller class, but that it belongs to the larger class. The separation between *UCQ*(*OBDD*) and *UCQ*(*FBDD*) extends even to the non-uniform definition of the complexity class. More precisely, we show $q_V \notin$ *UCQ*(*OBDD*), even if one assumes the non-uniform definition of *UCQ*(*OBDD*), and that $q_V \in$ *UCQ*(*FBDD*); similarly $q_W \notin$ *UCQ*(*FBDD*), even for a non-uniform definition of this class, and $q_W \in$ *UCQ*(*dDNNF*). These results are significant for the following reason. All lineage expressions for queries in *UCQ* are very simple: they are monotone, and have a DNF expression of polynomial size. This also applies to the lineage expressions of q_V and q_W. Thus, our lower bounds make a contribution 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]. But it is not very surprising that the "very simple" has no polynomial size FBDDs, since computing the probability of that formula is #P-hard: the "very simple" formula is precisely the lineage of the non-hierarchical query $R(x), S(x, y), T(y)$, for which computing the probability is #P-hard. In contrast, for both q_V and q_W one can compute the probability in polynomial time: hence, the fact that they do not admit polynomial size OBDDs or polynomial size FBDDs respectively is more surprising. On the other hand, we can use Bollig and Wegener's result to prove that, for every non-hierarchical query, its lineage has no polynomial size FBDD.

The lineage of the query q_V, that we use for the first major separation between *UCQ*(*OBDD*) and *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 $F(X_1, \dots, X_n) = X_{\sum_{i=1,n} X_i}$ (where $X_0 = 0$). Bryant [5] has shown that it has no polynomial size *OBDD*, while Gergov and Meinel [15] and independently Sieling and Wegener [24] 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 between *UCQ*(*FBDD*) and *UCQ*(*dDNNF*) 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 formula with exponential lower bound on d-DNNF size is presently known. In particular, we leave open the question whether $UCQ(dDNNF) \subsetneq UCQ(P)$. However, our algorithm in Sect. 6 suggests how d-DNNF 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*(*dDNNF*), 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 discuss results for non-uniform setting in Sect. 7 and conclude in Sect. 8.

2 Background and Definitions

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

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

$R(\bar{x})$ 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. Comma or $\wedge$ operator takes precedence over $\vee$. For example we write $R(x), S(x, y) \vee T(y)$ for $\exists x.(R(x) \wedge \exists y.S(x, y)) \vee \exists y.T(y)$.

A *query* is an expression as defined by (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 $\vee$. A conjunctive query admits an alternative representation, as a set of atoms, $R_1(\bar{x}_1), \dots, R_m(\bar{x}_m)$. Given two conjunctive queries q, q', the logical implication $q \Rightarrow q'$ holds iff there exists a homomorphism $q' \rightarrow q$ between their representations as sets of atoms [7].

Let D be a database instance. Denote X_t a distinct Boolean variable for each tuple $t \in D$. If $t \notin D$, $X_t \equiv \texttt{false}$. 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:

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

$$\Phi_{Q_1 \wedge Q_2} = \Phi_{Q_1} \wedge \Phi_{Q_2}, \qquad \Phi_{Q_1 \vee Q_2} = \Phi_{Q_1} \vee \Phi_{Q_2} \tag{3}$$

Given a probability $p(X_t) \in [0,1]$ for each Boolean variable X_t, we denote $P(\Phi_Q^D)$ the probability that the Boolean formula Φ_Q^D is true, when each Boolean variable X_t is set to 1 independently, with probability $p(X_t)$.

A *probabilistic database* is a pair (D, p) where D is a database and $p(t) \in [0,1]$ assigns a probability to each tuple $t \in D$. Given a Boolean query Q and a probabilistic database (D, p), the query probability $P(Q)$ is defined as $P(Q) = P(\Phi_Q^D)$, where in the latter expression each Boolean variable X_t has a probability equal to that of its corresponding tuple, $p(X_t) = p(t)$.

The *query evaluation problem on probabilistic databases* is the following: given a query Q and a probabilistic database (D, t), compute $P(Q)$. Usually we are interested in the *data complexity* of the query evaluation problem: for a fixed Q, determine the complexity of computing $P(Q)$ as a function of the input database (D, p).

Definition 1 *UCQ*(P) is the class of *UCQ* queries Q s.t. for any probabilistic database (D, p), 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 it here, since we will reuse some of the same concepts that characterize the class *UCQ*(P) to characterize various compilation targets.

We start by discussing connected queries. Consider a conjunctive query q given by the set of its atoms $R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m)$, and assume this representation is minimal, i.e., removing any atom results in an inequivalent query; it is known that this minimal representation is unique up to isomorphism [7]. Define the following undirected graph G: there is one node for each atom, and there is an edge from atom i to atom j if $R_i(\bar{x}_i)$ and $R_j(\bar{x}_j)$ share a common variable. We say that the query q is *connected* if the graph G is connected.

Lemma 1 *Suppose we restrict all conjunctive queries to be without constants. Let q be a conjunctive query. Then the following conditions are equivalent.* (1) *The query q is connected.* (2) *For every two conjunctive queries q_1, q_2, if $q_1 \wedge q_2 \Rightarrow q$ then either $q_1 \Rightarrow q$ or $q_2 \Rightarrow q$.* (3) *For every two conjunctive queries q_1, q_2, if $q \equiv q_1 \wedge q_2$ then either $q \equiv q_1$ or $q \equiv q_2$.*

Proof (1) implies (2). Assuming $q_1 \wedge q_2 \Rightarrow q$ we obtain a homomorphism $q \to q_1 \wedge q_2$. Since neither q_1 nor q_2 have constants, the homomorphism must map every variable in q to a variable in $q_1 \wedge q_2$. Since q is connected, the image of this homomorphism must be a connected graph, and, therefore, it is included either in q_1 or in q_2; this means that the homomorphism is either $q \to q_1$ or $q \to q_2$, implying either $q_1 \Rightarrow q$ or $q_2 \Rightarrow q$.

(2) implies (3). Assume $q_1 \wedge q_2 \equiv q$. In particular, $q_1 \wedge q_2 \Rightarrow q$, and by property (2) we have $q_1 \Rightarrow q$ or $q_2 \Rightarrow q$. Assuming the former, we derive $q_1 \Rightarrow q_1 \wedge q_2$, which further implies $q_1 \equiv q_1 \wedge q_2 \equiv q$. The latter case is symmetric.

(3) implies (1). Suppose that q is not connected. Suppose its minimal representation has m atoms. Let G be the graph corresponding to its minimal representation. We can partition its nodes into two sets, each with strictly less than m atoms, s.t. they do not share any variables. Thus, we have written $q = q_1 \wedge q_2$ where q_1, q_2 share no common variables and each has strictly less than m atoms. By condition (3) it follows that either $q_1 \Rightarrow q$ or $q_2 \Rightarrow q$. Assuming the former, we have $q \equiv q_1$, contradicting the fact that the minimal representation of q has m atoms. The latter case follows similarly too. □

The restriction to conjunctive queries without constants is necessary for the lemma to hold. Otherwise, consider the connected query $q = R(x, y), S(y, z)$, and $q_1 = R(x, a)$, $q_2 = S(a, z)$, where a is a constant and x, y, z are variables: we have $q_1 \wedge q_2 \Rightarrow q$ but neither $q_1 \Rightarrow q$ nor $q_2 \Rightarrow q$ holds.

We now define the key notions needed to characterize $UCQ(P)$, and which we need throughout this paper:

– A *component*, c, is a conjunctive query that is *connected*.
– Every *conjunctive query* can be written as a conjunction of components. That is, $q = c_1, c_2, \ldots, c_k$, s.t. c_i and c_j do not share any common variables, for all $i \neq j$. If $q = c_1, c_2, \ldots$ and $q' = c'_1, c'_2, \ldots$ are two conjunctive queries given as conjunction of components, and if they do not have any constants, then the logical 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, $d = c_1 \vee \cdots \vee c_k$. Given two disjunctive queries $d = c_1 \vee c_2 \vee \cdots$ and $d' = c'_1 \vee c'_2 \vee \cdots$, the logical implication $d \Rightarrow d'$ holds iff $\forall i.\exists j$ s.t. $c_i \Rightarrow c'_j$.
– A *UCQ* in DNF is a disjunction of conjunctive queries, $Q = q_1 \vee \cdots \vee q_m$. Given two queries in DNF, $Q = q_1 \vee q_2 \vee \cdots$ and $Q' = q'_1 \vee q'_2 \vee \cdots$, the logical implication $Q \Rightarrow Q'$ holds iff $\forall i.\exists j$ s.t. $q_i \Rightarrow q'_j$.
– A *UCQ* in CNF is a conjunction of disjunctive queries, $Q = d_1 \wedge \cdots \wedge d_m$. Given two queries in CNF, $Q = d_1 \wedge d_2 \wedge \cdots$ and $Q' = d'_1 \wedge d'_2 \wedge \cdots$, if they do not have any constants, then the implication $Q \Rightarrow Q'$ holds iff $\forall j.\exists i$ s.t. $d_i \Rightarrow d'_j$.

Obviously, any component is both a conjunctive query and a disjunctive query; also, every conjunctive query is a *UCQ* in DNF, and every disjunctive query is a *UCQ* in CNF.

The containment condition for DNF is due to Sagiv and Yannakakis [22]. The containment condition for CNF is from [11], and only holds if the queries have no constants. To see that this requirement is needed, consider the following three disjunctive queries: $d_1 = R(x, a)$, $d_2 = S(a, z)$, and $d = R(x, y), S(y, z)$, where a is a constant and x, y, z are variables. Define the following two *UCQ*'s: $Q = d_1, d_2$ and $Q' = d$. Both are in CNF, and $Q \Rightarrow Q'$, yet neither $d_1 \Rightarrow d$ nor $d_2 \Rightarrow d$ holds.

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, by repeatedly substituting a relation $R(A_1, \ldots, A_k)$ with 2 relations: $R_1 = \sigma_{A_i \neq a}(R)$ and $R_2 = \Pi_{A_1 \ldots A_{i-1} A_{i+1} \ldots A_k}(\sigma_{A_i = a}(R))$, for every attribute position i and every constant a that occurs in the query. For example, $R(x, a), S(x) \vee R(x, y), T(x)$ is rewritten as $R_2(x), S(x) \vee R_2(x), T(x) \vee R_1(x, y), T(x)$, where $R_1(x, y) = \sigma_{y \neq a}(R(x, y))$, and $R_2(x) = \pi_x(\sigma_{y=a}(R(x, y)))$.

