A Dichotomy for Non-repeating Queries with Negation in Probabilistic Databases

Robert Fink
University of Oxford

Dan Olteanu
University of Oxford & LogicBlox Inc.

ABSTRACT

This paper shows that any non-repeating conjunctive relational query with negation has either polynomial time or #P-hard data complexity on tuple-independent probabilistic databases. This result extends a dichotomy by Dalvi and Suciu for non-repeating conjunctive queries to queries with negation. The tractable queries with negation are precisely the *hierarchical* ones and can be recognised efficiently.

1. INTRODUCTION

Charting the tractability frontier of query evaluation lies at the foundation of probabilistic databases [23]. Existing probabilistic database management systems, such as MystiQ [6] and MayBMS/SPROUT [15], fundamentally rely on query tractability results as they provide exact evaluation techniques for tractable queries and approximate techniques for intractable queries. Thus far, complexity dichotomies are known for non-repeating conjunctive queries (a.k.a. conjunctive queries without self-joins) [6] and union of conjunctive queries [9] on tuple-independent probabilistic databases: The data complexity of any query in each of these languages is either #P-hard or in polynomial time.

This paper shows a similar complexity dichotomy for queries with negation in probabilistic databases. All tractable queries are precisely the *hierarchical* ones and can be recognised in LOGSPACE in the size of the query.

The query language considered in this paper is that of *relational algebra* queries constructed using non-repeating relation symbols, equi-joins, projections, and difference (union not allowed). We denote this language by $1\mathrm{RA}^-$. By non-repeating we mean that a relation symbol can occur at most once in the query. We also discuss extensions of $1\mathrm{RA}^-$, in particular non-repeating *relational calculus* queries with or without union, and their implications for tractability.

Following earlier work on query tractability in probabilistic databases, this paper considers the *tuple-independent model*, where every tuple in the input database is annotated by a Boolean random variable stating the probability of the existence of that tuple, and any two such variables are independent. For more complex probabilistic models, query tractability is quickly lost: for block-independent disjoint tables, tractability analysis essentially falls back to that for tuple-independent databases by restricting joins to key attributes, while for the general model of probabilistic c-tables, already selection or projection queries can be #P-hard [23].

The following theorem states the main result of this paper:

THEOREM 1. *The data complexity of any $1RA^-$ query Q on tuple-independent databases is polynomial time if Q is hierarchical and #P-hard otherwise.*

We next define the hierarchical property. Let Q be a $1\mathrm{RA}^-$ query. We denote by $[A]$ the equivalence class of attribute A in Q, as enforced by join and difference operators; for instance, given relations over schemas $X(A)$ and $Y(B)$, both the join $X \bowtie_{A=B} Y$ and the difference $X -_{A\leftrightarrow B} Y$ under the attribute mapping $A \leftrightarrow B$ enforce that $[A] = [B]$.

DEFINITION 1. *A $1RA^-$ query Q is* hierarchical *if for every pair of attribute classes $[A]$ and $[B]$ that have no attributes in Q's result, there is no triple of relation symbols R, S, and T in Q such that R has attributes in $[A]$ and not in $[B]$, S has attributes in both $[A]$ and $[B]$, and T has attributes in $[B]$ and not in $[A]$.*

The hierarchical property can be decided for $1\mathrm{RA}^-$ queries in LOGSPACE [7, 12]. In the special case of queries without the difference operator, the notion of hierarchical queries defaults to the one introduced previously for non-repeating conjunctive queries and also characterises all tractable queries within that class [6]. While the syntactic characterizations are equivalent, the tractability and hardness proofs for $1\mathrm{RA}^-$ are non-trivial generalizations of those for conjunctive queries. Careful treatment is needed for the interaction of projection and difference operators, which can encode universal quantification and can lead to hardness already for cases where one single input relation is probabilistic and all other relations are deterministic. A further source of complexity is the lack of commutativity and associativity of the difference operator, which leads to many incomparable minimal hard query patterns made out of difference and join operators. We next exemplify techniques used in the hardness and tractability proofs.

Hardness proof for non-hierarchical queries

We prove that every non-hierarchical query Q has #P-hard data complexity by reduction from the #P-hard model-counting problem for positive bipartite DNF formulas: Given any

PODS'14, June 22–27, 2014, Salt Lake City, UT, USA.

ACM 978-1-4503-2375-8/14/06. $15.00. http://dx.doi.org/10.1145/2594538.2594549.

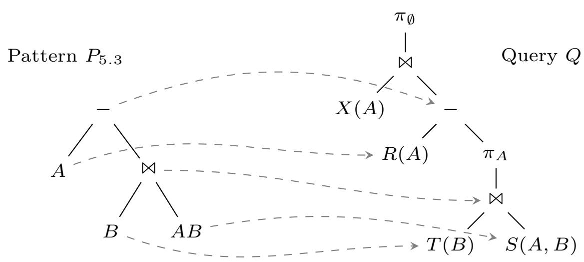


Figure 1: A query (right) matches a pattern (left).

R, X

A	$\mathbf{\Phi}$
1	$\top$
2	$\top$

S

A	B	$\mathbf{\Phi}$
1	x_1	$\top$
1	y_1	$\top$
1	y_2	$\bot$
2	x_1	$\top$
2	y_1	$\bot$
2	y_2	$\top$

$T \bowtie S$

A	B	$\mathbf{\Phi}$
1	x_1	$\neg\mathbf{x_1}$
1	y_1	$\neg\mathbf{y_1}$
1	y_2	$\bot$
2	x_1	$\neg\mathbf{x_1}$
2	y_1	$\bot$
2	y_2	$\neg\mathbf{y_2}$

$\pi_A(T \bowtie S)$

A	$\mathbf{\Phi}$
1	$\neg\mathbf{x_1} \vee \neg\mathbf{y_1}$
2	$\neg\mathbf{x_1} \vee \neg\mathbf{y_2}$

T

B	$\mathbf{\Phi}$
x_1	$\neg\mathbf{x_1}$
y_1	$\neg\mathbf{y_1}$
y_2	$\neg\mathbf{y_2}$

$R - \pi_A(T \bowtie S)$

A	$\mathbf{\Phi}$
1	$\mathbf{x_1 y_1}$
2	$\mathbf{x_1 y_2}$

Figure 2: Sketch of a hardness reduction for query Q in Figure 1. To avoid clutter (and in contrast to the naming convention used in Section 4), Q uses the same attribute names across multiple relations.

formula Ψ and the query Q, we construct an input database whose input tuples are annotated with variables in Ψ such that the result of Q becomes annotated with Ψ. To count the models of Ψ, we call an oracle that computes the probability P_Q of the query Q on a tuple-independent database where each variable has probability $1/2$. The number of models $\#\Psi$ is then $2^n P_Q$, where n is the number of variables in Ψ.

The starting point of our analysis is an alternative characterisation of the hierarchical property of queries via *matching* one of 48 minimal patterns; for each query, we craft a specific reduction depending on which pattern is matched. A *pattern* is a concise graphical representation of an infinite class of queries that satisfy certain structural properties. For example, the query Q in Figure 1 (right) is non-hierarchical as witnessed by the three relations R, S, T, and it matches the pattern shown in Figure 1 (left). Intuitively, the query matches the pattern, because the arrangement of the three relation symbols $R(A)$, $S(A,B)$, $T(B)$ and the operators connecting them in the query correspond to the structure of the attributes A and B and the operators in the pattern.

Example 1. We exemplify the reduction for query Q in Figure 1 and the formula $\Psi = x_1y_1 \vee x_1y_2$. The input relations and intermediate query results are shown in Figure 2. Each relation has a special column Φ that holds Boolean annotation formulas over variables in Ψ: Relations R and X have only true ($\top$) annotations, S has true and false ($\bot$) annotations, and all other relations have non-trivial annotations. Whereas the input relations are tuple-independent, the intermediate results exhibit correlated annotations. The query result is the projection on the empty set of the bottom-right relation; the annotation associated with the nullary result tuple is Ψ. Our filling of input tables may use variables as constants, e.g., for attribute B in tables S and T.

The reduction strategy is determined solely by the pattern matched by Q. The key challenge is to specify a database for the relation symbols that establish the match (R, S, T, in this example) such that they give rise to formula Ψ when Q is evaluated over this database. The remaining relation symbols (X, in this example) are populated such that they leave the annotations introduced by R, S, T unaltered. $\square$

Example 1 shows the power of negation: Our query Q can compute $\#\Psi$ for any positive 2DNF formula Ψ and is thus #P-hard already when *one* of its relations is uncertain (here, T) and all others are standard certain relations. In contrast, hardness can only be achieved for conjunctive queries when at least two input relations are uncertain.

Efficient algorithm for hierarchical queries

Our evaluation approach for hierarchical 1RA^- queries is to compile formulas annotating the query result into ordered binary decision diagrams (OBDDs), whose probabilities can be computed in time linear in their sizes [26]. While for hierarchical non-repeating conjunctive queries the OBDD sizes are independent of the query size and linear in the database size since the resulting formulas admit read-once representations [19], this is not the case for hierarchical 1RA^- queries, where the OBDD sizes remain linear in the database size, but may depend exponentially on the query size.

Example 2. The annotation of the result of the hierarchical Boolean query Q' on the database $\mathcal{D}$ in Figure 3 is

$$\begin{aligned}\Psi = & r_1\big[t_1(\neg u_1 \vee \neg v_1) \vee t_2(\neg u_1 \vee \neg v_2)\big] \vee \\ & r_2\big[t_1(\neg u_2 \vee \neg v_1) \vee t_2(\neg u_2 \vee \neg v_2)\big].\end{aligned}$$

The difference operator entangles the annotations of the participating relations in such a way that the resulting annotation Ψ is not a read-once formula; this entanglement is the pivotal intricacy introduced by the difference operator.

We show in Section 3 that for every tuple-independent database $\mathcal{D}$, the annotation of the result of Q' on $\mathcal{D}$ admits an OBDD of size $\mathcal{O}(|\mathcal{D}| \cdot f(Q))$, where $f(Q)$ is the OBDD *width* and only depends on the query size $|Q|$.

