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 MysiQ [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 relational symbols, equi-joins, projections, and difference (union not allowed). We denote this language by 1RA. By non-repeating we mean that a relation symbol can occur at most once in the query. We also discuss extensions of 1RA with negation. The tractable queries with negation are precisely the hierarchical ones and can be recognised efficiently.

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 1RA 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 \( X \times A \rightarrow B \ Y \) and the difference \( X - A \rightarrow B \ Y \) 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 1RA 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 characterisations are equivalent, the tractability and hardness proofs for 1RA 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
formula $\Psi$ and the query $Q$, we construct an input database whose input tuples are annotated with variables in $\Psi$ such that the result of $Q$ becomes annotated with $\Psi$. To count the models of $\Psi$, 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 $\Psi$.

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 $R(A), S(A, B), T(B)$ 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 $\Psi = x_1 y_1 \lor x_1 y_2$. The input relations and intermediate query results are shown in Figure 2. Each relation has a special column $\Phi$ that holds Boolean annotation formulas over variables in $\Psi$: 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 $\Psi$. 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 $\Psi$ 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.

Example 1 shows the power of negation: Our query $Q$ can compute $\#\Psi$ for any positive 2DNF formula $\Psi$ 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 [20]. 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 $D$ in Figure 3 is

$$\Psi = r_1([t_1(\neg u_1 \lor \neg v_1) \lor t_2(\neg u_1 \lor \neg v_2)] \lor \ r_2([t_1(\neg u_2 \lor \neg v_1) \lor t_2(\neg u_2 \lor \neg v_2)]).$$

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

We show in Section 5 that for every tuple-independent database $D$, the annotation of the result of $Q'$ on $D$ admits an OBDD of size $O(|D| \cdot f(Q))$, 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 $RC^3$. For $Q'$, this translation yields the $RC^3$ query

$$Q_{RC} = \exists A (R(A) \land \neg U(A)) \land \exists B T(B) \lor $$

$$\exists A R(A) \land \exists B (T(B) \land \neg V(B)),$$

where $Q_1$ and $Q_2$ are the formulas annotating the results of the two queries $Q_1$ and $Q_2$ on the database $D$ from Figure 3 are

$$\Psi_1 = (r_1 \land u_1 \lor r_2 \land u_2) \land (t_1 \lor t_2)$$

$$\Psi_2 = (r_1 \lor r_2) \land (t_1 \land v_1 \lor t_2 \land v_2),$$

and clearly $\Psi_1 \lor \Psi_2 \equiv \Psi$. The $RC^3$ 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 $\Psi_1$ and $\Psi_2$ admit OBDDs of size $O(|D|)$, as exemplified in the diagrams.
them via disjunctions and conjunctions. Constructed under the same global variable order, and it follows from classic results 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 \cite{DBLP:journals/vldb/LiC18} and a recent monograph \cite{DBLP:journals/corr/abs-1903-02453}. We next introduce a few necessary notions on the IRA and RC query languages and OBDDs.

The relational algebra query language IRA. We assume database schemas with unique attribute names. The set of attributes of a relation R is \( \text{sch}(R) \). A query \( Q \) is non-repeating if each relation symbol occurs at most once in \( Q \).

IRA is the class of non-repeating, union-free relational algebra queries composed of: Relation symbols; Equi-join: \( Q_1 \bowtie_{\rho} Q_2 \), where \( \rho \) is a conjunction of equality conditions \( \rho = (A_1 = B_1) \land \cdots \land (A_n = B_n) \) 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 \( \pi_{X} \) for a set \( X \) of attributes; Difference: \( Q_1 \setminus_{\rho} Q_2 \), where the attributes exported by \( Q_1 \) and \( Q_2 \) are \( \{A_1, \ldots, A_n\} \) and \( \{B_1, \ldots, B_n\} \) respectively, and \( \rho \) is the following conjunction of attribute mappings \( (A_1 \leftrightarrow B_1) \land \cdots \land (A_n \leftrightarrow B_n) \).

In \( Q_1 \bowtie_{\rho} Q_2 \) and \( Q_1 \setminus_{\rho} Q_2 \), we write \( A \in \rho \) to express that \( A \) contains an equality condition on \( A \), and \( (A=A') \in \rho \) or \( (A\leftrightarrow A') \in \rho \) to express that \( A \) 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 \( R(A_x), S(A_y, B_z), T(B_1) \) and then write the queries \( R \bowtie_{A, \ldots, A} S \) and \( (R \bowtie T) \setminus_{A, \ldots, A} S \) more concisely as \( R \bowtie S \) and \( (R \bowtie T) \setminus S \).

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 \( O \), \( Q \) has even polarity if the number of “−” operators between \( O \) (exclusive) and the root of \( Q \) (inclusive), for which \( O \) is a right descendant, is even, and has odd polarity otherwise. The pol function captures this notion: \( \text{pol}(Q, O) = 1 \) if \( O \) 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 \( E(Q) \), are defined recursively on the query structure:

- If \( Q = Q_1 \bowtie_{\rho} Q_2 \), then \( E(Q) = E(Q_1) \cup E(Q_2) \)
- If \( Q = Q_1 \setminus_{\rho} Q_2 \), then \( E(Q) = E(Q_1) \)
- If \( Q = \pi_{A}(Q_1) \), then \( E(Q) = \tilde{A} \)
- If \( Q = \sigma_{\rho}(Q_1) \), then \( E(Q) = E(Q_1) \)
- If \( Q = R \), then \( E(Q) = \text{sch}(R) \)

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

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

The relational calculus query language RC is the class of queries that are expressions \( \langle H \mid F \rangle \), where the query body \( F \) is a formula defined by the following grammar:

\[
F ::= R(\bar{X}) \mid \exists_X(F_1) \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg(F_1),
\]

and the query head \( H \) 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 \( Q \).