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, such that x occurs before y. For example, $R(x, y), R(y, z), R(x, z)$ is ranked because $x < y \wedge y < z \wedge x < z$ is consistent, while $R(x, y), S(y, x)$ 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, by repeatedly substituting a relation $R(A_1, \ldots, A_k)$ with three relations $R_1 = \sigma_{A_i < A_j}(R)$, $R_2 = \Pi_{A_1 \ldots A_{j-1} A_{j+1} \ldots A_k}(\sigma_{A_i = A_j}(R))$, $R_3 = \Pi_{A_1 \ldots A_j \ldots A_i \ldots A_k}(\sigma_{A_i > A_j}(R))$, for every two attributes A_i, A_j s.t. $i < j$. We give here the main intuition by illustrating with $q = R(x, y), R(y, x)$, and refer to [11] for further details. Denoting $R_1(x, y) = \sigma_{x<y}(R)$, $R_2(x) = \pi_x(\sigma_{x=y}(R(x, y)))$, $R_3(y, x) = \pi_{yx}(\sigma_{x>y}(R))$, we rewrite the query as $R_2(x) \vee R_1(x, y), R_3(x, y)$. The new query is ranked.

The reason for the first transformation is to ensure that the implication criteria for CNF expressions hold. 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 equation (1)): the *scope* of the variable w is the subexpression Q.

Definition 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 3 A disjunctive query *d has a separator* if it can be written as $d \equiv \exists w.Q$, such that w is a root variable, and for every two atoms g, g' using the same relational symbol R, the variable w occurs in the same position in g and in g'. In this case the variable w is called a *separator variable* in the expression $\exists w.Q$.

The hardness part of characterizing $UCQ(P)$ consists of showing that, if a disjunctive query has no separator then it is hard for #P: hence, it cannot be in $UCQ(P)$ unless $FP = $#P. Recall that we assume all queries to be without constants, ranked, and minimized.

Theorem 1 ([11]) *Let d be a 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 trivially has no separator. For example, consider $d = R() \vee S(x)$: the first component, $R()$, has no variables, and clearly d has no separator, e.g. if we write it as $\exists x.(R() \vee S(x))$ then x is not a root variable. However, it is always easy to get rid of the components without variables, then apply Theorem 1. Indeed, write the disjunctive query as $d = d_0 \vee d'$ where d_0 contains all components without variables and d' contains all components with variables. Thus, d_0 is a disjunction $R_1() \vee R_2() \vee \cdots$ of zero-ary relational symbols, and d' is a disjunction of components $c_1 \vee c_2 \vee \cdots$, each having at least one variable. None of the symbols $R_i()$ occurs in any component c_j, otherwise c_j would not be connected. Thus, d_0 and d' are independent probabilistic events, and $P(d) = 1 - (1 - P(d_0))(1 - P(d'))$, in other words computing $P(d)$ reduces to computing $P(d')$, and 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 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 no two queries among $Q[a_1/w], \ldots, Q[a_n/w]$ have any tuple in common. Furthermore, 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 1 Query q_1 in Table 1 has a separator, because[2] $q_1 \equiv \exists w.(R(w), S(w, y_1) \vee T(w), S(w, y_2))$. We can compute its probability as $P(q_1) = 1 - \prod_i (1 - P(R(a_i), S(a_i, y_1) \vee T(a_i), S(a_i, y_2)))$. 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 in different positions in $S(w, y_1)$ and $S(x_2, w)$. Therefore, h_1 is hard for #P.

Consider a UCQ in CNF: $Q = d_1 \wedge \cdots \wedge d_k$. 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

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

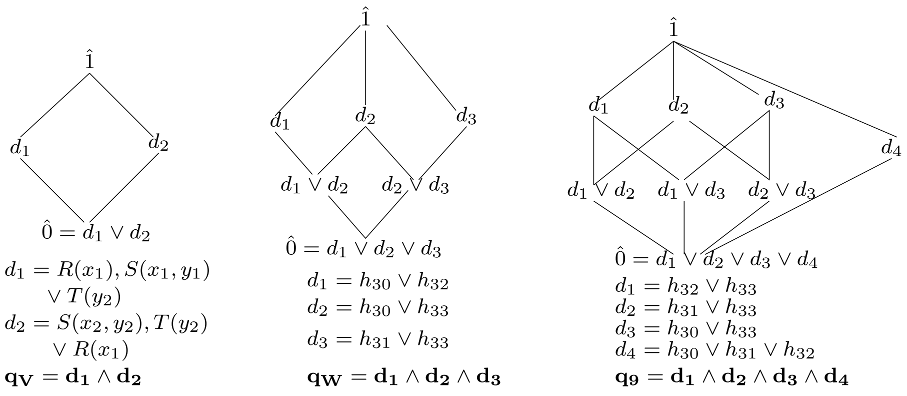

$\hat{1}$
d_1
d_2
$\hat{0} = d_1 \vee d_2$
$d_1 = R(x_1), S(x_1, y_1) \vee T(y_2)$
$d_2 = S(x_2, y_2), T(y_2) \vee R(x_1)$
$\mathbf{q_V = d_1 \wedge d_2}$
$\hat{1}$
d_1
d_2
d_3
$d_1 \vee d_2$
$d_2 \vee d_3$
$\hat{0} = d_1 \vee d_2 \vee d_3$
$d_1 = h_{30} \vee h_{32}$
$d_2 = h_{30} \vee h_{33}$
$d_3 = h_{31} \vee h_{33}$
$\mathbf{q_W = d_1 \wedge d_2 \wedge d_3}$
$\hat{1}$
d_1
d_2
d_3
d_4
$d_1 \vee d_2$
$d_1 \vee d_3$
$d_2 \vee d_3$
$\hat{0} = d_1 \vee d_2 \vee d_3 \vee d_4$
$d_1 = h_{32} \vee h_{33}$
$d_2 = h_{31} \vee h_{33}$
$d_3 = h_{30} \vee h_{33}$
$d_4 = h_{30} \vee h_{31} \vee h_{32}$
$\mathbf{q_9 = d_1 \wedge d_2 \wedge d_3 \wedge d_4}$

Fig. 1 CNF Lattices for the queries q_V, q_W, and q_9 from Table 2. 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; in q_9, the element $\hat{0}$ is not erasable

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 $d_\emptyset \equiv \texttt{false}$: all other elements correspond to non-trivial disjunctive queries d_s. The minimal element of the lattice is denoted $\hat{0}$, and corresponds to $\lambda(\hat{0}) = d_1 \vee \cdots \vee d_k$. Three examples are shown in Fig. 1.

We say x *covers* y if $y \leq x$ and there is no $z \in L$ s.t. $y < z < x$. We call x an *atom* if it covers $\hat{0}$; it is a co-atom if it is covered by $\hat{1}$. Denote by L^* the set of co-atoms. The *meet closure* of $S \subseteq L$ is the lattice: $\overline{S} = \{\bigwedge T \mid T \subseteq S\}$. Note that $\bigwedge \emptyset = \hat{1}$, the meet closure of any set S contains the maximal element $\hat{1}$.

The *Mobius function* of a lattice $(L, \leq)$ is the function $\mu_L : L \times L \to \mathbf{Z}$ defined by $\mu_L(x, x) = 1$, $\mu_L(x, y) = -\sum_{x<z\leq y} \mu_L(z, y)$. Note that $\mu_L(x, y) = 0$ whenever $x \not\leq y$. We will drop the L, i.e. denote μ_L by simply μ, henceforth when it is clear from the context. 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 4 (Safe queries) ([11]) (1) Let $Q = d_1 \wedge \cdots \wedge d_k$, 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 ([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 1.

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

Hierarchical Queries Let q be a conjunctive query, and denote $Vars(q)$ the set of variables used in the query and $at(x)$ the set of atoms containing a 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 5 Let Q be a query expression given by the grammar equation (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 see:

Proposition 1 *If Q is safe, then it is hierarchical.*

Proof By induction on the structure of Q. If $Q = d_1 \wedge \cdots \wedge d_k$, then each d_i corresponds to a co-atom x in the CNF lattice, hence $\mu(x, \hat{1}) = -1 \neq 0$, and therefore d_i must be safe, hence it is hierarchical by induction, hence Q is hierarchical. If $Q = d_0 \vee d_1$ and d_1 has a separator, $d_1 = \exists w.Q_1$, then $Q_1[a/w]$ is safe, hence it is hierarchical by induction, hence $\exists w.Q_1$ is hierarchical because w occurs in all atoms of Q_1. □

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 in general.

Inversions Inversions were first defined in [8]. In this paper we show how to use inversions to characterize $UCQ(RO)$ and $UCQ(OBDD)$. Let $Q = q_1 \vee \cdots \vee q_k$ 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 the following holds: x, y co-occur in some atom g , x', y' co-occur in some atom g', the atoms 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 $at(x) \nsubseteq at(y)$.

Definition 6 (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 occurring in the same non-hierarchical conjunctive query, such that $at(x) \cap at(y) \neq \emptyset$ and neither of the two sets $at(x)$, $at(y)$ contains the other. Consider the node (x, y) in the unification graph (such a node exists because $at(x) \cap at(y) \neq \emptyset$). Since we have both $x \succ y$ and $x \prec y$, the empty path starting and ending 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 equation (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: $\exists x_1, \exists x_2, \dots, \exists x_k$. In other words, each variable x_{i+1} is within the scope of x_i. Define π_g to be the permutation for which $g = R(x_{\pi_g(1)}, \dots, x_{\pi_g(k)})$.

Definition 7 A query expression Q given by the grammar equation (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. We have the following equivalence:

Proposition 2 *Q is inversion free iff it can be written as an inversion-free expression.*

Proof We first show how to write an inversion-free query as an inversion-free expression. Let Q be inversion free. We will define an order relation $x \ggg y$ on Q's variables s.t. (a) for every atom g, $\ggg$ is total over the set of variables occurring in g, and (b) if g, g' are two atoms with the same relation symbol R then the order imposed by $\ggg$ on the attributes of R is the same in g and g'. The order $\ggg$ gives us immediately the permutation π_g for every atom g; since Q can be written as a union of conjunctive queries, each of which is hierarchical, it follows that Q can be written as an inversion-free expression. To define $\ggg$, we first define a weaker relation $\gg$: $x \gg y$ if there exists a path in the unification graph from a node (x, y) to a node (x', y') s.t. $x' \succ y'$. Clearly $\gg$ is antisymmetric, because if we have both $x \gg y$ and $x \ll y$ then the graph has an inversion. We prove that $\gg$ is transitive. Indeed, suppose $x \gg y$ and $y \gg z$. By definition there exists a unification path $(y, z), (y_1, z_1), (y_2, z_2), \dots, (y_k, z_k)$ s.t. $y_k \succ z_k$. Consider the first edge of this path: there exists two atoms g, g_1 with the same relation name, g contains y, z, and g_1 contains y_1, z_1 on the same position. Then g must contain x as well (otherwise $x \prec y$ contradicting $x \gg y$). Denote x_1 the variable on the same position in g_1: since $x \gg y$ and $y \gg z$ we have $x_1 \gg y_1$ and $y_1 \gg z_1$. Repeating the same argument we find variables x_i s.t. $x_i \gg y_i$ and $y_i \gg z_i$, for $i = 1, k$. Since $y_k \succ z_k$, it also follows that $x_k \succ z_k$ (because $x_k \gg y_k$ implies $at(x_k) \supseteq at(y_k)$ hence $at(x_k) \not\subseteq at(z_k)$), proving that $x \gg z$. Therefore, $\gg$ defines a partial order on the set of variables. It is not a total order yet, because it may leave pairs of variables unordered. To make it a total order, we use the fact that the query Q is ranked, and define $x \ggg y$ to be: $x \gg y$ or ($x \not\ll y$ and there exists an atom g containing both x and y s.t. x occurs before y). It is easy to check that $\ggg$ is a partial order, and for any atom g it is total over its set of variables.