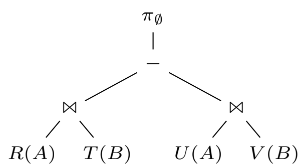
The underlying idea is to translate Q' into an equivalent disjunction of disjunction-free existential relational calculus queries such that each of the disjuncts gives rise to a compact OBDD and all OBDDs have compatible variable orders and can be combined efficiently into a single OBDD. We denote the language of such queries by $\text{RC}^\exists$. For Q', this translation yields the $\text{RC}^\exists$ query

$$\begin{aligned}Q_{RC} = & \underbrace{\exists_A\big(R(A) \wedge \neg U(A)\big) \wedge \exists_B T(B)}_{Q_1} \quad \vee \\ & \underbrace{\exists_A R(A) \wedge \exists_B\big(T(B) \wedge \neg V(B)\big)}_{Q_2}.\end{aligned}$$

The formulas annotating the results of the two queries Q_1 and Q_2 on the database $\mathcal{D}$ from Figure 3 are

$$\begin{aligned}\Psi_1 &= (r_1\neg u_1 \vee r_2\neg u_2) \wedge (t_1 \vee t_2)\\ \Psi_2 &= (r_1 \vee r_2) \wedge (t_1\neg v_1 \vee t_2\neg v_2).\end{aligned}$$

and clearly $\Psi_1 \vee \Psi_2 \equiv \Psi$. The $\text{RC}^\exists$ expressions Q_1 and Q_2 can be written such that (i) for each quantifier $\exists_X(Q')$ every relation symbol in Q' contains variable X, and (ii) the nesting order of the quantifiers is the same in both Q_1 and Q_2. Property (i) ensures that the formulas Ψ_1 and Ψ_2 admit OBDDs of size $\mathcal{O}(|\mathcal{D}|)$, as exemplified in the diagrams



R

A	$\mathbf{\Phi}$
1	$\mathbf{r_1}$
2	$\mathbf{r_2}$

T

B	$\mathbf{\Phi}$
1	$\mathbf{t_1}$
2	$\mathbf{t_2}$

U

A	$\mathbf{\Phi}$
1	$\mathbf{u_1}$
2	$\mathbf{u_2}$

V

B	$\mathbf{\Phi}$
1	$\mathbf{v_1}$
2	$\mathbf{v_2}$

$R \bowtie T$

A	B	$\mathbf{\Phi}$
1	1	$\mathbf{r_1 t_1}$
1	2	$\mathbf{r_1 t_2}$
2	1	$\mathbf{r_2 t_1}$
2	2	$\mathbf{r_2 t_2}$

$R \bowtie T - U \bowtie V$

A	B	$\mathbf{\Phi}$
1	1	$\mathbf{r_1 t_1 \neg(u_1 v_1)}$
1	2	$\mathbf{r_1 t_2 \neg(u_1 v_2)}$
2	1	$\mathbf{r_2 t_1 \neg(u_2 v_1)}$
2	2	$\mathbf{r_2 t_2 \neg(u_2 v_2)}$

Figure 3: Hierarchical query Q' and a database $\mathcal{D} = (R, T, U, V)$. The tables $R \bowtie T$ and $R \bowtie T - U \bowtie V$ show how the annotations of R, T, U, V are propagated by Q'.

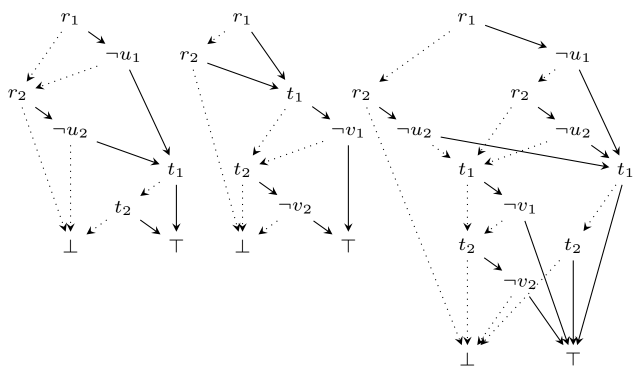


Figure 4: From left to right: OBDDs for Ψ_1, Ψ_2, and $\Psi = \Psi_1 \vee \Psi_2$ in Example 2.

of Figure 4. Property (ii) implies that these OBDDs can be constructed under the same global variable order, and it follows from classic results [26] that we can efficiently combine them via disjunctions and conjunctions. □

2. PRELIMINARIES

Due to lack of space, we defer the introduction of terminology for propositional formulas, their probabilistic interpretation when taken over Boolean random variables, as well as for probabilistic c-tables and annotation semirings to the extended version of this paper [12] and a recent monograph [23]. We next introduce a few necessary notions on the 1RA$^-$ and RC$^\exists$ query languages and OBDDs.

The relational algebra query language 1RA$^-$. We assume database schemas with unique attribute names. The set of attributes of a relation R is sch(R). A query Q is *non-repeating* if each relation symbols occurs at most once in Q.

1RA$^-$ is the class of non-repeating, union-free relational algebra queries composed of: *Relation symbols*; *Equi-join*: $Q_1 \bowtie_\rho Q_2$, where ρ is a conjunction of equality conditions $\rho = (A_1{=}B_1) \wedge \cdots \wedge (A_n{=}B_n)$ such that all A_i are attributes of Q_1 and all B_i are attributes of Q_2; *Projection*: $\pi_{A_1,\ldots,A_n}$ for attributes $A_1, \ldots, A_n$, or $\pi_{\bar{A}}$ for a set $\bar{A}$ of attributes; *Difference*: $Q_1 -_\rho Q_2$, where the attributes exported by Q_1 and Q_2 are $\{A_1, \ldots, A_n\}$ and $\{B_1, \ldots, B_n\}$ respectively, and ρ is the following conjunction of attribute mappings $(A_1{\leftrightarrow}B_1) \wedge \cdots \wedge (A_n{\leftrightarrow}B_n)$.

In $Q_1 \bowtie_\rho Q_2$ and $Q_1 -_\rho Q_2$, we write $A \in \rho$ to express that ρ contains an equality condition on A, and $(A{=}A') \in \rho$ or $(A{\leftrightarrow}A') \in \rho$ to express that ρ contains the equality condition $A{=}A'$ or $A{\leftrightarrow}A'$, respectively. When no confusion arises, we choose a schema with suggestive unique attribute names like $R(A_r), S(A_s, B_s), T(B_t)$ and then write the queries $R \bowtie_{A_r = A_s} S$ and $(R \bowtie T) -_{A_r \leftrightarrow A_s \wedge B_t \leftrightarrow B_s} S$ more concisely as $R \bowtie S$ and $(R \bowtie T) - S$.

We interchangeably use algebraic expressions and their ordered parse trees when referring to queries; in the latter case, the leaves are relations and inner nodes are algebra operators. Given a query Q and an operator Op in Q, Op has *even polarity* if the number of "$-$" operators between Op (exclusive) and the root of Q (inclusive), for which Op is a right descendant, is even, and has *odd polarity* otherwise. The pol function captures this notion: pol(Q, Op) is 1 if Op has odd polarity in Q, and 0 otherwise.

The equivalence class $[A]$ of an attribute A in Q is defined as in the introduction, where we consider the difference operators as joins on all attributes of its operands.

The attributes *exported* by a query Q, denoted $\mathcal{E}(Q)$, are defined recursively on the query structure:

$$
\begin{array}{ll}
\text{If } Q = Q_1 \bowtie_\rho Q_2, & \text{then } \mathcal{E}(Q) = \mathcal{E}(Q_1) \cup \mathcal{E}(Q_2) \\
\text{If } Q = Q_1 -_\rho Q_2, & \text{then } \mathcal{E}(Q) = \mathcal{E}(Q_1) \\
\text{If } Q = \pi_{\bar{A}}(Q_1), & \text{then } \mathcal{E}(Q) = \bar{A} \\
\text{If } Q = \sigma_\rho(Q_1), & \text{then } \mathcal{E}(Q) = \mathcal{E}(Q_1) \\
\text{If } Q = R, & \text{then } \mathcal{E}(Q) = \text{sch}(R)
\end{array}
$$

A query Q *exports* $[A]$ if there exists $A' \in [A]$ such that $A' \in \mathcal{E}(Q)$. Conversely, Q *does not export* $[A]$ if for all $A' \in [A]$ it holds that $A' \notin \mathcal{E}(Q)$. By $Q^{[A]}$, $Q^{[\neg B]}$, and $Q^{[A][\neg B]}$ we denote a query Q that exports $[A]$, does not export $[B]$, and respectively exports $[A]$ and not $[B]$.

We use $\pi_{-A_1,\cdots -A_n}(Q)$ as syntactic sugar for discarding $A_1, \ldots, A_n$, i.e., $\pi_{-A_1,\ldots,-A_n}(Q) = \pi_{\mathcal{E}(Q) - \{A_1,\ldots,A_n\}}(Q)$. Similarly, for an attribute A, the operator $\pi_{[A]}$ is a shortcut for $\pi_{A'}$ for any $A' \in [A]$, and the operator $\pi_{-[A]}$ denotes $\pi_{-A_1,\ldots,-A_n}$ where $[A] = \{A_1, \ldots, A_n\}$.

The relational calculus query language RC$^\exists$ is the class of queries that are expressions $\{\bar{H} \mid F\}$, where the query body F is a formula defined by the following grammar:

$$F ::= R(\bar{X}) \mid \exists_X(F_1) \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg(F_1),$$

and the query head $\bar{H}$ is the tuple of variable symbols that occur unquantified in F. In the sequel, we represent a query by its formula F alone. The *size* $|Q|$ of a query Q is the number of its relation symbols. A variable X is *root* in a query $\exists_X(Q)$ if X occurs in every relation symbol in Q [8].

Definition 2. *An RC$^\exists$ query Q is* canonicalised *if every occurrence of a relation symbol $R(\bar{X})$ in Q has the same query variables $\bar{X}$.*

Binary decision diagrams (BDDs) form a representation system for Boolean propositional formulas such as the annotations used in probabilistic databases. A BDD over a set $\mathbf{X}$ of variables is a directed acyclic graph where inner nodes are labeled with variables from $\mathbf{X}$ and terminal nodes are true ($\top$) and false ($\bot$). Each inner node has two outgoing edges, for the case its variable is set to true (solid edge) and false (dotted edge) respectively. Each root-to-leaf path in a BDD is a (possibly partial) assignment of variables.

A BDD is *ordered* (OBDD) if there is a total order Π on its variables such that the variables visited by each path are in Π-order. A *level* in an OBDD corresponds to all nodes labeled with the same variable. The *width*[1] of a BDD is the maximum number of edges crossing the section of the OBDD between the nodes of any two consecutive levels, where edges incident to the same node are counted as one.

In this paper, we make use of the following results:

LEMMA 1 ([26]). *Let Φ_1, Φ_2 be two formulas, Π be a fixed variable order on their variables, and O_1 and O_2 be Π-OBDDs of width w_1 and w_2 for Φ_1 and Φ_2, respectively. Then, Π-OBDDs for $\Phi_1 \wedge \Phi_2$ and for $\Phi_1 \vee \Phi_2$ can be constructed in time $O(|O_1|\cdot|O_2|)$ and have width at most $w_1 \cdot w_2$.*

Given an OBDD for a formula Ψ, the probability P_Ψ can be computed in time linear in the size of the OBDD.