**Definition 2.** An RC 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 \( X \) of variables is a directed acyclic graph where inner nodes are labeled with variables from \( X \) and terminal nodes are true (T) and false (⊥). 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 \( \Pi \) on its variables such that the variables visited by each path are in \( \Pi \)-order. A level in an OBDD corresponds to all nodes labeled with the same variable. The width 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** Let \( \Phi_1, \Phi_2 \) be two formulas, \( \Pi \) be a fixed variable order on their variables, and \( O_1 \) and \( O_2 \) be OBDDs of width \( w_1 \) and \( w_2 \) for \( \Phi_1 \) and \( \Phi_2 \), respectively. Then, II-OBDDs for \( \Phi_1 \land \Phi_2 \) and for \( \Phi_1 \lor \Phi_2 \) can be constructed in time \( O(|O_1|+|O_2|) \) and have width at most \( w_1 \cdot w_2 \).

**Example 3.** Figure 1 shows three OBDDs under the same variable order \( r_1, u_1, r_2, u_2, 2, 1, v_1, t_2, v_2 \). Solid lines denote the true-edges and dotted lines the false-edges. The path \( r_1 \rightarrow \neg u_1 \rightarrow r_2 \rightarrow \bot \) encodes that under any truth assignment \( \nu \) with \( \nu(r_1) = T \) and \( \nu(\neg u_1) \equiv \nu(r_2) \equiv \bot \), the expression \( \Phi_1 = (r_1 \lor u_1) \land (r_2 \lor u_2) \land (t_1 \lor t_2) \) 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 QUERYs

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

\( \Rightarrow \) \( Q_{RA} \) is equivalent to an RC\(^3\) query \( Q_{RC} \) that is

RC-hierarchical and \( \exists \)-consistent

\( \Rightarrow \) \( Q_{RC} \) can be compiled into an OBDD of size \( O(|D| \cdot 2^{Q_{RC}(\bar{D})}) \)

**Corollary** The probability \( P_\Phi \) can be computed in \( O(|D| \cdot 2^{Q_{RC}(\bar{D})}) \).

The reason for translating 1RA queries to RC\(^3\) queries is that relational calculus is more flexible and allows to unfold negated expressions as per \( \neg (Q_1 \lor Q_2) \equiv \neg Q_1 \lor \neg Q_2 \). Since the 1RA query \( Q_{RA} \) and the RC\(^3\) query \( Q_{RC} \) are equivalent for any input database \( D \), 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 \( O(|D| \cdot 2^{Q_{RC}(\bar{D})}) \).

The RC\(^3\) query \( Q_{RC} \) is a disjunction of disjunction-free RC\(^3\) expressions. In contrast to \( Q_{RA} \), \( Q_{RC} \) may have repeating relation symbols. It is hierarchical in a syntactically more restricted sense:

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

**Definition 3.** An \( Q_{RC} \) 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 \( RC_3 \) query is canonicalised if each relation symbol occurs only with the same variable symbols. In addition, the \( RC_3 \) 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 \( RC_3 \) query is \( \exists \)-consistent if there exists a total order \( \geq 3 \) of the variable symbols in \( Q \) such that \( X \geq 3 \) \( Y \) implies that there is no sub-query of the form \( \exists Y \exists X(Q') \) in \( Q \).

Intuitively, \( \exists \)-consistency for an \( RC_3 \) query that is a conjunction or disjunction of sub-queries means that these sub-queries have compatible join orders (i.e., non-contradicting \( \geq 3 \) 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 \( RC_3 \) queries:

\[
Q_1 = \exists A (M(A) \land \neg R(A)) \land \exists B N(B) \\
Q_2 = \exists A M(A) \land \exists B N(B) \land \neg T(B) \\
Q_3 = \exists A (M(A) \land U(A)) \land \exists B (N(B) \land V(B))
\]

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 \( D \), viz:

<table>
<thead>
<tr>
<th>( M )</th>
<th>( N )</th>
<th>( R )</th>
<th>( T )</th>
<th>( U )</th>
<th>( V )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( \Phi )</td>
<td>( B )</td>
<td>( \Phi )</td>
<td>( A )</td>
<td>( \Phi )</td>
</tr>
<tr>
<td>1 ( m_1 )</td>
<td>1 ( n_1 )</td>
<td>1 ( r_1 )</td>
<td>1 ( t_1 )</td>
<td>1 ( u_1 )</td>
<td>1 ( v_1 )</td>
</tr>
<tr>
<td>2 ( m_2 )</td>
<td>2 ( n_2 )</td>
<td>2 ( r_2 )</td>
<td>2 ( t_2 )</td>
<td>2 ( u_2 )</td>
<td>2 ( v_2 )</td>
</tr>
</tbody>
</table>

The annotations \( \Phi_i \) of \( Q_i \) (\( i = 1, 2, 3 \)) evaluated on \( D \) are

\[
\Phi_1 = (m_1 \land \neg r_1) \land (n_1 \lor n_2) \\
\Phi_2 = (m_1 \lor m_2) \land (n_1 \lor n_2) \\
\Phi_3 = (m_1 \land u_1) \lor (m_2 \land u_2) \land (n_1 \lor n_2) \\
\]

and can be represented by OBDDs of width 2 under the respective variable orders \( \Pi_1, \Pi_2, \Pi_3 \):