Now, suppose Q can be written as an inversion-free expression and still has an inversion from a node (x, y) to node (x', y') s.t. $x \succ y$ and $x' \prec y'$. Let π be the order in which variables are introduced in the inversion-free expression. Then x, y appear in the same order in π as x', y'. W.l.o.g., lets assume x occurs before y. Then x' occurs before y'. But we have $y' \succ x'$, hence x' couldn't have been a root variable when it was introduced which violates the fact that every inversion-free expression is also a hierarchical expression, a contradiction. This completes the proof. □

For example, consider q_1 in Table 1. On one hand we can write it as a union of conjunctive queries, $q_1 = R(x_1), S(x_1, y_1) \vee T(x_2), S(x_2, y_2)$. The unification graph has four nodes, $(x_1, y_1), (y_1, x_1), (x_2, y_2), (y_2, x_2)$, and two edges $((x_1, y_1), (x_2, y_2))$ and $((y_1, x_1), (y_2, x_2))$. We have both $x_1 \succ y_1$ (because $at(x_1) = \{R(x_1), S(x_1, y_1)\} \not\subseteq at(y_1) = \{S(x_1, y_1)\}$), and similarly $x_2 \succ y_2$. Hence, there is no inversion in the graph, and the query is inversion free. The proposition gives us an alternative way to see that, by writing the query as $q_1 = \exists x_1.R(x_1), \exists y_1.S(x_1, y_1) \vee \exists x_2.T(x_2), \exists y_2.S(x_2, y_2)$: 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, consider the query $h_1 = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$. Here $x_1 \succ y_1$ and $x_2 \prec y_2$, hence the edge $((x_1, y_1), (x_2, y_2))$ forms an inversion in the unification graph. One can see that we cannot write h_1 in a way that satisfies Definition 7: 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 obviously 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 8 *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 9 Let Q be a query expression given by the grammar equation (1). We say that Q is *hierarchical-read-once* if it is hierarchical (see Definition 5), 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 *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 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 Proposition 3 it cannot be written simultaneously as a hierarchical and read-once expression.

Theorem 3 *$Q \in UCQ(RO)$ 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.

Proof If: We prove by induction on the structure of a hierarchical-read-once expression Q that its lineage is read-once; this proof extends that of [20]. If $Q = Q_1 \vee / \wedge Q_2$, then Q_1, Q_2 have no relation symbols in common, and their lineage given by (3) is also read-once. If $Q = \exists x.Q_1$ then x must be a root variable, which implies that the lineages of $Q_1[a_1/x], \dots, Q_1[a_n/x]$ do not share any Boolean variables; hence, the lineage given by (2) is also read-once.

Only if: Assume $Q \in UCQ(RO)$; thus Φ_Q^D is read-once, for every database instance D; we show that Q must be equivalent to a hierarchical-read-once expression. First, we use Theorem 4 to argue that, if $Q \in UCQ(RO)$ then $Q \in UCQ(OBDD)$, hence Q is inversion-free. Referring to Definition 7, for every relation symbol R we denote π_R the permutation mapping the variable nesting order to the order in which they occur in an atom with relation symbol R.

We will construct a special database instance D: from the read-once-ness of Φ_Q^D, we will extract a hierarchical-read-once expression for Q. Let k be the total number of variables plus the total number of atoms in Q. We first construct the active domain for D. Start by choosing k constants $a_1, a_2, \dots, a_k$ called "root constants": these will be used to populate the attribute $\pi_R(1)$ of the relation R, for each relation symbol R. Next, choose k^2 constants a_{ij}, $1 \le i, j \le k$: these will populate the attribute $\pi_R(2)$ of each relation R, such that a_{ij} occurs only in those tuples that also contain a_i. Next, choose k^3 constants for the next level of the hierarchy, etc. This way we construct k^a tuples for a relation of arity a: note that the functional dependencies $\pi_R(i+1) \to \pi_R(i)$ hold for every relation R and every $i = 1, \dots, \mathit{arity}(R) - 1$.

Thus, we have fixed the database D. Next, we prove the following statement by induction: Let Q be any inversion-free query where the total number of variables plus atoms is at most k, and consider its lineage over our fixed database D, Φ_Q^D: if the lineage is read-once, then Q can be written as a hierarchical-read-once expression. Our induction proceeds on the structure of the read-once expression Φ_Q^D, abbreviated Φ_Q.

Case 1: Suppose $\Phi_Q = \phi_1 \wedge \phi_2$. Then ϕ_1, ϕ_2 have no common Boolean variables. We prove something stronger: that the variables come from disjoint sets of relations. Assume contrary, that both have a Boolean variable over the relational symbol R. To simplify the discussion we will assume R is unary; our argument extends in general too. Note that if m_1 is a minterm of ϕ_1 and m_2 is a minterm of ϕ_2, then $m_1 m_2$ must be a minterm of Φ_Q. This is because ϕ_1, ϕ_2 have no tuples in common, hence if another minterm $m_1' m_2' \Rightarrow m_1 m_2$, then $m_1' \Rightarrow m_1$ and hence m_1 couldn't have been a minterm of ϕ_1. Now, suppose $X_{R(a_1)}$ occurs in ϕ_1 and not in ϕ_2, and $X_{R(a_2)}$ occurs in ϕ_2 and not in ϕ_1. Then $X_{R(a_1)} X_{R(a_2)}$ occurs in some minterm in Φ_Q. Since the lineage is invariant under permutations of the active domain, for all $1 \le i < j \le k$ the term $X_{R(a_i)} X_{R(a_j)}$ occurs in some minterm of Φ_Q. Consider a third tuple of R, say $R(a_3)$. It must occur in either ϕ_1 or ϕ_2: assume w.l.o.g. it occurs in ϕ_2, and since Φ_Q has a minterm that contains $X_{R(a_2)} X_{R(a_3)}$, ϕ_2 must have a minterm that contains it. Hence, after conjoining with ϕ_1, we obtain a minterm in Φ_Q that contains $X_{R(a_1)} X_{R(a_2)} X_{R(a_3)}$, and, therefore, for every $1 \le i < j < l \le k$ there exists a minterm in Φ_Q containing $X_{R(a_i)} X_{R(a_j)} X_{R(a_l)}$. Repeating this argument leads us to conclude that Φ_Q has a minterm containing $X_{R(a_1)} \dots X_{R(a_k)}$: this is a contradiction because the minterms of Φ_Q cannot have more variables than the number of atoms in Q.

Thus, ϕ_1 contains only tuples over the relations $R_1, R_2, \dots$ and ϕ_2 contains only tuples over the relations $S_1, S_2, \dots$ Denote $Q_1 = Q[S_1 = S_2 = \dots = \mathtt{true}]$ the query obtained from Q by replacing all atoms referring to S_i with $\mathtt{true}$; similarly denote $Q_2 = Q[R_1 = R_2 = \dots = \mathtt{true}]$. The lineage of Q_1 on D is ϕ_1: hence, by induction hypothesis, Q_1 is equivalent to a hierarchical-read-once expression. Similarly Q_2. We prove now that $Q \equiv Q_1 \wedge Q_2$. Since every atom logically implies $\mathtt{true}$, we obtain immediately $Q \Rightarrow Q_1$ and $Q \Rightarrow Q_2$, hence $Q \Rightarrow Q_1 \wedge Q_2$. For the converse, write $Q_1 \wedge Q_2$ as a union of conjunctive queries $\bigvee_i q_i$, and let D_{q_i} be the canonical database for q_i: it suffices to prove that Q is true on D_{q_i} for every q_i. Both Q_1 and Q_2 are inversion-free, hence q_i is inversion free and the order π_R must be same in q_i and Q. Therefore the canonical database D_{q_i} satisfies all functional dependencies that hold in D and we can find an isomorphic copy of D_{q_i} in D, since we have chosen D "large enough". Set all Boolean variables corresponding to this copy to `true` and all others to `false`: we have $\phi_1 \wedge \phi_2 = \mathtt{true}$ (because $Q_1 \wedge Q_2$ is true on D_{q_i}), which implies $\Phi_Q = \mathtt{true}$, implying that Q is true on D_{q_i}.

Case 2: $\Phi_Q = \phi_1 \vee \phi_2$. We distinguish two cases:

Case 2.1: Every minterm in Φ_Q consists of tuples that have the same root constant. Thus, it may contain variables like $X_{R(a_1,a_{13})}X_{R(a_1,a_{15})}X_{S(a_1)}$ (same root constant a_1) but not $X_{R(a_1,a_{13})}X_{S(a_2)}$ (distinct root constants a_1, a_2). Then we claim Q must be a disjunctive sentence. Indeed, consider the DNF expression for $Q = q_1 \vee q_2 \vee \cdots$ and assume w.l.o.g. that q_1 is not connected, hence $q_1 = c \wedge c'$ where c, c' are two components. Then the lineage of q_1 includes minterms with mixed root constants, contradiction. Hence, Q is a disjunctive sentence. Now we use the fact that Q is inversion-free; in particular it has a separator, $Q = \exists x.Q_1$, and its lineage is $\Phi_Q = \bigvee_{i=1,k} \Phi_{Q_1[a_i/x]}$. Since Φ_Q is read-once, so is each $\Phi_{Q_1[a_i/x]}$ (since the latter is obtained from Φ_Q by setting to `false` all tuples with root constant a_j, for $j \neq i$). Hence, we apply induction hypothesis to $Q_1[a_i/x]$ and obtain a hierarchical-read-once expression: this proves that $\exists x.Q_1$ is hierarchical-read-once.

Case 2.2: $\Phi_Q = \phi_1 \vee \phi_2$ and there is at least one mixed minterm, containing tuples with two distinct root constants, say $X_{R_1(a_1,\bar{b})}X_{R_2(a_2,\bar{c})}$, and assume w.l.o.g. that this minterm appears in ϕ_1. We will show that all R_2-tuples occur in ϕ_1, i.e. ϕ_2 does not have R_2 tuples. We consider the case when R_1 and R_2 are distinct relational symbols: the case when they are the same symbol is similar. We use again the fact that Q is invariant under permutations of D to argue that Φ_Q must contain minterms that contain the tuples $X_{R_1(a_1,\bar{b})}X_{R_2(a_j,\bar{c}')}$, for any $j \leq k$. All these minterms must be in ϕ_1, since ϕ_2 may not contain $X_{R_1(a_1,\bar{b})}$. Thus, ϕ_1 must contain all tuples over R_2. With a similar argument, it must also contain all tuples over R_1.