EXAMPLE 3. Figure 4 shows three OBDDs under the same variable order $r_1, u_1, r_2, u_2, t_1, v_1, t_2, v_2$. Solid lines denote the *true*-edges and dotted lines the *false*-edges. The path $r_1 \xrightarrow{\top} \neg u_1 \xrightarrow{\perp} r_2 \xrightarrow{\perp} \perp$ encodes that under any truth assignment ν with $\nu(r_1) = \top$ and $\nu(\neg u_1) = \nu(r_2) = \perp$, the expression $\Psi_1 = (r_1 \neg u_1 \vee r_2 \neg u_2) \wedge (t_1 \vee t_2)$ becomes false. The width of the left two OBDDs is three: There are three edges with different sinks crossing from level of r_2 to $\neg u_2$ and respectively from t_1 to $\neg v_1$. The rightmost OBDD represents the disjunction of the two leftmost OBDDs (using the ITE algorithm [4]) and has width five. □

3. HIERARCHICAL 1RA$^-$ QUERIES

We show in this section the following result:

LEMMA 2. *Any hierarchical 1RA$^-$ query on tuple-independent databases has polynomial-time data complexity.*

PROOF. We prove the lemma via a sequence of steps:

Q_{RA} is a hierarchical 1RA$^-$ query

$\underset{\text{Lemma 3}}{\Rightarrow}$

Q_{RA} is equivalent to an RC$^\exists$ query Q_{RC} that is RC-hierarchical and $\exists$-consistent

$\underset{\text{Lemma 4}}{\Rightarrow}$

For any database $\mathcal{D}$, we can find an OBDD of size $\mathcal{O}(|\mathcal{D}| \cdot 2^{|Q_{RC}|})$ for the annotation Φ of the result $Q_{RC}(\mathcal{D})$

$\underset{\text{Corollary 1}}{\Rightarrow}$

The probability of Φ can be computed in $\mathcal{O}(|\mathcal{D}| \cdot 2^{|Q_{RC}|})$.

The reason for translating 1RA$^-$ queries to RC$^\exists$ queries is that relational calculus is more flexible and allows to unfold negated expressions as per $\neg(Q_1 \wedge Q_2) \equiv \neg Q_1 \vee \neg Q_2$. Since the 1RA$^-$ query Q_{RA} and the RC$^\exists$ query Q_{RC} are equivalent for any input database $\mathcal{D}$, the formulas annotating their results are equivalent too and thus have the same probability. We then show how Q_{RC}'s annotation can be compiled into an OBDD of size $\mathcal{O}(|\mathcal{D}| \cdot 2^{|Q_{RC}|})$.

The RC$^\exists$ query Q_{RC} is a disjunction of disjunction-free RC$^\exists$ expressions. In contrast to Q_{RA}, Q_{RC} may have repeating relation symbols. It is hierarchical in a syntactically more restricted sense:

[1]There is a different notion of BDD width in the literature that refers to the maximum number of nodes in any level.

DEFINITION 3. *An RC$^\exists$ query Q is* RC-hierarchical *if for every sub-query $\exists_X(Q')$ in Q it holds that X is root in Q'.*

Recall from Section 2 that a variable X is *root* in Q' if it appears in every relation symbol in Q', and that an RC$^\exists$ query is *canonicalised* if each relation symbol occurs only with the same variable symbols. In addition, the RC$^\exists$ queries obtained via rewriting can be written such that the nesting order of the existential quantifiers is the same over all of their disjunction-free expressions.

DEFINITION 4. *A canonicalised RC$^\exists$ query is* $\exists$-consistent *if there exists a total order $>_\exists$ of the variable symbols in Q such that $X >_\exists Y$ implies that there is no sub-query of the form $\exists_Y Q'(\exists_X)$ in Q.*

Intuitively, $\exists$-consistency for an RC$^\exists$ query that is a conjunction or disjunction of sub-queries means that these sub-queries have compatible join orders (i.e., non-contradicting $>_\exists$ orders). This means that their annotations, as well as the conjunction, disjunction, and negation of their annotations, can be compiled into OBDDs over the same variable order. In addition, the RC-hierarchical property effectively helps inferring from the order of quantifiers in the query a variable order for the OBDD that keeps its size only linear in the number of variables and thus in the database size but possibly exponential in the query size. We next illustrate these concepts via an example.

EXAMPLE 4. Consider the following three RC$^\exists$ queries:

$$Q_1 = \exists_A \big(M(A) \wedge \neg R(A)\big) \wedge \exists_B N(B)$$
$$Q_2 = \exists_A M(A) \wedge \exists_B \big(N(B) \wedge \neg T(B)\big)$$
$$Q_3 = \exists_A \big(M(A) \wedge U(A)\big) \wedge \exists_B \big(N(B) \wedge V(B)\big)$$

All three queries are RC-hierarchical since for each occurrence of $\exists_A$ and $\exists_B$, A and B, respectively, are root variables. Let us evaluate the queries over the database $\mathcal{D}$, viz:

M		N		R		T		U		V	
A	$\mathbf{\Phi}$	B	$\mathbf{\Phi}$	A	$\mathbf{\Phi}$	B	$\mathbf{\Phi}$	A	$\mathbf{\Phi}$	B	$\mathbf{\Phi}$
1	$\mathbf{m_1}$	1	$\mathbf{n_1}$	1	$\mathbf{r_1}$	1	$\mathbf{t_1}$	1	$\mathbf{u_1}$	1	$\mathbf{v_1}$
2	$\mathbf{m_2}$	2	$\mathbf{n_2}$	2	$\mathbf{r_2}$	2	$\mathbf{t_2}$	2	$\mathbf{u_2}$	2	$\mathbf{v_2}$

The annotations Φ_i of Q_i $(i = 1, 2, 3)$ evaluated on $\mathcal{D}$ are

$$\Phi_1 = (m_1\bar{r}_1 \vee m_2\bar{r}_2) \wedge (n_1 \vee n_2)$$
$$\Phi_2 = (m_1 \vee m_2) \wedge (n_1\bar{t}_1 \vee n_2\bar{t}_2)$$
$$\Phi_3 = (m_1u_1 \vee m_2u_2) \wedge (n_1v_1 \vee n_2v_2)$$

and can be represented by OBDDs of width 2 under the respective variable orders Π_1, Π_2, Π_3:

$$\Pi_1 : m_1, r_1, m_2, r_2, n_1, n_2$$
$$\Pi_2 : m_1, m_2, n_1, t_1, n_2, t_2$$
$$\Pi_3 : m_1, u_1, m_2, u_2, n_1, v_1, n_2, v_2$$

Now consider the query $Q_{123} = Q_1 \vee Q_2 \vee Q_3$; this query is canonicalised, RC-hierarchical, and $\exists$-consistent. The variable orders Π_1, Π_2, and Π_3 are compatible in the sense that they can be extended into an order Π_{123} over all variables:

$$\Pi_{123} : m_1, r_1, u_1, m_2, r_2, u_2, n_1, t_1, v_1, n_2, t_2, v_2$$

In the light of Lemma 1, the OBDDs of Φ_1, Φ_2, and Φ_3 can be combined to yield an OBDD of width at most 2^3 for the annotation $\Phi_1 \vee \Phi_2 \vee \Phi_3$ of query Q_{123}. □

3.1 From 1RA$^-$ to RC$^\exists$

At the core of the evaluation algorithm for hierarchical 1RA$^-$ queries is a rewriting of 1RA$^-$ queries into equivalent safe RC$^\exists$ queries. The rewriting procedure $[\![\cdot]\!]$ is the standard recursive inside-out translation from relational algebra to safe relational calculus (Lemma 5.3.11, [1]), with the addition that after each recursive translation step we "flatten" the resulting RC$^\exists$ query as follows:

- Every $\exists$ operator is pushed as deep as possible in the RC$^\exists$ query without pushing it past a $\neg$ operator: $\exists_X$ distributes over disjunctions and is pushed past conjuncts in which X does not appear. Lemma 3 shows that every $\exists_X$ operator can be pushed until X becomes root, i.e., X occurs in all relation symbols in its scope.
- Every $\neg$ operator is recursively pushed (as per $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$ and its dual) as deep as possible in the RC$^\exists$ query without pushing it past an $\exists$ operator.
- Conjunctions of disjunctions are eagerly expanded into disjunctions of conjunctions as per

$$(A \vee B) \wedge (C \vee D) \rightarrow AB \vee AC \vee BC \vee BD.$$

Our translation has several desirable properties:

Lemma 3. *For any hierarchical 1RA$^-$ query Q_{RA}, the translated RC$^\exists$ query $Q_{RC} = [\![Q_{RA}]\!]$ satisfies the following:*

(a) Q_{RC} is equivalent to Q_{RA}.

(b) Q_{RC} is canonicalised.

(c) Q_{RC} is a disjunction of disjunction-free RC$^\exists$ queries.

(d) For every variable X in Q_{RC}, Q_{RC} has no sub-query of the form $\exists_X(Q) \wedge Q'(\exists_X)$; here, $Q(\exists_X)$ denotes a query Q in which $\exists_X$ occurs.

(e) Q_{RC} is RC-hierarchical.

(f) The quantifiers in Q_{RC} can be ordered such that Q_{RC} is $\exists$-consistent.

Condition (d) permits sub-queries of the form $\neg\exists_X(Q) \wedge \neg\exists_X(Q')$ or $\exists_X(Q) \vee \exists_X(Q')$, but disallows, e.g., $\exists_X(Q) \wedge \exists_X(Q')$, $\exists_X(Q) \wedge \neg\exists_X(Q')$, $\exists_X(Q) \wedge \neg\exists_Y(Q'' \wedge \neg\exists_X(Q'''))$.

Example 5. Consider the following two 1RA$^-$ queries:

$$Q_a = \pi_\emptyset\Big[M(A) \bowtie N(B) - \big[R(A) \bowtie T(B) - U(A) \bowtie V(B)\big]\Big]$$

$$Q_b = \pi_\emptyset\Big[\pi_A\big(M(A) \bowtie N(B)\big) - \pi_A\big[R(A) \bowtie T(B) - U(A) \bowtie V(B)\big]\Big].$$

Query Q_a translates to Q_{123} from Example 4 (subsumed sub-queries removed to avoid clutter). Q_b is similar to Q_a, but with additional projections on A on both sides of the top-most difference operator, and translates to

$$\begin{aligned}[\![Q_b]\!] = {} & \exists_A\big(M(A) \wedge \neg R(A)\big) \wedge \exists_B N(B) \quad \vee \\ & \exists_A M(A) \wedge \exists_B N(B) \wedge \neg\exists_B T(B) \quad \vee \\ & \exists_A\big(M(A) \wedge U(A)\big) \wedge \exists_B N(B) \wedge \neg\exists_B\big(T(B)\neg V(B)\big).\end{aligned}$$

Like Q_{123}, the RC$^\exists$ query $[\![Q_b]\!]$ has three disjuncts, but the nesting orders of $\neg$ and $\exists_B$ operators in the second and third conjuncts differ from the corresponding order in Q_{123}. The translations of Q_a and Q_b satisfy Lemma 3: For example, for every operator $\exists_A$ (or $\exists_B$), A (or B) is a root variable in its scope (Property (e)), and the nesting orders of $\exists_A$ and $\exists_B$ are consistent in all sub-queries (Property (f)). □

The query translation can lead to large RC$^\exists$ queries: A conservative upper bound on their sizes would be a non-elementary function of the size of the input 1RA$^-$ query, explained by the rapid increase in the size and number of disjuncts when pushing down negations, projections, and conjunctions. A singly-exponential upper bound holds for 1RA$^-$ queries where for all projections $\pi_{-X}(Q)$ that are right descendants of a difference operator, attributes in the equivalence class $[X]$ occur in all relation symbols of Q (i.e., X is root in Q). The query Q_a in Example 5 satisfies this condition trivially, since it has no projection that is a right descendant of a difference operator. While this conservative upper bound suffices for the *data*-complexity argument in Lemma 2 since the blowup is in the size of the query only, it is not practical and better translation algorithms, which avoid the generation of subsumed disjuncts, are called for.