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

Now consider the query \( Q_{123} = Q_1 \lor Q_2 \lor Q_3 \); this query is canonicalised, RC-hierarchical, and \( \exists \)-consistent. The variable orders \( \Pi_1, \Pi_2, \) and \( \Pi_3 \) are compatible in the sense that they can be extended into an order \( \Pi_{123} \) 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 4, the OBDDs of \( \Phi_1, \Phi_2, \) and \( \Phi_3 \) can be combined to yield an OBDD of width at most 2 for the annotation \( \Phi_1 \lor \Phi_2 \lor \Phi_3 \) of query \( Q_{123} \).
3.1 From 1RA\textsuperscript{−} to RC\textsuperscript{3}

At the core of the evaluation algorithm for hierarchical 1RA\textsuperscript{−} queries is a rewriting of 1RA\textsuperscript{−} queries into equivalent safe RC\textsuperscript{3} queries. The rewriting procedure \cite{ref} is the standard recursive inside-out translation from relational algebra to safe relational calculus (Lemma 5.3.11, \cite{ref}), with the addition that after each recursive translation step we “flatten” the resulting RC\textsuperscript{3} query as follows:

- Every \( \exists X \) operator is pushed as deep as possible in the RC\textsuperscript{3} query without pushing it past a \( \neg \) operator: \( \exists X \) distributes over disjunctions and is pushed past conjunctions in which \( X \) does not appear. Lemma \[\text{Lemma 3}\] 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 \( \neg(A \land B) \rightarrow \neg A \lor \neg B \) and its dual) as deep as possible in the RC\textsuperscript{3} query without pushing it past an \( \exists \) operator.
- Conjunctions of disjunctions are eagerly expanded into disjunctions as per

\[
(A \lor B) \land (C \lor D) \rightarrow AB \lor AC \lor BC \lor BD.
\]

Our translation has several desirable properties:

**Lemma 3.** For any hierarchical 1RA\textsuperscript{−} query \( Q_{1RA} \), the translated RC\textsuperscript{3} query \( Q_{RC} = [Q_{1RA}] \) satisfies the following:

1. \( Q_{RC} \) is equivalent to \( Q_{1RA} \).
2. \( Q_{RC} \) is canonicalised.
3. \( Q_{RC} \) is a disjunction of conjunction-free RC\textsuperscript{3} queries.
4. For every variable \( X \) in \( Q_{RC} \), \( Q_{RC} \) has no sub-query of the form \( \exists X(Q) \land Q'(\exists X) \); here, \( Q(\exists X) \) denotes a query \( Q \) in which \( \exists X \) occurs.
5. \( Q_{RC} \) is RC-hierarchical.
6. The quantifiers in \( Q_{RC} \) can be ordered such that \( Q_{RC} \) is \( \exists \)-consistent.

Condition \[\text{Condition (d)}\] permits sub-queries of the form \( \neg \exists X(Q) \lor \exists X(Q') \) or \( \exists X(Q) \lor \exists X(Q') \), but disallows, e.g., \( \exists X(Q) \land \exists X(Q') \lor \exists X(Q') \land \neg \exists X(Q) \).

**Example 5.** Consider the following two 1RA\textsuperscript{−} queries:

\[
Q_a = \pi_{\theta} \left[ M(A) \land N(B) \lor \left[ R(A) \land T(B) \lor U(A) \land V(B) \right] \right]
\]
\[
Q_b = \pi_{\theta} \left[ \pi_A \left( M(A) \land N(B) \right) \right]
\]

\[
- \pi_A \left[ R(A) \land T(B) \lor U(A) \land V(B) \right] \right] .
\]

Query \( Q_a \) translates to \( Q_{123} \) from Example 2 (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

\[
[Q_a] = \exists_A \left[ M(A) \land \neg R(A) \right] \land \exists_B N(B) \lor
\exists_A M(A) \land \exists_B N(B) \land \neg \exists_B T(B) \lor
\exists_A M(A) \land U(A) \land \exists_B N(B) \land \neg \exists_B T(B) \lor V(B) .
\]

Like \( Q_{123} \), the RC\textsuperscript{3} query \( [Q_a] \) 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\textsuperscript{3} queries: A conservative upper bound on their sizes would be a non-elementary function of the size of the input 1RA\textsuperscript{−} 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\textsuperscript{−} queries where for all projections \( \pi \rightarrow X(Q) \) 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 \( \Phi \) of the RC\textsuperscript{3} query \( Q_{RC} \) obtained from \( Q_{1RA} \) as per Lemma 3. This OBDD has a total order \( \Pi \) 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

\[
Q = \exists_X \left[ R(X) \land \exists_Y (S(X,Y) \land \neg T(X,Y)) \right] .
\]

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 \( R(a) \land \exists_Y (S(a,Y) \land \neg T(a,Y)) \); 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 \( b \) 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

\[
R(1), S(1,1), T(1,1), S(1,2), T(1,2), S(1,3), T(1,3), \ldots
\]

(tuples with \( X = 1 \))

\[
R(2), S(2,1), T(2,1), S(2,2), T(2,2), S(2,3), T(2,3), \ldots
\]

(tuples with \( X = 2 \)) and so on.