Denote $R_1, R_2, \ldots$ the relation symbols that occur only in ϕ_1 and $S_1, S_2, \ldots$ the other symbols. By our assumption, at least one symbol is in the first list (because there is a mixed minterm in ϕ_1) and at least one symbol is in the second list (because ϕ_2 is not empty). We prove that no minterm contains tuples with relation symbols from both lists. Indeed, if the minterm is mixed, then we have seen that its symbols appear either only in ϕ_1 (hence they are all R_i symbols), or only in ϕ_2 (hence they are all S_j symbols). Suppose the minterm contains a unique root constant, say a_1, i.e. the minterm contains $X_{R_i(a_1,\ldots)}X_{S_j(a_1,\ldots)}$. Since the minterms are closed under isomorphisms of the domain, there are minterms containing $X_{R_i(a_2,\ldots)}X_{S_j(a_2,\ldots)}$, $X_{R_i(a_3,\ldots)}X_{S_j(a_3,\ldots)}$, etc. All these must belong to ϕ_1 (because they contain R_i); hence ϕ_1 contains all tuples over S_j, and therefore S_j must also be in the list $R_1, R_2, \ldots$

Define $Q_1 = Q[R_1 = R_2 = \cdots = \mathtt{false}]$ i.e. formula obtained from Q by setting all relation symbols R_i to false. Similarly, define $Q_2 = Q[S_1 = S_2 = \cdots = \mathtt{false}]$. We show that $Q = Q_1 \vee Q_2$, and since by induction hypothesis Q_1, Q_2 have a hierarchical-read-once expression, so does Q. First note that $Q_i \Rightarrow Q$, $i = 1, 2$, since `false` implies anything, hence $Q_1 \vee Q_2 \Rightarrow Q$. We will now show $Q \Rightarrow Q_1 \vee Q_2$, and here we use an argument similar to the above. Let $Q = \bigvee q_i$ and let D_{q_i} be a canonical database for q_i. Since D was chosen large enough, there exists an isomorphic copy of D_{q_i} in D, and, consequently, a minterm in Φ_Q consisting of the conjunction of its tuples. This minterm either consists of R_i tuples, hence $D_{q_i} \models Q_2$, or of S_j tuples, hence $D_{q_i} \models Q_1$. □

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 $Q \in \mathit{UCQ}(\mathit{RO})$. For example, one can check that q_2 in Table 1 is not in $\mathit{UCQ}(\mathit{RO})$, 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 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 Φ: $\Phi \equiv \Phi_u$ 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 $X_1, \dots, X_n$), 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 $X_1, \dots, X_n$ 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.

One can construct an *OBDD* for any read-once formula Φ in time 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 10 *UCQ*(*OBDD*) is the class of queries Q s.t. for every database D, one can construct an *OBDD* for Φ_Q^D in time polynomial in $|D|$.

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

Theorem 4 *$Q \in \mathit{UCQ}(\mathit{OBDD})$ iff it is inversion-free.*

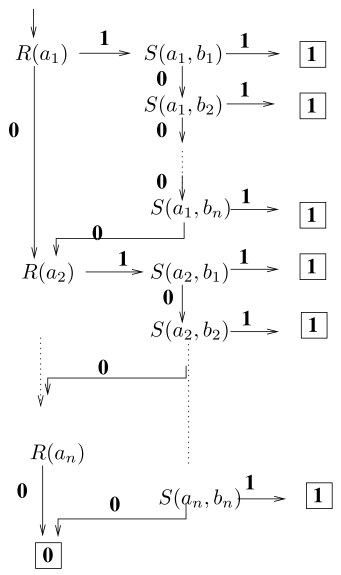


Fig. 2 *OBDD* for the query $R(x), S(x, y)$ (cf. [20])

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 $q_2 \in UCQ(OBDD) - UCQ(RO)$.

The significance of this result is the following. Olteanu and Huang [20] showed that for any hierarchical query Q, one can construct an *OBDD* for Φ_Q^D in time $O(|D|)$, 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, in two stages: first showing that one can construct in PTIME an *OBDD* for inversion-free formulae, and every query with inversion has exponential size *OBDD* over some database.

4.1 Tractable Queries

Given an *OBDD* of Φ over variables $\bar{x} = \{x_1, x_2, \ldots, x_n\}$ 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. $|\{\Phi_{x_{\pi(1)} \ldots x_{\pi(k)} = \bar{b}} \mid \bar{b} \in \{0, 1\}^k\}|$. The width of an *OBDD* is the maximum width at any level. If the width is 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 the *OBDD* is linear. Note that if the size of *OBDD* is polynomial, then the construction of the *OBDD* can be done in PTIME in our setting, since the lineage of a *UCQ* is always a monotone formula and checking the equivalence of monotone formulas is in PTIME.

We first need to define the notion of a shared BDD. A *shared* BDD for a set of formulas $\Phi_1, \Phi_2, \dots, \Phi_m$ 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, $1 \le i \le m$. This means a node reached by following the assignments $\bar{x}$ from the root can be thought of as representing a set of subformulae $\Phi_{1\bar{x}}, \Phi_{2\bar{x}}, \dots, \Phi_{k\bar{x}}$. 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 2 (cf. [27]) *Let Φ_1, Φ_2 be two Boolean functions and consider a fixed variable order Π. If there exist Π-OBDD 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 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 2, 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 (Definition 7). 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, $R_1 < R_2 < \cdots$. 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 $R(a_{\pi_R(1)}, a_{\pi_R(2)}, \dots, a_{\pi_R(k)})$ is associated to the string $a_1 a_2 \dots a_k R$. 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 2. If $Q = \exists x.Q_1$, then $\Phi_Q = \bigvee_{a \in ADom(D)} \Phi_{Q[a/x]}$. Let the active domain consist of $a_1 < a_2 < \cdots < a_n$, in this order. The *OBDD*s for $\Phi_{Q[a_1/x]}, \dots, \Phi_{Q[a_n/x]}$ 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. $\square$

Corollary 1 *If a set of components $c_1, c_2, \dots, c_m$ 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):

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

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 $h_{k0}, h_{k1}, \dots, h_{kk}$, 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* (Proposition 7). 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 $R(a_1), \dots, R(a_n)$, relation T has n tuples $T(b_1), \dots, T(b_n)$, 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 6 *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 $X_1, X_2, \dots$; 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 $v_1, v_2, \dots$ 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 $v_1 \dots v_h$ have been eliminated) is the size of the set $\{\Phi[(v_1 \dots v_h) = \bar{b}] \mid \bar{b} \in \{0,1\}^h\}$. 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 $v_1, v_2, \dots, v_h$ of Π. Define h to be the first level for which $h_1 + h_2 = n$. Denote $\mathbf{X}^{set} = \{X_i \mid X_i \in \{v_1, \dots, v_h\}\}$ 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 $\Phi_{k0} = \bigvee_{ij} X_i Z^1_{ij}$. Suppose there exists j s.t. $\forall i.(X_i \in \mathbf{X}^{set} \Rightarrow Z^1_{ij} \in \mathbf{Z}^{unset})$; 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 $Z^1_{ij} \in \mathbf{Z}^{set}$.

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. $Z^1_{ij} \in \mathbf{Z}^{set}$: then include (i, j) in S. Note that the cardinality of S is $n - h_2 = h_1$.

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

Next, for each $p = 1, \dots, k-1$, consider the *OBDD* for $\Phi_{kp} = \bigvee_{ij} Z^p_{ij} Z^{p+1}_{ij}$. Forall $(i, j) \in C_p$ we have $Z^p_{ij} \in \mathbf{Z}^{set}$ and $Z^{p+1}_{ij} \in \mathbf{Z}^{unset}$. 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 $\Phi_{kk} = \bigvee_{ij} Z^k_{ij} Y_j$. Forall $(i, j) \in C_k$ we have $Z^k_{ij} \in \mathbf{Z}^{set}$ and $Y_j \in \mathbf{Y}^{unset}$. Using the same argument, we obtain $n_k \geq 2^{c_k}$.

Putting everything together we obtain:

$$\sum_{i=1,k} n_i \geq \sum_{i=1,k} 2^{c_i} \geq k 2^{\frac{\sum_i c_i}{k}}$$

$$= k 2^{\frac{h_1}{k}} > k 2^{\frac{n}{2k}}$$

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

Proposition 7 *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 $\Omega(k 2^{n/2k})$.*

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 Proposition 6 this is sufficient to prove that Q has a no compact *OBDD*.

Proof Write $Q = \bigvee q_j$ in DNF, and let $(x_0, y_0), (x_1, y_1), \dots, (x_k, y_k)$ be an inversion in Q. Assume w.l.o.g. that the inversion is of minimal length: this implies there exist atoms $r, s_1, s'_1, \dots, s_k, s'_k, t$ with the following properties: $r \in at(x_0) - at(y_0)$,

$t \in at(y_k) - at(x_k)$, and for every $i = 1, k$, s_i contains x_{i-1}, y_{i-1}, s'_i contains x_i, y_i, they unify, and the unification equates $x_{i-1} = x_i$ and $y_{i-1} = y_i$. In particular, the atoms s_i and s'_i have the same relation symbol. Assume that x_i, y_i are variables in the query q_{j_i}, for $i = 0, k$. We assume that these k queries are distinct: if not, simply create a fresh copy of the query, creating new copies of its variables. Thus, q_{j_0} contains the atoms r, s_1, query q_{j_1} contains the atoms s'_1, s_2 and so on. Next, we perform variable substitutions in the queries $q_{j_0}, \dots, q_{j_k}$ in order to equate all variables in s_i and s'_i, except for $x_{i-1}, y_{i-1}, x_i, y_i$. In other words, all atoms along the inversion path have the same variables, except for the variables forming the actual inversion. For example, if the queries were $R(x_0, u_0)$, $S_1(x_0, y_0, u_0)$; $S_1(x_1, y_1, u_1)$, $S_2(x_1, y_1, u_1, v_1)$; $S_2(x_2, y_2, u_2, v_2), \dots$ then we equate $u_0 = u_1 = u_2 = \cdots$ and $v_1 = v_2 = \cdots$ This is possible in general because Q is ranked: we only equate variables between q_{j_i} and q_{j_l}, but not within the same q_{j_i}. We now construct the database D as follows. Its active domain consists of all constants $a_1, \dots, a_n, b_1, \dots, b_n$ and all variables $z \in Vars(q_{j_i})$ s.t. $z \neq x_i$, $z \neq y_i$, for $i = 0, k$. For each $i = 0, k$, and each $j = 1, n$, $l = 1, n$, let $q_{j_i}[a_j, b_l]$ denote the set of tuples obtained by substituting x_i with a_j and y_i with b_l. Define D to be the union of all these sets: $D = \bigcup_{i,j,l} q_{j_i}[a_j, b_l]$. Because of our earlier variable substitutions, $s_i[a_j, b_l]$ and $s'_i[a_j, b_l]$ are the same tuple: this tuple corresponds to the tuple $S_i(a_j, b_l)$ in the bipartite graph. Similarly, $r(a_j)$ and $t(b_l)$ correspond to the tuples $R(a_j)$ and $T(b_l)$ in the bipartite graph. Thus, the bipartite graph D_0 is isomorphic to a subset of the database D. Consider now any OBDD for Φ_Q^D, over a fixed variable ordering Π. We can obtain an OBDD for h_{ki} for every $i = 0, k$ as follows. Assume $0 < i < k$. Then we keep unchanged the Boolean variables corresponding to $S_i(a_j, b_l)$ and $S_{i+1}(a_j, b_l)$, (that is, the atoms $s'_i[a_j, b_l]$ and $s_{i+1}[a_j, b_l]$). All other Boolean variables corresponding to tuples in $q_{j_i}[a_j, b_l]$ are set to `true`; all remaining Boolean variables are set to `false`. Then the lineage Φ_Q^D becomes the lineage $\Phi_{h_{ki}}^{D_0}$. The case $i = 0$ is similar (here we keep unchanged the Boolean variables corresponding to $R(a_j)$ and $S_1(a_j, b_l)$), and so is the case $i = k$. Thus, we obtain $k + 1$ OBDD's for all queries h_{ki}, and all use the same variable order Π. The claim follows now from Proposition 6. $\square$