3.2 OBDD Construction

The last step in the proof of Lemma 2 is the OBDD compilation of the annotation Φ of the RC$^\exists$ query Q_{RC} obtained from Q_{RA} as per Lemma 3. This OBDD has a total order Π over the Boolean variables annotating the input tuples that can be derived from the structure of Q_{RC}. Let us first exemplify the construction of this order.

Example 6. Consider the query

$$Q = \exists_X\big[R(X) \wedge \exists_Y(S(X,Y) \wedge \neg T(X,Y))\big].$$

Since X is a root variable, the OBDDs for different values of X are independent and can be concatenated. For each value a in the active domain of X, we construct the OBDD for the query $R(a) \wedge \exists_Y(S(a,Y) \wedge \neg T(a,Y))$; one good variable order for this OBDD is the sequence of the annotation of $R(a)$ and all annotations of $S(a,b)$ and of $T(a,b)$ for all values b in the active domain of Y. If we write $R(1)$ for the annotation of tuple (1) in R, and similarly for S and T (all values being positive integers), then the overall variable order is

$$\begin{aligned}& R(\mathbf{1}), S(\mathbf{1},1), T(\mathbf{1},1), S(\mathbf{1},2), T(\mathbf{1},2), S(\mathbf{1},3), T(\mathbf{1},3)\ldots, \\ & \qquad \text{(tuples with } X=\mathbf{1}) \\ & R(\mathbf{2}), S(\mathbf{2},1), T(\mathbf{2},1), S(\mathbf{2},2), T(\mathbf{2},2), S(\mathbf{2},3), T(\mathbf{2},3)\ldots, \\ & \qquad \text{(tuples with } X=\mathbf{2}) \text{ and so on.}\end{aligned}$$

The annotations are ordered in lexicographically ascending order: We first consider all annotations with $X = 1$, then all annotations with $X = 2$, etc. For all annotations with $X = 1$, we first consider those with $Y = 1$, then those with $Y = 2$, etc. This variable order leads to a compact OBDD because the order of random variables annotating bindings of query variables X, Y in the relations R, S, T is compatible with the nesting order of the quantifiers $\exists_X$ and $\exists_Y$. □

Lemma 4. *For any RC$^\exists$ query Q_{RC} that satisfies the properties of Lemma 3, the annotation Φ of Q_{RC} on a tuple-independent database $\mathcal{D}$ can be represented by an OBDD of size $\mathcal{O}(|\mathcal{D}| \cdot 2^{|Q_{RC}|})$.*

PROOF. We prove the lemma for Boolean queries Q_{RC}; the general case follows trivially. Let the relation symbols in Q_{RC} be $R_1, \ldots, R_n$, the variables be $X_1, \ldots, X_m$, and let $\mathrm{ADom}(X_i)$ be the active domain of variable X_i. The annotation of tuple $\bar{A}$ of relation R_i is denoted by $R_i(\bar{A})$, e.g., the annotation of tuple (a, b) in relation R_1 is $R_1(a, b)$. We assume without loss of generality that the order of the query variables $X_1, \ldots, X_m$ is such that $X_i >_\exists X_j \Leftrightarrow i < j$ with respect to the nesting order $>_\exists$ defined by the $\exists$-consistency of Q_{RC}; that is, $i < j$ allows for the quantifier nesting $\exists_{X_i} Q(\exists_{X_j})$, but not $\exists_{X_j} Q(\exists_{X_i})$. Since Q_{RC} is canonicalised and $\exists$-consistent (Lemma 3), we can assume without loss of generality that in each relation symbol R the query variables occur in $>_\exists$ order (we can always relabel the query and database schema such that the query variables occur in $>_\exists$-order). For example, Q_{RC} may contain $R(X_1, X_5, X_7)$, but not $R(X_7, X_1, X_5)$. Furthermore, we assume a total order over the active domain of the database such that for any $x_i \in \mathrm{ADom}(X_i)$ and $x_j \in \mathrm{ADom}(X_j)$ it holds that $x_i < x_j \Leftrightarrow i < j$; similarly for relation names: $R_1 < R_2 < \cdots < R_n$, where in addition the relation names are not part of the active domains of query variables and occur before the domain constants in this order.

We define a total order Π on the annotations of the tuples in $\mathcal{D}$ as follows. We first associate with every annotation $R(\bar{A})$ the string $\mathrm{string}(R(\bar{A})) = \bar{A}R$, e.g., annotation $R_2(A_7, B_2, C_7)$ is associated with the string $A_7B_2C_7R_2$. The order Π is then defined as

$$R(\bar{A}) <_\Pi R'(\bar{A}') \Leftrightarrow \mathrm{string}(R(\bar{A})) <_{\mathrm{lex}} \mathrm{string}(R(\bar{A}'))$$

where $<_{\mathrm{lex}}$ is the lexicographic order on strings as defined by the total order of the active domain of the database and the relation names. Note that Π is uniquely defined by the order of the relation symbols and the order on the active domain of $\mathcal{D}$. However, different orders on the former and the latter give rise to different orders Π.

We show by structural induction over Φ that it has a Π-OBDD of width $2^{|Q_{RC}|}$ where $|Q_{RC}|$ denotes the number of relation symbols in Q_{RC}:

- The base case is a relation symbol $R(\bar{A})$ which corresponds to a trivial Π-OBDD with one variable $R(\bar{A})$ and width 2.

- If $Q_{RC} = Q_1 \wedge Q_2$ or $Q_{RC} = Q_1 \vee Q_2$, then by induction hypothesis the annotations of Q_1 and Q_2 have Π-OBDDs of width $2^{|Q_1|}$ and $2^{|Q_2|}$, respectively. Then by Lemma 1, the annotation of Q_{RC} has a Π-OBDD of width $2^{|Q_1|} \cdot 2^{|Q_2|} = 2^{|Q_{RC}|}$.

- If $Q_{RC} = \neg Q$, then by induction hypothesis Q has a Π-OBDD of width $2^{|Q|}$. Swapping the $\top$ and $\perp$ nodes in this OBDD yields the required Π-OBDD for Q_{RC}.

- If $Q_{RC} = \exists_{X_i} Q$, then for every $A_l \in \mathrm{ADom}(X_i)$ the annotations Φ_l of queries $Q[A_l/X_i]$ are over disjoint sets of variables because Q_{RC} is RC-hierarchical by Lemma 3 and hence X_i is root in Q. Moreover, each Φ_l has a Π-OBDD of width $2^{|Q|}$ by induction hypothesis. Let $\mathrm{ADom}(X_i) = \{A_1, \ldots, A_h\}$ such that $A_k <_{\mathrm{lex}} A_l$ if and only if $k < l$. The annotation Φ of Q_{RC} is the disjunction $\bigvee_{A_l \in \mathrm{ADom}(X_i)} \Phi_l$. Since the formulas Φ_l are over disjoint sets of variables for distinct values of l, an OBDD for their disjunction is obtained by their concatenation in which the $\perp$ node of the OBDD for Φ_l is replaced by the root node of the OBDD for Φ_{l+1}.

  It remains to show that this construction yields an OBDD over order Π. First, note that the OBDD for each Φ_l is over order Π by induction hypothesis; we next show that for any two annotations $R(\bar{A}_k)$ in Φ_k and $R'(\bar{A}_l)$ in Φ_l with $k < l$, it holds that $R'(\bar{A}_k) <_\Pi R'(\bar{A}_l)$; by the definition of $<_\Pi$, this is equivalent to showing $\bar{A}_k R <_{\mathrm{lex}} \bar{A}_l R'$. The strings $\bar{A}_k$ and $\bar{A}_l$ are identical in the first $i-1$ places since, by construction, the variables X_j with $j < i$ are set to the same constants. The lexicographic order of $\bar{A}_k$ and $\bar{A}_l$ — and hence the Π-order of $R(\bar{A}_k)$ in Φ_k and of $R'(\bar{A}_l)$ in Φ_l — is determined by the values of X_i in $\bar{A}_k$ and in $\bar{A}_l$; this value is A_l in $\bar{A}_l$ and A_k in $\bar{A}_k$. Since we concatenate the OBDDs in the order $\Phi_1 \to \cdots \to \Phi_h$ and since $A_1 <_{\mathrm{lex}} \cdots <_{\mathrm{lex}} A_h$ it follows that $\bar{A}_k <_{\mathrm{lex}} \bar{A}_l$ and thus $R(\bar{A}_k) <_\Pi R'(\bar{A}_l)$. The constructed OBDD has width $2^{|Q_{RC}|} = 2^{|Q|}$, because the concatenation leaves the width unchanged. $\square$

The OBDD construction in the above proof shows that conjunction, disjunction, negation, and existential quantification of $\mathrm{RC}^\exists$ queries representing rewritings of $1\mathrm{RA}^-$ queries correspond to analogous operations on OBDDs representing the annotations of such queries. In particular, the width of the resulting OBDD is bounded above by the product of the widths of the input OBDDs. This is a conservative upper bound that allows a uniform and simple treatment of $\mathrm{RC}^\exists$ operations in the proof. A tighter bound can be obtained via a more specific analysis: Any non-repeating RC-hierarchical $\mathrm{RC}^\exists$ query Q admits an OBDD of width at most $|Q|$ and size linear in the input database size and independent of the query size [19]. This tighter bound on the OBDD width can be immediately extended to $\exists$-consistent conjunction and disjunction of such queries $Q_1, \ldots, Q_n$: The resulting OBDD has width $|Q_1| \cdot \ldots \cdot |Q_n|$, which is smaller than $2^{|Q_1| + \cdots + |Q_n|}$ as used in the proof.