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\textsuperscript{3} query \( Q_{RC} \) that satisfies the properties of Lemma 3, the annotation \( \Phi \) of \( Q_{RC} \) on a tuple-independent database \( D \) can be represented by an OBDD of size \( O(|D| \cdot 2^{|Q_{RC}|}) \).
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 \( ADom(X_i) \) be the active domain of variable \( X_i \). The annotation of tuple \( A \) of relation \( R_i \) is denoted by \( R_i(A) \), 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 \( X_i \prec X_j \) \( \iff i < j \) with respect to the nesting order greater than \( > \) defined by the \( \exists \)-consistency of \( Q_{RC} \); that is, \( i < j \) allows for the quantifier nesting \( \exists x_i(\exists x_j) \), but not \( \exists x_j(\exists x_i) \). Since \( Q_{RC} \) is canonicalised and \( \exists \)-consistent (Lemma 4), we can assume without loss of generality that in each relation symbol \( R \) the query variables occur in \( > \) order (we can always relabel the query and database schema such that the query variables occur in \( > \) order). For example, \( Q_{RC} \) may contain \( R(X_1, X_2, X_3) \), 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 \( x_i \in ADom(X_i) \) and \( x_j \in ADom(X_j) \) it holds that \( x_i < x_j \iff i < j \); 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 \( \Pi \) on the annotations of the tuples in \( D \) as follows. We first associate with every annotation \( R(A) \) the string \( \text{string}(R(A)) = AR \), e.g., annotation \( R_2(A_7, B_2, C_7) \) is associated with the string \( A_7B_2C_7R_2 \). The order \( \Pi \) is then defined as

\[
R(A) <_\Pi R'(A') \iff \text{string}(R(A)) <_{\text{lex}} \text{string}(R'(A'))
\]

where \( <_{\text{lex}} \) 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 \( \Pi \) is uniquely defined by the order of the relation symbols and the order on the active domain of \( D \). However, different orders on the former and the latter give rise to different orders \( \Pi \).

We show by structural induction over \( \Phi \) that it has a II-OBDD of width \( 2^{\lceil Q_{RC} \rceil} \) where \( |Q_{RC}| \) denotes the number of relation symbols in \( Q_{RC} \):

- The base case is a relation symbol \( R(A) \) which corresponds to a trivial II-OBDD with one variable \( R(A) \) and width \( 2 \).
- If \( Q_{RC} = Q_1 \land Q_2 \) or \( Q_{RC} = Q_1 \lor Q_2 \), then by induction hypothesis the annotations of \( Q_1 \) and \( Q_2 \) have II-OBDDs of width \( 2^{\lceil Q_1 \rceil} \) and \( 2^{\lceil Q_2 \rceil} \), respectively. Then by Lemma 4, the annotation of \( Q_{RC} \) has a II-OBDD of width \( 2^{\lceil Q_1 \rceil} \cdot 2^{\lceil Q_2 \rceil} = 2^{\lceil Q_{RC} \rceil} \).
- If \( Q_{RC} = \neg Q \), then by induction hypothesis query \( Q \) has a II-OBDD of width \( 2^{\lceil Q \rceil} \). Swapping the \( \top \) and \( \bot \) nodes in this OBDD yields the required II-OBDD for \( Q_{RC} \).
- If \( Q_{RC} = \exists X_i Q \), then for every \( A_i \in ADom(X_i) \) the annotations \( \Phi_i \) of queries \( Q[A_i/X_i] \) are over disjoint sets of variables because \( Q_{RC} \) is RC-hierarchical by Lemma 4, and hence \( X_i \) is root in \( Q \). Moreover, each \( \Phi_i \) has a II-OBDD of width \( 2^{\lceil Q \rceil} \) by induction hypothesis. Let \( ADom(X_i) = \{A_i, \ldots, A_k\} \) such that \( A_k <_{\text{lex}} A_l \) if and only if \( k < l \). The annotation \( \Phi_i \) of \( Q_{RC} \) is the disjunction \( \bigvee_{A_i \in ADom(X_i)} \Phi_i \). Since the formulas \( \Phi_i \) are over disjoint sets of variables for distinct values of \( l \), an OBDD for their disjunction is obtained by their concatenation in which the \( \bot \) node of the OBDD for \( \Phi_i \) is replaced by the root node of the OBDD for \( \Phi_i+1 \).

It remains to show that this construction yields an OBDD over order \( \Pi \). First, note that the OBDD for each \( \Phi_i \) is over order \( \Pi \) by induction hypothesis; we next show that for any two annotations \( R(A_k) \) in \( \Phi_k \) and \( R'(A_l) \) in \( \Phi_l \) with \( k < l \), it holds that \( R'(A_k) <_\Pi R'(A_l) \); by the definition of \( <_\Pi \), this is equivalent to showing \( A_k R <_{\text{lex}} A_l R' \). The strings \( A_k \) and \( A_l \) 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 \( A_k \) and \( A_l \) — and hence the \( \Pi \)-order of \( R(A_k) \) in \( \Phi_k \) and of \( R'(A_l) \) in \( \Phi_l \) — is determined by the values of \( X_i \) in \( A_k \) and in \( A_l \); this value is \( A_i \) in \( A_k \) and \( A_i \) in \( A_l \). Since we concatenate the OBDDs in the order \( \Phi_1 \rightarrow \cdots \rightarrow \Phi_n \) and since \( A_1 <_{\text{lex}} \cdots <_{\text{lex}} A_n \) it follows that \( A_k <_{\text{lex}} A_l \) and thus \( R(A_k) <_\Pi R'(A_l) \). The constructed OBDD has width \( 2^{\lceil Q_{RC} \rceil} = 2^{\lceil Q \rceil} \), because the concatenation leaves the width unchanged.

The OBDD construction in the above proof shows that conjunction, disjunction, negation, and existential quantification of RC queries representing rewritings of 1RA 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 RC operations in the proof. A tighter bound can be obtained via a more specific analysis: Any non-repeating RC-hierarchical RC query \( Q \) admits an OBDD of width at most \( |Q| \) and size linear in the input database size and independent of the query size. This tighter bound on the OBDD width can be immediately extended to \( \exists \)-consistent conjunction and disjunction of such queries \( Q_1, \ldots, Q_k \). The resulting OBDD has width \( |Q_1| \cdot \cdots \cdot |Q_k| \), which is smaller than \( 2^{\lceil Q_1 \rceil + \cdots + \lceil Q_k \rceil} \) as used in the proof.