If Q has an inversion of length 0, then it is non-hierarchical and as we discuss later in Theorem 7 $Q \notin UCQ(FBDD)$, 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, 24]. 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 for which one can construct a *FBDD* in PTIME, but no polynomial size *OBDD* exists.

Definition 11 *UCQ*(*FBDD*) is the class of queries Q s.t. for any database D, one can construct an *FBDD* of Φ_Q^D in time polynomial in $|D|$.

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

Example 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:

$$\begin{aligned} q_V &= \big(R(x_1), S(x_1, y_1) \vee T(y_3)\big) \wedge \big(S(x_2, y_2), T(y_2) \vee R(x_3)\big) \\ &= d_1 \wedge d_2 \end{aligned}$$

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

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

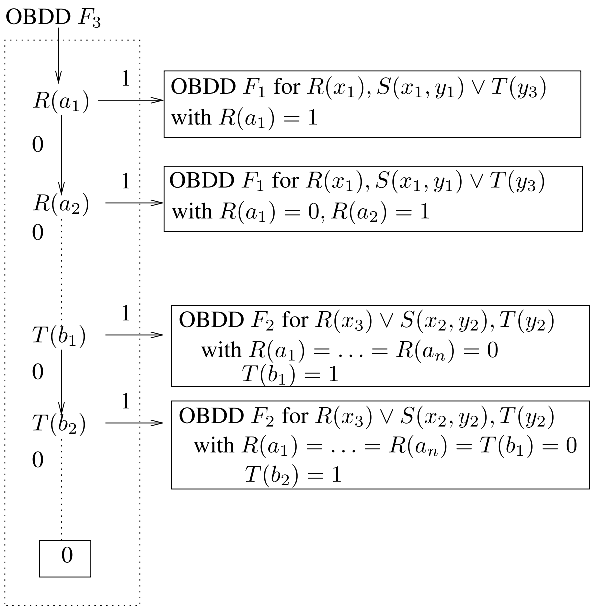
Each of d_1, d_2, d_3 is inversion-free, hence they have *OBDD*s, denote them F_1, F_2, F_3. Of course, F_1 and F_2 use different variable orderings and cannot be combined into an *OBDD* for q_V. Consider the database given by the bipartite graph (Sect. 4) and assume the following order on the active domain: $a_1 < \cdots < a_n < b_1 < \cdots < b_n$. 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 to continue with either F_2 or 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. Figure 3 illustrates the construction. The FBDD inspects the variables in this order: $X_{R(a_1)}, X_{R(a_2)}, \ldots, X_{R(a_n)}, X_{T(b_1)}, \ldots, X_{T(b_n)}$. (This is F_3.) Each edge $X_{R(a_i)} = 0$ leads to the next node, $X_{R(a_{i+1})}$, etc. Consider now an edge $X_{R(a_i)} = 1$. Here we know d_2 is true, but we still need to evaluate d_1. We create a new copy of F_1 where we set all variables $X_{R(a_1)}, \ldots, X_{R(a_{i-1})}$ to 0 (i.e. eliminate these nodes and redirect their incoming edge to their 0-child) and set $X_{R(a_i)}$ to 1. Then we connect the edge $X_{R(a_i)} = 1$ to the root node of this copy of F_1. Similarly, we connect an edge $X_{T(b_j)} = 1$ to the root of a copy of F_2 where we set $X_{R(a_1)} = \cdots = X_{R(a_n)} = X_{T(b_1)} = \cdots = X_{T(b_{j-1})} = 0$ and $X_{T(b_j)} = 1$. The result is an FBDD of size[3] $O(n^3)$ (since F_1, F_2 have sizes $O(n^2)$).

Thus:

Proposition 8 $q_V \in$ *UCQ*(*FBDD*) $-$ *UCQ*(*OBDD*).

[3] In this particular example one could reduce the size to $O(n^2)$ by sharing nodes among the multiple copies of F_1 and similarly for F_2.

Fig. 3 *FBDD* for the query q_V


OBDD F_3
$R(a_1)$
1
0
OBDD F_1 for $R(x_1), S(x_1, y_1) \vee T(y_3)$ with $R(a_1) = 1$
$R(a_2)$
1
0
OBDD F_1 for $R(x_1), S(x_1, y_1) \vee T(y_3)$ with $R(a_1) = 0, R(a_2) = 1$
$T(b_1)$
1
0
OBDD F_2 for $R(x_3) \vee S(x_2, y_2), T(y_2)$ with $R(a_1) = \ldots = R(a_n) = 0$ $T(b_1) = 1$
$T(b_2)$
1
0
OBDD F_2 for $R(x_3) \vee S(x_2, y_2), T(y_2)$ with $R(a_1) = \ldots = R(a_n) = T(b_1) = 0$ $T(b_2) = 1$
0

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, 24], 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 12 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 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* Definition 4 we define here *rf-safe queries*[4]:

[4] r is for restricted, since we do not have a full characterization yet.

Definition 13 (1) Let $Q = d_1 \wedge \cdots \wedge d_k$, 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 *Every rf-safe query is in UCQ*(*FBDD*).

Proof of Theorem 5 We start with a definition. Consider a monotone, Boolean expression, written as a disjunction of minterms: $\Phi = \bigvee_{i=1,n} T_i$. We also view Φ as a set of minterms, writing $T_i \in \Phi$. Each minterm T_i is a set of variables: thus, $T_i \Rightarrow T_j$ means that, as sets, $T_j \subseteq T_i$.

Definition 14 Let Φ, Φ' be two monotone, Boolean expressions, s.t. $\Phi' \Rightarrow \Phi$. We say that Φ *dominates* Φ' if for every minterm $T' \in \Phi'$, and for every Boolean variable $X \in T'$, one of the following conditions hold: (a) either X does not occur in Φ, or (b) there exists a minterm $T \in \Phi$ s.t. $X \in T$ and $T \subseteq T'$.

The following is easy to check:

Lemma 3 *If the disjunctive query d dominates d', then for any database D, the lineage Φ_d^D dominates $\Phi_{d'}^D$.*

Consider an FBDD for Φ, and a node x. Any path from the root to x corresponds to an assignment of a subset of Boolean variables; we call that an *assignment at x*.

Call an FBDD for Φ *greedy* if each sink node x labeled 1 has an additional label consisting of (a) a set $s \subseteq [n]$ and (b) and index i, such that, for any assignment at x, the following properties hold: $T_i = 1$ (i.e. all variables in T_i are set to 1 by the assignment). (2) For any $j \in s$, $T_j = 0$. (3) For any $k \notin s$, if the assignment sets a value of a variable in T_k, then that variable is in T_i (hence it is set to 1).

A greedy FBDD does exactly what the name says: it evaluates the Boolean DNF expression greedily. If a variable $X = 0$ then it skips all minterms that contain X: these minterms now belong to the set s. If a variable $X = 1$, then it continues to read only variables Y that co-occur with X in some minterm.

Let $\Phi = \bigwedge_{i=1,m} \Phi_i$, where each Φ_i is a monotone, Boolean expression. Let $(L, \leq)$ be the CNF-lattice for Φ constructed as follows. Its elements x are in one-to-one correspondence with Boolean expressions $\Phi_s = \bigvee_{i \in s} \Phi_i$, where $s \subseteq [m]$, up to logical equivalence (i.e. if $\Phi_{s_1} \equiv \Phi_{s_2}$ then they correspond to the same element x); $\lambda(x) = \Phi_s$ denotes the Boolean expression associated to x. And $x \leq y$ if $\lambda(y) \Rightarrow \lambda(x)$.

Lemma 4 *Let $\Phi = \bigwedge_{i=1,m} \Phi_i$, and $(L, \leq)$ be its CNF lattice. Suppose that, for all $x \leq y$, $\lambda(x)$ dominates $\lambda(y)$. Let F_x be a greedy FBDD of size n_x for $\lambda(x)$, for each $x \in L$. Then there exists a greedy FBDD for Φ, of size $O(m \cdot (\prod n_x))$.*

Proof (Sketch) We construct the FBDD by generalizing the idea of Example 2. Let $x_0 = \hat{0}$ be the smallest element in the lattice. The FBDD starts with F_{x_0}. Consider a sink node. If it is labeled 0, then we leave it labeled 0: we know that $\Phi_1 = \cdots = \Phi_m = 0$, hence so is Φ. If it is labeled 1, then we have two additional labels: a minterm T_i (known to be 1), and a set of minterms, TT (all known to be 0). Let $s \subseteq [m]$ be the set s.t. $i \in s$ iff T_i does not imply Φ_i: thus, from that sink node we have to continue to evaluate Φ_s. We assume, inductively, to have an FBDD for Φ_s. Then we modify it, by setting all variables in T_i to 1, and setting all other variables occurring in the set of minterms TT to 0: dominance ensures that the new FBDD still computes correctly Φ_s. □

The proof of Theorem 5 follows now directly from the last two lemmas. □

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

$$\begin{aligned} q_T = {} & \big(T_1(x), A(x), V(x,y,z) \vee T_3(z), C(z) \vee T_2(y), B(y), D(y)\big), \\ & \big(T_2(y), B(y), V(x,y,z) \vee T_1(x), A(x) \vee T_3(z), C(z)\big), \\ & \big(T_3(z), C(z), V(x,y,z) \vee T_1(x), A(x) \vee T_2(y), B(y), D(y)\big) \end{aligned}$$

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

Theorem 6 $q_W \notin$ *UCQ*(*FBDD*).

We will return to this query in the next section.