We can now use both Lemmata 1 and 4 to obtain the polynomial-time computation of query probability:

COROLLARY 1 (LEMMATA 1, 4). *Let Q_{RC} be a $\mathrm{RC}^\exists$ query satisfying the properties of Lemma 3. For any tuple-independent database $\mathcal{D}$, the probability of the query result $Q_{RC}(\mathcal{D})$ can be computed in time $\mathcal{O}(|\mathcal{D}| \cdot 2^{|Q_{RC}|})$.*

4. NON-HIERARCHICAL $1\mathrm{RA}^-$ QUERIES

We show in this section the following result:

LEMMA 5. *The data complexity of any non-hierarchical $1\mathrm{RA}^-$ query is #P-hard.*

PROOF. Given a $1\mathrm{RA}^-$ query Q and any 2DNF formula Ψ, we use a reduction from the model-counting problem $\#\Psi$ by means of a construction of a database $\mathcal{D}$ such that Ψ and the query result $Q(\mathcal{D})$ have the same probability. The reduction depends on structural properties of Q. We show that the non-hierarchical property is equivalent to *matching a pattern* from the list of all possible patterns made up of inner nodes that are difference or join operators and leaves that correspond to three relations $R^{[A][\neg B]}$, $S^{[A][B]}$, and $T^{[B][\neg A]}$ for two distinct attribute classes $[A]$ and $[B]$. The notion of a match is then refined to that of an *annotation-preserving*

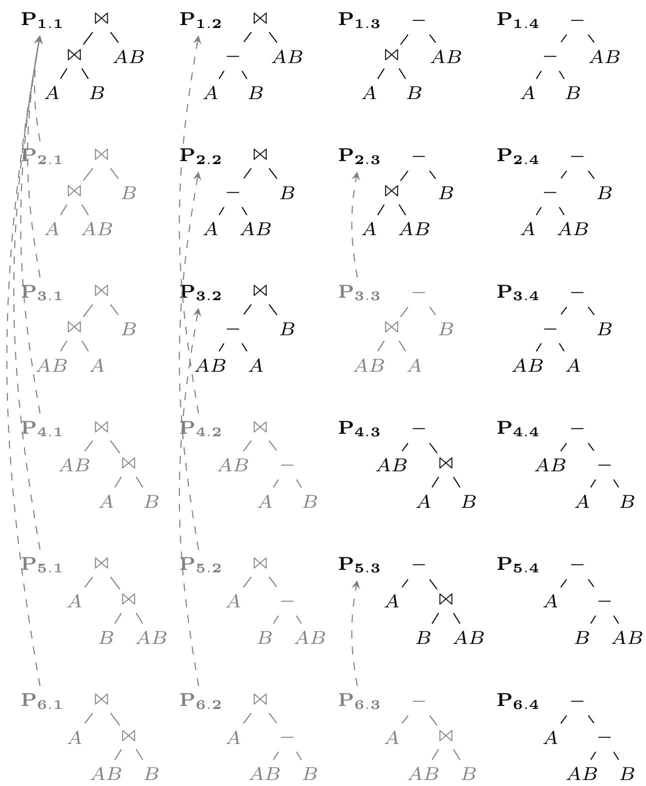


Figure 5: The 24 query patterns $P_{1.1}$, ..., $P_{6.4}$. The 10 grey patterns can by reduced to other patterns as indicated by the arrows, since the labels A and B are symmetric and can be swapped, and the join ($\bowtie$) operator is commutative and its sub-queries can also be swapped. Further 24 patterns can be obtained by swapping A and B in the above patterns.

match, for which a database construction scheme is possible such that the query result becomes annotated by Ψ.

The proof steps are summarised as follows:

$$\begin{array}{c} Q \text{ is non-hierarchical} \\ \Leftrightarrow_{\text{Proposition 1}} \\ Q \text{ has a match with a pattern in Figure 5} \\ \Leftrightarrow_{\text{Lemma 7}} \\ Q \text{ has an annotation-preserving match with a pattern} \\ \Rightarrow_{\text{Lemma 8}} \\ Q \text{ is hard for \#P.} \end{array}$$ □

4.1 Database construction scheme

Our database construction scheme prescribes how to populate relations used in a non-hierarchical query such that the query result is annotated with a desired 2DNF formula. It particularly focuses on two distinguished attributes $[A]$ and $[B]$ that witness the non-hierarchical property of the query.

We assume two finite sets of constants, $\mathbf{A}$ and $\mathbf{B}$, and a constant ■ distinct from those in $\mathbf{A}$ and $\mathbf{B}$. In this section, the projection operator π_A^Φ is used to symbolise the projection on attribute A and the annotation column Φ; in contrast, π_A selects only column A, neglecting the annotations of tuples. The notation $(a_1, \ldots, a_n | \Phi(a_1, \ldots, a_n))$ denotes a tuple $(a_1, \ldots, a_n)$ annotated with formula $\Phi(a_1, \ldots, a_n)$.

Preserving the data of one attribute

We commence by analysing queries with one distinguished attribute A. Let Φ be a total function on $\mathbf{A}$. A relation Q is *A-reducible to* $(\mathbf{A}, \Phi)$ if the $[A]$-attributes of Q are filled with all values from $\mathbf{A}$, all non-$[A]$-attributes are filled with ■, and the annotation of a tuple identified by $a \in \mathbf{A}$ is $\Phi(a)$:

$$\pi^\Phi_{[A]}(Q) = \{(a|\Phi(a)) \mid a \in \mathbf{A}\}$$
$$\pi_C(Q) = \{(■)\} \qquad \text{for any attribute } C \text{ with } C \notin [A].$$

By $\text{red}_A(Q) = \mathbf{A}|\Phi$ we denote that Q is A-reducible to $(\mathbf{A}, \Phi)$. Queries that do not export $[A]$ are called $\emptyset$-reducible to a nullary function Φ (denoted $\text{red}_\emptyset(Q) = ■|\Phi$) if

$$\pi^\Phi_\emptyset(Q) = \{(\Phi)\}$$
$$\pi_C(Q) = \{(■)\} \qquad \text{for any attribute } C.$$

We next define three classes of relations $\mathcal{Q}^A$, $\mathcal{Q}_{\text{fill}}$, and $\mathcal{Q}_\emptyset$ that are characterised by their A-reductions; let $\Phi_\top$ be the constant function $\Phi_\top(.) = \top$.

$$Q^{[A]} \in \mathcal{Q}^A \quad \text{if} \quad \text{red}_A(Q) = \mathbf{A}|\Phi \tag{1}$$
$$Q^{[A]} \in \mathcal{Q}_{\text{fill}} \quad \text{if} \quad \text{red}_A(Q) = \mathbf{A}|\Phi_\top \tag{2}$$
$$Q^{[\neg A]} \in \mathcal{Q}_{\text{fill}} \quad \text{if} \quad \text{red}_\emptyset(Q) = ■|\Phi_\top \tag{3}$$
$$Q \in \mathcal{Q}_\emptyset \quad \text{if} \quad Q = \emptyset \tag{4}$$

In Equation (1), Φ can also be $\neg\Phi$. Queries $\mathcal{Q}^A$ are relations in which the values of $[A]$-attributes are populated with values from $\mathbf{A}$, and values for non-$[A]$-attributes are set to ■. There is a functional dependency $[A] \to \Phi$ such that every tuple is represented by its $[A]$-value a and has a corresponding annotation $\Phi(a)$ or $\neg\Phi(a)$. Queries $\mathcal{Q}_{\text{fill}}$ are similar to Q_A-queries, but every tuple is annotated with $\top$. Queries $\mathcal{Q}_\emptyset$ are simply empty relations.

Example 7. Given the domain $\mathbf{A} = \{x_1, x_2, x_3\}$, the following relation X over the distinguished attribute A and two attributes B, C with $B, C \notin [A]$ satisfies the properties of a $\mathcal{Q}^A$-query, and relation Y is a $\mathcal{Q}_{\text{fill}}$-query.

$\mathcal{Q}^A$-relation X

A_x	B_x	C_x	$\mathbf{\Phi}$
x_1	■	■	$\mathbf{x_1}$
x_2	■	■	$\mathbf{x_2}$
x_3	■	■	$\mathbf{x_3}$

$\mathcal{Q}_{\text{fill}}$-relation Y

A_y	B_y	C_y	$\mathbf{\Phi}$
x_1	■	■	$\top$
x_2	■	■	$\top$
x_3	■	■	$\top$

In relation X we use the same symbols x_i both as data values for A and annotations; the functional dependency $A_x \to \Phi$ is thus trivially satisfied by $\Phi(x_i) = \mathbf{x_i}$. □

Figure 6 shows how $\mathcal{Q}^A$, $\mathcal{Q}_{\text{fill}}$, and $\mathcal{Q}_\emptyset$-queries are propagated through query operators: Given query classes $\mathcal{Q}_1$ and $\mathcal{Q}_2$, the right-most column ($\mathcal{Q}_1$ *Op* $\mathcal{Q}_2$) in the table shows the class to which a query that combines two queries from those respective classes by operator *Op* belongs.

Example 8. Continuing Example 7, the equi-join $X \bowtie Y$ (on the corresponding A, B, C attributes) of $\mathcal{Q}^A$-query X and $\mathcal{Q}_{\text{fill}}$-query Y yields the following relation:

$\mathcal{Q}^A$-query $X \bowtie Y$

A_x	A_y	B_x	B_y	C_x	C_y	$\mathbf{\Phi}$
x_1	x_1	■	■	■	■	$\mathbf{x_1}$
x_2	x_2	■	■	■	■	$\mathbf{x_2}$
x_3	x_3	■	■	■	■	$\mathbf{x_3}$

$\mathcal{Q}_1$	Op	$\mathcal{Q}_2$	$\mathcal{Q}_1\ Op\ \mathcal{Q}_2$
$\mathcal{Q}^A$	$\bowtie$	$\mathcal{Q}_{\text{fill}}$	$\mathcal{Q}^A$
	$-$	$\mathcal{Q}_\emptyset$	$\mathcal{Q}^A$
$\mathcal{Q}^{AB}$	$\bowtie$	$\mathcal{Q}_{\text{fill}}$	$\mathcal{Q}^{AB}$
	$-$	$\mathcal{Q}_\emptyset$	$\mathcal{Q}^{AB}$
$\mathcal{Q}_{\text{fill}}$	$\bowtie$	$\mathcal{Q}^A$	$\mathcal{Q}^A$
		$\mathcal{Q}^{AB}$	$\mathcal{Q}^{AB}$
		$\mathcal{Q}_{\text{fill}}$	$\mathcal{Q}_{\text{fill}}$
	$-$	$\mathcal{Q}^A$	$\mathcal{Q}^A$
		$\mathcal{Q}^{AB}$	$\mathcal{Q}^{AB}$
		$\mathcal{Q}_\emptyset$	$\mathcal{Q}_{\text{fill}}$

Figure 6: Class membership of queries connecting classes $\mathcal{Q}^A$, $\mathcal{Q}_{\text{fill}}$, and $\mathcal{Q}_\emptyset$ with operators $\bowtie$, $-$.

This join satisfies the conditions of a $\mathcal{Q}^A$-query as suggested by the rule $\mathcal{Q}^A \bowtie \mathcal{Q}_{\text{fill}} \rightarrow \mathcal{Q}^A$ in Figure 6. Similarly, the difference of $Y - X$ is also a $\mathcal{Q}^A$-query:

$\mathcal{Q}^A$-query $Y - X$			
A_y	B_y	C_y	$\mathbf{\Phi}$
x_1	■	■	$\neg\mathbf{x_1}$
x_2	■	■	$\neg\mathbf{x_2}$
x_3	■	■	$\neg\mathbf{x_3}$

Now let $Q^{[A]}$ be a query that contains a $\mathcal{Q}^A$-relation $X^{[A]}$. We can populate the relations of Q such that Q is a $\mathcal{Q}^A$-query, i.e., that Q satisfies the above properties for $\mathcal{Q}^A$:

LEMMA 6. *Given a query Q, a distinguished attribute A of Q, and a distinguished relation X^A of Q that satisfies Equation* (1)*, the remaining relations of Q can be filled such that Q satisfies Equation* (1).