We can now use both Lemmata 4 and 3 to obtain the polynomial-time computation of query probability:

**Corollary 1.** (Lemmata 4, 3). Let \( Q_{RC} \) be a RC3 query satisfying the properties of Lemma 3. For any tuple-independent database \( D \), the probability of the query result \( Q_{RC}(D) \) can be computed in time \( O(|D| \cdot 2^{\lceil Q_{RC} \rceil}) \).

### 4. NON-HIERARCHICAL 1RA \(^-\) QUERIES

We show in this section the following result:

**Lemma 5.** The data complexity of any non-hierarchical 1RA query is \( \#P \)-hard.

**Proof.** Given a 1RA query \( Q \) and any 2DNF formula \( \Psi \), we use a reduction from the model-counting problem \( \#\Psi \) by means of a construction of a database \( D \) such that \( \Psi \) and the query result \( Q(D) \) 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 \( \bar{R}^{A_i[\cdot \sim B_j]} \), \( S[A_i][B_j] \), and \( T[B_j][\cdot \sim A_i] \) 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 be 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.

For a database construction scheme is possible such that the query result becomes annotated by $\Phi$. The proof steps are summarised as follows:

$$Q \text{ is non-hierarchical} \iff \text{Proposition 4} \iff Q \text{ has a match with a pattern in Figure 5} \iff \text{Lemma 5} \iff Q \text{ has an annotation-preserving match with a pattern} \Rightarrow \text{Lemma 6} \iff Q \text{ is hard for } \#P.$$

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, $A$ and $B$, and a constant $\Box$ distinct from those in $A$ and $B$. In this section, the projection operator $\pi_\Box$ is used to symbolise the projection on attribute $A$ and the annotation column $\Phi$; in contrast, $\pi_A$ 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 $\Phi$ be a total function on $A$. A relation $Q$ is $A$-reducible to $(A, \Phi)$ if the $[A]$-attributes of $Q$ are filled with all values from $A$, all non-$[A]$-attributes are filled with $\Box$, and the annotation of a tuple identified by $a \in A$ is $\Phi(a)$:

$$\pi_{[A]}(Q) = \{(a|\Phi(a)) \mid a \in A\}$$
$$\pi_C(Q) = \{\Box\} \quad \text{for any attribute $C$ with $C \notin [A]$.}$$

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

$$\pi_{[A]}(Q) = \{\Box\} \quad \text{for any attribute $C$.}$$

We next define three classes of relations $Q^A$, $Q_{\Box}$, and $Q_\Phi$ that are characterised by their $A$-reductions; let $\Phi_T$ be the constant function $\Phi_T(\cdot) = \top$.

$$Q^{[A]} \in Q^A \quad \text{if } \text{red}_A(Q) = A\Phi \quad (1)$$
$$Q^{[A]} \in Q_{\Box} \quad \text{if } \text{red}_A(Q) = A\Box \Phi_T \quad (2)$$
$$Q^{[\neg A]} \in Q_{\Box} \quad \text{if } \text{red}_\Phi(Q) = \Box\Phi_T \quad (3)$$
$$Q \in Q_\Phi \quad \text{if } Q = \emptyset \quad (4)$$

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

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

- $Q^A$-relation $X$

<table>
<thead>
<tr>
<th>$A_x$</th>
<th>$B_x$</th>
<th>$C_x$</th>
<th>$\Phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\top$</td>
</tr>
</tbody>
</table>

- $Q_{\Box}$-relation $Y$

<table>
<thead>
<tr>
<th>$A_y$</th>
<th>$B_y$</th>
<th>$C_y$</th>
<th>$\Phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\top$</td>
</tr>
</tbody>
</table>

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

Figure 6 shows how $Q^A$, $Q_{\Box}$, and $Q_\Phi$-queries are propagated through query operators: Given query classes $Q_1$ and $Q_2$, the right-most column $(Q_1 \text{ Op } Q_2)$ in the table shows the class to which a query that combines two queries from those respective classes by operator $\text{Op}$ belongs.

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

- $Q^A$-query $X \bowtie Y$

<table>
<thead>
<tr>
<th>$A_x$</th>
<th>$A_y$</th>
<th>$B_x$, $B_y$</th>
<th>$C_x$, $C_y$</th>
<th>$\Phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_1$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$x_2$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$x_3$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\top$</td>
</tr>
</tbody>
</table>
Figure 6: Class membership of queries connecting classes $Q^A$, $Q_{hll}$, and $Q_\emptyset$ with operators $\Join_1 \ldots$.

This join satisfies the conditions of a $Q^A$-query as suggested by the rule $Q^A \Join Q_{hll} \rightarrow Q^A$ in Figure 5. Similarly, the difference of $Y$ and $X$ is also a $Q^A$-query:

<table>
<thead>
<tr>
<th>$Q_1$</th>
<th>$Op$</th>
<th>$Q_2$</th>
<th>$Q_3$</th>
<th>$Op$</th>
<th>$Q_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q^A$</td>
<td>$\Join_1$</td>
<td>$Q_{hll}$</td>
<td>$Q^A$</td>
<td>$-$</td>
<td>$Q_\emptyset$</td>
</tr>
<tr>
<td>$Q^{A,B}$</td>
<td>$\Join_1$</td>
<td>$Q_{hll}$</td>
<td>$Q^{A,B}$</td>
<td>$-$</td>
<td>$Q_\emptyset$</td>
</tr>
<tr>
<td>$Q_{hll}$</td>
<td>$\Join$</td>
<td>$Q^{A,B}$</td>
<td>$Q^A$</td>
<td>$-$</td>
<td>$Q_\emptyset$</td>
</tr>
</tbody>
</table>

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

**Lemma 6.** Given a query $Q$, a distinguished attribute $A$ of $Q$, and a distinguished relation $X^{[A]}$ of $Q$ that satisfies Equation (4), the remaining relations of $Q$ can be filled such that $Q$ satisfies Equation (4).