Proof Consider the following three queries:

$$\begin{aligned} d_1 &= R(x)S_1(x,y) \vee S_2(x,y)S_3(x,y) \\ d_2 &= R(x)S_1(x,y) \vee S_3(x,y)T(y) \\ d_3 &= S_1(x,y)S_2(x,y) \vee S_3(x,y)T(y) \end{aligned}$$

Thus, $q_W = d_1 \wedge d_2 \wedge d_3$. We first prove a surprising fact that given an FBDD for q_W, we can construct a shared FBDD for d_1, d_2, d_3. Note that in general it is not possible to split an FBDD into a shared FBDD: we exploit the properties of d_1, d_2, d_3 to do this.

Lemma 5 *Given any FBDD for q_W of size N, there exists a shared FBDD for d_1, d_2, d_3 of size polynomial in N and data instance.*

Call a node *shared* if for any two paths $\bar{x}, \bar{y}$ from root to the node, $d_{i\bar{x}} = d_{i\bar{y}}$, for $i = 1, 2, 3$. Hence if all nodes were shared, then the FBDD would also be shared. To prove the lemma, we show that if a node is not shared then the subformula represented by that node is actually an inversion-free query, so we could just replace the FBDD below that node with a shared OBDD. Hence, one can make every node shared, and therefore the FBDD shared.

Proof of Lemma 5 (Transformation into a shared FBDD) Each node x in an FBDD for F represents a Boolean expression F_x, obtained by setting some variables in F. If two paths $P1, P2$, lead to the same x, then $F[P1] = F[P2]$, where $F[P1]$ denotes the formula obtained by applying the partial assignment $P1$. Let F be the lineage of $q_W = d_1 \wedge d_2 \wedge d_3$, and write it as $F = G_1 \wedge G_2 \wedge G_3$, where G_i is the lineage of d_i. In general, two paths $P1, P2$ in the *FBDD*(q_W) that lead to the same node do not have to equate d_1, i.e. we may have $G_1[P1] \neq G_1[P2]$.

Definition 15 An FBDD for g_W is called *shared* if for any two paths $P1, P2$, if $F[P1] = F[P2]$ then for all $i = 1, 2, 3$, $G_i[P1] = G_i[P2]$.

This lemma is significant, because it says that we can transform *FBDD*(q_W) to compute each of the three queries d_1, d_2, d_3 separately. In general, this is not possible for an arbitrary conjunction of Boolean formulas. What makes this work in our case is that, whenever a node x fails to keep track separately of the three subqueries then the formula at x depends only on d_1, d_2 or only on d_2, d_3. Both these queries are inversion-free, hence in both cases we can construct a shared OBDD for the remaining computation of the two queries, replacing the rest of the *FBDD*(q_W)

Call a node x *shared* if for any two paths $P1, P2$ leading to x, we have $G_i[P1] = G_i[P2]$, for $i = 1, 2, 3$. We will leave shared nodes unchanged. If x is an non-shared node, then we show how to replace it with a shared OBDD for the remaining formulas. (Some of x's descendants may become unreachable, and they can be removed later.)

Suppose x is a non-shared node. Let $P1, P2$ be two paths leading to a node x in the *FBDD*(F). Then $F_x = F[P1] = F[P2]$. Suppose $G_1[P1] \neq G_1[P2]$.

Case 1: For every pair of constants a, b, the path $P1$ sets either $S_3(a, b) = 0$ or $T(b) = 0$. Then $G_2[P1]$ has only terms $R(a'), S_1(a', b')$ and similarly $G_3[P1]$ has only terms $S_1(a', b'), S_2(a', b')$. Denote the three queries below:

$$d_1 = R(x), S_1(x, y) \vee S_2(x, y), S_3(x, y)$$

$$e_2 = R(x), S_1(x, y)$$

$$e_3 = S_1(x, y), S_2(x, y)$$

Let D_1, D_2, D_3 be the set of tuples that are unset in $G_1[P1]$, $G_2[P1]$, and $G_3[P1]$. Then F_x is the conjunction of the lineages of d_1, e_2, e_3 on these three databases. Since d_1, e_2, e_3 are inversion-free, we can compute a shared OBDD that evaluates all three of them in parallel on D_1, D_2, D_3 (Corollary 1): thus, we have separated the FBDD at x and below x.

Case 2: There exists a pair of tuples $X = S_3(a,b)$, $Y = T(b)$ s.t. $P1$ leaves them either unset, or set to 1. Set $X = Y = 1$. Then G_2, G_3 become `true`, and therefore $G_1[P, X = 1, Y = 1] = F[P, X = 1, Y = 1]$, for $P \in \{P1, P2\}$. Since $F[P1] = F[P2]$ we obtain $G_1[P1, X = 1, Y = 1] = G_1[P2, X = 1, Y = 1]$. Hence, the only way in which $G_1[P1]$ and $G_1[P2]$ may differ is that either one has the term $S_2(a,b)$ and the other has $S_2(a,b), S_3(a,b)$, or one is `true` and the other has the term $S_3(a,b)$. The first case is impossible because it means that one of the two paths has set $S_3(a,b)$ to `true`. The second case implies $F_x = G_2[P] \wedge G_3[P]$ for $P \in \{P_1, P_2\}$ and we repeat the argument above.

This completes the case when G_1 differs on two paths. The case when G_3 differs is similar and omitted. So suppose $G_1[P1] = G_1[P2]$ and $G_3[P1] = G_3[P2]$. Then, we prove that we also have $G_2[P1] = G_2[P2]$. Indeed, this follows immediately by inspecting the three query lineages: G_1 is $\bigvee_{a,b} R(a)S_1(a,b) \vee S_2(a,b), S_3(a,b)$ and G_2 is $\bigvee_{a,b} R(a), S_1(a,b) \vee S_3(a,b), T(b)$. Thus, the subformula of $R(a), S_1(a,b)$ in $G_1[P1]$ is the same as that in $G_2[P1]$; since $G_1[P1] = G_1[P2]$, it means that this part of G_2 is the same in $P1$ and in $P2$. Similarly, from $G_3[P1] = G_3[P2]$ we conclude that the part $S_3(a,b), T(b)$ in G_2 is the same in $P1$ and $P2$. Hence, $G_2[P1] = G_2[P2]$. □

Let $h_1' = R(x_1), S(x_1, y_1), S(x_2, y_2), T(y_2)$. We can show that

Lemma 6 *$h_1' \notin UCQ(FBDD)$.*

We start at the root and choose 2^{n-1} paths (our database is bipartite as in Sect. 4) as follows: for each node we go in both directions if the given node isn't a prime implicant in the subformula at that node, otherwise we choose only the 0-edge. Then we exploit the fact that each of these paths has only *set* a limited number of variables to show that any two paths can differ on the assignment of only a few variables, hence most of them must end in distinct subformulas.

Proof of Lemma 6 Define Φ_1 to be $\bigvee_{i,j=1}^{n} r_i s_{i,j}$, Φ_2 as $\bigvee_{i,j=1}^{n} s_{i,j} t_j$ and F_n to be $\Phi_1 \wedge \Phi_2$. Then we show $\mathrm{FBDD}(\mathrm{F}_n) = n^{\Omega(\log(n))}$. We start at the root and choose 2^{n-1} paths $\mathcal{P}$ as follows: for each node we go in both directions if the given node isn't a prime implicant in either Φ_1 or Φ_2: we call such nodes *branching*; otherwise we choose only the 0-edge. We ignore redundant (i.e. the two branches for 0, 1 point to the same node) nodes. We stop a path after we have branched on $n-1$ nodes. We can always find $n-1$ nodes to branch on since our function cannot become 0 before we branched on $n-1$ nodes. This is because each branching variable can set at most n^3 minterms to 0. We have n^4 minterms to set to 0 and that can't be done with $n-1$ branching nodes. The set of variables on a branch p are denoted by $\mathit{Vars}(p)$ and $p(v)$ denotes the assignment of the variable v in p. The Boolean formula $f_p = \mathrm{F}_n[p(\mathbf{x})/\mathbf{x}]$, $\mathbf{x} = \mathit{Vars}(p)$ is obtained by applying the assignments on path p to F_n.

Definition 16 Given a Boolean formula f, call a variable v *determined*, if there exists $b \in \{0,1\}$ s.t. for any $p \in \mathcal{P}$ $f_p = f$ implies $v \in \mathit{Vars}(p)$ and $p(v) = b$. Conversely any variable $x \notin \mathit{Vars}(f)$ that is not *determined* is called *undetermined*.

Now the essence of the proof is that two paths that result in the same function can only differ on the *undetermined* variables. The following lemma characterizes the variables that can be *undetermined*.

Lemma 7 *Given a path p and a variable $x \notin Vars(f_p)$; x is undetermined for f_p iff $x = s_{i,j}$ for some $1 \leq i, j \leq n$ and*

1. *$r_i = 0$ or $s_{i,j1} = 1$ for some $1 \leq j1 \leq n$ on p and*
2. *$t_j = 0$ or $s_{i1,j} = 1$ for some $1 \leq i1 \leq n$ on p.*

Proof Follows immediately by doing a simple case analysis. □

Note that for any f_p, the number of paths q s.t. $f_q = f_p$ is bounded above by $2^{\#\text{undetermined variables in} f_p}$. This is because q and p can only differ on the assignment of undetermined variables. We now bound the number of undetermined variables for any path p. Let n_r, n_t be the number of $\mathbf{r}, \mathbf{t}$ variables set to 0 on path p; n_s be the number of $\mathbf{s}$ variables set to 1. Then by Lemma 7 the number of undetermined variables is at most $(n_r + n_s)(n_t + n_s)$. Let m_r, m_t, m_s similarly be the number of *branching* variables amongst n_r, n_t, n_s. Then, $m_s = n_s$ and $n_r \leq m_r + m_s$, $n_t \leq m_t + m_s$. The first equality $m_s = n_s$ holds because non-branching variables are always set to 0. Also a non-branching variable from $\mathbf{r}, \mathbf{t}$ is set to 0 iff it becomes a prime implicant; which can only happen if one of the variables from $\mathbf{s}$ is set to 1 and conversely setting one variable from $\mathbf{s}$ makes at most one prime implicant. This proves the second and third inequality.

Hence the number of undetermined variables is at most $(m_r + 2m_s)(m_t + 2m_s) \leq 4(m_r + m_s + m_t)^2$. Now consider all paths p where $l = m_r + m_s + m_t = \frac{\log(n)}{8}$. Consider the complete binary tree of depth $n - 1$ on the branching variables. Flip the edges coming out of $\mathbf{r}, \mathbf{t}$ nodes in the tree, i.e. 0 to 1 and vice-versa. We map any such path p to a path in this tree by choosing opposite assignment to variables from $\mathbf{r}, \mathbf{t}$. This is a 1-1 mapping. And the set of such paths in our tree is exactly the set of paths where the number of nodes tested to be 1 are l, which is $\binom{n-1}{l}$. Hence the size of the FBDD is at least $\frac{\binom{n-1}{l}}{2^{4l^2}}$ which for $l = \frac{\log(n)}{8}$ gives the required bound. □

Now to finish the proof, we finally show a reduction from a shared FBDD of d_1, d_2, d_3 to h'_1.