PROOF. We first identify the set $\mathcal{OP}_-$ of difference operators in Q that do not have X as a right descendant and partition the relations of Q into three sets:

$$\begin{aligned}\text{rels}_X &= \{X\}\\ \text{rels}_\emptyset &= \text{relations right descendants of a } \mathcal{OP}_- \text{ operator}\\ \text{rels}_{\text{fill}} &= \text{all other relations}\end{aligned}$$

We populate every $\text{rels}_{\text{fill}}$ relation as a $\mathcal{Q}_{\text{fill}}$-query, and every $\text{rels}_\emptyset$ relation as a $\mathcal{Q}_\emptyset$-query. For the former, it suffices to populate each $[A]$ attribute of a $\text{rels}_{\text{fill}}$-relation with $\mathbf{A}$, and each non-$[A]$-attribute with ■. The following inductive argument shows that every operator on the path in Q between X and the root of Q is a $\mathcal{Q}^A$-query: First, this trivially holds at X itself. Now let Op be an operator on the path between X and the root of Q. We have the cases:

- $Q_L \bowtie Q_R$, where without loss of generality Q_L contains X. Then, Q_L is a $\mathcal{Q}^A$-query, Q_R contains a relation from $\text{rels}_{\text{fill}}$ and is a $\mathcal{Q}_{\text{fill}}$-query. Hence, $Q_L \bowtie Q_R$ is a $\mathcal{Q}^A$-query.
- $Q_L - Q_R$, where Q_L contains X. Then the difference operator is in $\mathcal{OP}_-$ and Q_R is a $\mathcal{Q}_\emptyset$-query, Q_L is a $\mathcal{Q}^A$-query, and hence $Q_L - Q_R$ is a $\mathcal{Q}^A$-query.
- $Q_L - Q_R$, where Q_R contains X. Then, Q_R is a $\mathcal{Q}^A$-query, Q_L contains a relation from $\text{rels}_{\text{fill}}$ and is a $\mathcal{Q}_{\text{fill}}$-query. Hence, $Q_L - Q_R$ is a $\mathcal{Q}^A$-query. □

If X has even polarity in Q, then the annotation $\Phi_Q(a)$ of a tuple (a) in $\pi_{[A]}(Q)$ is the same as the corresponding annotation $\Phi_X(a)$ of a tuple (a) in $\pi_{[A]}(X)$; if X has odd polarity in Q, then $\Phi_Q(a) = \neg\Phi_X(a)$.

Preserving the data of two attributes

We can extend the above technique to queries that contain relations over two distinguished attributes A and B whose values we would like to preserve; we only sketch this next.

Let Φ^{AB} be a total function on $\mathbf{A}\times\mathbf{B}$, and let Φ^A be a total function on $\mathbf{A}\cup\mathbf{A}\times\mathbf{B}$ such that $\Phi^A(a) \equiv \bigvee_{b\in\mathbf{B}} \Phi^A(a,b)$ for all $a \in \mathbf{A}$. As before, a relation Q is *A-reducible to* $(\mathbf{A}, \Phi^A)$ if

$$\begin{aligned}\pi^{\Phi}_{[A]}(Q) &= \{(a|\Phi^A(a)) \mid a \in \mathbf{A}\}\\ \pi_C(Q) &= \{(■)\} \qquad \text{for any attribute } C \text{ with } C \notin [A].\end{aligned}$$

Similarly, Q is AB-reducible to $(\mathbf{A}\times\mathbf{B}, \Phi^{AB})$ if

$$\begin{aligned}\pi^{\Phi}_{[A][B]}(Q) &= \{(a, b|\Phi^{AB}(a,b)) \mid a \in \mathbf{A}, b \in \mathbf{B}\}\\ \pi_C(Q) &= \{(■)\} \text{ for any attribute } C \text{ with } C \notin [A]\cup[B].\end{aligned}$$

By $\text{red}_{AB}(Q) = \mathbf{A}\times\mathbf{B}|\Phi^{AB}$ we denote that Q is AB-reducible to $(\mathbf{A}\times\mathbf{B}, \Phi^{AB})$. We define additional classes of queries:

$$Q^{[A][\neg B]} \in \mathcal{Q}^A \text{ if } \text{red}_A(Q) = \mathbf{A}|\Phi^A \tag{5}$$
$$Q^{[A][B]} \in \mathcal{Q}^A \text{ if } \text{red}_{AB}(Q) = \mathbf{A}\times\mathbf{B}|\Phi^A \tag{6}$$
$$Q^{[A][B]} \in \mathcal{Q}^{AB} \text{ if } \text{red}_{AB}(Q) = \mathbf{A}\times\mathbf{B}|\Phi^{AB} \tag{7}$$
$$Q^{[A][\neg B]} \in \mathcal{Q}_{\text{fill}} \text{ if } \text{red}_A(Q) = \mathbf{A}|\Phi_\top \tag{8}$$
$$Q^{[A][B]} \in \mathcal{Q}_{\text{fill}} \text{ if } \text{red}_{AB}(Q) = \mathbf{A}\times\mathbf{B}|\Phi_\top \tag{9}$$
$$Q^{[\neg A][\neg B]} \in \mathcal{Q}_{\text{fill}} \text{ if } \text{red}_\emptyset(Q) = ■|\Phi_\top \tag{10}$$
$$Q \in \mathcal{Q}_\emptyset \text{ if } Q = \emptyset \tag{11}$$

In Equations (5)–(7), Φ^A and Φ^{AB} can also be negated. Queries from these classes are propagated by query operators as depicted in Figure 6. Lemma 6 can be extended to the case of two attributes A and B:

- For a distinguished relation $X^{A\neg B}$ of Q that satisfies Equation (5), the remaining relations of Q can be filled such that Q satisfies Equation (5) if Q exports $[A]$ but not $[B]$, or Equation (6) if Q exports $[A]$ and $[B]$.
- For a distinguished relation X^{AB} of Q, the remaining relations in Q can be filled such that Q satisfies Equation (7) if Q exports $[A]$ and $[B]$.

4.2 Patterns and matches

We next define hard minimal query patterns and matches.

DEFINITION 5. *A* (query) pattern *P over attributes A, B and relational operators Op_1, $Op_2 \in \{\bowtie, -\}$ is a binary tree with leaves A, B, AB, root node Op_1, and inner node Op_2.*

There are $2\cdot 2\cdot 2\cdot 6 = 48$ different patterns: There are two distinct unlabeled binary trees with three leaves, the two operators can each be either $\bowtie$ or $-$, and there are 6 possible orders of the labels A, AB, and B. Figure 5 shows 24 of the 48 patterns and omits for each pattern the symmetric pattern obtained by swapping leaves A and B.

DEFINITION 6. *A $1RA^-$ query Q* matches *a pattern P over attributes A and B if there is mapping from the nodes A, B, AB, Op_1, and Op_2 of P to relations $R^{[A][\neg B]}$, $T^{[\neg A][B]}$, $S^{[A][B]}$, and operators Op_1 and respectively Op_2 in the parse tree of Q that preserves ancestor-descendant relationships.*

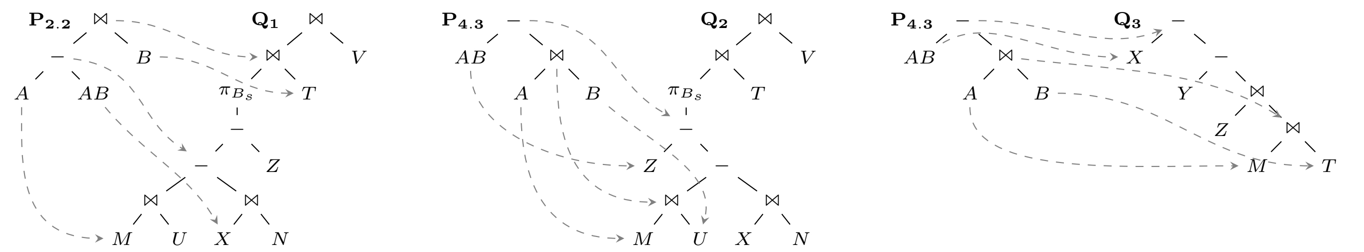


Figure 7: Patterns $P_{2.2}$ and $P_{4.3}$ and parse trees of queries Q_1, Q_2, Q_3 over the schema $M(A_m)$, $N(A_n)$, $T(B_t, C_t)$, $U(B_u)$, $V(B_v, C_v)$, $X(A_x, B_x)$, $Y(A_y, B_y)$, $Z(A_z, B_z)$. Q_1 is an (M, X, T)-match of pattern $P_{2.2}$; it also matches other patterns and is an annotation-preserving (M, X, T)-match of $P_{2.2}$, since Op_2 (the least common ancestor of M and X) is left-deep. Although Q_2 is an (M, X, T)-match of $P_{2.2}$, it is not an annotation-preserving match of $P_{2.2}$, since Op_2 is a right descendant of the top-most difference operator. However, Q_2 is an annotation-preserving (M, Z, U)-match of pattern $P_{4.3}$. Query Q_3 is an annotation-preserving (M, X, T)-match of pattern $P_{4.3}$.

We also say that Q is an (R, S, T)-match of P to emphasise which relations establish the match. Figures 1 and 7 show examples of queries matching patterns. Pattern matching is intimately linked to the non-hierarchical property:

PROPOSITION 1. *A 1RA$^-$ query is non-hierarchical if and only if it matches one of the patterns in Figure 5.*

The notion of a match is further specialised to that of an *annotation-preserving match.* Whereas the database construction scheme detailed in Section 4.1 does not work for general matches, it does work for annotation-preserving matches. We first define left-deep operators.

DEFINITION 7. *An operator Op is* left-deep *in a 1RA$^-$ query Q if Op is a* left *descendant of every difference operator on the path between the root of Q and Op.*

EXAMPLE 9. In Figure 7, the bottom-most difference operator in Q_1 is left-deep, while the bottom-most difference operator in Q_2 is not left-deep. □

DEFINITION 8. *A 1RA$^-$ query Q is an* annotation-preserving match *of a pattern P over attributes A and B if:*

1. *Q is an (R, S, T)-match of P;*
2. *For every difference operator Op_- in Q, if Op_1 is a right descendant of Op_-, then Op_- does not export $[A]$ or $[B]$.*
3. *If Op_2 is a left descendant of Op_1 in Q, then Op_2 is left-deep in the sub-query rooted at Op_1.*

We say that Q is an *annotation-preserving (R, S, T)-match* of P to emphasise the relations establishing the match. Figure 7 shows examples of annotation-preserving matches.

We next look closer at the connection between matches and annotation-preserving matches. Lemma 7 establishes next that any query that matches a pattern necessarily also has an annotation-preserving match with a (possibly different) pattern; furthermore, the relation symbols that establish the annotation-preserving match can be found by exploring the query tree in left-to-right depth-first in-order.

LEMMA 7. *Let Q be a 1RA$^-$ query and $o_1, \ldots, o_n$ be the sequence of its parse tree nodes in left-to-right depth-first in-order, and $Q_1, \ldots, Q_n$ be the corresponding sequence of sub-queries rooted at $o_1, \ldots, o_n$. If Q_i is the first sub-query in the above order that matches a pattern in Figure 5, then Q_i is an annotation-preserving match with a pattern.*

EXAMPLE 10. Consider query Q_2 in Figure 7. The sub-query rooted at the top-most difference operator is the first one to match a pattern and also has an annotation-preserving (M, Z, U)-match with $P_{4.3}$. □

4.3 Hardness reductions

The 24 patterns in Figure 5 are the smallest hard patterns for 1RA$^-$, and any query that is an annotation-preserving match of one of them is hard for #P.

LEMMA 8. *The data complexity of any 1RA$^-$ query that is an annotation-preserving match of one of the patterns in Figure 5 is #P-hard.*

Putting together Proposition 1 and Lemmata 7 and 8, we obtain that the data complexity of all non-hierarchical 1RA$^-$ queries is #P-hard.