**Proof.** We first identify the set $OP_-$ of difference operators in $Q$ that do not have $X$ as a right descendant and partition the relations of $Q$ among three sets:

- $rels_X = \{X\}$
- $rels_{\emptyset} = \text{relations right descendants of an } OP_- \text{ operator}$
- $rels_{hll} = \text{all other relations}$

We populate each $rels_{hll}$ relation as a $Q_{hll}$-query, and every $rels_{\emptyset}$ relation as a $Q_\emptyset$-query. For the former, it suffices to populate each $[A]$ attribute of a $rels_{hll}$-relation with $A$, and each non-[A]-attribute with $\emptyset$. The following inductive argument shows that every operator on the path in $Q$ between $X$ and the root of $Q$ is a $Q^A$-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:

- $Q_L \Join Q_R$, where without loss of generality $Q_L$ contains $X$. Then, $Q_L$ is a $Q^A$-query, $Q_R$ contains a relation from $rels_{hll}$ and is a $Q_{hll}$-query. Hence, $Q_L \Join Q_R$ is a $Q^A$-query.
- $Q_L \rightarrow Q_R$, where $Q_L$ contains $X$. Then the difference operator is in $OP_-$. $Q_R$ is a $Q_\emptyset$-query, $Q_L$ is a $Q^A$-query, and hence $Q_L \rightarrow Q_R$ is a $Q^A$-query.
- $Q_L \rightarrow Q_R$, where $Q_R$ contains $X$. Then, $Q_R$ is a $Q^A$-query, $Q_L$ contains a relation from $rels_{hll}$ and is a $Q_{hll}$-query. Hence, $Q_L \rightarrow Q_R$ is a $Q^A$-query.

If $X$ has even polarity in $Q$, then the annotation $\Phi_Q(a)$ of a tuple $(a)$ in $\pi_{A}(X)$ 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 $\Phi^{A\,B}$ be a total function on $A \times B$, and let $\Phi^A$ be a total function on $A \cup B \times B$ such that $\Phi^A(a) \equiv \bigvee_{b \in B} \Phi^{A\,B}(a, b)$ for all $a \in A$. As before, a relation $Q$ is $A$-reducible to $(A, \Phi^A)$ if

$$\pi^A_{\Phi}(Q) = \{(a|\Phi^A(a)) \mid a \in A\}$$ $$\pi^C_{\Phi}(Q) = \{(\emptyset)\} \text{ for any attribute } C \text{ with } C \notin [A].$$

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

$$\pi_{[A]\,[B]}(Q) = \{(a,b|\Phi^{A\,B}(a,b)) \mid a \in A, b \in B\}$$ $$\pi_C(Q) = \{(\emptyset)\} \text{ for any attribute } C \text{ with } C \notin [A] \cup [B].$$

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

- $Q^{[A]\,[B]} \in Q^A$ if $\text{red}_{A}(Q) = A|\Phi^A$ (5)
- $Q^{[A]\,[B]} \in Q^{A\,B}$ if $\text{red}_{AB}(Q) = A \times B|\Phi^A$ (6)
- $Q^{[A]\,[B]} \in Q^{A\,B}$ if $\text{red}_{AB}(Q) = A \times B|\Phi^{A\,B}$ (7)
- $Q^{[A\,B]} \in Q_{hll}$ if $\text{red}_{A}(Q) = A|\Phi^A$ (8)
- $Q^{[A\,B]} \in Q_{hll}$ if $\text{red}_{AB}(Q) = A \times B|\Phi^A$ (9)
- $Q^{[A\,B]} \in Q_{hll}$ if $\text{red}_{B}(Q) = B|\Phi^A$ (10)
- $Q \in Q_{\emptyset}$ if $Q = \emptyset$ (11)

In Equations (5)–(9), $\Phi^A$ and $\Phi^{A\,B}$ can also be negated. Queries from these classes are propagated by query operators as depicted in Figure 8. Lemma 2 can be extended to the case of two attributes $A$ and $B$:

- For a distinguished relation $X^{A\,-B}$ 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 $X^{A\,B}$ 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 $Op_1, Op_2 \in \{\times, \ldots\}$ 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 $\times$ or $\rightarrow$, and there are 6 possible orders of the labels $A, AB, B$. Figure 8 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 $A, B, AB, Op_1, Op_2$ to relations $R^{[A]\,[B]}$, $T^{[A]\,[B]}$, $S^{[A\,B]}$, and operators $Op_1$ and respectively $Op_2$ in the parse tree of $Q$ that preserves ancestor-descendant relationships.
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 $O_p$ is left-deep in a 1RA$^-$ query $Q$ if $O_p$ is a left descendant of every difference operator on the path between the root of $Q$ and $O_p$.

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 $O_{p_{-}}$ in $Q$, if $O_{p_{1}}$ is a right descendant of $O_{p_{-}}$, then $O_{p_{-}}$ does not export $[A]$ or $[B]$.
3. If $O_{p_{2}}$ is a left descendant of $O_{p_{1}}$ in $Q$, then $O_{p_{2}}$ is left-deep in the sub-query rooted at $O_{p_{1}}$.