Lemma 8 *Given a shared FBDD for q_W of size N, there exists an FBDD for h'_1 of size at most N.*

This is, perhaps, the most surprising step, because it seems to reduce a query that is hard for #P (h'_1) to a query that is in PTIME (q_W). However, what we describe below is not a reduction: instead is a transformation of an FBDD.

The goal of the reduction is to set $S_1(a, b) = \neg S_2(a, b) = S_3(a, b)$ in *FBDD*(q_W): it can be easily seen (by inspecting the definition of the two queries) that the new FBDD computes h'_1. Since we have shown in Lemma 6 that h'_1 has no polynomial size FBDD, this completes the proof.

The difficulty of this step is to show that the FBDD has enough memory to remember if any of $S_1(a,b)$, $S_2(a,b)$, or $S_3(a,b)$ was set. We show that it does have enough memory, by using the fact that it is a *shared* FBDD, i.e. it computes the queries d_1, d_2, d_3 simultaneously.

Proof of Lemma 8 Start with the shared FBDD for q_W, given by Lemma 5. We want to enforce

$$S_1(a,b) = \neg S_2(a,b) = S_3(a,b) \tag{4}$$

for every tuple $S_1(a,b)$.

Modify each node x as follows. If it is a test for $R(a)$ or for $T(b)$, then make no change: it will continue to be a test for the same variable. If it is a test for $S_i(a,b)$, $i = 1, 3$, then we will either replace it with a test for $S(a,b)$, or will "know" the value of $S_i(a,b)$ and will follow only the 0 or the 1 branch. We give the details next.

Suppose node x tests for $S_1(a,b)$. Denote $G_{i,x}$, $i = 1, 2, 3$ the three Boolean expressions at x: the FBDD is shared, so it can keep track of them separately.

Step 1: Inspect $G_{3,x}$. Recall that the original G_3 contained $S_1(a,b), S_2(a,b) \vee S_3(a,b), T(b)$. There are a few cases: (a) $G_{3,x}$ does not contain $S_1(a,b)$ at all: then we know $S_2(a,b) = 0$ on all paths leading to x. (b) $G_{3,x}$ contains $S_1(a,b)$ (a prime implicant). Then on all paths to x must set $S_2(a,b) = 1$. In both (a) and (b) we know how to set $S_1(a,b)$ according to (4). (c) $G_{3,x}$ contains $S_1(a,b), S_2(a,b)$: then we know $S_2(a,b)$ is unset, and continue with step 2.

Step 2. At this point we know $S_2(a,b)$ is unset. Inspect G_1. The original G_1 was $R(a), S_1(a,b) \vee S_2(a,b), S_3(a,b)$. We know $S_2(a,b)$ is unset, hence the cases are: (a) $G_{1,x}$ does not contain $S_2(a,b)$: then we know $S_3(a,b) = 0$ on all paths to x. (b) $G_{1,x}$ contains $S_2(a,b)$ (a prime implicant). Then on all paths to x, $S_3(a,b) = 1$. In cases (a) and (b) we know how to set $S_1(a,b)$ according to the constraint equation (4). (c) $G_{1,x}$ contains $S_2(a,b), S_3(a,b)$. Then we know $S_3(a,b)$ is also unset, and it means we can read $S(a,b)$.

So far we have assumed that neither $G_{1,x}$ nor $G_{3,x}$ are `true`. Suppose $G_{3,x}$ is true. Let x be a maximal node where G_3 becomes true, i.e. the same doesn't hold for any of the children of x. Due to our previous construction when we transformed the FBDD into a shared one, the entire subgraph reachable from x is isolated. First, we consider all variables S_1, S_2, S_3 already set at x, and update the subgraph under x accordingly. We will need to cope with the variables read within this subgraph. Here, we first rewrite the query $d_1, d_2 = R(x), S_1(x,y) \vee S_2(x_1,y_1), S_3(x_1,y_1), S_3(x_2,y_2), T(y_2)$. Given our constraint equation (4), this query is equivalent to $R(x), S_1(x,y)$. We simply replace the entire subtree at x with an OBDD for $R(x), S(x,y)$. This completes the proof. □

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*.[5] 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, bipartite 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 7 (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 any interior node labeled with variable X in the *FBDD*, 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-DNNF 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 a d-DNNF$^\neg$.

Definition 17 *UCQ*(*dDNNF*) is the class of queries Q s.t. for any database D, one can construct a d-DNNF for $\neg\Phi_Q^D$ in time polynomial in $|D|$. *UCQ*(*dDNNF*$^\neg$) is the class of queries Q s.t. one can construct a d-DNNF$^\neg$ for Φ_Q^D in time polynomial in $|D|$.

$UCQ(FBDD) \subseteq UCQ(dDNNF) \subseteq UCQ(dDNNF^\neg) \subseteq UCQ(P)$; the first inclusion is can be shown to be strict.

Proposition 9 *$q_W \in UCQ(dDNNF) - UCQ(FBDD)$.*

[5]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 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 polynomial size d-DNNF but not *FBDD*. The previous separation of *FBDD* and d-DNNF 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 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"; in [2] the authors 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 has polynomial size *OBDD* and $\Phi_1 \wedge \Phi_2 \equiv \texttt{false}$. 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 criterion for a query $Q = d_1 \wedge \cdots \wedge d_m$, to be in $UCQ(dDNNF^{\neg})$, which is quite interesting because it explains the border between d-DNNF and PTIME in terms of lattice-theoretic concepts. We need to define some lattice theoretic concepts first. Let the CNF lattice of Q be $(L, \leq)$.

We now describe the construction algorithm. If $m = 1$ then Q is a disjunctive query; in this case it must have a separator (assuming FP $\neq$ #P (Theorem 1)), $d_1 = \exists w.Q_1$ and we write: $\neg Q = \bigwedge_{a \in ADom(D)} Q_1[a/w]$. The $\wedge$ operator is "decomposable", i.e. its children are independent.

If $m \geq 2$, we express $Q = Q_1 \wedge Q_2$, and consider the following derivation for $\neg Q$, where we write $\vee^d$ to indicate that a $\vee$ operation is disjoint:

$$\begin{aligned}
\neg(Q_1 \wedge Q_2) &= \neg Q_1 \vee \neg Q_2 \\
&= \neg Q_1 \vee^d [Q_1 \wedge \neg Q_2] \\
&= \neg Q_1 \vee^d \neg[\neg Q_1 \vee Q_2] \\
&= \neg Q_1 \vee^d \neg\big[(\neg Q_1 \wedge \neg Q_2) \vee^d Q_2\big] \\
&= \neg Q_1 \vee^d \neg\big[\neg(Q_1 \vee Q_2) \vee^d Q_2\big] \qquad (5)
\end{aligned}$$

The effect of the decomposition above is that it reduces Q to three subqueries, namely Q_1, Q_2, and $Q_1 \vee Q_2$, whose CNF lattices are meet-sublattices of L, obtained as follows. Let $Q = Q_1 \wedge Q_2$, where $Q_1 = d_{l1} \wedge d_{l2} \wedge \cdots$ and $Q_2 = d_{o1} \wedge d_{o2} \wedge \cdots$. Denote by $v_1, \ldots, v_m, u_1, \ldots, u_k$ the co-atoms of this lattice, such that $v_1, v_2, \ldots$ are the co-atoms corresponding to $d_{l1}, d_{l2}, \ldots$ and $u_1, u_2, \ldots$ are the co-atoms for $d_{o1}, d_{o2}, \ldots$.

- The CNF lattice of Q_1 is $\overline{M}$, where $M = \{v_1, \ldots, v_m\}$.
- The CNF lattice of Q_2 is $\overline{K}$, where $K = \{u_1, \ldots, u_k\}$.
- The CNF lattice of $Q_1 \vee Q_2 = \bigwedge_{i,j}(d_{li} \vee d_{oj})$ is $\overline{N}$, where $N = \{v_i \wedge u_j \mid i = 1, m; j = 1, k\}$. Here $v_i \wedge u_j$ denotes the lattice-meet, and corresponds to the query-union.

Note that each of the three lattices above, $\bar{M}, \bar{K}, \bar{N}$ is a strict subset of L.

This justifies the following definition of *d-safe queries*, analogous to *safe queries* Definition 4.

Definition 18 (1) Let $Q = Q_1 \wedge Q_2$. Then Q is *d-safe* if $Q_1, Q_2, Q_1 \vee Q_2$ are all d-safe. (2) Let $d = c_1 \vee \cdots \vee c_k$ be a disjunctive query, and let $d = d_0 \vee d_1$, 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 8 *If Q is d-safe, then it is in $UCQ(dDNNF^{\neg})$.*

To illustrate this algorithm, we now show how to construct a d-DNNF for q_W.

Proof of Proposition 9 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 express Q_W as $d_1 \wedge (d_2 \wedge d_3)$, and using (5) get

$$\neg Q_W = \neg d_1 \vee^d \neg\big[\neg\big(d_1 \vee (d_2 \wedge d_3)\big) \vee^d (d_2 \wedge d_3)\big]$$

d_1 is hierarchical-read-once, and $d_2 \wedge d_3$ is inversion-free; hence they both have compact d-DNNF. $d_1 \vee (d_2 \wedge d_3) = d_{12}$ is inversion-free and hence it also admits a compact d-DNNF. □

We prove now that every d-safe query is also safe. Fix a lattice L. Every non-empty subset $S \subseteq L - \{\hat{1}\}$ corresponds to a query, $\bigwedge_{u \in S} \lambda(u)$. We define a nondeterministic function NE that maps a non-empty set $S \subseteq L - \{\hat{1}\}$ to a set of elements $NE(S) \subseteq \overline{S}$, as follows. If $S = \{v\}$ is a singleton set, then $NE(S) = \{v\}$. Otherwise, partition S non-deterministically into two disjoint, non-empty sets $S = M \cup K$, define $N = \{v \wedge u \mid v \in M, u \in K\}$, and define $NE(S) = NE(M) \cup NE(K) \cup NE(N)$. Thus, $NE(S)$ is non-deterministic, because it depends on our choice for partitioning S. The intuition is the following: in order for the query $\bigwedge_{u \in S} \lambda(u)$ to be d-safe, all lattice points in $NE(S)$ must also be d-safe: they are "non-erasable".