The proof of Lemma 8 goes over each pattern case and shows hardness via a reduction from the #2DNF problem: Let Q be a query that is an annotation-preserving (R, S, T)-match for a pattern P, and let $\Psi = \bigvee_{(i,j)\in E} x_i y_j$ be a 2DNF formula with $|E|$ clauses over disjoint variable sets $\mathbf{X}$ and $\mathbf{Y}$. We construct in polynomial time a tuple-independent database $\mathcal{D}$ using the database construction scheme in Section 4.1 such that the annotation of the query result $Q(\mathcal{D})$ is either Ψ and hence $P_{Q(\mathcal{D})} = P_\Psi = \#\Psi \cdot 2^{-|\text{vars}(\Psi)|}$, or $\neg\Psi$ and then $P_{Q(\mathcal{D})} = 1 - P_\Psi$.

We next give reductions for patterns $P_{4.3}$ and $P_{5.3}$; all reductions are given in an extended paper [12]. Pattern $P_{1.1}$ is the only one needed to show hardness of non-hierarchical 1RA$^-$ queries without difference, i.e., of non-repeating conjunctive queries studied in prior work [6]. The reduction for pattern $P_{5.3}$ establishes that a query matching $P_{5.3}$ can be hard already when constrained to databases in which one relation is probabilistic and all other relations are certain.

Reduction for pattern $P_{4.3}$. We use the illustration of a query matching $P_{4.3}$ in Figure 8(left). By Definition 8, a query Q that is an annotation-preserving match of $P_{4.3}$

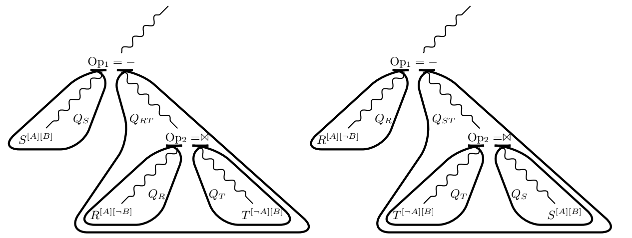


Figure 8: Schematic illustration of a query that is an annotation-preserving match of pattern $P_{4.3}$ (left) or $P_{5.3}$ (right). A curly path indicates that other operators may occur on it.

satisfies the following structural constraint: If Op_1 is a right descendant of a difference operator, then this operator does not export $[A]$ or $[B]$. Furthermore, attributes $[A]$ and $[B]$ are exported by every operator on the paths from S to R and from S to T, respectively. We encode the 2DNF formula Ψ as a database $\mathcal{D}$ such that the annotation of the query result $Q(\mathcal{D})$ is Ψ, if the polarity of Op_2 is odd in Q_{RT}. In case of even polarity, we derive a database $\mathcal{D}$ and another formula Υ from Ψ such that $P_{Q(\mathcal{D})} = P_\Upsilon$ and linearly many calls to an oracle for P_Υ suffice to compute $\#\Psi$.

Case 1: Odd polarity $(\mathrm{pol}(Q_{RT}, Op_2) = 1)$**.** We fill the relations R, S, T such that Q_R is a $\mathcal{Q}^A$-query, Q_T is a $\mathcal{Q}^B$-query, and Q_S is a $\mathcal{Q}^{AB}$-query, and for all three relations the annotation functions are the identity. In other words, R consists of a tuple with A-value x_i and annotation x_i for each variable $x_i \in \mathbf{X}$ that occurs in Ψ; T consists of a tuple with B-value y_j and annotation y_j for each variable $y_j \in \mathbf{Y}$ that occurs in Ψ; S consists of a tuple with (A, B)-values (x_i, y_j) and annotation $\top$ for each clause $x_i y_j$ in Ψ. Note that when used outside annotations, the variables are considered constants in relations R, S, T. For the remaining relations, we distinguish two cases: (1) Any relation that appears on the right side of a difference operator different from Op_1 and Op_2, is set to $\emptyset$. (2) Any relation with an $[A]$ attribute and no $[B]$ attribute is filled like R, but with annotations $\top$. Symmetrically, any relation with a $[B]$ attribute and no $[A]$ attribute is filled like T, but with annotations $\top$. Relations with both $[A]$ and $[B]$ attributes are filled with the Cartesian product of $\mathbf{X}$ and $\mathbf{Y}$ and annotations $\top$. In all of the above cases, any attribute that is neither in $[A]$ nor in $[B]$ is filled with constant $\blacksquare$.

Since Op_2 has odd polarity in Q_{RT} and since both $[A]$ and $[B]$ are exported by every operator on the path between Op_1 and Op_2, Q_{RT} and $Q_S - Q_{RT}$ are $\mathcal{Q}^{AB}$-queries with annotations

$$\mathrm{red}_{AB}(Q_{RT}) = \mathbf{X} \times \mathbf{Y} | \neg\Phi_{RT}, \Phi_{RT}(x_i, y_i) = x_i y_i$$
$$\mathrm{red}_{AB}(Q_S - Q_{RT}) = \mathbf{X} \times \mathbf{Y} | \Phi_{RST},$$
$$\Phi_{RST}(x_i, y_j) = \begin{cases} x_i y_j & \text{if } (i,j) \in E \\ \bot & \text{if } (i,j) \notin E. \end{cases}$$

The final projection $\pi_{-[A]-[B]}$ yields one answer tuple, whose annotation is the disjunction of all clauses in Ψ.

Case 2: Even polarity $(\mathrm{pol}(Q_{RT}, Op_2) = 0)$**.** Let Θ be the set of assignments of variables $\mathbf{X} \cup \mathbf{Y}$. Then the number of models of Ψ is defined by $\#\Psi = \sum_{\theta \in \Theta : \theta \models \Psi} 1$. If we partition Θ into disjoint sets $\Theta_0 \cup \cdots \cup \Theta_{|E|}$, such that $\theta \in \Theta_i$ if and only if θ satisfies exactly i clauses of Ψ, then this sum can equivalently by written as

$$\#\Psi = \sum_{\theta \in \Theta_1 : \theta \models \Psi} 1 + \cdots + \sum_{\theta \in \Theta_m : \theta \models \Psi} 1 = |\Theta_1| + \cdots + |\Theta_{|E|}|.$$

We next show how to compute $|\Theta_i|$ (and hence $\#\Psi$) using an oracle for P_Υ, with Υ defined below. Let $\mathbf{Z} = \{z_1, \ldots, z_{|E|}\}$ be a set of variables disjoint from $\mathbf{X} \cup \mathbf{Y}$ and define Υ as

$$\Upsilon = \bigvee_{i=1}^{|E|} \neg z_i \wedge \neg\psi_i \quad \text{or, equivalently} \quad \neg\Upsilon = \bigwedge_{i=1}^{|E|} (z_i \vee \psi_i) \quad (12)$$

We fix the probabilities of variables in $\mathbf{X}$ and $\mathbf{Y}$ to $1/2$ and of variables in $\mathbf{Z}$ to $p_z \in [0,1]$. The probability $1 - P_\Upsilon = P_{\neg\Upsilon}$ can be expressed by conditioning on the number of satisfied clauses of Ψ:

$$P_{\neg\Upsilon} = \sum_{k=0}^{|E|} \underbrace{P\left(\neg\Upsilon \middle| \begin{matrix}\text{exactly } k \text{ clauses} \\ \text{of } \Psi \text{ are satisfied}\end{matrix}\right)}_{p_z^{|E|-k}} \cdot \underbrace{P\left(\begin{matrix}\text{exactly } k \text{ clauses} \\ \text{of } \Psi \text{ are satisfied}\end{matrix}\right)}_{\frac{1}{2}^{|\mathbf{X}|+|\mathbf{Y}|} \cdot |\Theta_k|}$$

$$= \frac{1}{2}^{|\mathbf{X}|+|\mathbf{Y}|} \sum_{k=0}^{|E|} p_z^{|E|-k} |\Theta_k|$$

Intuitively, the first term simplifies to $p_z^{|E|-k}$, because if exactly k clauses ψ_i are satisfied in $\neg\Upsilon$, then in order to satisfy the remaining $|E| - k$ clauses $(z_i \vee \psi_i)$ at least $|E| - k$ of the z_i must be satisfied, and this occurs with probability $p_z^{|E|-k}$. This is a polynomial in p_z of degree $|E|$, with coefficients $|\Theta_0|, \ldots, |\Theta_{|E|}|$. The $|E| + 1$ coefficients can be derived from $|E| + 1$ pairs (p_z, P_Υ) using Lagrange's polynomial interpolation formula. We conclude that $|E| + 1$ oracle calls to P_Υ suffice to determine $\#\Psi = \sum_{i=0}^{|E|} |\Theta_i|$.

It remains to show how Υ can be encoded as the annotation of a query that is an annotation-preserving match of $P_{4.3}$; given this encoding, any algorithm that evaluates $P_{Q(\mathcal{D})}$ constitutes the above oracle. Formula Υ is encoded using the database construction scheme from Case 1, where the annotation of a tuple with (A, B)-values (x_i, y_j) corresponding to clause $\psi_k = x_i y_j$ in Ψ becomes $\neg z_k$. Then, the annotation of a tuple with (A, B)-values (x_i, y_j) in the result of the sub-query rooted at Op_1, i.e., $Q_S - Q_{RT}$, becomes $\neg z_k \wedge \neg\psi_k$. The final projection $\pi_{-[A]-[B]}$ yields one result tuple, whose annotation is the disjunction of the annotation of $Q_S - Q_{RT}$ which is exactly Υ.

Reduction for pattern $P_{5.3}$**.** We use the illustration of a query matching $P_{5.3}$ in Figure 8 (right). We only describe here the case when $[B]$ is not exported by Op_1, in which case the sub-query Q_{ST} contains a projection operator $Op_\pi = \pi_{-[B]}$ such that every operator between Op_π and Op_1 exports $[A]$ but not $[B]$, and every operator between Op_π and Op_2 exports $[A]$ and $[B]$. Let Q_π be the sub-query rooted at Op_π.