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 8 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 $a_1, \ldots, a_n$ 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 $a_1, \ldots, a_n$. 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 $Q_3$ 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 $\#2$DNF problem: Let $Q$ be a query that is an annotation-preserving $(R, S, T)$-match for a pattern $P$, and let $\Psi = \bigvee_{(i, j) \in E} x_{i,j}$ be a 2DNF formula with $|E|$ clauses over disjoint variable sets $X$ and $Y$. 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 $\Psi$ and hence $P_{Q(\mathcal{D})} = P_\Psi = \#\Psi \cdot 2^{-|\vars(\Psi)|}$, 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. 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. 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 $P_{4.3}$...
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 $\Psi$ as a database $D$ such that the annotation of the query result $Q(D)$ is $\Psi$, if the polarity of $Op_2$ is odd in $Q_{RT}$. In case of even polarity, we derive a database $D$ and another formula $\Upsilon$ from $\Psi$ such that $P_{Q(D)} = P_{\Upsilon}$ and linearly many calls to an oracle for $P_{\Upsilon}$ suffice to compute $\#\Psi$.

**Case 1: Odd polarity** ($\pol(Q_{RT}, Op_2) = 1$). We fill the relations $R,S,T$ such that $Q_R$ is a $Q^A$-query, $Q_T$ is a $Q^B$-query, and $Q_S$ is a $Q^{AB}$-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 $x_i \in X$ that occurs in $\Psi$; $T$ consists of a tuple with $B$-value $y_i$ and annotation $y_i$ for each variable $y_i \in Y$ that occurs in $\Psi$; $S$ consists of a tuple with $(A,B)$-values $(x_i,y_j)$ and annotation $\top$ for each clause $x_i y_j$ in $\Psi$. Note that when used outside annotations, 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 $X$ and $Y$ and annotations $\top$. In all of the above cases, any attribute that is neither in $[A]$ nor in $[B]$ is filled with constant $\top$.

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 $Q^{AB}$-queries with annotations

$$\text{red}_{AB}(Q_{RT}) = X \times Y \mid \Phi_{RT}, \Phi_{RT}(x_i, y_j) = x_i y_j$$

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

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

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

$$\#\Psi = \sum_{\theta \in \Theta_0} 1 + \cdots + \sum_{\theta \in \Theta_{|E|}} 1 = |\Theta_0| + \cdots + |\Theta_{|E|}|.$$  

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

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

We fix the probabilities of variables in $X$ and $Y$ to 1/2 and of variables in $Z$ 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 $\Psi$:

$$P_{\neg \Upsilon} = \sum_{k=0}^{|E|} P(\text{exactly } k \text{ clauses of } \Psi \text{ are satisfied}) \cdot P(\text{exactly } k \text{ clauses of } \Psi \text{ are satisfied}) \cdot \frac{1}{2^{|X| + |Y|} \sum_{k=0}^{|E|} p_z^{E-k} \cdot |\Theta_k|}$$

Intuitively, the first term simplifies to $p_z^{E-k}$, because if exactly $k$ clauses $\psi_i$ 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 $(p_z, P_{\Upsilon})$ using Lagrange’s polynomial interpolation formula. We conclude that $1 - P_{\Upsilon} + 1$ oracle calls to $P_{\Upsilon}$ suffice to determine $\#\Psi = \sum_{i=0}^{|E|} |\Theta_i|$.

It remains to show how $\Upsilon$ can be encoded as the annotation of a query that is an annotation-preserving match of $P_{3,3}$, given this encoding, any algorithm that evaluates $P_{Q(D)}$ constitutes the above oracle. Formula $\Upsilon$ is encoded using the database construction scheme from Case 1, where the annotation of a tuple with $(A,B)$-values $(x_i,y_j)$ corresponds to clause $\psi_k = x_i y_j$ in $\Psi$ 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 $\Upsilon$.

**Reduction for pattern $P_{3,3}$**. We use the illustration of a query matching $P_{3,3}$ 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 $Op_p = \pi_{-[B]}$ such that every operator between $Op_p$ and $Op_1$ exports $[A]$ but not $[B]$, and every operator between $Op_p$ and $Op_2$ exports $[A]$ and $[B]$. Let $Q_S$ be the sub-query rooted at $Op_p$. The first step is to show that one may without loss of generality assume that $Op_p$ is left-deep in $Q_S$. Assume to the contrary that there is a difference operator $Op_\text{red}$ between $Op_p$ and $Op_2$ that has $Op_2$ as a right descendant; clearly, $Op_\text{red}$ exports $[A]$ and $[B]$ and hence its left sub-query contains relations $X^{[-][B]}$ and $Y^{[-][B]}$, or it contains a relation $Z^{[-][B]}$. In the former case, $Q$ is an annotation-preserving $(R,S,Y)$-match of pattern $P_{3,4}$; in the latter case, $Q$ is an
Figure 9: Relations R, S, T for the hardness reduction of a query with an annotation-preserving match for pattern $P_{0.3}$ where (1) $O_{p_1}$ does not export $[B]$ and (2) the projection operator $\pi_{-[B]}$ on the path between $O_{p_1}$ and $O_{p_2}$ 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 $\Psi = \psi_1 \lor \psi_2 = x_1 y_1 \lor x_1 y_2$.

annotation-preserving $(R, Z, T)$-match of pattern $P_{0.4}$. In both cases, the new $O_{p_2}$ is $O_{p_1}$, and left-deep in $Q_\Psi$.

Next, two cases can be analysed separately depending on the polarity of $O_{p_4}$ in $Q_{ST}$.