Call an element $z \in L$ *erasable* if there exists a non-deterministic choice for $NE(L^*)$ that does not contain z. Recall that L^* is the set of co-atoms of L. The intuition is that, if z is erasable, then there exists a sequence of applications of rules from Definition 18, which avoids computing z; in other words, it "erases" z from the list of queries in the lattice for which it needs to compute the *d-DNNF*$^{\neg}$, and therefore Q_z is not required to be *d*-safe. We prove that only queries Q_z where $\mu_L(z, \hat{1}) = 0$ can be erased:

Lemma 9 *If z is erasable in L, then $\mu_L(z, \hat{1}) = 0$.*

Proof We prove the following claim, by induction on the size of the set S: if $z \notin NE(S)$, $z \neq \hat{1}$, then $\mu_{\overline{S}}(z, \hat{1}) = 0$ (if $z \notin \overline{S}$, then we define $\mu_{\overline{S}}(z, \hat{1}) = 0$). The lemma follows by taking $S = L^*$ (the set of all co-atoms in L).

If $S = \{v\}$, then $NE(S) = \{v\}$ and $\overline{S} = \{v, \hat{1}\}$: therefore, the claim hold vacuously. Otherwise, let $S = M \cup K$, and define $N = \{v \wedge u \mid v \in M, u \in K\}$. We have $NE(S) = NE(M) \cup NE(K) \cup NE(N)$. If $z \notin NE(S)$, then $z \notin NE(M)$, $z \notin NE(K)$, and $z \notin NE(N)$. By induction hypothesis $\mu_{\overline{M}}(z, \hat{1}) = \mu_{\overline{K}}(z, \hat{1}) = \mu_{\overline{N}}(z, \hat{1}) = 0$. Next, we notice that (1) $\overline{M}, \overline{K}, \overline{N} \subseteq \overline{S}$, (2) $\overline{S} = \overline{M} \cup \overline{K} \cup \overline{N}$ and (3) $\overline{M} \cap \overline{K} = \overline{N}$. Then, we apply the definition of the Möbius function directly, using a simple inclusion-exclusion formula:

$$\begin{aligned}\mu_{\overline{S}}(z, \hat{1}) &= -\sum_{u \in \overline{S}, z < u \leq \hat{1}} \mu_{\overline{S}}(u, \hat{1}) \\ &= -\Bigg(\sum_{u \in \overline{M}, z < u \leq \hat{1}} \mu_{\overline{S}}(u, \hat{1}) + \sum_{u \in \overline{K}, z < u \leq \hat{1}} \mu_{\overline{S}}(u, \hat{1}) - \sum_{u \in \overline{N}, z < u \leq \hat{1}} \mu_{\overline{S}}(u, \hat{1})\Bigg) \\ &= \mu_{\overline{M}}(z, \hat{1}) + \mu_{\overline{K}}(z, \hat{1}) - \mu_{\overline{N}}(z, \hat{1}) = -0 - 0 + 0 = 0 \qquad \square\end{aligned}$$

The lemma implies immediately:

Proposition 10 *For any UCQ query Q, if Q is d-safe, then it is safe. The converse does not hold in general: query q_9 is safe but is not d-safe.*

It is conjectured that $q_9 \notin UCQ(dDNNF^{\neg})$. Note that the proposition only states that q_9 is not d-safe, but it is not known whether d-safety is a complete characterization of $UCQ(dDNNF^{\neg})$.

Proof We prove the statement by induction on Q. We show only the key induction step, which is when $Q = \bigwedge_i d_i$, and L is its CNF lattice. Let $Z \subseteq L$ denote the nodes corresponding to d-unsafe queries: if Q is d-safe, then all elements in Z are erasable. This implies that $\forall z \in Z$, $\mu(z, \hat{1}) = 0$. Hence, we can apply Möbius' inversion formula to the lattice L, and refer only to queries that are d-safe; by induction hypothesis, these queries are also safe, implying that Q is safe.

We show that q_9 is safe, but is not d-safe. We will denote the lattice points with query $d_i \vee d_j \vee \cdots$ in Fig. 1 as $d_{ij\ldots}$. The query at $\hat{0}$ is the only hard query (since it is equivalent to h_3), and $\mu(\hat{0}, \hat{1}) = 0$. On the other hand, we prove that $\hat{0}$ cannot be erased. Indeed, the co-atoms of the lattice are $L^* = \{d_1, d_2, d_3, d_4\}$; given the symmetry of d_1, d_2, d_3, there are only three ways to partition the co-atoms into two disjoint sets $L^* = M \cup K$:

- $M = \{d_1, d_2, d_3\}$, $K = \{d_4\}$. In this case the lattice $\overline{M}$ is $\{\hat{0}, d_{12}, d_{13}, d_{23}, d_1, d_2, d_3, \hat{1}\}$, and $\mu_{\overline{M}}(\hat{0}, \hat{1}) = -1$, proving that this query is unsafe, and, therefore, d-unsafe.
- $M = \{d_1, d_2\}$, $K = \{d_3, d_4\}$. In this case the lattice $\overline{K}$ is $\{\hat{0}, d_3, d_4, \hat{1}\}$, and has $\mu_{\overline{K}}(\hat{0}, \hat{1}) = 1$, hence, by the same argument, is d-unsafe.
- $M = \{d_1\}$, $K = \{d_2, d_3, d_4\}$. Here, too, $\overline{K} = \{\hat{0}, d_{23}, d_2, d_3, d_4, \hat{1}\}$, and $\mu_{\overline{K}}(\hat{0}, \hat{1}) = 1$.

$\square$

7 Results on Non-uniform Classes

In this section we look at the non-uniform compact classes for different targets T.

Definition 19 For target $T \in \{\text{OBDD}, \text{FBDD}, \text{d-DNNF}\}$, denote by $UCQ^n(T)$ the class of all queries $Q \in UCQ$ s.t. Φ_D^Q has a compilation in T of size polynomial in $|D|$ for all D.

Note that in case of RO, a formula is either RO or not RO. Furthermore, thanks to the result due to Gurvich [19], this can be done in PTIME for monotone formulas that form the lineages of UCQ. On the other hand, a formula may have a compact OBDD, FBDD, or d-DNNF, but our algorithm may not be able to construct it. Hence $UCQ^n(T) \supseteq UCQ(T)$.

For OBDD though, it follows from the results of Sect. 4, that

Proposition 11 $UCQ^n(OBDD) = UCQ(OBDD)$.

The proof follows from Proposition 7, which implies that queries not in $UCQ(OBDD)$ do not have a polynomial size OBDD, i.e., $UCQ - UCQ(OBDD) \subseteq UCQ - UCQ^n(OBDD)$. Hence $UCQ^n(OBDD) \subseteq UCQ(OBDD)$, which means the two sets must be equal.

We do not have a full characterization for FBDD, d-DNNF, so we do not know if the same is true for them. The main separation results though still hold for non-uniform classes as well.

Proposition 12

$$UCQ^n(OBDD) \subsetneq UCQ^n(FBDD)$$
$$UCQ^n(FBDD) \subsetneq UCQ^n(dDNNF)$$
$$UCQ^n(FBDD) \subsetneq UCQ(P)$$

Proof We can construct an FBDD for q_V in PTIME (Theorem 5), but it doesn't always admit a compact OBDD (Proposition 7). This proves the first result. Similarly one can construct a d-DNNF for q_W in PTIME (Proposition 9), but it admits no polynomial size FBDD (Theorem 6). This proves the last two separations. □

8 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*(*dDNNF*), are the first examples of “simple” Boolean expressions (meaning: monotone, 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].

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

References

1. Bollig, B., Wegener, I.: A very simple function that requires exponential-size read-once branching programs. Inf. Process. Lett. **66**, 53–57 (1998)
2. Bollig, B., Wegener, I.: Complexity theoretical results on partitioned (nondeterministic) binary decision diagrams. Theory Comput. Syst. **32**, 487–503 (1999). doi:10.1007/s002240000128
3. Bryant, R.E.: Symbolic manipulation of boolean functions using a graphical representation. In: DAC, pp. 688–694 (1985)
4. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986)
5. Bryant, R.E.: On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. IEEE Trans. Comput. **40**(2), 205–213 (1991) doi:10.1109/12.73590
6. Cadoli, M., Donini, F.M.: A survey on knowledge compilation. AI Commun. **10**(3,4), 137–150 (1997)
7. Chandra, A., Merlin, P.: Optimal implementation of conjunctive queries in relational data bases. In: Proceedings of 9th ACM Symposium on Theory of Computing, Boulder, Colorado, pp. 77–90 (1977)
8. Dalvi, N., Suciu, D.: The dichotomy of conjunctive queries on probabilistic structures. In: PODS, pp. 293–302 (2007)
9. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDB J. **16**(4), 523–544 (2007)
10. Dalvi, N., Suciu, D.: Management of probabilistic data: foundations and challenges. In: PODS, pp. 1–12. ACM Press, New York (2007)
11. Dalvi, N.N., Schnaitter, K., Suciu, D.: Computing query probability with incidence algebras. In: PODS, pp. 203–214 (2010)
12. Darwiche, A.: On the tractable counting of theory models and its application to belief revision and truth maintenance. CoRR cs.AI/0003044 (2000)
13. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. **17**(1), 229–264 (2002)
14. Gál, A.: A simple function that requires exponential size read-once branching programs. Inf. Process. Lett. **62**(1), 13–16 (1997). doi:10.1016/S0020-0190(97)00041-0. http://www.sciencedirect.com/science/article/B6V0F-3SNV288-T/2/39afb175413bd7ee03397bb582be0161
15. Gergov, J., Meinel, C.: Efficient boolean manipulation with obdd’s can be extended to fbdd’s. IEEE Trans. Comput. **43**(10), 1197–1209 (1994)
16. Golumbic, M.C., Mintz, A., Rotics, U.: Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. Discrete Appl. Math. **154**(10), 1465–1477 (2006)
17. Green, T., Karvounarakis, G., Tannen, V.: Provenance semirings. In: PODS, pp. 31–40 (2007)
18. Green, T.J.: Containment of conjunctive queries on annotated relations. In: ICDT, pp. 296–309 (2009)
19. Gurvich, V.: Repetition-free boolean functions. Usp. Mat. Nauk **32**, 183–184 (1977)
20. Olteanu, D., Huang, J.: Using OBDDs for efficient query evaluation on probabilistic databases. In: SUM, pp. 326–340 (2008)
21. Roy, S., Perduca, V., Tannen, V.: Faster query answering in probabilistic databases using read-once functions. In: Proceedings of the 14th International Conference on Database Theory, ICDT’11, pp. 232–243. ACM, New York (2011). doi:10.1145/1938551.1938582

22. Sagiv, Y., Yannakakis, M.: Equivalences among relational expressions with the union and difference operators. J. ACM **27**, 633–655 (1980)
23. Sen, P., Deshpande, A., Getoor, L.: Read-once functions and query evaluation in probabilistic databases. In: VLDB (2010)
24. Sieling, D., Wegener, I.: Graph driven bdds—a new data structure for boolean functions. Theor. Comput. Sci. **141**(1&2), 283–310 (1995)
25. Tannen, V.: Provenance for database transformations. In: EDBT, p. 1 (2010)
26. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. SIAM, Philadelphia (2000)
27. Wegener, I.: BDDs–design, analysis, complexity, and applications. Discrete Appl. Math. **138**(1–2), 229–251 (2004)