The first step is to show that one may without loss of generality assume that Op_2 is left-deep in Q_π. Assume to the contrary that there is a difference operator Op_- between Op_π and Op_2 that has Op_2 as a right descendant; clearly, Op_- exports $[A]$ and $[B]$ and hence its left sub-query contains relations $X^{[A][\neg B]}$ and $Y^{[\neg A][B]}$ or it contains a relation $Z^{[A][B]}$. In the former case, Q is an annotation-preserving (R, S, Y)-match of pattern $P_{5.4}$; in the latter case, Q is an

R	
A_r	$\mathbf{\Phi}$
1	$\top$
2	$\top$

T	
B_t	$\mathbf{\Phi}$
x_1	$\neg\mathbf{x_1}$
y_1	$\neg\mathbf{y_1}$
y_2	$\neg\mathbf{y_2}$

S		
A_s	B_s	$\mathbf{\Phi}$
1	x_1	$\top$
1	y_1	$\top$
1	y_2	$\bot$
2	x_1	$\top$
2	y_1	$\bot$
2	y_2	$\top$

$Q_T \bowtie Q_S$		
A_s	B_s	$\mathbf{\Phi}$
1	x_1	$\neg\mathbf{x_1}$
1	y_1	$\neg\mathbf{y_1}$
1	y_2	$\bot$
2	x_1	$\neg\mathbf{x_1}$
2	y_1	$\bot$
2	y_2	$\neg\mathbf{y_2}$

$Q_\pi = Q_{ST}$	
A_s	$\mathbf{\Phi}$
1	$\neg\mathbf{x_1} \vee \neg\mathbf{y_1}$
2	$\neg\mathbf{x_1} \vee \neg\mathbf{y_2}$

Q_{RST}	
A_r	$\mathbf{\Phi}$
1	$\mathbf{x_1 y_1}$
2	$\mathbf{x_1 y_2}$

Figure 9: Relations R, S, T for the hardness reduction of a query with an annotation-preserving match for pattern $P_{5.3}$ where (1) Op_1 does not export $[B]$ and (2) the projection operator $\pi_{-[B]}$ on the path between Op_1 and Op_2 has even polarity in Q_{ST} (the sub-query containing both relations S and T). Only attributes $[A]$ and $[B]$ are depicted, and it is assumed that R, S, T have even polarity in their respective sub-queries Q_R, Q_S, and Q_T. The database is with respect to the formula $\Psi = \psi_1 \vee \psi_2 = x_1y_1 \vee x_1y_2$.

annotation-preserving (R, Z, T)-match of pattern $P_{6.4}$. In both cases, the new Op_2 is Op_- and left-deep in Q_π.

Next, two cases need to be analysed separately depending on the polarity of Op_π in Q_{ST}.

Case 1: Even polarity ($\mathrm{pol}(Q_{ST}, Op_\pi) = 0$). Let $\mathbf{N} = \{1, \dots, |E|\}$ be the set of integers that numbers consecutively the clauses in Ψ: $\Psi = \psi_1 \vee \dots \vee \psi_{|E|}$. We set relation R to contain a tuple (n) annotated with $\top$ for every clause number $n \in \mathbf{N}$. Relation S contains all tuples (n, v) where $n \in \mathbf{N}$ is a clause number and $v \in \mathbf{X} \cup \mathbf{Y}$ is a variable from Ψ; (n, v) is annotated with $\top$ if clause n contains variable v, and with $\bot$ otherwise. Relation T has a tuple (v) annotated with $\neg v$ for each variable v in Ψ. Figure 9 exemplifies how R, S, T are filled for a query matching $P_{5.3}$ and for formula $\Psi = x_1y_1 \vee x_1y_2$ and how these annotations are propagated through the query.

Case 2: Odd polarity ($\mathrm{pol}(Q_{ST}, Op_\pi) = 1$). Intuitively, since the number of difference operators between the root of the query and the relations S and T is even, they act equivalently to a sequence of join operators for query annotations: We fill the relations such that Q_T is a $\mathcal{Q}^B$-query, Q_S is a $\mathcal{Q}^{AB}$-query, Q_R is a $\mathcal{Q}^A$-query, and then Q_{ST} is a $\mathcal{Q}^A$-query, where for relations R and T the annotation functions are the identity and for relation S, the anotation function is $\top$ for all tuples (x_i, y_j) corresponding to clauses in Ψ and $\bot$ otherwise.

5. BEYOND 1RA$^-$ QUERIES

In this section we discuss the effect of various extensions of 1RA$^-$ on query tractability.

A dichotomy for full relational algebra seems unattainable since key reasoning tasks for such queries, such as equivalence, emptiness, or subsumption, are undecidable: Given two equivalent queries, one hard and one tractable, we thus cannot decide whether their union is tractable. Restrictions on the use of negation, e.g., guarded negation [3], enable decidability of query equivalence and can pave the way to a complexity dichotomy for (possibly repeating) relational queries with guarded negation in probabilistic databases.

5.1 Non-repeating relational algebra

If we add the union operator to the language 1RA$^-$, we need a different syntactic characterisation of the tractable queries, since the hierarchical property is not defined for queries with union. An immediate attempt would consider all (union-free) sub-queries obtained by choosing one term at each union and checking whether all of them are hierarchical. This approach fails since such sub-queries are not necessarily $\exists$-consistent. For instance, the non-repeating relational algebra query $Q = \pi_\emptyset[S(A,B) - (R(A) \bowtie S_1(A,B) \cup T(B) \bowtie S_2(A,B))]$ has two hierarchical union-free sub-queries under $\pi_\emptyset$: $S(A,B) - (R(A) \bowtie S_1(A,B))$ and $S(A,B) - (T(B) \bowtie S_2(A,B))$. However, these sub-queries cannot be rewritten to $\exists$-consistent RC$^\exists$ queries, since they have roots A and B respectively; it can be further shown that Q is #P-hard.

An alternative characterisation would be to check $\exists$-consistency and the RC$^\exists$-hierarchical property of the RC$^\exists$ expression Q_r representing the rewriting of a non-repeating relational algebra query Q described in Section 3.1. Then Q is tractable when Q_r is $\exists$-consistent and RC-hierarchical. Checking these properties can be done efficiently in the size of the input RC$^\exists$ query, yet Q_r may be much larger than Q (as per discussion at the end of Section 3.1). It is open whether the characterisation of tractable non-repeating relational algebra queries can be done more efficiently than following this procedure via $\exists$-consistency, which incurs the non-trivial time to rewrite the input query.

5.2 Non-repeating RC$^\exists$

There are subtle differences between 1RA$^-$ and non-repeating RC$^\exists$ that revolve around RC$^\exists$'s flexibility to allow disjunction and negation on sub-queries of different schemas. For instance, the non-repeating RC$^\exists$ queries $S(x,y) \wedge \neg R(x)$ and $S(x,y) \wedge (R(x) \vee T(y))$ cannot be expressed in 1RA$^-$. Whereas the former query is tractable, the latter is #P-hard: This means that 1RA$^-$ cannot express both tractable and hard queries that are expressible in non-repeating RC$^\exists$.

For non-repeating RC$^\exists$, the RC-hierarchical property alone does *not* characterise the tractable queries, even when we take away disjunction. Indeed, the RC$^\exists$ query equivalent to the 1RA$^-$ query from Figure 3, i.e., $Q = \exists_A \exists_B R(A) \wedge S(B) \wedge \neg(U(A) \wedge V(B))$, does not satisfy the RC-hierarchical property since neither A nor B are root in the expression and they cannot be pushed further down. However, as for 1RA$^-$ queries, we can rewrite a non-repeating RC$^\exists$ query Q into an RC$^\exists$ query Q_r as outlined in Section 3.1, e.g., $Q_r = \exists_A[R(A) \wedge \neg U(A)] \wedge \exists_B S(B) \vee \exists_A R(A) \wedge \exists_B[S(B) \wedge \neg V(B)]$ for the above query Q, and then again Q is tractable when Q_r is RC-hierarchical and $\exists$-consistent.

6. RELATED WORK

Negation is a substantial source of complexity already for databases with incomplete information and without probabilities [2]. In probabilistic databases, the MystiQ system supports a limited class of NOT EXISTS queries [25]. A framework for the exact and approximate evaluation of full relational algebra queries (thus including negation) in probabilistic databases is part of SPROUT [13, 11]. Further work looks at approximating queries with negation [18].

Our dichotomy is in line with and contributes to a succession of complexity results for queries on probabilistic databases: Starting from a first example of a #P-hard query [14], polynomial-time/#P-hard dichotomies have been established by Dalvi and Suciu for non-repeating conjunctive queries [5] and unions of conjunctive queries (UCQs) [9]; a trichotomy has been proven for positive queries with HAVING aggregates [22]; the precise tractability frontier for so-called quantified queries such as relational division and set equivalence, which can be expressed as repeating queries with nested negation, is also known [13]. Our result strictly generalises the dichotomy for non-repeating conjunctive queries. It corrects an earlier statement by the authors (Theorem 6.4 in [13]). Whereas tractable $1\mathrm{RA}^-$ queries can be characterised efficiently by the hierarchical syntactic property, for UCQs no such efficient decision procedure is known. Further complexity results are known for inequality joins [19, 20] and queries with aggregates and group-by clauses [10].

The closest in spirit to the proof techniques in this paper are those for the UCQ dichotomy result [9]. The algorithm for tractable UCQ queries translates them into relational calculus expressions that have root variables and satisfy properties similar to what we call *canonicalised*. These properties are captured by the notion of *separator* variables. Similar to the case of root variables in our algorithm, the existence of a separator variable ensures that the annotations of the query expression are independent for different valuations of the separator variable. Our notion of $\exists$-consistency for queries with negation is inspired by the notion of inversion-freeness for UCQ queries.

The vast majority of hardness reductions in the above works are from the #P-hard model-counting problem for positive (2)DNF formulas [24, 21]. The complexity class #P was originally defined by Valiant [24].

OBDDs have been proposed by Bryant [4]. The first connection between polysize OBDDs and tractable queries has been shown for hierarchical non-repeating conjunctive queries [19]. The class of inversion-free UCQs is equivalent to the class of UCQ queries that admit polysize OBDDs [17]. UCQs with inequalities have also been characterised in terms of their corresponding OBDDs [16].

An overview of various topics in probabilistic databases has been compiled recently [23].

Acknowledgment. The authors would like to thank Dan Suciu for discussions on this work.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, **78**(1), 1991.

[3] V. Bárány, B. ten Cate, and M. Otto. Queries with guarded negation. *PVLDB*, 5(11), 2012.

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

[5] N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*, 2004.

[6] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, **16**(4), 2007.

[7] N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, 2007.

[8] N. Dalvi and D. Suciu. "The Dichotomy of Conjunctive Queries on Probabilistic Structures". In *PODS*, 2007.

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

[10] R. Fink, L. Han, and D. Olteanu. Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5), 2012.

[11] R. Fink, J. Huang, and D. Olteanu. Anytime approximation in probabilistic databases. *VLDB J.*, 22(6), 2013.

[12] R. Fink and D. Olteanu. A dichotomy for non-repeating queries with negation in probabilistic databases. Technical report, U. Oxford, 2014.

[13] R. Fink, D. Olteanu, and S. Rath. Providing Support for Full Relational Algebra Queries in Probabilistic Databases. In *ICDE*, 2011.

[14] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, 1998.

[15] J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: A probabilistic database management system. In *SIGMOD*, 2009.

[16] A. K. Jha and D. Suciu. On the tractability of query compilation and bounded treewidth. In *ICDT*, 2012.

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

[18] S. Khanna, S. Roy, and V. Tannen. Queries with difference on probabilistic databases. *PVLDB*, 4(11), 2011.

[19] D. Olteanu and J. Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, 2008.

[20] D. Olteanu and J. Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*, 2009.

[21] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4), 1983.

[22] C. Ré and D. Suciu. The Trichotomy of HAVING Queries on a Probabilistic Database. *VLDB J*, 18(5), 2009.

[23] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases.* Morgan & Claypool Publishers, 2011.

[24] L. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, **8**, 1979.

[25] T.-Y. Wang, C. Ré, and D. Suciu. Implementing NOT EXISTS predicates over a probabilistic database. In *QDB/MUD*, 2008.

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