**Case 1: Even polarity** $(\text{pol}(Q_{ST}, O_{p_4}) = 0)$. Let $N = \{1, \ldots, |E|\}$ be the set of integers that numbers consecutively the clauses in $\Psi$: $\Psi = \psi_1 \lor \cdots \lor \psi_{|E|}$. We set relation $R$ to contain a tuple $(n, v)$ annotated with $\top$ for every clause number $n \in N$. Relation $S$ contains all tuples $(n, v)$ where $n \in N$ is a clause number and $v \in X \cup Y$ is a variable from $\Psi$; $(n, v)$ 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 in $\Psi$. Figure 9 exemplifies how $R$, $S$, $T$ are filled for a query matching $P_{0.3}$ and for formula $\Psi = x_1 y_1 \lor x_1 y_2$ and how these annotations are propagated through the query.

**Case 2: Odd polarity** $(\text{pol}(Q_{ST}, O_{p_4}) = 1)$. 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 $Q^H$-query, $Q_S$ is a $Q^A$-query, $Q_R$ is a $Q^A$-query, and then $Q_{ST}$ is a $Q^A$-query, where for relations $R$ and $T$ the annotation functions are the identity and for relation $S$, the annotation function is $\top$ for all tuples $(x_i, y_j)$ corresponding to clauses in $\Psi$ 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 3-consistent. For instance, the non-repeating relational algebra query $Q = \pi_0[S(A, B) \land (R(A) \land S_2(A, B))]$ has two hierarchical union-free sub-queries under $\pi_0$: $S(A, B) \land (R(A) \land S_2(A, B))$ and $S(A, B) \land (T(B) \lor S_2(A, B))$. However, these sub-queries cannot be rewritten to 3-consistent RC3 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 3-consistency and the RC²-hierarchical property of the RC³ expression $Q_\Psi$ representing the rewriting of a non-repeating relational algebra query $Q$ described in Section 5.1. Then $Q$ is tractable when $Q_\Psi$ is 3-consistent and RC-hierarchical. Checking these properties can be done efficiently in the size of the input RC³ query, yet $Q_\Psi$ 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 3-consistency, which incurs the non-trivial time to rewrite the input query.

5.2 Non-repeating RC³

There are subtle differences between 1RA⁻ and non-repeating RC³ that revolve around RC³’s flexibility to allow disjunction and negation on sub-queries of different schemas. For instance, the non-repeating RC³ queries $S(x, y) \land \neg R(x)$ and $S(x, y) \land (R(x) \lor T(y))$ 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 RC³.

For non-repeating RC³, the RC-hierarchical property alone does not characterise the tractable queries, even when we take away disjunction. Indeed, the RC³ query equivalent to the 1RA⁻ query from Figure 3, i.e., $Q = \exists_3 \exists_B R(A) \land S(B) \land (U(A) \lor V(B))$, 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 RC³ query $Q$ into an RC² query $Q'$ as outlined in Section 3.1. e.g., $Q' = \exists_3 [R(A) \land U(A)] \land \exists_3 [S(B) \land \neg V(B)]$ for the above query $Q$, and then again $Q'$ is tractable when $Q'$ is RC-hierarchical and 3-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 \cite{dalvi2007efficient} \cite{dalvi2004efficient}. Further work looks at approximating queries with negation \cite{wegener2004approximation}.

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 \cite{dalvi2004efficient}, polynomial-time/\#P-hard dichotomies have been established by Dalvi and Suciu for non-repeating conjunctive queries \cite{dalvi2004efficient} and unions of conjunctive queries (UCQs) \cite{dalvi2007efficient}; a trichotomy has been proven for positive queries with HAVING aggregates \cite{fink2013aggregation}; the precise tractability frontier for so-called量化 queries such as relational division and set equivalence, which can be expressed as repeating queries with nested negation, is also known \cite{fink2013aggregation}. Our result strictly generalises the dichotomy for non-repeating conjunctive queries. It corrects an earlier statement by the authors (Theorem 6.4 in \cite{fink2013aggregation}). Whereas tractable IRA\textsuperscript{−} 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 \cite{fink2013aggregation} and queries with aggregates and group-by clauses \cite{fink2012anytime}.

The closest in spirit to the proof techniques in this paper are those for the UCQ dichotomy result \cite{dalvi2007efficient}. 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 \(3\)-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 \cite{valiant1979the}. \#P was originally defined by Valiant \cite{valiant1979the}.

OBDDs have been proposed by Bryant \cite{bryant1986graph}. The first connection between poly-size OBDDs and tractable queries has been shown for hierarchical non-repeating conjunctive queries \cite{bryant1991graphs}. The class of inversion-free UCQs is equivalent to the class of UCQ queries that admit poly-size OBDDs \cite{bryant1991graphs}. UCQs with inequalities have also been characterised in terms of their corresponding OBDDs \cite{bryant1986graph}.

An overview of various topics in probabilistic databases has been compiled recently \cite{wegener2004approximation}.

\textbf{Acknowledgment.} The authors would like to thank Dan Suciu for discussions on this work.

7. REFERENCES


\cite{barany2012queries} V. Bárány, B. ten Cate, and M. Otto. Queries with guarded negation. \textit{PVLDB}, 5(11), 2012.


\cite{fink2013aggregation} R. Fink, L. Han, and D. Olteanu. Aggregation in probabilistic databases via knowledge compilation. \textit{PVLDB}, 5(5), 2012.


\cite{provan1983complexity} J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. \textit{SIAM J. Comput.}, 12(4), 1983.

\cite{real2009trichotomy} C. Ré and D. Suciu. The Trichotomy of HAVING Queries on a Probabilistic Database. \textit{VLDB J}, 18(5), 2009.

